UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE MATERIALES
PCeIM

HIDRODINÁMICA DE OBJETOS EMBEBIDOS EN FLUJOS FUERTES.

TESIS
QUE PARA OPTAR POR EL GRADO DE
DOCTOR EN CIENCIAS E INGENIERÍA DE MATERIALES

PRESENTA:

M. EN C. ALFREDO SANJUAN SANJUAN

TUTOR PRINCIPAL

DR. ENRIQUE GEFFROY AGUILAR
INSTITUTO DE INVESTIGACIONES EN MATERIALES

COMITÉ TUTOR:

DR. ANTONMARÍA MINZONI ALESSIO, IIMAS
DR. MARCO ANTONIO REYES HUESCA, F. I.

Ciudad Universitaria, Cd. Mx., mayo de 2017

JURADO ASIGNADO:

Presidente:	Dr. Fermín Alberto Viniegra Heberlein
Primer vocal:	Dr. Enrique Geffroy Aguilar
Segundo vocal:	Dr. Francisco Javier Solorio Ordaz
Tercer Vocal:	Dra. Catalina Stern Forgach
Secretario:	Dr. Panayiotis Georgios Panayotaros

Lugar donde se realizó la tesis:

INSTITUTO DE INVESTIGACIONES EN MATERIALES, UNAM

AGRADECIMIENTOS

Este trabajo lo dedico a todas las personas que me han apoyado a lo largo de este proyecto, sin su ayuda habría sido imposible culminarlo.

Quiero agradecer a mi tutor, Enrique Geffroy Aguilar, quién me brindó su apoyo incondicional desde que pisé por primera vez su oficina. Estos agradecimientos también son dedicados a Tim Minzoni Alessio y a Marco Antonio Reyes Huesca, mi comité tutor, por esas charlas en torno al proyecto realizado. Además, los tres no solo fueron un apoyo académico, sino que a nivel personal me dejaron ser parte de su vida como amigos y se los agradezco.

Quiero agradecer a los sinodales por la retroalimentación recibida para finalizar esta tesis, gracias a Catalina Stern, Francisco Solorio, Panayiotis Panayotaros y a Fermín Viniegra.

Quiero agradecer a mi familia, a mis padres Ambrosio y Paula por la paciencia que han tenido a lo largo de estos años. A mis hermanos Rosario y Gerardo; y a mi esposa Luz Marcela.

En estos años que estuve desarrollando la investigación de este trabajo he conocido a muchas personas en el grupo del Laboratorio de Reología Óptica, a todos ustedes les dedico esta tesis y les agradezco su ayuda. A Israel, por tener la paciencia de explicarme cada detalle de los experimentos que realizó en el TRM, a Minerva, a Carlos, Joana, Eduardo, Liz, Laylet y a las personas que, aunque no son parte formal del grupo han estado con nosotros, gracias Sofía, Itzel, Mildred, y Violeta.

Por último, quiero agradecer el apoyo ante el comité académico que tuvo Raúl Herrera Becerra.

Gracias a todos por el apoyo incondicional.

ABSTRACT

Two phase flows are important in many industrial application covering food and cosmetic emulsions, transport of particulate fluids, exploitation of crude oil, among many others. On the long term, the relevance of multiphase flows may be even more significant, in at least two branches of knowledge. Material science advances will require a detailed analysis of processes with systems composed of more than one phase, especially their surface energies or interfacial tensions, which play an important role when particles smaller that a micron are involved. Also, which may become most relevant, is the capture and sequestration of CO_2 —by adsorption methods— demanding a better understanding of multiphase equilibria and transport processes in multiphase flows, regardless of the use of liquids or zeolites as substrates.

However, in order to propose a model for the macroscopic description of any of these possible applications —via simple rheological constitutive equations— having a detailed balance of microscopic stress fields is essential. In general, the quest for detailed dynamics of two-phase flows is still a rather out-of-reach endeavor. Thus, a rather complex, associated problem but much simpler than the complete flow is a detailed study of *the micro-hydrodynamics of a single drop* embedded in a shearing flow. Although this problem may seem at quick sight of a rather simplified nature, it already addresses many of the relevant phenomena and basic principles valid for the large set of problems mentioned above. Thus, it is within this field that the objective of my work is set; and I attempt to provide a detailed view of some prevailing questions using numerical simulations.

A complete understanding of the particle or drop dynamics in the suspending fluid is still lacking mainly because of the non-linear nature of the occurring phenomena, especially due to an interface that is deformable by the stress field. In particular, complex phenomena are observed whenever large deformation of the drops occurs, which are most frequently encountered with strong flows —with rates of elongation greater that the rate of vorticity.

Thus, the issue of this thesis was to observe the drop deformation of drops immersed in *elongational flows with vorticity*, in contrast to the previous work published to date that addresses problems in flows such as *simple shear flows*, or 2-, 3-dimensional purely elongational flows. These flows hardly cover the flow regimes regularly found in

nature or encountered in applications and provide a limited insight into the ample spectrum of possible deformation dynamics of drops. Here I present the predicted dynamics of drops for a large family of flows that encompasses all the previously mentioned flows under a variety of parameters of relevance for applications, such as interfacial tension, ratio of viscosities of the fluids, strength of the flow and type of applied flow.

Using comparisons between my numerical results with earlier experimental, numerical and theoretical studies (Acrivos & Lo, 1978), (Bentley & Leal, 1986b), (Kwak & Pozrikidis, 1998), (Reyes, 2005), (Ha & Leal, 2001), (Rosas I. Y., 2013), and (Zhao & Shaqfeh, 2011), I attempt to offer a systematic and complete overview of all results consistent with the basic hydrodynamic equations. In particular, I address the question of whether deformed drops present middle cross-section of a circular form, or whether the prevalent form is an ellipsoidal shape. Once this point is settled on the most general shape, the next point addressed is whether these asymmetries imply different relaxation mechanisms for the different measures of deformation for the main axes. Furthermore, with this information, an attempt to evaluate the interfacial tension of the drop for a retracting drop is carried out.

Subsequently, I attempt to describe the dependence of the dynamics of these different shape forms on parameters such as the viscosity ratio, the shear rate of the flow, the capillary number, and the parameters that describe the flow types (here evaluated). With these sets of data, for the time-evolution of drop deformations, is possible to elucidate the existence of a large class of drop forms, as well as the associated set of solutions for these two-phase flows. The surprising aspect of these results is that drop shapes are not only non-circular, but stable form of drop exists that resemble more a *guarache* shape (a highly flattened out drop), which in turn implies the possibility of a large class of internal and external flows, and thus imply a rather complex impact on the stress fields of the fluid not previously evaluated. Here, the implication for appropriate macroscopic models for the internal structure of these rather simple flows could be a serious setback when assuming the possibility of simple models. Nonetheless, I attempt to provide some discernment of about the topics mentioned above.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

CHAPTER 1.

Study of the deformation of a drop

In the science of materials, multiphase fluids, *i.e.*, fluids composed by two or more different substances, are frequently found in industrial applications, *e.g.* crude oil or pharmaceutical industry, et cetera. Some examples of application of two-phase fluids are: (a) industrial separators, whose function is to separate various substances; (b) sprays or reactors, to manipulate several substances in order to make another fluid that combines properties of the separate phases, or to produce another (new) material, (c) polymeric manipulation or (d) analysis of multiphase microfluids as DNA analysis. These applications motivate the study of multiphase fluids.

In particular, these complex systems are classified as emulsions —whenever the two phases are liquids—, dispersions (for a liquid phase that supports solid particles), or colloids —when particles are smaller than a few microns. Understanding of the behavior of these systems requires assuming a different subset of basic physical phenomena and is of importance for many relevant applications. The variety of phenomena observed implies a rather complex nature being the result of changes in the structure of the flow at multiple scales of length and time, simultaneously.

However, these observed phenomena imply an extreme complexity, their full physical understanding has not been up to date amenable. Rather, a more limited set of parameters or experimental phenomena has been the acceptable approach, albeit with a more limited accessible scope. A possible starting point of relevance for the

above observations —(Taylor, 1932), (Lamb, 1945), (Batchelor, 1967)— is the *study of deformation of a single drop induced by an equally simple flow.* This has been the starting pattern for theoretical studies since the seminal Ph. D. thesis of A. Einstein, the experimental work by G. I. Taylor, and many important numerical simulations carried out in the previous decades. Thus, this work focuses on *the fluid mechanics of a two-phase fluid*: a pair of immiscible fluids that may have different values for the relevant properties as will be discussed in the next Section.

The earliest studies of multiphase fluid showed the extreme difficulty of this topic of research. Lord Rayleigh in 1879 studied the capillary effects in jets, (Rayleigh, 1879) observing the detailed instabilities of two phase flows. Albert Einstein made a treatise to analyze a simple prototype of multiphase fluid (Einstein, 1906). In Einstein's work, the viscosity of a suspension of little spherical particles immersed in a continuum fluid is reported —these rigid solid spheres could model the presence of little drops with an enormous viscosity compared to that of the continuum fluid. The next seminal work comes until 1932 when Taylor used a more detailed description of the second phase (Taylor, 1932). Taylor assumed that particles were not rigid, as Einstein did, presenting a theoretical analysis of a *2D-flow* applied on the continuum phase, with a single particle immersed. This approximation is the simplest configuration of a two-phase fluid which encompasses the basic fundamental phenomena, and which in turn incorporates the mathematical methods required for such line of work.

With these previous reports, the detailed study of the drop dynamics —a single drop of a fluid called the disperse phase— embedded in a second fluid —called the continuum phase— established one of the basic tools for further advances. Sir Geoffrey Ingram Taylor said in the first treatise of experimental device of drop deformation that "When one liquid is at rest in another liquid of the same density it assumes the form of the spherical drop. Any movement of the outer fluid (apart from pure rotation or translation) will distort the drop owing to the dynamical and viscous forces which then act on its surface", (Taylor, 1934). It is assumed that the two fluids are immiscible, and their interaction is described by the forces that generate the deformation of the interface separating both fluids. Since this work, the study of drop deformation has been fundamental to understanding the fluid mechanics of multiphase fluids phenomenon, and here I expand our understanding of the applied forces to those induced by strong flows that are well describe for the continuum phase.

Many important subsequent theoretical studies to Taylor (Taylor, 1932), have been published by R. Cox, Barthès-Biesel, A. Acrivos, E. J. Hinch, and J. M. Rallison among the most relevant during the last century: (Taylor, 1964), (Cox, 1969), (Frankel & Acrivos, 1970), (Barthès-Biesel, 1973), (Acrivos & Lo, 1978), (Rallison, 1978), (Astarita, 1979), (Hinch & Acrivos, 1979), (Rallison, 1980), (Hinch & Acrivos, 1980), (Brady & Acrivos, 1982), and (Rallison, 1984). At the same time, experimental devices were used to observe drop deformations immersed in a fluid within a well-controlled flow. Especially, for *simple shear flow* machines: (Rumscheid & Mason, 1961), (Torza, Cox, & Mason, 1972), (Grace, 1982), (Guido & Villone, 1997), (Guido, Greco, & Villone, 1999), (Guido & Villone, 1999), (Mo, 2000), (Guido & Greco, 2001), (Wannaborworn, Mackley, & Renardy, 2002), (Yu, Bousmina, & Zhou, 2004). As well as flows of an elongational type: (Taylor, 1934), (Bentley & Leal, 1986a), (Bentley & Leal, 1986b), (Stone, Bentley, & Leal, 1986), (Stone & Leal, 1989a), (Stone & Leal, 1989b), (Ha & Leal, 2001) (Rosas, Reyes, Minzoni, & Geffroy, 2014), (Escalante, Reyes, Rosas, & Geffroy, 2015), and (Rojas, 2016). These two type of flows complement each other, however showing rather distinctive and unique phenomena which depends on the parameter space; many of these features are discussed in this thesis.

The previous studies showed that the flow type may lead to a systematic classification of the shapes of the drops, with more cylindrical shapes for elongational types of flows —an elongated prolate with a quasi-circular waist—, while the *simple shear flows* cause deformations that flatten the waist of the ellipsoid along the direction of vorticity of the flow. These complex shapes appear to require multiple solutions to the governing equations, which are difficult to probe experimentally and very difficult to solve with the theoretical tools currently available.

Finally, numerical models have been used to study those cases when the experimental device or the theoretical predictions cannot fill gaps in our understanding of the drop dynamics: (Youngren & Acrivos, 1975), (Rallison & Acrivos, 1978), (Rallison, 1981), (Unverdi & Tryggvason, 1991), (Kennedy, Pozrikidis, & Skalak, 1994), (Loewenberg & Hinch, 1996), (Zinchenko, 1997), (Pozrikidis, 1997), (Kwak & Pozrikidis, 1998), (Coulliette & Pozrikidis, 1998), (Primo, Wrobel, & H., 2000), (Khayat, 2000), (Pozrikidis, 2000), (Yuriko, Renardy, & Renardy, 2000), (Cristini, Blawzdziewicz, & Loewenberg, 2000), (Huo, Lowengrub, & Shelley, 2001), (Blawzdziewicz, Cristini, & Loewenberg, 2003), (Kim & Lowengrub, 2004), (Bazhlekov, 2004), (Reyes, 2005), (Khismatullin, Renardy, & Renardy, 2006), (Subramanian & Koch, 2006), (Young, Blawzdziewicz, Cristini, & Goodman, 2008), (Mählmann & Papageorgiou, 2009) (Sohn, Yu-Hau, Li, Voigt, & Lowengrub, 2010), (Zhao & Shaqfeh, 2011), (Reyes, Minzoni, & Geffroy, 2011), (Ramalingam, Ramkrishna, & Basaran, 2012), (Lalanne, Tanguy, & Risso, 2013), (Spann, Zhao, & Shaqfeh, 2014), (Escalante, Reyes, Rosas, & Geffroy, 2015).

This work addresses the dynamics of drop deformations immersed in an immiscible fluid that occurs under a large class of linear *2D-flows*, a class that has been poorly studied until the experimental work of Rosas, (Rosas, Reyes, Minzoni, & Geffroy, 2014). It is based on a numerical technique that describes the 3-dimensional evolution of shape of the drop using a *Boundary element method* algorithm. In Chapter 1, I present the fundamentals of the theoretical and numerical implementation of the algorithms. Then in Chapter 2, my numerical results are matched against theoretical, experimental and other numerical results of drop deformation, including conventional flows such as *simple shear flow* or *extensional 2D-flows*. Chapter 2 addresses mainly the correct calibration of the numerical method implemented for this work.

Chapter 3 and 4 address flow effects on drops with small or large viscosities. Chapter 3 presents results in strong flows of drop deformations with very low ratios of viscosity. The numerical data were compared with the theoretical results of Taylor and Cox. Experimental data of Rosas (Rosas I. Y., 2013) were used to calibrate the numerical results. Then, I present an analysis of the *3D-effects* of the shape of the drop as a consequence of the imposed flow. Chapter 4 presents results when the drop has a larger viscosity than the fluid that contains the drop. Again, the experimental data of Rosas were used and the *3D effects* of the drop were analyzed.

The deformation attained by the drop along different directions is also presented in detail in Chap. 3 and Chap. 4. The classification of different shapes and its correlation with different type of flows motivated an analysis of the characteristic time-scales for the elongation of the principal axes of the drop. Chapter 5 presents an overview of the time-scales of drop deformation. The first part of Chap. 5 focuses on the time required to attain the steady state deformation. After the deformation process is arrested, retraction of the drop to its spherical shape is predicted. However, different axes have different deformation, and also clearly different time-scales. The second part of Chap. 5 focuses in the retraction times associated for all three axes. Thus, under this new environment, evaluation of the interfacial tension during the process of retraction is rather complex. And in the last part of Chap. 5, I investigate possible corrections to the standard techniques for determination of interfacial tension by matching experimental against numerical retraction processes. The necessity to carry out this comparison is due to the fact that those techniques assume shapes of the drop that differ from the shapes and time-scales predicted in latter Chapters.

The effects of *the intensity of the flow*, *G*, on the drop form have not being observed in detail in previous works. In Chapter 6, the importance of *the intensity of the flow* in the stationary state of deformation is evaluated. As the strength of the flow increases, the role of interfacial tension becomes non-homogenous, becoming weaker normal to the direction of the flow; that is, reduced by the presence of vorticity. Consequently, the observed solution at weak shear rates shows a bifurcation with well-defined critical values; future possible studies are described as a result of this work.

In Chapter 7, I investigate *the slender body theory for drops in extensional flows*. The earlier study of *extensional flows* by Acrivos, Lo and Hinch is based on asymptotic domain expansions assuming an axisymmetric behavior: as a result of a slender ellipsoid of revolution. Hence, its principal limitation appears when solutions of the non-axisymmetric shapes are looked for. With the BEM-3D numerical method new questions are also addressed, based upon the transition of drop shape classes observed in extensional *3D*- versus *2D-flows*. The former flow always induces axisymmetric shapes, while the latter flow is neutral in the third dimension, and the observed shapes resemble more to “squashed ellipsoids” perpendicular to the plane of the flow. Those questions are studied as well for the transition observed in the linear planar flows. Chapter 8 is the extension of Chapter 7 because another bifurcation of the shape deformation can be observed. This latter bifurcation of the solution takes place for drop deformations observed during the transition from *simple shear flow* to *extensional 2D-flow*.

Chapter 9 contains my brief overview of the detailed dynamics of the deformation of a drop when embedded in a large class of flows, and a discussion of the results of this numerical study. The problems solved and unsolved by the method and new branches of study in the drop deformations in strong flows.

1.1 Background

Properties of multiphase fluids are well characterized under statics conditions. However, when these systems are under flow, the presence of surfactants or multiple length-scales, the approximations in the calculus of the global properties or the complex nature of the observed phenomena, the dynamic of inter-phases —essential to the global properties of this kind of fluid— dictate that the description of the fluid dynamics turns to be hard. In this Section, I describe all of the assumptions made for the study of the deformation of a drop immersed in another fluid (this being the simplest case of a multiphase fluid). To incorporate all effects mentioned above, it is necessary to characterize simultaneously the fluid dynamics of each fluid as well as the stress balance at all points of the interface. However, this case presents a large variety of technical difficulties, with some depending mostly on the numerical approximation and others due to the complex nature of the fluid dynamics of two fluids with an evolving interface. In the worst cases, there is a combination of those possible situations.

Therefore, my analysis of drop deformation assumes two Newtonian fluids, Fig. 1.1. The first (inner) phase is the drop with viscosity μ_2, while the continuum phase viscosity is μ_1. The viscosity ratio is $\lambda_\mu = \mu_2/\mu_1$, and the interfacial tension on the interface is γ. The two fluids have the same density and both are immiscible. There is not surfactant and there are no Marangoni stresses present (these due to a heterogeneous distribution of surfactant).

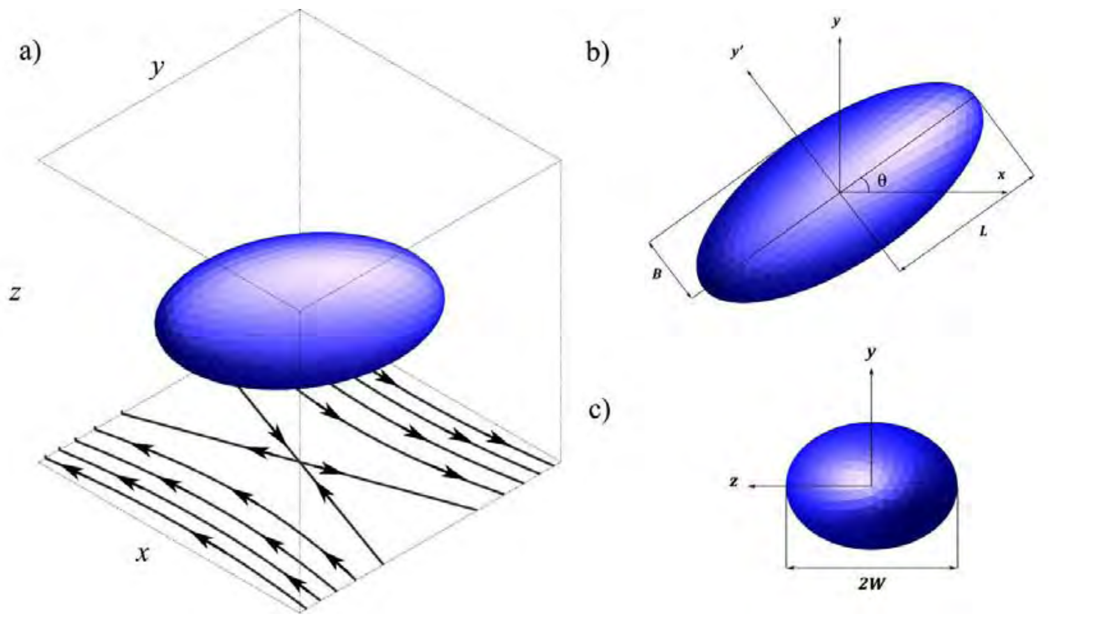

Figure 1.1. *Scheme of a two-phase fluid. Drop in stationary shape.* $\lambda_\mu = 0.012$, $\alpha = 0.13$ *and* $Ca = 0.40$. *(a) 3D-view of the drop within an applied 2D-flow shown on the xy-plane. (b) Conventional characterization of deformation on the xy-plane: L and B. (c) Cross section of the drop: W being the third deformation length scale.*

Creeping-flow conditions are assumed. In the creeping-flow regime the motion of the fluids is governed by the Stokes equations Eq. (1.1), and continuity equation Eq. (1.2), (Leal, 2007), for each fluid.

$$\mu\, \nabla^2 \boldsymbol{u} = \nabla p\,, \tag{1.1}$$

$$\nabla \cdot \boldsymbol{u} = 0\,. \tag{1.2}$$

The velocity field $\boldsymbol{u}$ is continuous over the drop interface S. Tractions exerted on the two sides of the interface between the two fluids have two different values, with a corresponding discontinuity:

$$\Delta \boldsymbol{f} = \boldsymbol{f_1} - \boldsymbol{f_2} = (\boldsymbol{\sigma}_1 - \boldsymbol{\sigma}_2) \cdot \hat{\boldsymbol{n}}\,, \tag{1.3}$$

where $\hat{\boldsymbol{n}}$ is the normal vector pointing out of $\boldsymbol{S}$ and the stress tensor is represented by $\boldsymbol{\sigma}$. In this work, a constant value of interfacial tension γ (Pozrikidis, 1997) is assumed; *i.e.*, $\boldsymbol{\nabla} \cdot \hat{\boldsymbol{n}}$ is equal to twice the mean curvature κ_{m} at that point on the interface, and

$$\Delta \boldsymbol{f} = \gamma\, \hat{\boldsymbol{n}}\, \boldsymbol{\nabla} \cdot \hat{\boldsymbol{n}} = 2\gamma\, \kappa_{\mathrm{m}} \hat{\boldsymbol{n}}\,. \tag{1.4}$$

The drop is subjected to (immersed in) a two-dimensional linear flow:

$$\boldsymbol{u}(x, y, z) = \frac{G}{1+\alpha}(y, -\alpha x, 0) \;\; when \;\; \boldsymbol{x}(x, y, z) \to \infty, \tag{1.5}$$

where G *is the intensity of the rate of deformation tensor of* $\boldsymbol{u}(x, y, z)$. The expression for G, Eq. (1.6) shows the definition made by Makosco, (Makosco, 1994). The linear flows given by Eq. (1.5) were used in all simulations except for the analysis of *extensional flows* in Chap. 7 and Chap. 8. There, the analysis of the flow is like those of Eq. (1.5), but

$$G = (|\boldsymbol{II}_{2\boldsymbol{D}}|)^{½}, \tag{1.6}$$

is estimated as Eq. (1.6) in all cases for *2D-effects*. The *flow-type* parameter is α (Bentley & Leal, 1986b), (Reyes, Minzoni, & Geffroy, 2011), (Rosas, Reyes, Minzoni, & Geffroy, 2014), (Escalante, Reyes, Rosas, & Geffroy, 2015), Eq. (1.7).

The parameter α measures the relative contribution of vorticity versus the contribution of the rate deformation of the flow, (Reyes, Minzoni, & Geffroy, 2011).

$$\alpha = \frac{\|\boldsymbol{D}\| - \|\overline{\boldsymbol{W}}\|}{\|\boldsymbol{D}\| + \|\overline{\boldsymbol{W}}\|}\,. \tag{1.7}$$

The term $\boldsymbol{D}$ corresponds to the *rate deformation tensor* $\boldsymbol{D} = \frac{1}{2}(\boldsymbol{\nabla u} + \boldsymbol{\nabla u}^T)$, (Makosco, 1994). $\overline{\boldsymbol{W}}$ is the *objective vorticity tensor* defined by Astarita $\overline{\boldsymbol{W}} = \boldsymbol{W} - \boldsymbol{\Omega}$, (Astarita, 1979). The class of *2D-flows*, *i.e.*, the configuration of linear planar flows can be understood using Fig. 1.2. Figure 1.2 shows the most relevant flows in 2D.

Makosco establishes the measure of *the intensity of the flow* as a measure of the second invariant of twice the rate of deformation tensor, *i.e.*:

$$\boldsymbol{II}_{2D} = \frac{1}{2}(tr(\boldsymbol{D})^2 - tr(\boldsymbol{D}^2)). \tag{1.8}$$

In order to apply the correct intensity of the flow in these numerical simulations, Makosco' s expression was used together with the contribution of the α-parameter:

$$\sqrt{|II_{2D}|} = (1+\alpha). \tag{1.9}$$

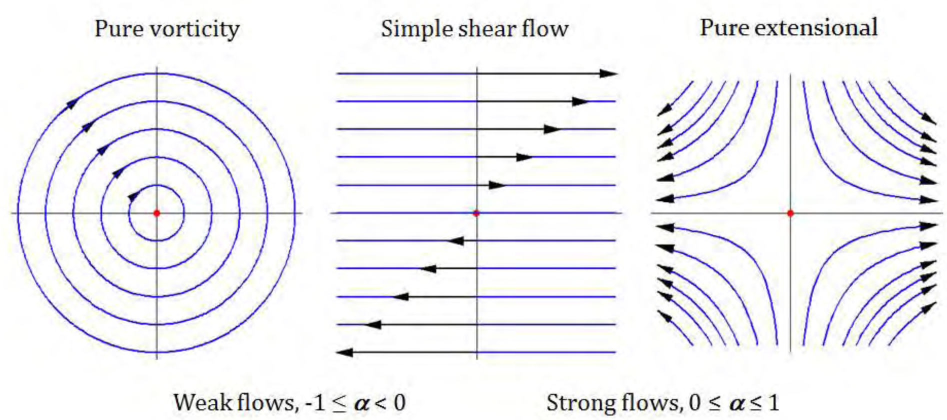

Figure 1.2 *The better known planar linear flows in 2D. Definition of weak flows and strong flows using the parameter α.*

Here, the parameter α values range from zero to one, thus Eq. (1.5) corresponds to a class of *2D-strong flows*. When the value of $\alpha = 0$, the strong flow with the highest content of vorticity is produced (Fig. 1.2, center), which corresponds to *simple shear flow* and with the velocity gradient in the y direction. The value of G is equal to the flow shear rate $G = \dot{\gamma}$. For a value of $\alpha = 1$ we have a pure *two-dimensional extensional flow*—no vorticity (Fig. 1.2 right)—; the principal axes of deformation are at $x = y$, the compressional axes being at $x = -y$; *the intensity of the flow* is the *strain rate* $G = \dot{\varepsilon}$.

Thus, the dynamic of drop deformation appears to be characterize by three dimensionless numbers. The viscosity ratio λ_μ, the capillary number *Ca,* Eq. (1.10) and the *flow-type* parameter α, Eq. (1.11). Later, other parameters appear to be equally important.

The capillary number characterizes the ratio between viscous stresses —imposed by the flow— and capillary forces that resist the deformation and drive the drop towards the equilibrium shape; were r_0 is the non-deformed radius of the drop.

$$Ca = \frac{r_0 \mu G}{\gamma}. \tag{1.10}$$

The results presented in Chapter 3 and Chapter 4 were calculated for a range of values of the *flow-type* parameter α close to *simple shear flow*, *i.e.*, $\alpha = 0.03$, $\alpha = 0.05$ and $\alpha = 0.13$. Even though these values appear small, these flows are capable of significant deformations of a drop, while maintaining a good degree of control on the magnitude of deformation when compared with pure elongational flows. These types of flows have until recently been studied experimentally are no numerical data has been published to date, (Rosas I. Y., 2013).

The applied flows cover a range of capillary numbers from $0.0 < Ca \leq Ca_{cr}$. For the smallest values of Ca used here, the drop shape does not present significant or relevant differences of its principal shape parameters; for values of $Ca \simeq Ca_{cr}$, the deformation is not yet sufficient to cause drop break up. In Chapter 3, the continuum phase is more viscous than the drop: $\lambda_\mu = 0.012$. In contrast, the drop has a larger viscosity than the continuum phase, $\lambda_\mu = 15.68$, for results presented in Chap. 4. The values of *flow-type* parameter, capillary numbers and the viscosity rate were chosen so that numerical predictions match the values of the experimental results of Rosas, (Rosas, Reyes, Minzoni, & Geffroy, 2014).

The principal measure of drop deformation used during most of the last century corresponds to Taylor's deformation (Taylor, 1934):

$$D_T = \frac{(L - B)}{(L + B)}, \tag{1.11}$$

were L and B being the axes of the drop shown in Fig. 1.1b. Additionally, to Taylor's deformation a secondary scale is used that corresponds to the ratio between the lengths of the axes of the deformed drop normalized by the initial radius. Later, other measures are introduced.

1.2 Numerical method

The Boundary Integral formulation was proposed by Ladyzhenskaya, (Ladyzhenskaya, 1963) within the framework of hydrodynamic potentials. Then, Brebbia formalized the Boundary-Integral Equation Method (BIEM), (Brebbia, 1978) and introduced the terminology *Boundary element method* (BEM). In this Section, the basic equations necessary for the numerical implementation of this computational method are presented. The numerical method used is the *3D-collocation boundary element method* (Pozrikidis, 1992)

In the Stokes regime, Eq. (1.1), the Stokes Equation can be rewritten as

$$\boldsymbol{\nabla}\cdot\boldsymbol{\sigma} = -\boldsymbol{\nabla}\, p + \mu\, \boldsymbol{\nabla}^2\boldsymbol{u}\,, \tag{1.12}$$

were $\boldsymbol{\sigma} = -p\boldsymbol{I} + \mu(\boldsymbol{\nabla}\, \boldsymbol{u} + \boldsymbol{\nabla}\, \boldsymbol{u}^T)$ is the stress tensor.

Let us consider two unrelated Stokes flows with velocities $\boldsymbol{u}$ and $\boldsymbol{u}^*$ and associated stress tensors $\boldsymbol{\sigma}$ and $\boldsymbol{\sigma}^*$. The Lorentz reciprocal theorem or Lorentz Theorem establishes the possibility of evaluating a solution if another known solution exists. For the Stokes regime (Happel & Brenner, 1963), (Leal, 2007), and (Pozrikidis, 1997). It implies that

$$\boldsymbol{\nabla}\cdot(\boldsymbol{u}^*\cdot\boldsymbol{\sigma} - \boldsymbol{u}\cdot\boldsymbol{\sigma}^*) = 0. \tag{1.13}$$

This is the counterpart of Green's second identity for harmonic functions (Lorentz, 1907). In other words, any solution can be expressed in terms of known solutions without having to solve the Stokes Equation explicitly.

Integrating Eq. (1.13) over a volume control, and using the divergence theorem to obtain

$$\iint_S (\mu\boldsymbol{u}^*\boldsymbol{f} - \mu^*\boldsymbol{u}\boldsymbol{f}^*)dS = 0, \tag{1.14}$$

were $\boldsymbol{f} = \boldsymbol{\sigma}\cdot\hat{\boldsymbol{n}}$, and $\hat{\boldsymbol{n}}$ is the unit normal vector point out the volume of control.

The complete velocity field of a drop immerse in another fluid can be calculated using Eq. (1.14). The use of analytical free-space solutions is an important feature of *Boundary element method*. These fundamental hydrodynamic solutions are applied in the governing equation Eq. (1.12), to solve the velocity field of the interface.

The hydrodynamic solution corresponds to Green's function of Stokes flow. This function provides the velocity and pressure fields that satisfies the continuity equation Eq. (1.2).

$$-\nabla p^* + \mu \nabla^2 \boldsymbol{u}^* + \boldsymbol{b}\delta(\boldsymbol{x} - \boldsymbol{x}_0) \equiv 0 \qquad (1.15)$$

were the vector $\boldsymbol{b}$ is a constant vector, and $\delta(\boldsymbol{x} - \boldsymbol{x}_0)$ is the tridimensional delta function centered at $\boldsymbol{x}_0 \in S$. Equation (1.15) is a representation of a modified stress tensor of the Green's function. Physically, Green's function expresses the flow due to a point force located at $\boldsymbol{x}_0$ (named *pole* or *source point*), along the direction and strength of $\boldsymbol{b}$; valid in the absence or presence of boundaries. Green's functions are named *fundamental solutions* or *propagators*.

Solving the velocity field for this *point source* in Eq. (1.15) implies that

$$\boldsymbol{u}^*(\boldsymbol{x}) = \frac{1}{8\pi\mu}\boldsymbol{G}(\boldsymbol{x}, \boldsymbol{x}_0) \cdot \boldsymbol{b}, \qquad (1.16)$$

$$\boldsymbol{p}^*(\boldsymbol{x}) = \frac{1}{8\pi}\boldsymbol{p}(\boldsymbol{x}, \boldsymbol{x}_0) \cdot \boldsymbol{b}, \quad \text{and} \qquad (1.17)$$

$$\boldsymbol{\sigma}^*(\boldsymbol{x}) = -\boldsymbol{p}(\boldsymbol{x}, \boldsymbol{x}_0)\boldsymbol{I} + \nabla\boldsymbol{G}(\boldsymbol{x}, \boldsymbol{x}_0) + \nabla\boldsymbol{G}(\boldsymbol{x}, \boldsymbol{x}_0)^{\boldsymbol{T}} = \frac{1}{8\pi}\boldsymbol{T}(\boldsymbol{x}, \boldsymbol{x}_0) \cdot \boldsymbol{b} \ ; \qquad (1.18)$$

were $\boldsymbol{G}(\boldsymbol{x}, \boldsymbol{x}_0)$, $\boldsymbol{p}(\boldsymbol{x}, \boldsymbol{x}_0)$ and $\boldsymbol{T}(\boldsymbol{x}, \boldsymbol{x}_0)$ are the Kernels or the Green's functions.

For an infinitely unbounded flow, as in this work, the free-space Green's functions are used. These functions are:

$$\boldsymbol{G}(\boldsymbol{x}, \boldsymbol{x}_0) = -\frac{\delta}{r} + \frac{\widehat{\boldsymbol{x}}\widehat{\boldsymbol{x}}}{r^3}, \qquad (1.19)$$

$$\boldsymbol{p}(\boldsymbol{x}, \boldsymbol{x}_0) = 2\frac{\widehat{\boldsymbol{x}}}{r^3} \text{ and} \qquad (1.20)$$

$$\boldsymbol{T}(\boldsymbol{x}, \boldsymbol{x}_0) = -6\frac{\widehat{\boldsymbol{x}}\widehat{\boldsymbol{x}}\widehat{\boldsymbol{x}}}{r^5} \ ; \qquad (1.21)$$

were $\boldsymbol{G}(\boldsymbol{x}, \boldsymbol{x}_0)$ is named *Stokeslet* or *Ossen-Burgers tensor*, $\boldsymbol{T}(\boldsymbol{x}, \boldsymbol{x}_0)$is the stress field called *Stresslet* (which is a symmetric tensor), $\widehat{\boldsymbol{x}} = \boldsymbol{x} - \boldsymbol{x}_0$, and $r = \|\widehat{\boldsymbol{x}}\|$.

The velocity field at the interface is obtained using *point sources* (Stokeslets and Stresslets) on the interface of the drop, in conjunction with *Lorentz theorem*. Considering, on the surface of the drop, Stokeslets and Stresslets for every collocation point respectively (Leal, 2007), (Prosperetti & Tryggvason, 2007), and (Pozrikidis, 1992), then, the numerical scheme is given by:

$$\boldsymbol{u}^{\infty}(\boldsymbol{x}) - \frac{1}{8\pi\mu_1}\int_S \Delta\boldsymbol{f}(\boldsymbol{x})\cdot\boldsymbol{G}(\boldsymbol{x},\boldsymbol{x_0})dS(\boldsymbol{x})$$

$$+ \frac{1-\lambda_\mu}{8\pi}\int_S \boldsymbol{u}_0(\boldsymbol{x})\cdot\boldsymbol{T}(\boldsymbol{x},\boldsymbol{x_0})\cdot\widehat{\boldsymbol{n}}(\boldsymbol{x})dS(\boldsymbol{x}) = \begin{cases} \boldsymbol{u}_1(\boldsymbol{x}) & 1.22\ a), \\ \dfrac{1+\lambda_\mu}{8\pi}\boldsymbol{u}_0(\boldsymbol{x}) & 1.22\ b), \\ \lambda_\mu \boldsymbol{u}_2(\boldsymbol{x}) & or\ \ 1.22\ c). \end{cases}$$

In this equation, there are three cases. Case (b) solves the velocity field when $\boldsymbol{x}$ is on the surface of the drop. With this information, case (a) calculates the velocity field when $\boldsymbol{x}$ is outside of the drop —exterior flow. Finally, Case (c) calculates the velocity field for $\boldsymbol{x}$ being inside the drop.

The flow imposed on the continuum phase is determine by $\boldsymbol{u}^{\infty}(\boldsymbol{x})$: the velocity field faraway. The second term on the left side of Eq. (1.22b) is known as the Single-Layer Potential and evaluates the projection of the velocity field of a distribution of point forces with the stresses across the interface. The third term of Eq. (1.22b) is known as the Double-Layer Potential and is used to evaluate the velocity field at the interface. The *boundary element method* has the advantage of reducing a 3D computation problem into a *2D-evaluation*; *i.e.*, the method requires to solve the velocity field on the surface of the drop, using Eq. (1.22b) and with this information is possible to know the velocity field inside and outside of the drop.

1.2.1 Numerical implementation of the computation of drop deformation in strong flows

The numerical scheme solves the Stokes flow equation for an instant of time, Eq. (1.22b). The evolution of the drop shape is carried out using the velocities obtained at the collocation points on the interface to advance each point for a small-time interval. By means of an interpolation subroutine the velocity at all the nodes is calculated and then the new position of the drop (at $\boldsymbol{x}_0(t+\Delta t)\in S$) is evaluated. For the evaluation of the Single Layer Potential, an approximation of the local curvature of the drop is used on curved triangles using Eq. (1.4). A quadratic interpolation throughout these triangles is carried out to obtain the mean curvature of the elements of the drop. In this manner, the mesh of the interface and the algebraic system of equations of the boundary elements is smaller than those of other more conventional numerical schemes, such as *e.g.*, finite differences. The next Subsections explain, in a detailed form, every component of the numerical algorithms written for this numerical study. The accompanying CD-ROM contains copies of the Fortran 2010 codes written for these simulations.

1.2.2 Performance of the mesh used in the method in the time

Boundary element method in 3D is a powerful algorithm to calculate the evolution of a drop immersed in another fluid when a *2D-flow* is applied by the continuum phase. The advantage with respect to other numerical method is the simplification of the *3D-problem* to a surface problem. However, the real cost is due to numerical computations not being that easy to carry out. First, the numerical method needs to evaluate all of the geometry parameters of the drop, such as position, curvature, (this will be explained in the next Subsection), among others, because information about each element is required to solve the singular integrations on the mesh. In Eq. (1.22), the Stresslet Eq. (1.19) and the Stokeslet, Eq. (1.21) are present. When the algebraic system is evaluated, the velocity field on the surface is known for an instant of time. Thus, the process is repeated in time to have the dynamic evolution of the drop deformation. In general, those steps are simple. However, every step needs an efficient algorithm to calculate in a fast manner, all parameters of each element; *i.e.*, to have information of all elements conforming the surface of the drop.

A useful numerical scheme must be efficient in the execution of the algorithms — in this case of the geometry, and the behavior of the deformed drop— given the necessity to provide an accurate benchmark of the drop model versus the experiments. Numerical methods use large arrays, which implies a considerable time to carry out the numerical computation. The time used mainly depends of the number of elements of the drop, (Grinfeld, 2010). The mesh used in this numerical method is based on the partition of an elementary octahedron. The minimal number of elements n of the mesh is 8, then 32, then 128, etc., Eq. (1.23).

$$n = 8 * 4^k. \tag{1.23}$$

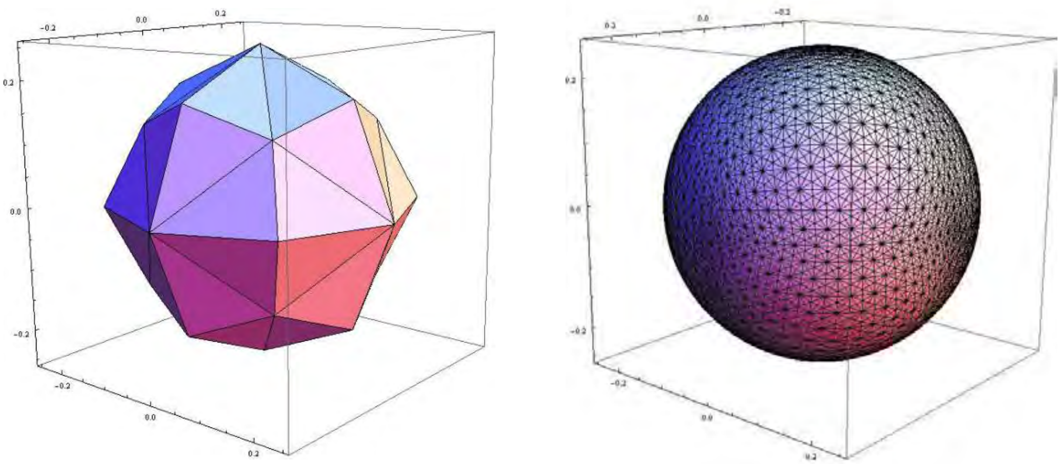

Figure 1.3 *Drop of 0.25 mm of radius. In the left part, the mesh has no subdivision, 8 elements (curved triangles). In the right, the same drop with a mesh of 2048 elements, subdivision equal $k = 4$.*

The k is the number of partition that is applied on the initial octahedron. When the partition $k = 4$, the number of elements is 2048. In Figure 1.3, a simple mesh with 8 elements (left) is shown, and a final mesh after the partitioning process (right) with 2048 elements.

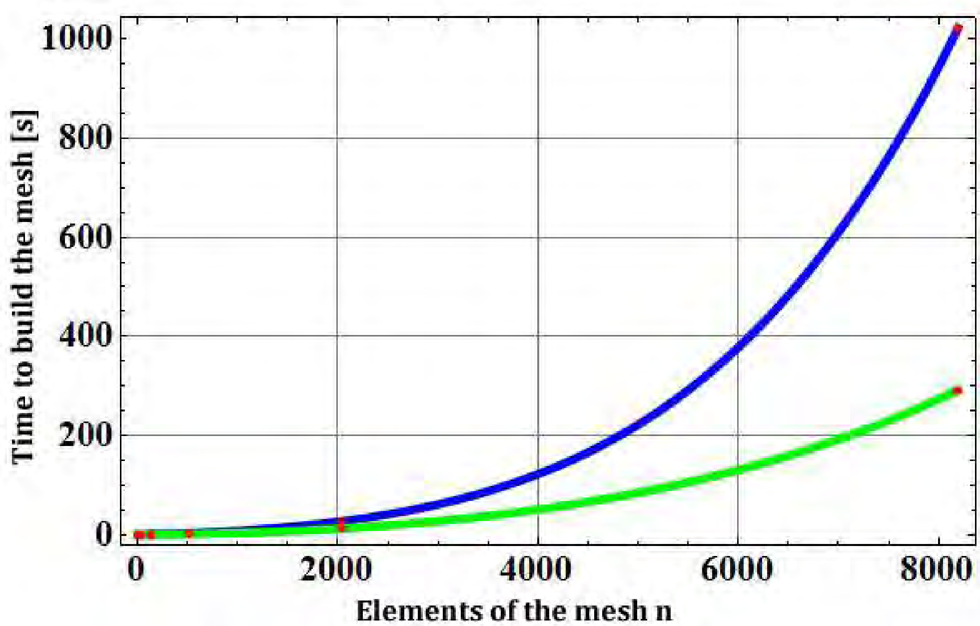

Figure 1.4 *Time to calculate the algebraic system vs. number of element of the mesh. Blue line indicates the time using a one tread algorithm. Green line is the time using 4 computational threads.*

Figure 1.4 shows the time required to calculate the complete mesh with different number of elements using a computer with a two-cores, dual processor HP w4300. Figure 1.4 shows the growth of the computation time as the number of elements increases. The blue line plot is a nonlinear function. However, the blue model in Fig. 1.4 is based on sequential programming. The green plot is the execution time for the same algorithm but using a parallelized scheme. The difference is that one tread is used for this algorithm (blue line), on the other hand, 4 computational threads (green line) are used to generate the same mesh. For a mesh of 8,192 elements, as shown in Fig. 1.4, the multithreading model is eight times faster than the sequential algorithm.

Parallel programming techniques reduces —in linear form— the required time for the generation of the mesh. However, this example proves unambiguously the necessity to extensively apply optimized parallel codes for these studies.

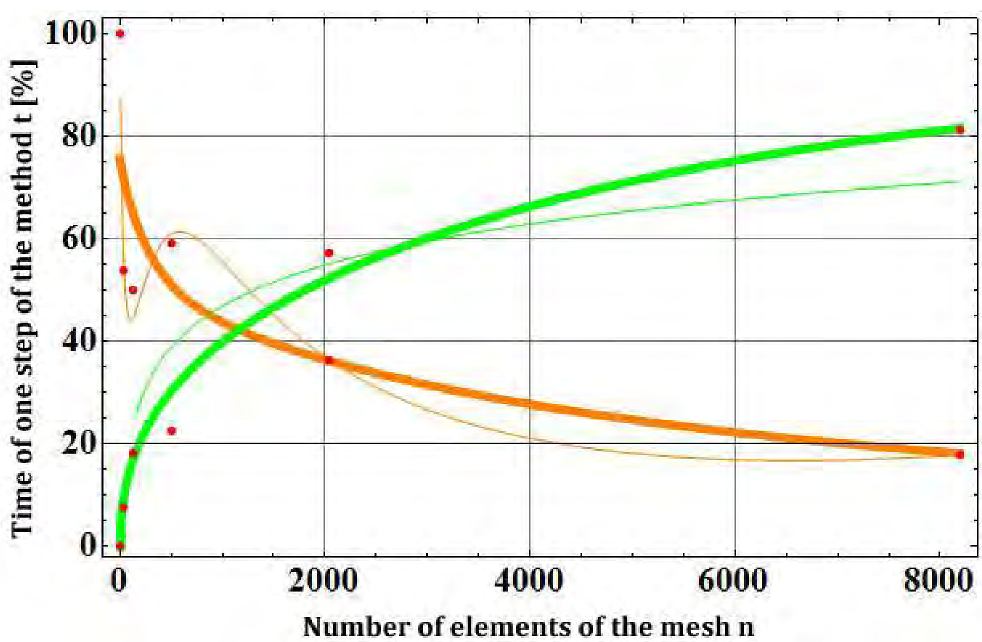

Figure 1.5 *Time to build one step of the dynamic of drop deformation vs. number of elements of the mesh, green. Time to calculate the solution of the algebraic system vs. number of elements, orange. Time is in percentage of CPU Time.*

Figure 1.5 shows how the numerical method employs the total time, in the same machine: (1) to generate the geometry mesh, to estimate the curvature and characteristics of the mesh and (2) to build the algebraic system to solve the problem of BEM-3D (orange plot); the green lines correspond to the time used to solved the algebraic system.

These results were obtained only for one step of time evolution of the drop deformation. The total time for a numerical experiment corresponds to the times for one step multiplied by the total number of steps, divided by the number of treads used in the algorithm. Most of the results presented here were carried out with 8 treads using HP Z800, Z440 and Z640 workstations. So, the time lapse shown in Fig 1.5 is at least twice faster nowadays.

The latest numerical experiments do not report the time required for every step, because that information implies dedicating machine-time to calculate this datum of little use. In terms of efficiency of the code I preferred to disregard the one step time information, and focus in the total simulation time. In general, most numerical runs take approximately 3 days using the fastest workstation.

1.2.3 Oriented parallel programing in the numerical scheme

Another important objective of my numerical program of BEM-3D was to build a practical code which allows expeditious modifications or changes or some characteristics of the numerical scheme without changing major portions of the code. The numerical codes were optimized up to 80%, *i.e.*, the parallelization techniques are applied to most algorithms, except those needed to calculate the curvature of each element because these are not readily parallelizable algorithms.

The parallel methods applied use the newest procedures of Fortran 2010 together with the OMP-Libraries of Intel. The final code can simulate different kinds of flows, covering the possibility to study all *2D-flows* from shear to *extensional flow* (strong flows), as well as elongational flows without vorticity. The code can be used to study other *3D-flows* such as ABC flow, extensional *3D-flows* or the Poiseuille flow.

The numerical model of a drop designed for this work can produce information about cross-sections of the drop as no other model published earlier shows. As well, the method employs the newest algorithm for evaluation of the curvature of each element of the interface, which may probe indispensable for studying highly deformed drops. To test the validity of the new algorithms, every subroutine or algorithm can be replaced, in the fast manner, by an older, validated one, or a new procedure. This feature of the model's code was used extensively —in particular, are essential in the calibration of the method— to determine the best choices and the optimum space of parameters, to evaluate predictive differences between diverse algorithms, and to choose the best option for performing the numerical simulations.

1.2.4 Curvature in BEM 3D

For BEM-3D calculations, the correct determination of the local curvature values is most relevant, because at least two curvature measures are required to correctly describe a *3D-surface*. As well, these values of curvature directly affect the stability of the numerical equations used to calculate the velocity and stress fields. It is due to the jump of stresses present across the drop boundary —evaluated with Eq. (1.4)—, which is strongly dependent on local values of curvature. For this reason, in the development of numerical scheme the curvature needs to be well defined. The simplest scheme most frequently used corresponds to a mapping of curved triangles in the $\xi\eta$*-plane* (Cools & Rabinowitz, 1993), (Pozrikidis, 1998).

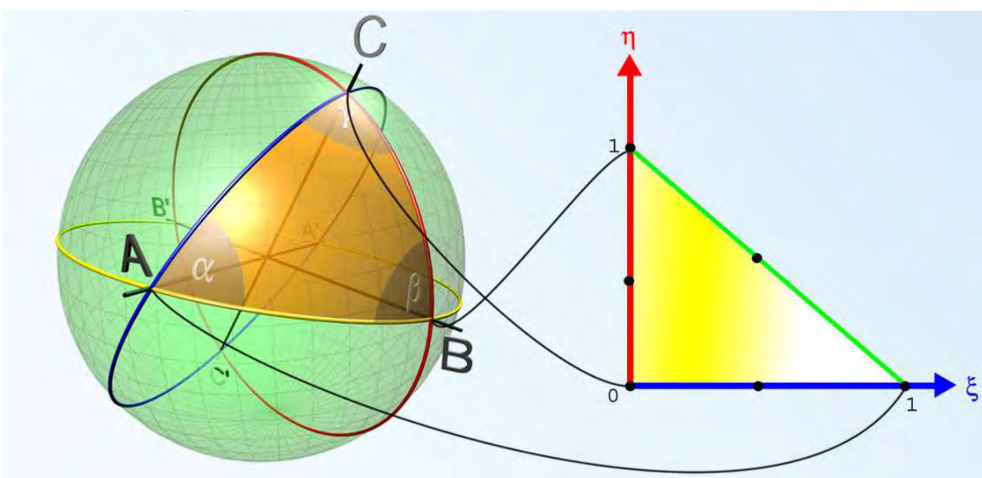

Figure 1.6 *Mapping of curved triangles from 3D-space to 2D-space.*

Every element used in this numerical method is built with 6 nodes to represent a curved triangle Fig. 1.3 and Fig. 1.6. Many numerical methods used plane triangles (Kennedy, Pozrikidis, & Skalak, 1994), (Zinchenko, 1997), (Khayat, 2000), (Bazhlekov, 2004), (Prosperetti & Tryggvason, 2007), (Spann, Zhao, & Shaqfeh, 2014), etc. In general, the plane triangle is made by 3 nodes. In Chapter 2, the numerical method was calibrated using the same parameters of the predecessor methods to compare the previous results with the new method.

Then, using the appropriate Gaussian weights, the pointwise curvature is calculated, by evaluating a contour integral over the element (Pozrikidis, 1997). However, Bazhlekov showed (Bazhlekov, 2004) that the calculated curvature has an error of 14% regardless of the number of the nodes defining the element. Many numerical BEM methods (Kennedy, Pozrikidis, & Skalak, 1994), (Zinchenko, 1997), (Khayat, 2000), (Bazhlekov, 2004), (Prosperetti & Tryggvason, 2007), (Spann, Zhao, & Shaqfeh, 2014), etc., use plane triangles —made with 3 nodes only— to describe the surface. Thus, this type of computation predicts only the correct behavior for slightly deformed drops.

More elongated drops demand improving the calculation of its local curvatures. The first step was to employ *curved triangles* because the drop's curvature is better represented in this manner. For curved triangles, every element of the interface is built with 6 nodes; Figs. 1.3 and 1.6. In Chapter 2, the numerical method was calibrated using the same parameters of those earlier methods to compare previous prediction against the new method.

For highly elongated drops, curvature properties have important and different behaviors: (a) near the tips, both radii have approximately same values and signs; for (b) the central region of the drop, curvature signs are the same, but one of the radius in quite large relatively to the smallest radius; and (c) for drops with the classical dumbbell shape, in the waist region, one of the curvatures has a negative sign —it is here only—, and under this conditions, that the mechanism of drop break-up takes place. Therefore, a consistent and accurate analysis of the curvature needs to be able to predict smooth changes-of-sign behavior for the lengthwise radius, a condition of concavity that is always present when the drop is strongly deformed. It is only then that the transition to break up of the drop will be reproduced in the numerical simulations.

However, the *boundary element method* in 3D cannot represent the *final stages of the break-up* of the drop. But the correct evaluation of the curvatures make possible studying the behavior of a drop closest to the break-up phase. With this motivation, a review of the differential geometry was made.

To develop a new method for acquiring the median curvature of each element, it was necessary to review the First and the Second Fundamental Forms of a surface (Levi, 1980), (Stoker, 1989). Then a new curvature was obtained using the initially proposed weights. A comparison was made between a spherical surface and the geometry mesh employed. The calculated values were better than using the method of contour integral on the element, (Pozrikidis, 1997). However, the error was around 0.5% of the spherical value.

In order to have an accurate estimation of the curvature, classical methods increase the number of elements of the interface as the deformation of the drop becomes more elongated. As a consequence, classical BEM methods usually employ techniques that subdivide most elements of the mesh, increasing the local precision with smaller elements. However, the simulation time grows as a linear function of the number of elements, in the best of cases. On the other hand, the numerical method used in this work does not need to refine the mesh to have an accurate approximation of the curvature, while using the computation time to evaluate the time evolution rather than for mesh refinement. In other words, the method is sufficiently accurate to estimate the curvature, in a stable manner, as well as faster.

For drop deformations less than $D_T < 0.5$, approximation of the local curvature with the median curvature value is enough. However, when the deformation increases, there is a significant reduction of the curvature radius transversal to the flow. Hence, if the median curvature is used to represent the local curvature, the effects of this transversal curvature is inhibited by the much larger radius of the plane tangential to the flow due to the elongated elements in the cross section. So, for elongated drops, all curvatures of cross sections of the drop will be significantly larger than its tangential curvatures. But, elements will not preserve this information: locally, the curvature in the element becomes asymptotically small along the normal direction to the cross section; *i.e.*, the curvature parallel to the flow in the element will dominate the median curvature values, regardless of degree of elongation. The consequence of this underestimation of the local curvature implies that simulations reach a maximum drop elongation, regardless of the strength of the flow. Evaluation of the correct elongation requires a larger —*positive*— value in the transversal curvature —than the value of tangential curvature, *positive or negative*, to the flow. Only then highly deformed drops are possible to model, and to reach *a negative median curvature*, *i.e.*, and in this way to have a more elongated state of deformation. However, highly deformed drops are no longer stationary because a transient behavior towards rupture becomes dominant. Being aware of these limitations, all my numerical experiments were performed to have a maximum stationary deformation $D_T < 0.5$.

Except for the numerical simulation of Spann, *et al.*, (Spann, Zhao, & Shaqfeh, 2014), most previous numerical methods have drops surfaces with less than 1000 elements. So, in my numerical method I have employed 256 elements for the same circumstances (numerical experiments) of other published works, in order to be consistent with the evaluation and calibration of this method. Spann´s simulations used two sizes of meshes; the first mesh has about 2880 faces, the second 5120 faces. Our method uses 2048 curved-triangle elements, meaning that each curved triangle is equivalent to 4 plane triangles. In fact, 2048 curved element in my code have a similar precision than a mesh of plane triangles of 8196 elements. At the end, the new numerical method of BEM-3D is optimal for precision, efficiency in time and structure of the mesh.

1.2.5 Numerical Accuracy

In Chapter 2, I present the necessary simulations required for the proper calibration of the method versus previous numerical methods results. For this reason, a similar number of elements (512) is used. The numerical simulation results presented in this work use a surface of 2048 curved elements. To obtain the dynamics of the drop deformation, the time advance was performed using different methods. In Chapter 2, the calibration of the method, uses Euler´s method to estimate the deformation of the drop in *simple shear flow* with rate of viscosity of $\lambda_\mu = 1$. However, for the comparison with experimental data, the numerical mesh had 2048 elements or higher

A fourth-order Adams-Bashforth-Moulton method (Lambert, 1973) was employed in Chapter 2 to calibrate the method with the experimental data (Guido & Villone, 1997). Chapter 3 and Chapter 4 used this method too. The rest of simulation presented in this work used a third-order Runge-Kutta method.

The total time required to attain the stationary shape of the drop deformation is $Time = Gt$. In all numerical experiments the deformation of the drop reaches a final stationary deformation. The time employed in each numerical simulation was compared with the experimental data of Guido *et al.*, (Guido & Villone, 1997) and Rosas (Rosas, Reyes, Minzoni, & Geffroy, 2014).

CHAPTER 2.

Calibration of numerical method

Since Taylor's experimental work (Taylor, 1934), the study of drop deformation has been focused mainly under two kinds of flows: known as *simple shear flow* and *extensional 2D-flow* (in some cases named hyperbolic flow). These flows are two-dimensional flows *i.e.*; the flow field is planar.

For years, theories and analyses have assumed that the cross section of the drop is a slightly deformed *axisymmetric* ellipsoid —a prolate, with equal lengths for both short axes—, which simplifies the analysis of drop dynamics while facilitating the study of drops dimensions mainly on the plane parallel to the flow (that is, the plane of observation of the experiments); see Fig. 1.1b. In recent years, *e.g.*, Kennedy *et al.*, (Kennedy, Pozrikidis, & Skalak, 1994) showed that during stationary states, the cross section of the drop (on the plane perpendicular to that of the flow) was not nearly circular. Subsequently, Guido *et al.*, (Guido & Villone, 1997) showed experimentally the same fact. Both observations were carried out applying *shear flows* on the continuum phase. In the cases of 2D *extensional flows*, the work of Acrivos and Lo (Acrivos & Lo, 1978), Hinch and Acrivos (Hinch & Acrivos, 1979) established that a drop comes to a steady state of deformation with the cross section being also non-circular. However, there are no experiments in this kind of flows were the evolution of cross section was measured in a clear manner.

Taking advantage of this archive, one way to calibrate my new numerical method was to predict the evolution of drop deformations and to compare these results with previous observation and predictions reported in the literature. This Chapter is a summary of the different calibrations methods with respect to predecessor numerical methods and experimental data.

2.1 The numerical method vs previous numerical methods

Taylor was the first person to present a theory predicting the dynamics of drop deformation in *simple shear flows* (Taylor, 1932). Add to this, Taylor made the first experimental device to check those predictions (Taylor, 1934). Then Mason *et al,* carried out the first experiments of drop deformations in *simple shear flows* (Rumscheid & Mason, 1961), (Torza, Cox, & Mason, 1972). About the same time, Grace made a series of experiments in the same regime but for different viscosities of the fluids (Grace, 1982). These experiments were employed to calibrate the theoretical predictions by (Rallison, 1981), (Kennedy, Pozrikidis, & Skalak, 1994), (Khayat, 2000), and (Yuriko, Renardy, & Renardy, 2000). Using theoretical models for drops, Cox (Cox, 1969), and Acrivos and Hinch (Hinch & Acrivos, 1979) suggest that all cross sections of the drop are circular for this flow. Today the study of drop deformation in *simple shear flow* is vast; the works of Guido *et al.*, (Guido & Villone, 1997), showed that cross sections are rarely circular. Kennedy *et al.*, showed, using the numerical analysis, the same ellipsoidal shapes for the cross section, (Kennedy, Pozrikidis, & Skalak, 1994). Even so, since that work no other numerical analysis made detailed measurements of the drop shape in the perpendicular direction of the flow. Based in the previous work of Acrivos (Hinch & Acrivos, 1979), Hinch proved that cross section is non-circular in general. Hinch assumed the axisymmetric case because his equations become too complicated to work with them, and to extract useful results. The 3D numerical method here presented makes a number of improvement to the earlier models, hence the need to be properly calibrated against experiments and other predictions under "known conditions". My first calibration addresses the values of deformation obtained under steady flow conditions. The Calibration Section focuses on numerical data obtained by simulations by Rallison —

BEM 2D— (Rallison, 1981), Kennedy *et al* BEM 3D (Kennedy, Pozrikidis, & Skalak, 1994), Renardy *et al* VOF (Yuriko, Renardy, & Renardy, 2000). The simulations of Kennedy used a larger number of elements of the mesh —512 plane-triangles— when using a *boundary element method* technique. Renardy's model uses a *volume of fluid algorithm*. In this case, the method employed a mesh of 512 elements. Figure 2.1, shows a comparison of the drop deformation reached when a stationary state is attained, for different capillary numbers. The drop and the continuum phase have a ratio of viscosities of $\lambda_\mu = 1$. All numerical methods show a consistent behavior for capillary number less than $Ca < 0.35$. Theoretical results, on one hand, show that the theory of Taylor and Cox is also consistent with these data. On the other hand, the theory of Barthès-Biesel goes to a state of breakup before the experimental data of Mason *et al.* For larger capillary numbers, *i.e.*, $0.35 < Ca < Ca_{Cr}$, numerical predictions diverge from the experiments. The VOF method provides similar results to the experimental data; the reason of the small differences is because BEM methods solved the case of an unbounded drop, while VOF models take into account the presence of a wall where the *shear flow* conditions are applied. That is, VOF has information of bounded problems in drop deformation in shear rate. However, for other plane flows VOF does not have the versatility of the BEM-3D here proposed, because it is necessary to model the boundaries that generate the *2D-flow*, as Reyes mentioned, (Reyes, 2005).

A second calibration of my numerical method was carried out addressing the non-symmetric nature of the dimensions of the cross-section of drops data of Kennedy *et al.* (Kennedy, Pozrikidis, & Skalak, 1994). The length of the principal axis of deformation is L, while the shorter axes are now B —normal to L and the flow direction— and W *perpendicular* to the flow plane.

The shape asymmetry observed with both methods occurs readily for low values of the Capillary number and the coincidence is best for lower values of the viscosity ratio. Both meshes used are similar and the results are shown in Fig. 2.2.[1].

[1] The numerical simulation of our BEM3D implementation takes approximately 24 hrs. to do all cases shown in Fig. 2.2. Kennedy, *et al.*, simulations took weeks running on a supercomputer of those days. These improvements are the result of the use of new processors and the optimization of the code described in Chap. 1.

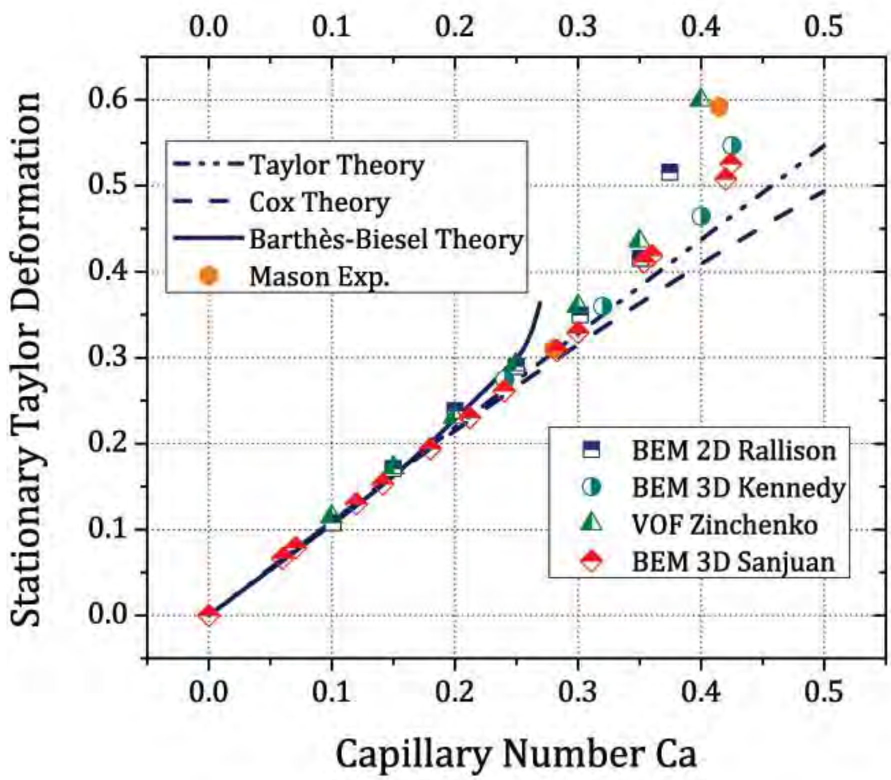

Figure 2.1 *Stationary Taylor's Drop Deformation vs. Capillary Number. Simple shear flow, $\alpha = 0.0$ and $\lambda_\mu = 1.0$.*

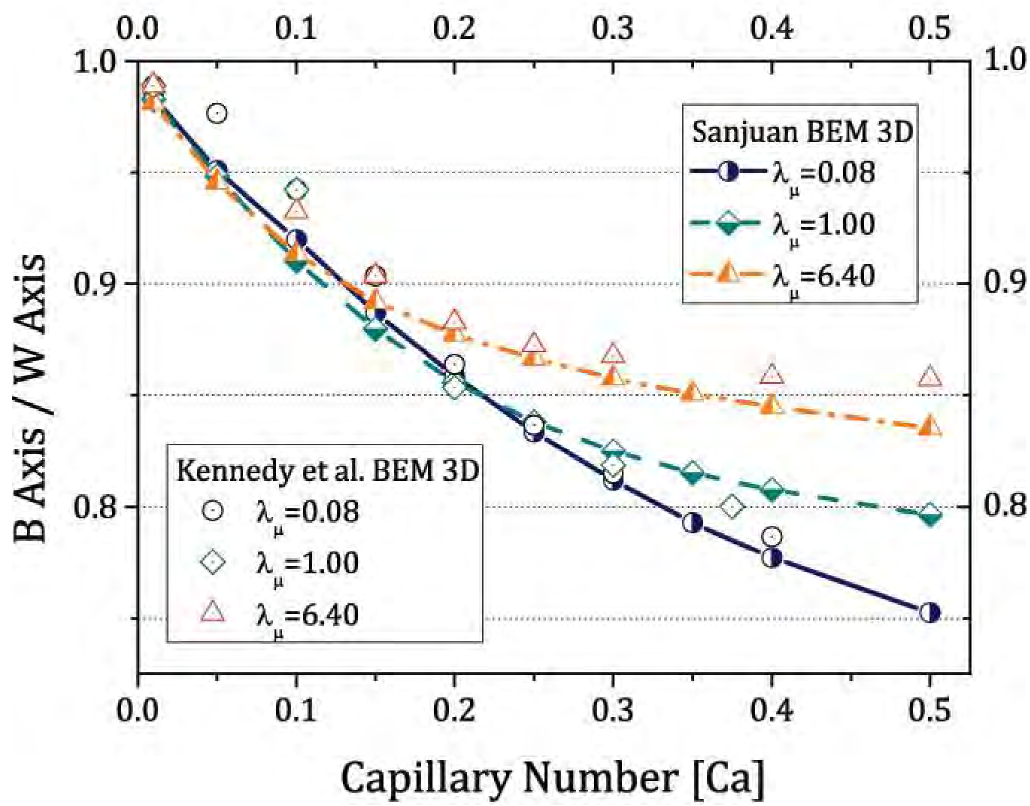

Figure 2.2 *B/W ratio vs. Capillary Number for different λ_μ values.*

2.2 Experiments in *simple shear flow* compared with experimental data

Now, my model predictions are compared to the experimental data of Guido *et al.*, (Guido, Greco, & Villone, 1999). Here I have two objectives: (1) to compare numerical vs. *experimental values* of the cross section that Guido and his group observed in *simple shear flows*; and (2) to study the proper physical assumptions of the model when ratios of viscosity are different of 1. When modeling viscosity ratios different than 1, the contribution of the Double Layer Potential —second integral of Eq. (1.22)— is non-zero. When $\lambda_\mu = 1$, it is a common practice that *boundary element methods* employ only the Single Layer Potential, and the numerical methods are faster. When $\lambda_\mu \neq 1$, to estimate the stresslets contribution on the surface of the drop, hence to calculate the stress and interfacial tension appropriately requires the second integral. This integral depends on the viscosity ratio λ_μ [2];

Figure 2.3 shows the values of the numerical simulation for a drop —of radius 25 μm and $\lambda_\mu = 1.4$— versus the experimental data of Guido; for values of $\lambda_\mu = 1.3$ and $\lambda_\mu = 1.4$. These *normalized values* of the 3 principal axes were obtained for steady states of deformation. Lines in Fig. 2.3 are Taylor's model predicted values of the semi-axes of a drop when it is modeled as a perfect ellipsoid (Taylor, 1932). These results indicate that it is a very good estimation of the drop deformation for regimes of small capillary numbers: $Ca < 0.15$. However, as the capillary number increases beyond $Ca \simeq 0.25$, Taylor's model clearly underestimates the growth of the *L-axis*, with associated larger values for the length of the *B-axis*; that is, Taylor's theory predicts a weaker deformation of the drop. Add to this, the model predicts axis-symmetry in the *W-* and *B-axis*, when the experiments made by Guido and the numerical simulations show a clear departure from the axis-symmetric form. In Fig. 2-3, the comparison is limited to $Ca \lesssim 0.5$ mainly because evaluation of the axes lengths in the work of Guido (Guido & Villone, 1997), presents larger errors due to

[2] Computation of this Double Layer Potential for $\lambda_\mu \ll 1$ turns out to be quite difficult. For this reason, there are not that many simulations with small values of λ_μ.

higher order on the shape form observed in the images. So, the information used to calibrate the code could not be the best option.

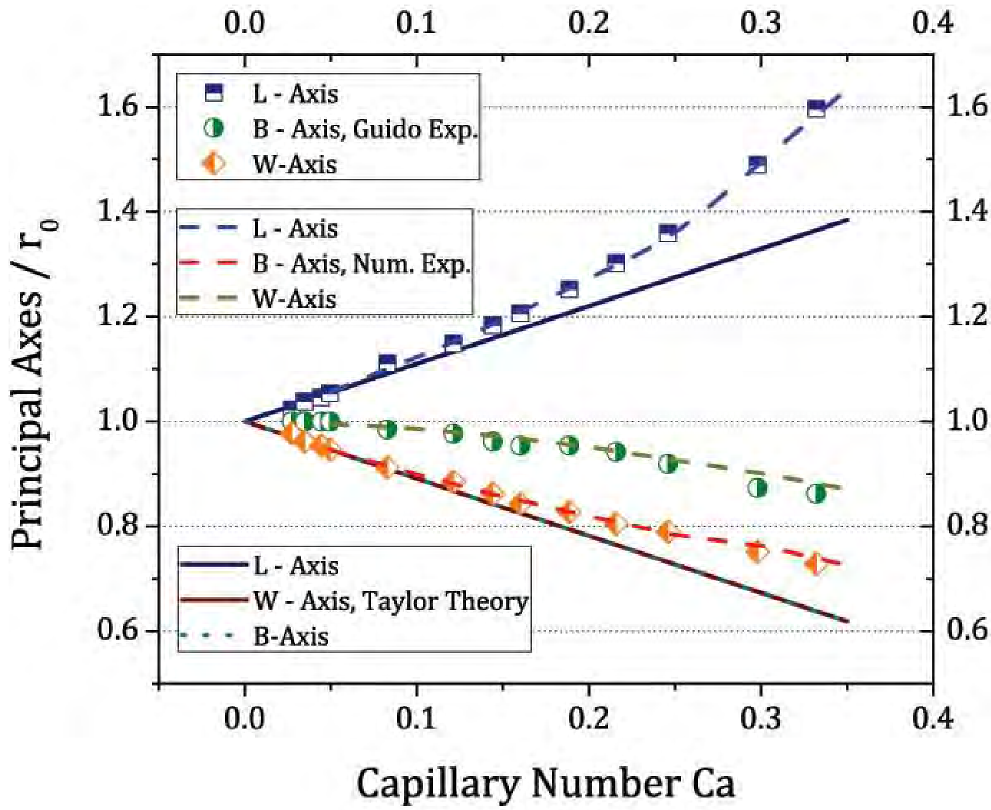

Figure 2.3 *Principal axes of the drop vs. Capillary Number.* $\lambda_{\mu} = 1.4$, $\alpha = 0.0$.

Figure 2.4 shows the comparisons of the "shapes" of a drop from two complementary visualization planes reported by Guido (right) and subjected to *simple shear flow* with $Ca = 0.46$, and radius of $25\ \mu$m. The top shapes correspond to the typical *xy-plane* projection in experimentation, Fig. 1.1b): the observed inclination corresponds to the orientation angle of the drop in the flow. The bottom shapes correspond to the *xz-plane* projection.

For the *xz-plane* projection, the drop is tilted with an angle of orientation (as shown in the *xy-plane*) and its photo (with optical axis along *y-axis*) models poorly the real deformation (magnification of the lens is a complex function of the master transfer function and coordinates x, y, z), for this reason, the numerical and experimental images may have different shapes near the ends of the drop —the left part appears to be larger than the right because the left part is above the focal plane.

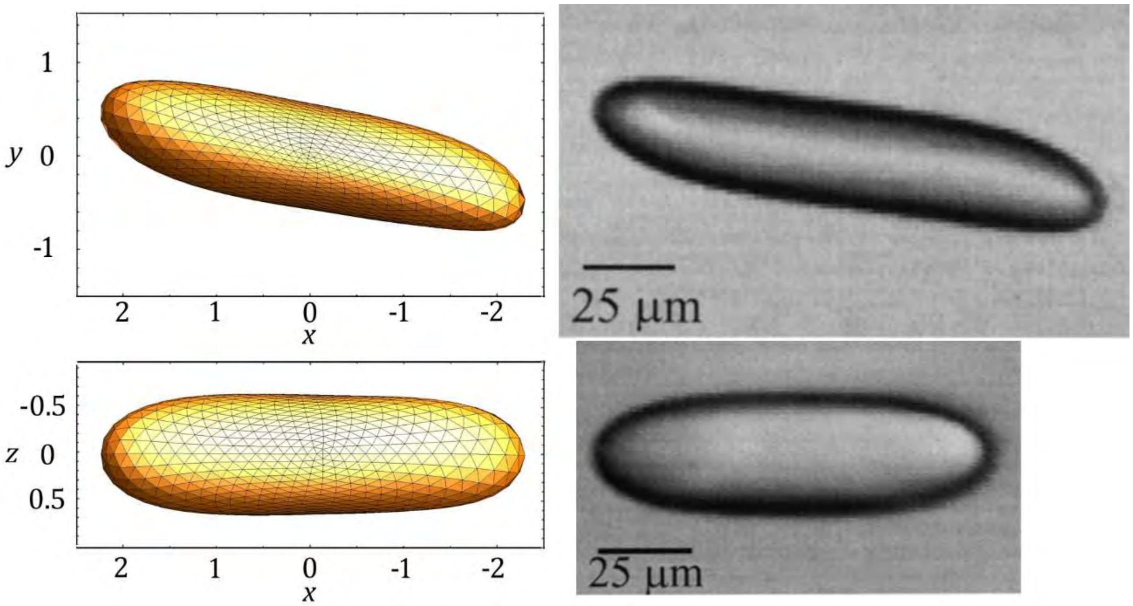

Figure 2.4 *Comparison between Numerical Simulation of BEM3D and Guido's Experimental Data for a drop deformation. Simple shear flow,* $\alpha = 0.0$*,* $\lambda_\mu = 1.4$*,* $Ca = 0.46$*. The experimental data were* $D = 0.70$*, and* $\theta = 32.05°$*. The numerical values were* $D = 0.67$*, and* $\theta = 34.8°$*.*

For the shapes on the left, these are the numerical result for a drop at $Ca = 0.46$. The radius of the drop is 25μm. However, this image is modeled with all lengths of the drop as a ratio *wrt*. the initial radius. Thus, the simulated shapes have different length scales. Besides, observing the image of the experimental drop, the main length scale is approximately 4 times the initial radius. Unfortunately, the experimental projection of the drop does not correspond to the same drop as the *xy-plane* projection. Comparing the scale of 25 μm, the *xz-plane* projection is shorter than the *xy-plane* projection implying a strong discrepancy about the correctness of the deformation. For the experimental conditions of Fig. 2.4, the drop attain a critical deformation. This situation implies that there is not a stationary shape, so if the photographs were not taken at the same time, the dimension would not have been the same. These circumstances could have happened in the experimental images. However, the numerical simulation predicts the shapes.

2.3 Comparisons of numerical vs. experimental data for *extensional flow*

For any *2D-flow*, it is hard to expect that the cross-section behavior would not be *asymmetric* —especially for large Capillary numbers— in contrast with earlier predicted theoretical or experimental results. Whenever a third axis does not contribute to the deformation of the flow, that is, is essentially passive, it should be expected to contribute to the shape of the drop in a different manner than the principal axes on the plane of the flow. Likewise, this condition ought to be observed ($i.e., L \neq B \neq W$) whenever the rate of deformation of the three axes are different, even when the flow is 3D. Hence, only with pure elongational flow we may expect that the deformation on two perpendicular directions might have the same values, with the third axis being the principal deformation scale.

The *deformation of drops immersed in strong flows* has not being studied intensively. Most of the earlier works are those of Taylor and of Leal and collaborators (Bentley & Leal, 1986a) Stone, Milliken, etc., only the work of Rosas *et al.*, addressed these flows more recently. The assumption of $L \neq B, B = W$ dominated in previous studies, (Rallison, 1980). Now, this assumption is now known to be of too limited validity and, for this reason, it becomes desirable to calibrate my numerical method with experimental data about cross-sectional dimensions, (Guido & Villone, 1997). If a drop in *simple shear flows* shows non-circular cross sections, *in 2D strong flows* the cross section will most probably not, as will be show. Figure 2.5 present the shapes of the drop (perpendicular) projections on two planes: xy- and zy-plane, *i.e.*, elongation along *y-axis*, compression along *x-axis* and neutral *wrt*. the *z*-axis. It is worth remarking that the drop becomes clearly a rather flat ellipsoid, resembling a typical Mexican *guarache*[3]. *Guarache* in this study is refer to a drop deformed with the cross section flattened, the principal axes *W-axis* is bigger than *B-axis*, and the size

[3] *Guarache* (wa'ratʃe) is a Mexican food consisting of an oblong, fried masa base; the name *guarache* is derived from the shape of the masa, similar to the popular sandals of the same name.

of *W-axis* is equal or major than the initial radius of the drop. Figure 2.5 (d) indicates the profile of a drop with $Ca = 0.15$ and $D_T = 0.37$.

The advantages of my *3D-boundary element method*, described here, for studying *asymmetric forms of drops* are: (1) the code was optimized to improve the description of the drop surface with more elements (2048) —results show a different behavior in the dynamic of drop deformation analyzed in the next Chapter—; (b) The code is capable of simulating different kinds of flows, from *shear flow* to *extensional flow*, and hence studying the effects of deformation along the third neutral axis; and finally, (3) conservation of volume of fluid can be carefully monitored to achieve good prediction on the relative deformation along all axes. However, with this code is not possible to study the complex geometry of some experimental devices such as Four-Roll-Mills or Two-Roll-Mills, mainly because a detailed mesh of the description of the rollers surfaces is required, and such efforts will be addressed in the future.

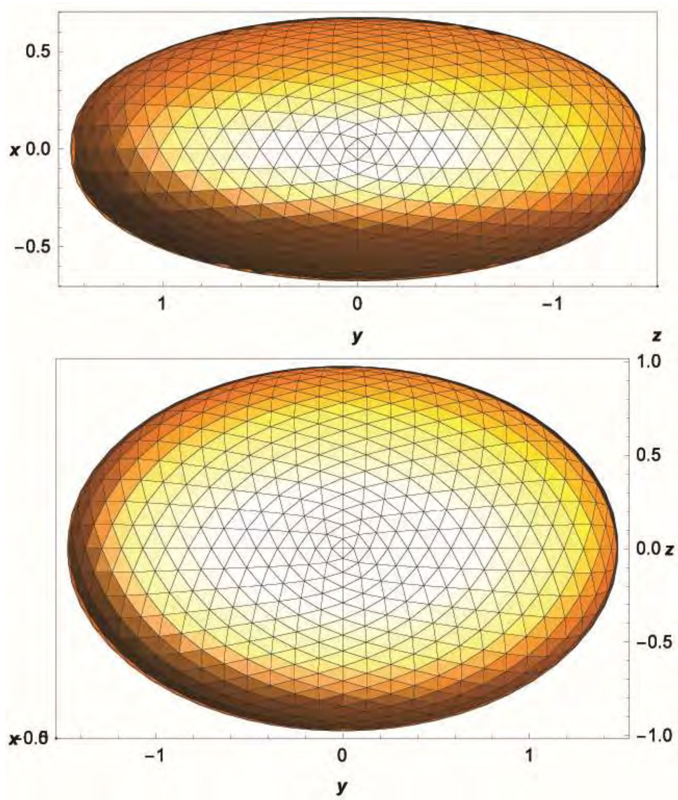

Figure 2.5 Deformation of the drop induced by a 2D-elongational flow. Elongation along y-axis, compression along x-axis and a neutral deformation for z-axis. xy-plane, above; zy-plane below.

The experimental studies of the length scales of drop deformations, induced by *extensional flows*, is scarcer compared with the data on deformations in *simple shear* *flows*. A comparative analysis like that possible for *simple shear flows* (see previous Chapter) also presents other difficulties —*i.e.*, data of stationary states of deformations are more difficult to measure because critical capillary number are smaller, with most experiment inducing the rupture of the drops. Add to this, that there is not an analysis of cross-sections of drops on any kind of fluid in *extensional flows*.

Taylor, Acrivos and Hinch elucidate theoretically the deformation dynamics of drops in *2D-elongational flow* using an asymptotic approximation to describe the form of the drop (Acrivos & Lo, 1978), (Hinch & Acrivos, 1979) and (Hinch & Acrivos, 1980). Furthermore, it is possible to compare as well my method versus the experimental data of (Stone, Bentley, & Leal, 1986), (Stone & Leal, 1989a) and (Stone & Leal, 1989b) albeit only for the L and B dimensions.

Capillary Number	D_T obtained by Stone and Leal	D_T obtained by BEM-3D	*B-axis*/r0, BEM-3D	*W-axis*/r0, BEM-3D
0.05	0.08	0.08	0.93	1.00
0.10	0.18	0.16	0.85	0.98
0.13	0.32	0.33	0.78	0.98
0.15	0.37	0.38	0.72	0.95
0.175	0.47	0.45	0.65	0.87

Table 2.1 *Comparison of deformation of drops deformed in extensional 2D-flow. Experimental data of Stone and Leal, and the numerical experiment using BEM-3D. The last two columns remark the difference between the B and W-axis.*

Figure 2.6 shows the steady state deformation for different capillary numbers and with a viscosity ratio of 1. The right most figures are sketches of the drop forms reported by Stone and Leal, (Stone & Leal, 1989a) for the flow plane of a 2D purely elongational flow; for these experiments, no data exists about the *xz-plane* shapes. The second and third column show drop forms for the *xy-plane* and *xz-plane* as predicted by this code simulations. The comparison is for the same capillary number and viscosity ratio. Figures indicate planes of observation similar to those shown in Figure 2.5: the *xy-plane* is the plane of the flow and the *xz-plane* depicts the cross-section of the drop. Figure 2.6(a) corresponds to a drop deformed with $Ca = 0.05$ and a stationary deformation of $D_T = 0.08$; the value of *W-* and *B-axis* are presented in Table 2.1; (b) shows a drop with $Ca = 0.1$ and $D_T = 0.18$; (c) shows a drop with $Ca = 0.13$, $D_T = 0.32$.

The numerical comparison is in the left part of Figure 2.6. In this part of the image is evident the change of the profile of the drop when the observation is like experimental devices, (left up). However, in the other direction, (left down), the drop has different dimensions in *extensional 2D-flow*.

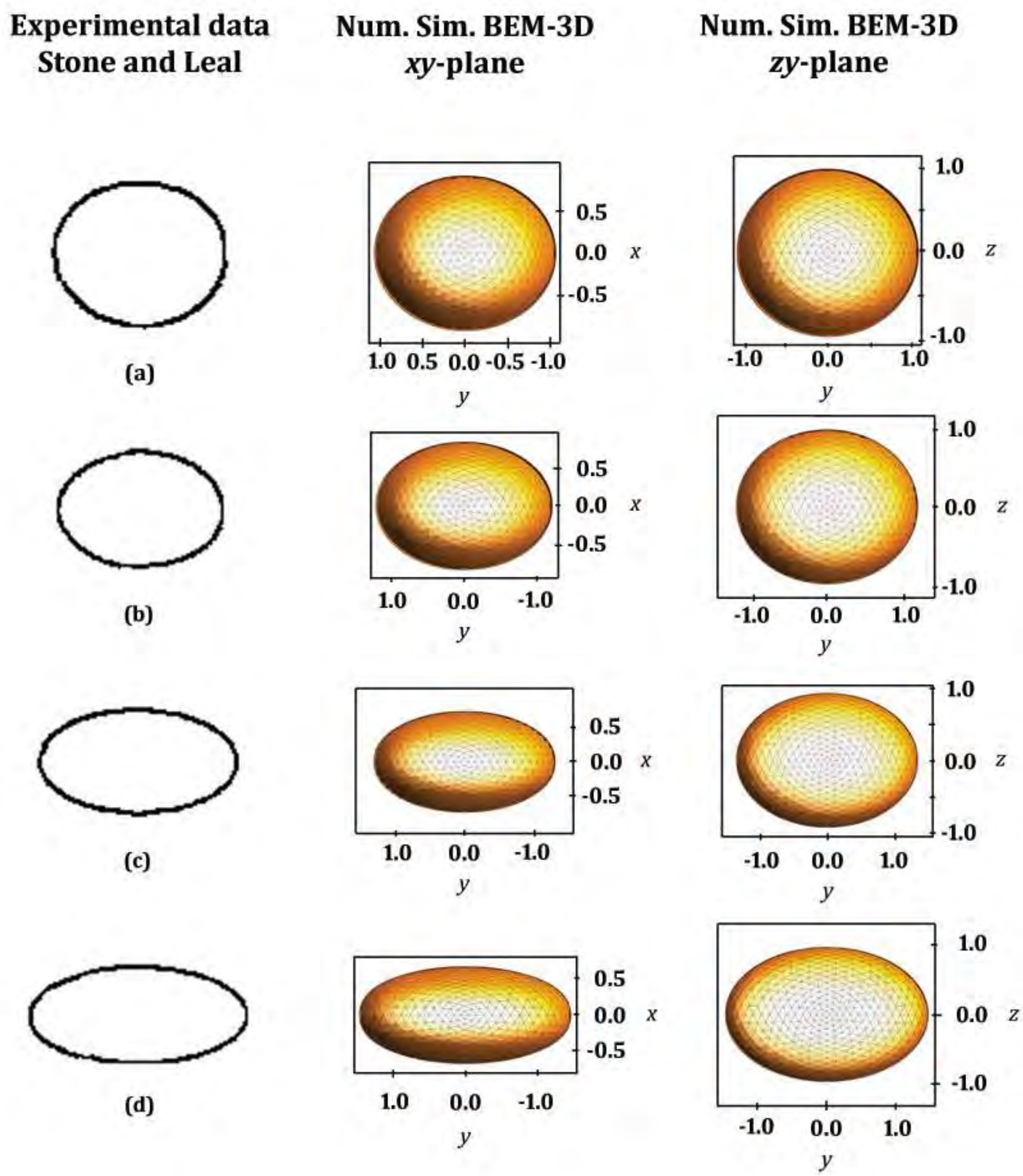

Figure 2.6 *Comparison of steady shape of deformation between experimental data vs numerical simulation of a drop in extensional 2D-flow with* $\lambda_\mu = 1.0$*. a)* $Ca = 0.05$ *and* $D_T = 0.08$*; b)* $Ca = 0.10$ *and* $D_T = 0.18$*; and c)* $Ca = 0.13$ *, and* $D_T = 0.31$.

I address a discussion of drop deformations induced by pure *extensional flows* in Chapter 7. Given that the nature of this phenomenon is quite complex, it is necessary to have a detailed explanation of the observed phenomena in this regime. Chapter 7 begins with an analysis based in a theory developed in the 70's, which predicts the stationary state of deformation considering the axis-symmetric behavior in the cross section. The comparison will be made of the *extensional flow* in 3D, as Acrivos and Hinch made, (Acrivos & Lo, 1978) and (Hinch & Acrivos, 1979). Finally, the comparison with the *extensional flow* in 2D will be made.

CHAPTER 3.

Shape of a drop immersed in a fluid under an elongational flow with vorticity; small ratios of viscosity

In Chapter 1 the viscosity ratio, λ_μ , was introduced. Here, the following results correspond to a drop immersed in another much viscous fluid $\lambda_\mu \ll 1$. For these cases, a large deformation of the drop is possible, but the orientation of the drop does not coincide neither with the direction of the flow nor the orientation of the principal axes. For these cases, the time evolution of the deformation of the drop provides information about the maximum possible deformations but no less relevant are steady state orientations of the drop in the flow. Thus, given that the shape of the drop is best described by three length scales, here I present the time-evolution of all axes of the drop, as well as particular features shown during the process before reaching the stationary state[4]. Plots of the evolution of deformations for multiple capillary numbers are plotted in each Graph; all traces correspond to the application of *a steady elongational flow* —with $\alpha = 0.13$. The data for other types of flows — $\alpha = 0.03$ and $\alpha = 0.05$— present a similar behavior. The values of stationary shape, deformation and orientation were used in figures to shows the comparison with the flow $\alpha = 0.13$.

[4] This chapter was submitted in June of 2016 to Journal of Physics; it is currently published. The final version of this work is on the Appendix A.

3.1 Time evolution of drop deformation

For a drop immersed in a steady flow, and when the viscosity ratio is small, if the capillary number is less than the critical capillary number, it deforms until it reaches a stationary shape (Taylor, 1934), (Bentley & Leal, 1986a), (Rosas, Reyes, Minzoni, & Geffroy, 2014). Figure 3.1 shows the time-evolution of the shape of the drop from a spherical (initial) to the final elongated, stationary shape. For these traces, the various capillary numbers were obtained by increasing the value of *G*, Eq. (1.6). One can observe that the stationary state is reached for different times and those values vary as the capillary number changes. The analysis of drop deformation presented in this Chapter focuses on the stationary shape detected in the numerical simulations.

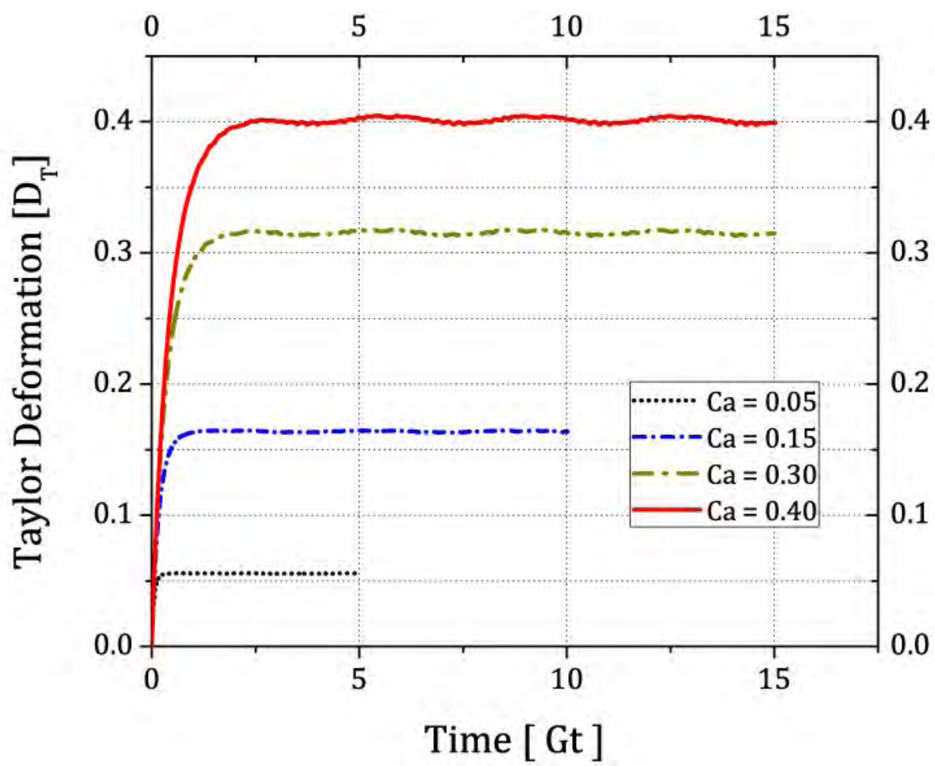

Figure 3.1 *Taylor's Deformation of the drop vs. Time.* $\lambda_\mu = 0.012$, $\alpha = 0.13$.

3.2 Attained stationary states

The numerical experiments explore effects of the applied flow on the drop shape. Figure 3.2 shows the time evolution of the 3 principal axes of the drop shape under the application of steady flow condition. The simulation is stopped when the drop deformation no longer changes; see Figs. 3.1 and 3.2. The deformation values numerically attained match those obtained experimentally by Rosas (Rosas, Reyes, Minzoni, & Geffroy, 2014) within $\pm 5\%$ for $0.05 \leq Ca \leq 0.25$ and $\pm 1\%$ for $0.3 \leq Ca \leq 0.4$.

Figure 3.2 shows that the stationary state is reached as early as 10% of the total simulation time; corresponding to approximately 1 unit of dimensionless time; the drop remains in the stationary shape for the rest of the simulation. Although small oscillations in the stationary values persist, the amplitude of these oscillations is less than $\pm 1.5\%$ of the average long-time value. The same behavior was observed for different capillary values. Chapter 6 presents an analysis of these oscillations in the stationary shape of the drop, with possible phenomena explaining these scenarios.

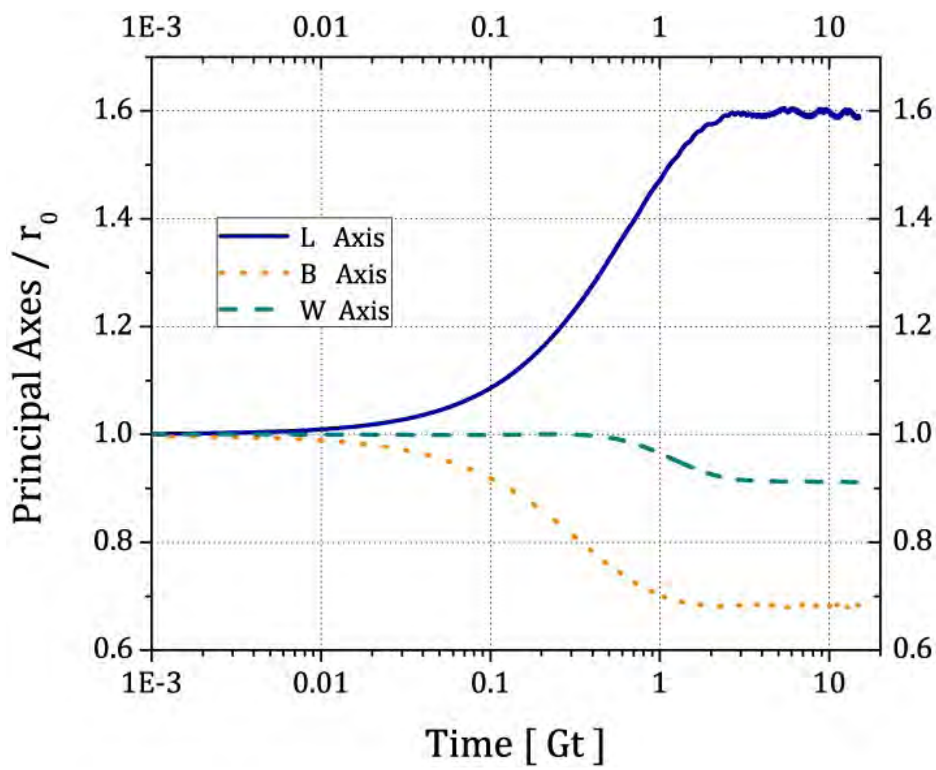

***Figure 3.2** Evolution of the three principal axes of the drop in a flow with $\alpha = 0.13$, $Ca = 0.4$ and total time $Gt = 15$.*

As well, evolution of lengths for the *B- and W-axes* indicates that the cross section of the drop is not circular, for *B* values are significantly smaller. To observe the non-symmetric form of the drop, the ratio of *B-* vs. *W-axes* is presented in Fig. 3.3. On one hand, it can be observed that in the initial 10% of the evolution and with $Ca = 0.4$,

B/W decreases and undershoots its stationary value. On the other hand, the rates of change for B and L are very similar, and are mainly determined by the magnitude of the vorticity. However, the changes of length of W *wrt* B clearly occurs at a faster pace; actually, based upon Fig. 3.2, the rate appears to be about four times faster for $Ca = 0.4$. The *B-axis* length changes at a rate similar to the *L-axis*, while the elongation of the *W-axis* lags in time. Up to time $Gt = 1$, the *W-axis* does not change significantly; it changes about the time when the other two axes have reached their stationary values.

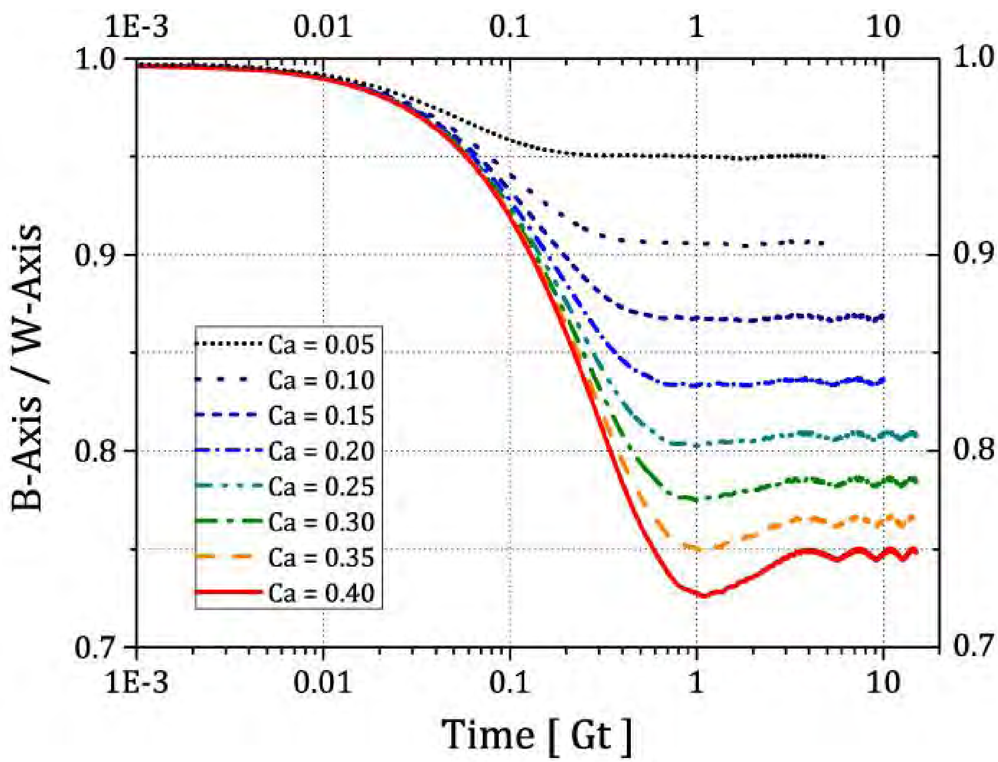

Figure 3.3 *Ratio of the B wrt. W axes in numerical simulations for a flow with* $\alpha = 0.13$ *and total time Gt =15 for different values of Ca.*

For *simple shear flows*, my simulations produce values for the ratios of the cross section that are similar to the numerical results previously reported (Guido & Villone, 1997) and (Kennedy, Pozrikidis, & Skalak, 1994), having as well the same general behavior, even though the viscosity ratio reported is larger.

Figure 3.4 presents the effects of the type of flow on the ratio of length for B and W. Even though higher values of α imply less vorticity, the *2D-character* of these *elongational flows* is still present. The drops retain a more *guarache* shape than for flows with more vorticity. It will seem to imply that the neutral deformation axis is a dominant effect that may persist even for pure *2D-elongational flow*. That is, when $Ca = 0.4$, and for flows with $\alpha = 0.03$, the deformation achieved is less than 85% of the values reached for flows of $\alpha = 0.13$.

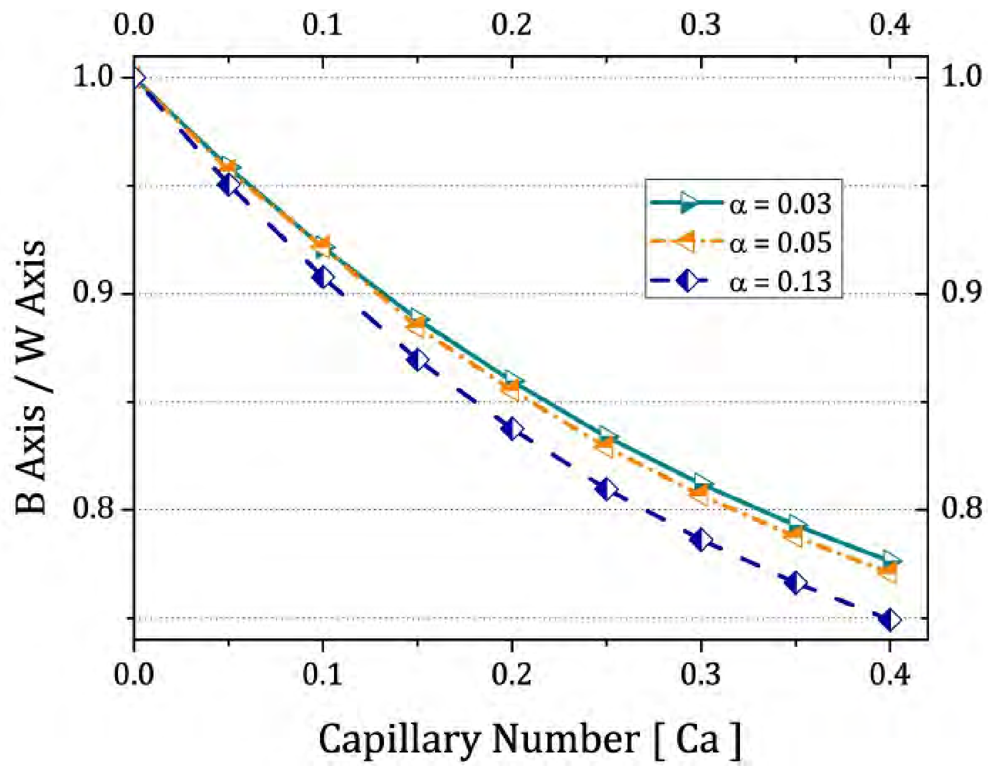

Figure 3.4 *Ratio of the B- wrt. the W-axes under stationary state conditions. Simulations traces correspond to different elongational flows with* $\alpha = 0.03$ *,* $\alpha = 0.05$ *and* $\alpha = 0.13$ *and for different values of Ca*

Figure 3.5 presents a comparison of (a) experimental results (Rosas, Reyes, Minzoni, & Geffroy, 2014), (b) the simulated values for $\alpha = 0.13$, and (c) the analytical results of Taylor-Cox (Taylor, 1932) (Taylor, 1934) and (Cox, 1969) for $\alpha = 0.0$. The compared property is the stationary Taylor deformation values on the *xy-plane*.

Cox (Cox, 1969) and Taylor (Taylor, 1934) theories for *simple shear flows* establish a linear relationship between deformation *vs*. the capillary number. However, it is clear that the dependence observed in simulations of stronger flows is slightly non-linear; albeit theory, simulations and experiments agreeing fairly well. Chapter 7 presents a more detailed comparison between numerical simulations and the experimental data of Rosas (Rosas, Reyes, Minzoni, & Geffroy, 2014).

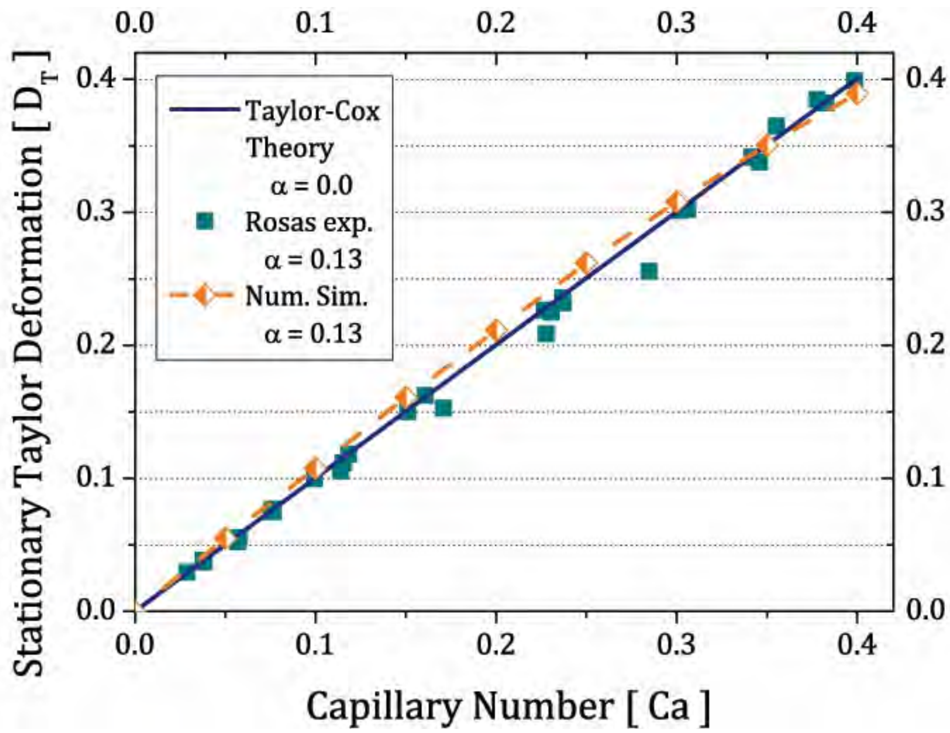

Figure 3.5 *Deformation of the drop for numerical simulations and experimental data for α= 0.13 and analytical prediction of Taylor Cox in simple shear flow α= 0.0 for different values of Ca*

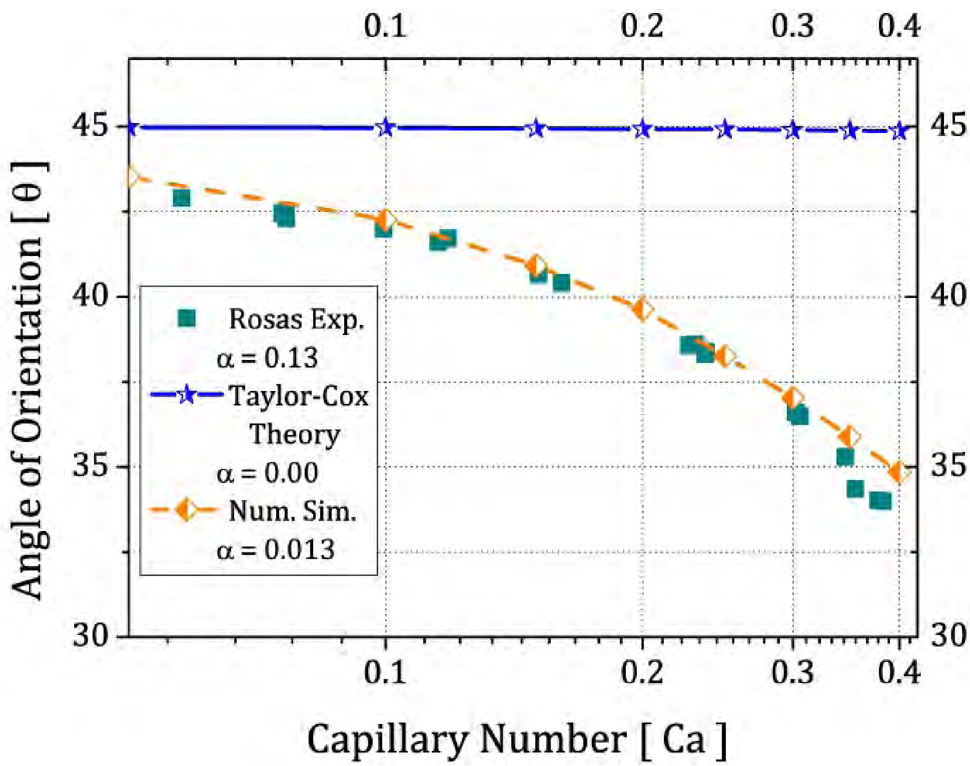

Figure 3.6 *Orientation of the drop for numerical simulations and experimental data for* $\alpha = 0.13$ *and analytical prediction of Taylor Cox in simple shear flow* $\alpha = 0.0$ *for different values of Ca*

The second parameter that characterizes the shape of the deformed drop in the flow is the orientation angle; see Fig. 1.1(b). Figure 3.6 shows the orientation of the principal axes of the drop as it elongates. This parameter does play an important role because the principal axis of deformation —*L-axis*— is located at 45° degrees with respect to flow direction, and the drop tends to align itself closer to the flow direction for higher *Ca* number flows; that is, the drops elongate and rotate away from the 45° orientation. Figure 3.6, shows that the orientation angle values for drops predicted by Taylor and Cox theory remain fixed at 45°, contrary to all experimental evidence.

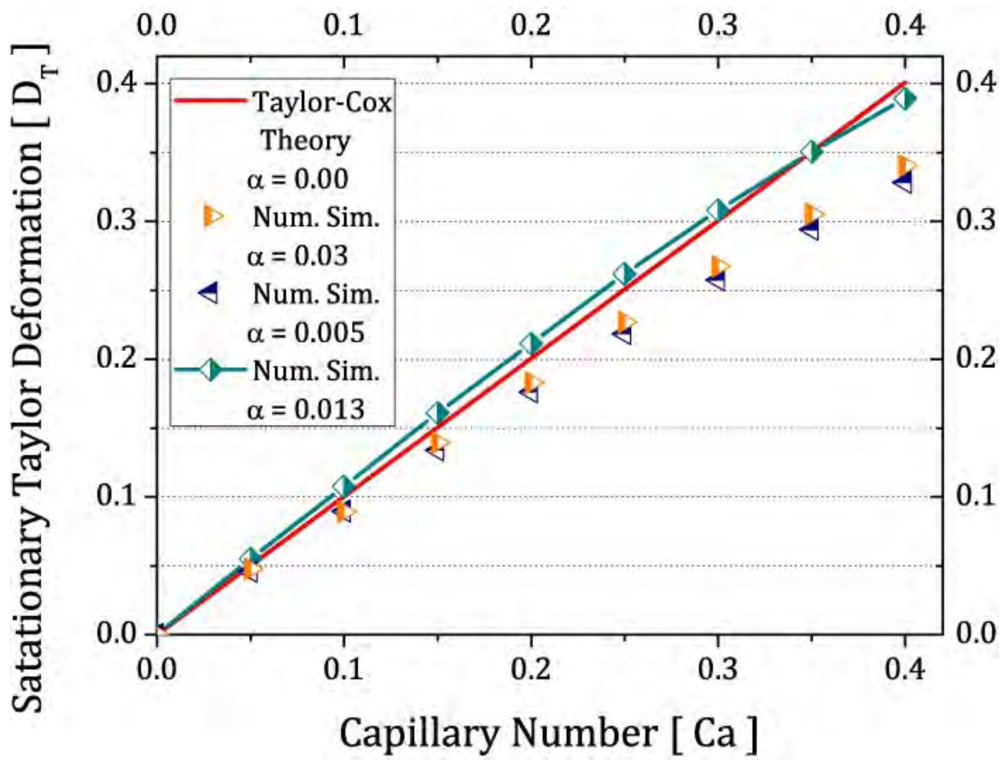

Figure 3.7 *Deformation of the drop for numerical simulations for flow with* $= 0.03$ *,* $\alpha = 0.05$ *and* $\alpha = 0.13$ *13 and analytical prediction of Taylor Cox in simple shear flow* $\alpha = 0.0$ *for different values of Ca.*

In Figure 3.7, the stationary principal deformations vs. the *Ca* number were plotted for the different types of flows. As in Fig. 3.4, the behavior of the relationship between Taylor deformation and capillary number appears to be non-linear, however under predicting the deformation with respect to the theoretical predictions of Cox and Taylor. That is, Taylor-Cox theory does not rotate the drop, hence tends to cause a larger deformation. The experiments and these simulations do rotate the drop towards the flow direction, away from the principal axis of deformation —*L-axis*—, causing a smaller deformation. Thus, the equivalence in deformation of theory, $\alpha = 0.0$, and a flow with $\alpha = 0.13$ may be understood in this manner.

Finally, the measurements of the principal axes of the drop are shown in Fig 3.8. These values were obtained in the stationary shape of the drop. This figure is similar as the Fig. 2.3 from Chapter 2. The analysis of the L, B and *W-axis* of the ellipsoidal drop, in steady state, clearly show deviations from the axisymmetric shape as the flow type parameter α increases. The behavior of the *W-axis* is similar —remains unchanged for the values $\alpha = 0.03$, 0.05 and 0.13— indicating an effect due mainly to the 2D character of the flow, and regardless of the vorticity of the applied flow.

The major changes observed occur for the *L-* and *B-axis*, with a reduction of vorticity of the flow inducing a greater elongation of the drop along the principal deformation axis —*L-axis*—.

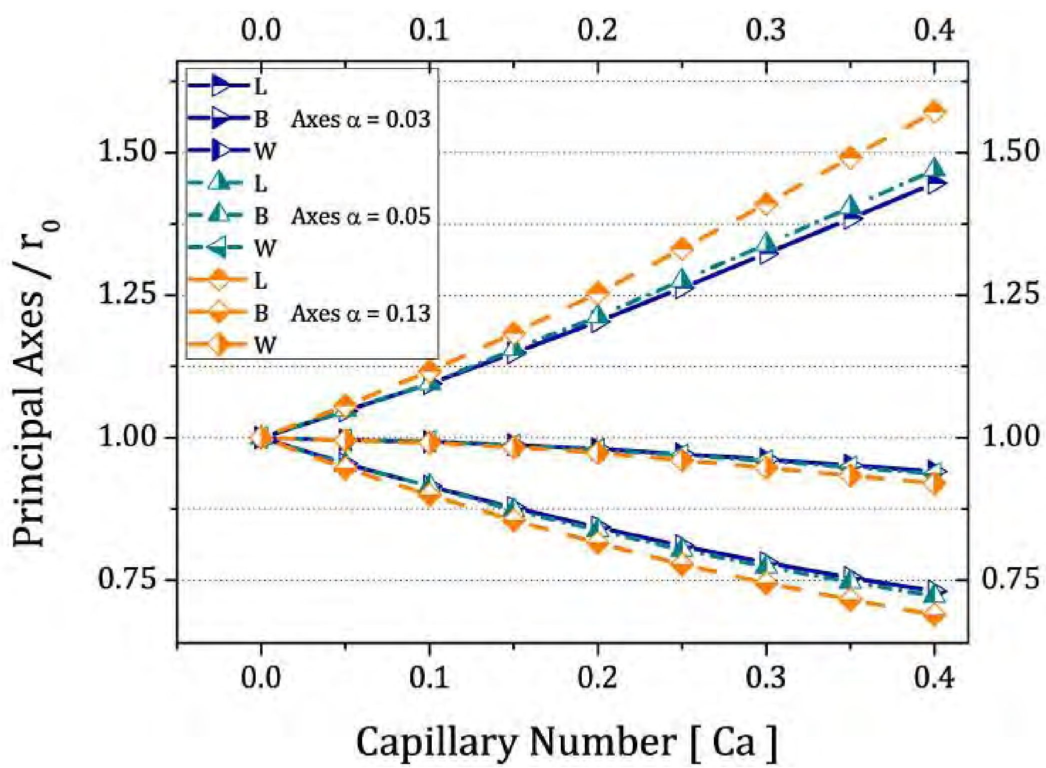

Figure 3.8 *Lengths (normalized) of the principal axes of drops under stationary flow with* $\alpha = 0.03,\ 0.05$ *and* 0.13*, for different values of Ca.*

Figure 3.8 shows that in general, stationary drop shapes do not have an axisymmetric cross section. The simulations here presented for drop forms induced in *2D-flows* do not appear to match the axisymmetric case established by G. I. Taylor and Cox, in agreement with previous results of Kennedy *et al.*, and Guido *et al.*, for the *simple shear flows*. This is likely the case for Ca < 0.4, and may be correct even if the capillary number is as small as $Ca = 0.05$. For Ca > 0.4, the deformation in the xy-*plane* continues to increase, but stronger than the numerical simulation shows, Taylor-Cox model overestimate the possible deformations. On the other hand, the simulated behavior appears consistent with the experimental data of Rosas (Rosas, Reyes, Minzoni, & Geffroy, 2014).

The difference between the *simple shear flow*, $\alpha = 0$, and flows with stronger degree of elongation induced larger drop deformations while the angle of orientation —principal *L-axis*— rotates away from 45 degrees. Another important observation is the increase in the principal *L-axis* value for stationary states when the flow parameter α becomes larger than simple shear rate flow, $\alpha = 0.13$; the relationship between the axis rate of growth and the capillary values goes away from the linear behavior as Fig. 3.8. shows. In Chapter 7, the non-linear behavior is more obvious because $\alpha = 1$.

Other differences can be observed, between the shapes of projections and the ellipses. For example, the drop with $D_T = 0.2328$ and the $angle\ = 1.34°$ does not have an ellipse for projection on the *xy-plane*. Its shape resembles a higher order deformation, more like a sigmoid, as has been reported for drops in *simple shear flow*, Fig 2.4. The next two comparisons are similar, *i.e.*; they are not perfect elliptic shapes. So, as we can see in the next chapter, a drop deformed in a strong flow does not have a circular cross-section, nor the elongated shapes are ellipses, when the ratio of viscosity is large. These figures (shown in Fig. 4.7) could give us information of the stationary states attained.

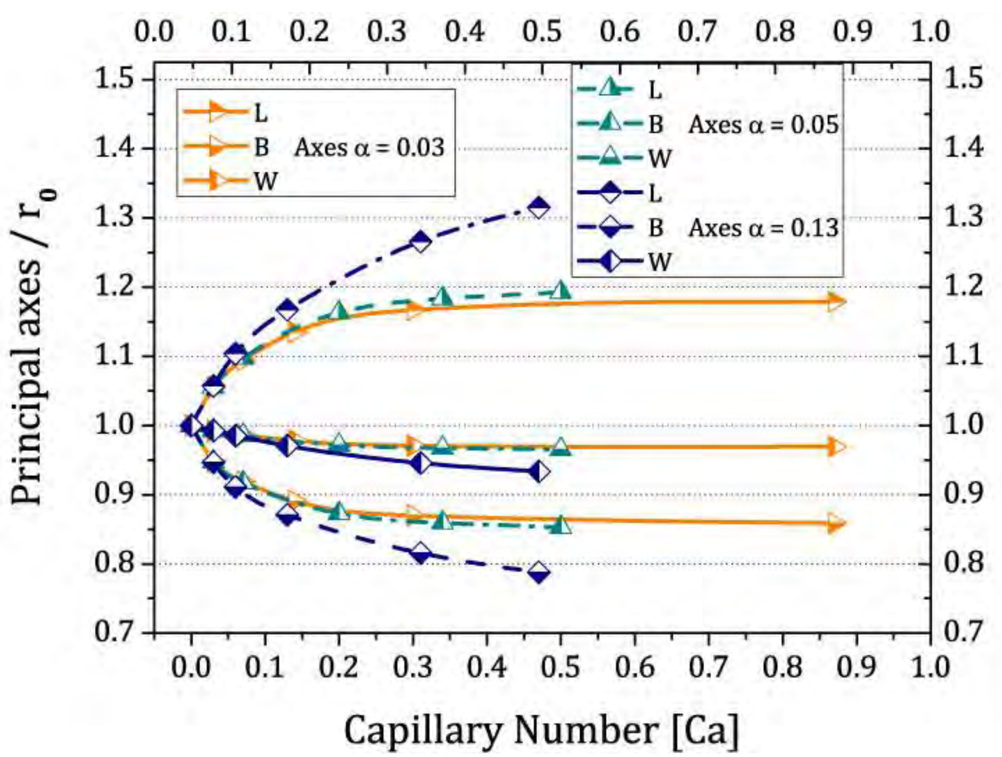

Figure 4.9 *Stationary values of the principal axes of the drop for a flow with* $\alpha = 0.03, 0.05$ *and* 0.13*, for different values of Ca.*

Figure 4.8 present the oscillatory behavior of the drop time evolution in a polar plot, for different capillary numbers and when the ratio of viscosity is large, $\lambda_\mu = 16$. The starting point is a spherical drop without orientation; once a flow is applied, the drops deforms and rotates. As the capillary number increases, its deformation grows, oscillations increase (present more cycles) and the trajectory is a full spiral which goes to a final steady state at long times. Figure 4.9 shows the principal axes of the drop in this regime of flow. The evolution of drops as *Ca* increases goes to a stationary state of deformation. For $\alpha = 0.13$, the first idea based in Fig. 4.4 and Fig. 4.5, is that a stationary shape in this flow is not attained. However, Fig. 4.9, shows that the behavior of the drop deformation in this regime may reach a final steady shape.

The unusual behavior of the drop length scales in a flow with $\alpha = 0.13$, implies the need to analyze carefully the deformation for elongated drops in this regimen. The shapes obtained in this flow are like ones in Fig. 4.7 *i.e.*; the shapes are not ellipses. The distortion of the shape is due to the change of the instantaneous directions of the flow. That is, the principal axes of deformation *L-axis*, is near the direction of the axis of elongation. However, when the drop turns around, going beyond the outflow axes, the *L-axis* of the drop now subjected to a compressional kinematics. The images show in Fig. 4.7 indicate the distortion of the drop due to the flow around the drop. To have the best information of the sigmoid shapes it is necessary to have a code capable given us information of the flow inside and outside the drop to understand the formation of the sigmoid shapes and the transition of the regime near $\alpha = 0.13$.

CHAPTER 4.

Shape of a drop immersed in a fluid under an elongational flow with vorticity; large ratios of viscosity

If the viscosity of the drop is very high, one can expect that the deformation of an embedded drop in a flow might resemble more closely to that of a rigid particle. However, the drop still deforms, although slightly. The orientation of the drop will also differ too. Deformation of drops induced by flows where the ratio of viscosity, $\lambda_\mu >> 1$, thus present a different behavior compared with the cases studied in Chapter 3, and my observations are here presented.

Cox (Cox, 1969) developed a theory to predict the deformation of drops in *simple shear flows* when drops have a higher viscosity than the continuum phase. Using Taylor's theory, Cox presented an alternative (using an approximation with spherical harmonics, (Lamb, 1945)) but appropriate when the ratio of viscosity is larger than the values used by Taylor. Cox uses the inverse of capillary number and finds correlations between Taylor's deformation, the capillary number and the orientation of the drop, then, he compares his results with the experimental data of Taylor and Mason, (Taylor, 1934), (Rumscheid & Mason, 1961).

Years later, Leal *et al.*, produced a large collection of experimental and theoretical results on the behavior of drop deformation in strong flows (using a Four-Roll Mill) with viscosity ratios larger than 1, *e.g.*, (Bentley & Leal, 1986b). However, the observation of rotation of drops in those strong flows was not studied in detail until the work of Rosas *et al.*, (Rosas, Reyes, Minzoni, & Geffroy, 2014) and (Rosas I. Y., 2013); mainly because the induced rotation by strong flows with significant vorticity is larger —thus, easier to observe— than those observed with the Four-Roll Mill device used by Leal and coworkers. In contrast, as the Two Roll Mill device produces strong flows, the deformations obtained are bigger than those predicted for *simple shear flow* with the same capillary numbers.

In order to understand the role played by the orientation angle and deformation on an equal footing, Cox presented polar plots of deformation D_T and orientation θ. The time-parameterized numerical trajectories $D_T\ vs.\theta$ of the drop here presented are similar as those predicted by Cox for *simple shear flow* —earlier experimental trajectories by Mason *et al.*, are unfortunately too noisy to be of relevance for a comparative study. Rosas presents the first detailed parameterized plots for deformation showing that Cox model is qualitatively good, although some differences persist.

In this Chapter, the numerical study focuses on phenomena observed with in viscous drops. The ratio of viscosity is an important parameter in drop deformation since Taylor's work (Taylor, 1934) thus the importance of an analysis of deformation and rotation in the regime $\lambda_\mu \gg 1$ is important. Here an important tool is the use of $D_T\ vs.\theta$ trajectories.

4.1 Time evolution of drop deformation

Numerically, the BEM-3D method calculates the Double Layer Potential in order to have the time evolution of the drop deformation, with a significant increment of the time computation. Also, as Rosas found (Rosas, Reyes, Minzoni, & Geffroy, 2014), the runtime to observe the full evolution of drop deformation is much longer, compared to the computation time for all cases presented in Chapter 3. As an example, for a capillary number $Ca = 0.30$, in the case of $\lambda_\mu = 0.012$, the time to simulate the drop deformation to achieve the stationary states together with the retraction time towards the spherical drop is, in slowest case no more than $Gt = 25$. For the cases of this Chapter with the same capillary number but with a $\lambda_\mu = 16$, the minimum simulation time to observe the same behavior is near $Gt = 60$. For $Ca = 0.87$, to attain the steady state and to observe the retraction of the drop $Gt \approx 100$.

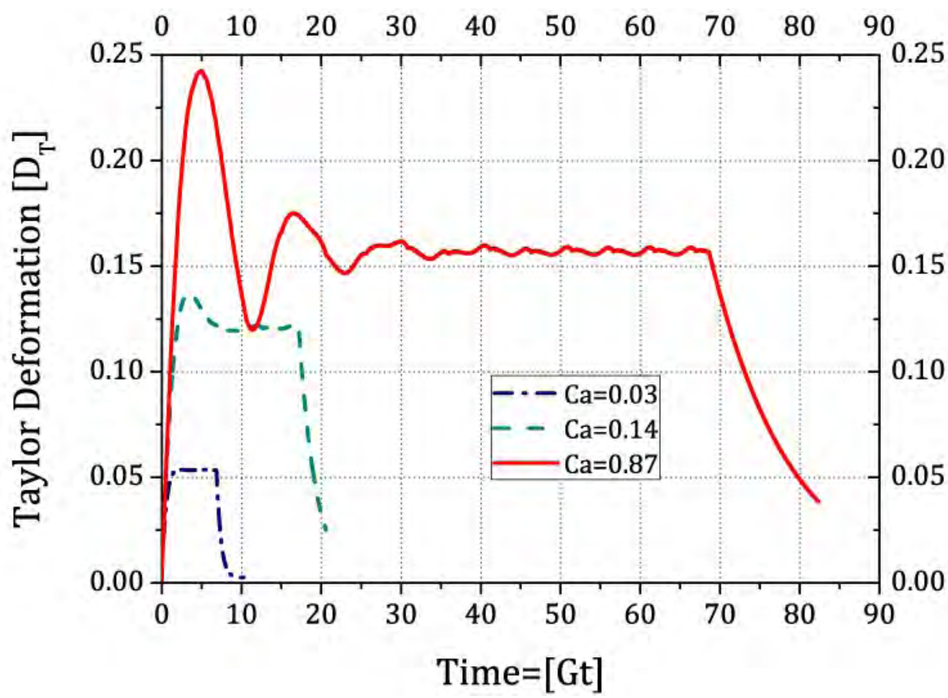

Figure 4.1 *Taylor's Deformation of drops in the time. $\lambda_\mu = 16$, $\alpha = 0.03$.*

Figure 4.1 shows typical curves of deformation histories for $Ca = 0.03$, $Ca = 0.14$, and $Ca = 0.87$ with viscosity ratio $\lambda_\mu = 16$, and $Gt = 83$. The flow parameter values are the same as Rosas used with the Two-Roll Mill (TRM) device (Rosas, Reyes, Minzoni, & Geffroy, 2014). For drops with $\lambda_\mu = 16$, the dynamics of drop deformation for $\alpha = 0.03$, 0.05 and 0.13 are similar, so the Figures presented are only for the *strong flow* with a value of $\alpha = 0.03$. However, in subsequent discussions the behavior observed for all flows used will be described.

4.2 Stationary states attained

As was explained in Chapter 3, the drop deformation reaches a stationary deformation for flows with a capillary number less than the critical value $Ca < Ca_{cr}$.

In other words, initial drop shapes correspond to a no-flow state (spherical shape); then, as the flow is applied on the continuum phase, the drop begins to deform. The high viscosity of the drop causes a weaker deformation of the drop than those cases observed in Chapter 3. The stresses on the drop, as a function of its viscosity must be balanced instantly with the stresses applied by the external flow. The drop deformation dynamic takes place as a competition between deformation-orientation *vs.* maintaining the equilibrium shape. At the end, the drop deformation attains a stationary shape, if $Ca < Ca_{cr}$.

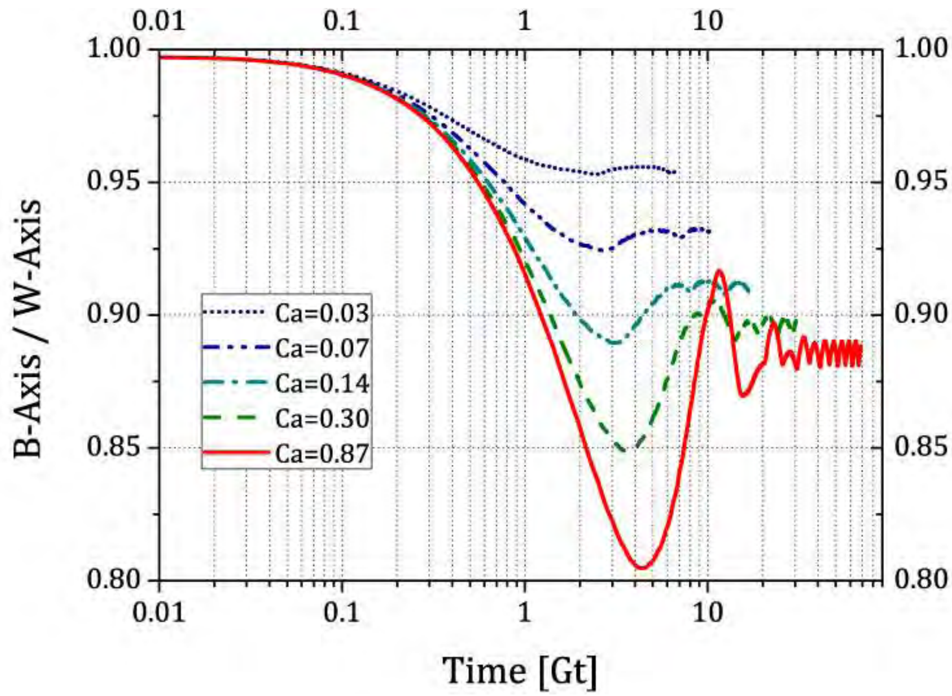

Figure 4.2 *Ratio of the B wrt. W axes in numerical simulations for a flow with α=0.03 and total time Gt =75 for different values of Ca.*

As shown in Figure 4.1, when $Ca = 0.03$, the drop deformations have a similar behavior in time as was observed in Chapter 3, but with a smaller length. The drop reaches a stationary shape, and if the flow is turned off, the drop returns to the spherical shape. When the capillary number increases a little more, the competition (as function of the rate of viscosity) is more evident. There is an overshot for the case of $Ca = 0.14$. Afterwards, the drop goes to the stationary shape. For the last two cases, *i.e.*, $Ca = 0.3$ and $Ca = 0.87$, the main overshoots are followed with multiple oscillations, with their maxima and minima attenuating in time until the drop shape attains an stationary deformation. Those oscillations are a consequence of the competition mentioned and due to phenomena with multiple retraction times.

In a similar way as in Chapter 3, data for the B/W-*axis*, shown in Fig. 4.2, indicates that the cross section of the drop is not circular and is time dependent, and may pulsate slightly. In Fig. 4.2, during the initial 10% of the evolution of B/W, the main oscillation occurs, with the amplitude decreasing subsequently until the stationary value is attained.

Figure 4.3 shows that stationary states are reached after 40% of the total simulation time; with the drop remaining in the stationary shape for the rest of the simulation. Although small oscillations in the stationary values persist, the amplitude of these oscillations is less than $\pm 2\%$ of the average long-time value. The same behavior was observed for a wide range of capillary values; see Fig. 4.1. When $\lambda_\mu = 16$ the rates of change for the *B*- and *W-axes* are very similar, constant, and appear to be mainly determined by the magnitude of the flow vorticity, in contrast to the behavior observed for drops of small viscosity.

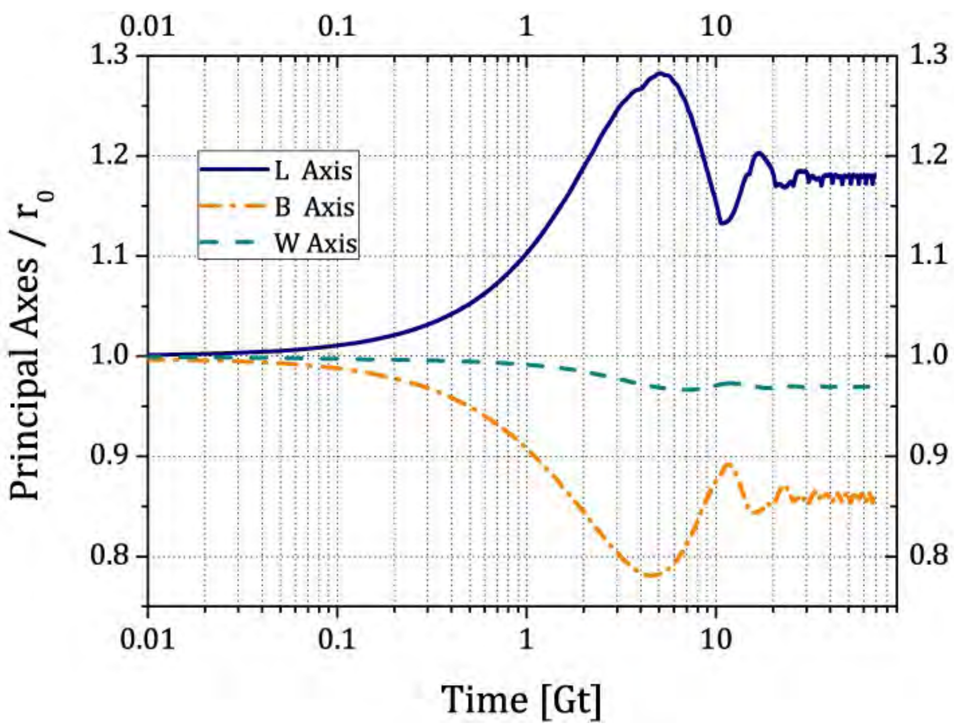

Figure 4.3 *Evolution of the three principal axes of the drop in a flow with* $\alpha = 0.03$*, Ca=0.87 and total time Gt =75.*

Figure 4.4 presents the ratio between the *B*- and *W*-axes after the steady state deformation is attained. After the steady state is reached, and for $\alpha = 0.03$, the cross-section shape remains close to elliptical, with the ratio being around 0.88. For $\alpha = 0.05$, the behavior is very similar. However, when $\alpha = 0.13$, this ratio decreases (more elliptic shape) as the capillary number augments. That is, vorticity decreases with smaller α values, but the *2D-character* of the flow remain dominant.

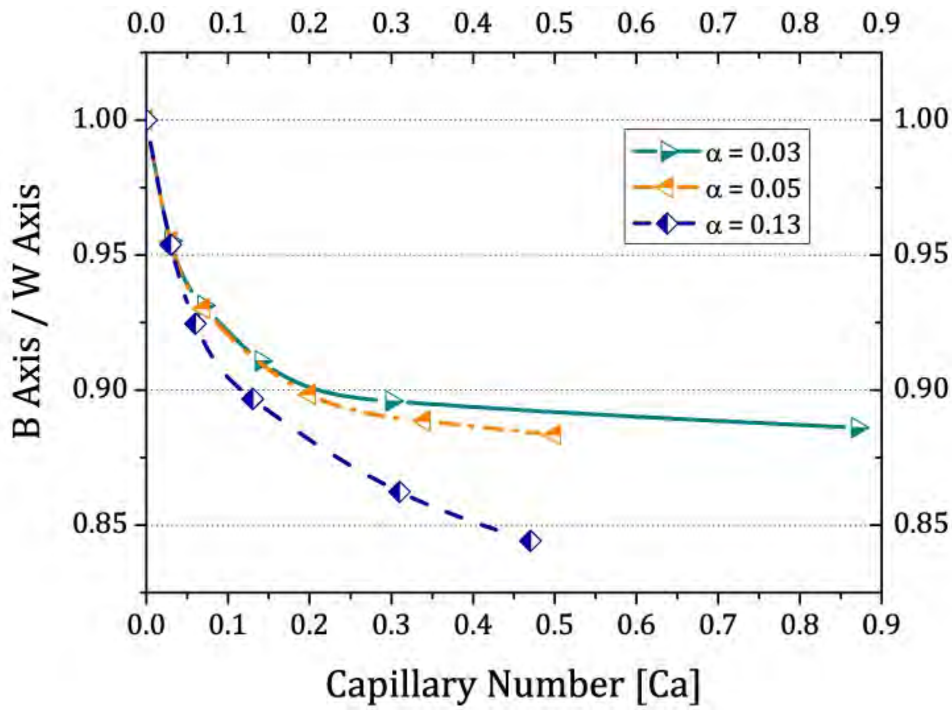

Figure 4.4 *Stationary ratio of the B wrt. the W axis in numerical simulations for flow with α = 0.03, α = 0.05 and α = 0.13 for different values of Ca.*

Figure 4.5 presents a comparison of multiple theoretical, numerical and experimental observations —most data correspond to $\lambda_\mu \simeq 16$. The theoretical model is that of Cox (corresponding to the red trace) for *simple shear flow*, while the experiments cover a very large range of values for the flow type, α. The experimental values are those obtained by Bentley and Leal, (Bentley & Leal, 1986b) for the larger values of the flow type, while those of Rosas address flows with more vorticity. For these high viscosity drops, there is a limit for the stationary deformation values in *simple shear flow*. Curves are similar as those of Cox theory predictions for *simple shear flow*; even for $\alpha \lesssim 0.13$, a maximum deformation in steady state has been observed by

Rosas. For small capillary numbers, the stationary deformation of drops depends on the capillary number, increasing as well under the more *elongational flows*.

In Figure 4.5 it is possible to infer the maximum values of steady state deformations, for different values of the *parameter* α. In particular, the experimental traces may show an unbounded deformation, indicating that the flow is close to its critical value of *Ca*, beyond which rupture of the drop may occur. Comparing to experimental data from Bentley and Rosas, (Bentley & Leal, 1986a) (Rosas I. Y., 2013), it is clear that a flow with $\alpha = 0.13$ appears not to induce the rupture of drops, in contrast to a flow with $\alpha = 0.2$, which indicates a flow type readily capable of rupturing a drop even with such high viscosity. Thus, between $0.13 < \alpha < 0.2$ a regime transition must occur.

That is, the existence of a critical capillary number, may indicate a flow type transition between a deformed drop or an unstable flow solution. For these reasons those results motivate a careful study of *strong flows* near the instability.

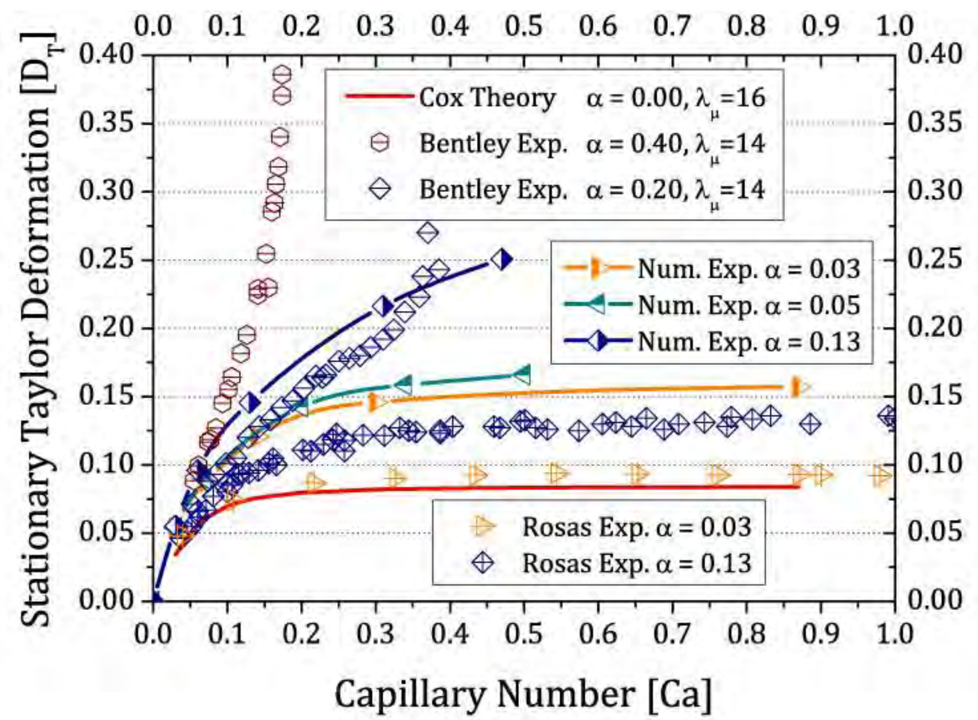

Figure 4.5 *Deformation of the drop for numerical simulations, experimental data of Rosas for* $\alpha = 0.03$, $\alpha = 0.05$ *and* $\alpha = 0.13$; *Experimental data of Bentley for* $\alpha = 0.4$ *and* $\alpha = 0.2$ *and analytical prediction of Taylor Cox in simple shear flow* $\alpha = 0.0$ *for different values of Ca.*

These limiting Ca values depend not only on the flow type α but also on the ratio of viscosity λ_μ. There is consistency in the values of the stationary deformation values at this limit; *i.e.*, the strong flows applied generate less vorticity than *simple shear flow*, so the stationary deformation values must be larger than the *shear flow* case; weaker rotation does not inhibit elongation as it is shown in Fig. 4.5. Thus, for $\alpha = 0.13$ the attained deformation in stationary states is larger than the weaker flows. For a given flow, higher viscosity implies that the deformation may be small while the vorticity tends to rotate away from the principal axis of deformation. Thus, higher viscosity drop may deform to a stable elongation at higher values of the ratio of viscosities λ_μ.

Furthermore, at the critical limit conditions, the length scales of the deformation are more difficult to define because no steady elongation in observed; *i.e.*, employing the information of Fig. 4.4, the steady shape of cross section in this flow shows a deviation which is far from the horizontal limit, and which implies an ever-increasing *L-axis* length. In fact, the fluctuation in the deformation of the drop implies an oscillation in the angle of orientation of the drop too, see Fig. 4.2.

There are noticeable differences for the stationary deformations, between deformations simulated numerically and the experimental data of Rosas. And there may be various possible explanations for these differences. One basic possible effect may be the fact that the Two Roll Mill device required a control scheme to maintain the drop near the stagnation point permanently. Together with the non-idealities of the mill, fluctuations in the external flow are only natural, as Rosas explains in his work (Rosas I. Y., 2013), Chapter 3.[5] At the end, rotation dominates the deformation before the stationary value. On the other hand, numerical methods do not have the possibility to simulate all possible non-ideal "laboratory" conditions. Please note that in Chapter 6, a possible new explanation about the differences between the numerical simulations and the experiments is discussed. This last mechanism may indicate why

[5] These fluctuations occur with a frequency comparable with the diffusivity of vorticity, reducing mainly the rate of deformation, without altering the rotation of the drop in the experiments, hence causing a smaller deformation of the drop by the experimental device vs. numerical simulations.

the numerical simulation do not show as many oscillations as the experimental data reported (Rosas I. Y., 2013).

Figure 4.6 shows the orientation of the drops under different flows. Bentley experiments mainly show the limit of resolution of the experimental device. The Four Roll Mill has the best control of the drop as the parameter α goes near values of one. In the other extreme, as the *parameter* α decreases, the control of the drop degrades because there are two rolls that participate marginally in the control of the drop position, (Bentley & Leal, 1986a). This problem is shown on the orientation angle data, because the values appear to have perceptible fluctuations.

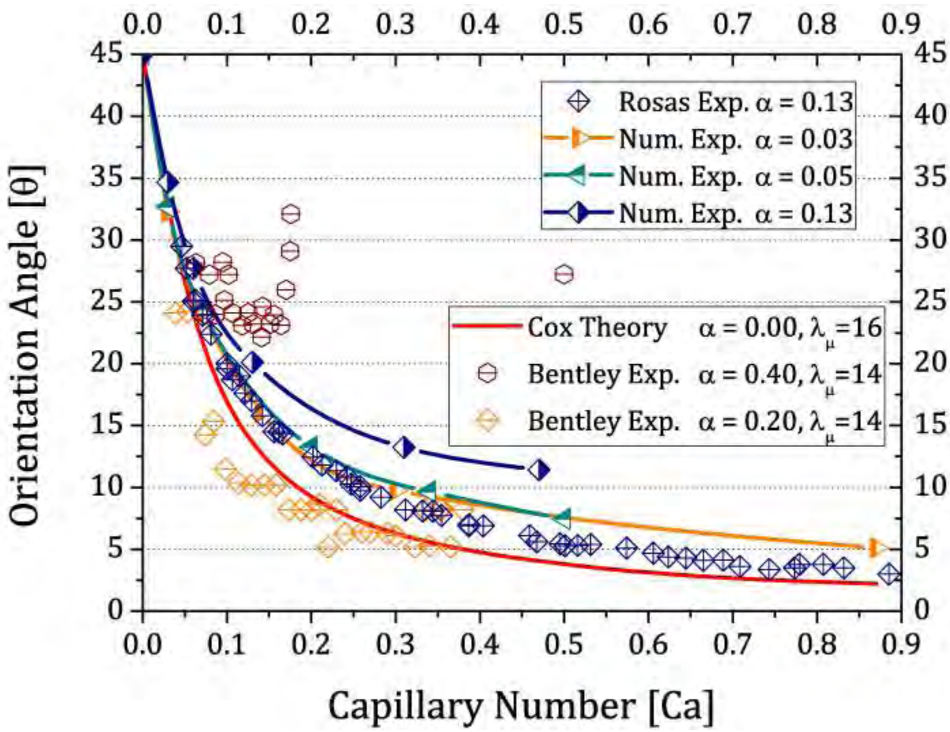

Figure 4.6 *Orientation of the drop for numerical simulations, experimental data of Rosas for $\alpha = 0.03$, $\alpha = 0.05$ and $\alpha = 0.13$; Experimental data of Bentley for $\alpha = 0.4$ and $\alpha = 0.2$ and analytical prediction of Taylor Cox in simple shear flow $\alpha = 0.0$ for different values of Ca.*

The orientation angles present a smooth behavior on the experimental data of Rosas. In Fig. 4.6, only the experimental data of $\alpha = 0.13$ is presented, and not data for $\alpha = 0.03$ and $\alpha = 0.05$, because the latter show the same behavior as Cox theory. Numerical data have a similar behavior as the experimental data for $\alpha = 0.13$. However, the angle of orientation is systematically larger than the experimental data, indicating that simulations must present a stronger deformation for the same flow conditions: α and Ca. For example, the trace of numerical results of the orientation angles, for $\alpha = 0.03$, is similar to the experimental data of $\alpha = 0.13$. As noted before, fluctuations may explain discrepancies between the numerical and the experimental data, for a reduction of rate of deformation does not reduce simultaneously vorticity.

The numerical data shown in Fig. 4.6 indicate that for a flow with $\alpha = 0.03$ the orientation of the drop attains a value like that of *simple shear flow*, red curve. However, when the flow parameter increases, to $\alpha = 0.13$, the orientation goes to larger values (closer to the out-flow axes). Also, when the flow applied in the continuum phase generates a deformation greater than the stationary deformation, no stationary state is attained and the drop will be oriented parallel to the angle of the out-flow axes (Reyes, 2005), (Rosas, Reyes, Minzoni, & Geffroy, 2014) and (Escalante, Reyes, Rosas, & Geffroy, 2015).

In summary, the deformation seen with the numerical method is larger than that observed with experimental data, while the orientation angles of the simulations are lower than the results of Rosas. In Chapter 6 these discrepancies are studied under the light of a new phenomenon of drop deformation not observed until this work.

Figure 4.7 shows the numerical images (colored images) of deformed and rotated drops in a flow with $\lambda_\mu = 16$ and $\alpha = 0.03$ in the *xy-plane* (Fig 1.1b). These images correspond to data of a drop with $Ca = 0.87$, both the deformation as well as the orientation angle are in the *xy-plane*. The black shapes are the projections on the *xy-plane* for the numerical images. The length of the deformation and the orientation angle are calculated evaluating the maximum distance from the center of the drop *wrt* the nodes of the mesh. The typical procedure to determine deformation and orientation is to use an adjusted ellipse to the projected image. The last set of images observed in Fig. 4.7 corresponds to different ellipses, adjusted with the *ImageJ*® program based on the projection of the drop simulated. As the information of the Fig. 4.7 indicates, there are very small differences of parameters between the numerical data and the *ImageJ*® analysis.

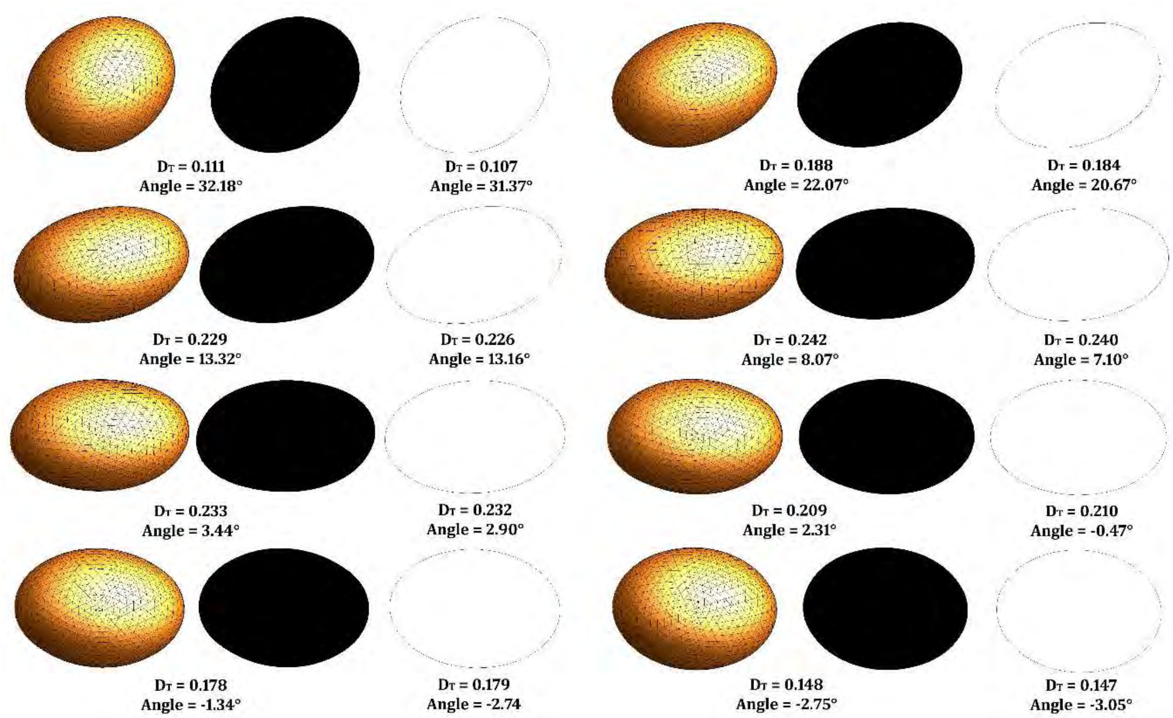

Figure 4.7 *Drop evolution in the time.* $Ca = 0.87$ *wich*$\lambda_{\mu} = 16$ *and* $\alpha = 0.03$*. The color images are the numerical shape obtained with BEM3D, the black pictures are the projection on the xy-plane; within these projections, ellipses were fitted.*

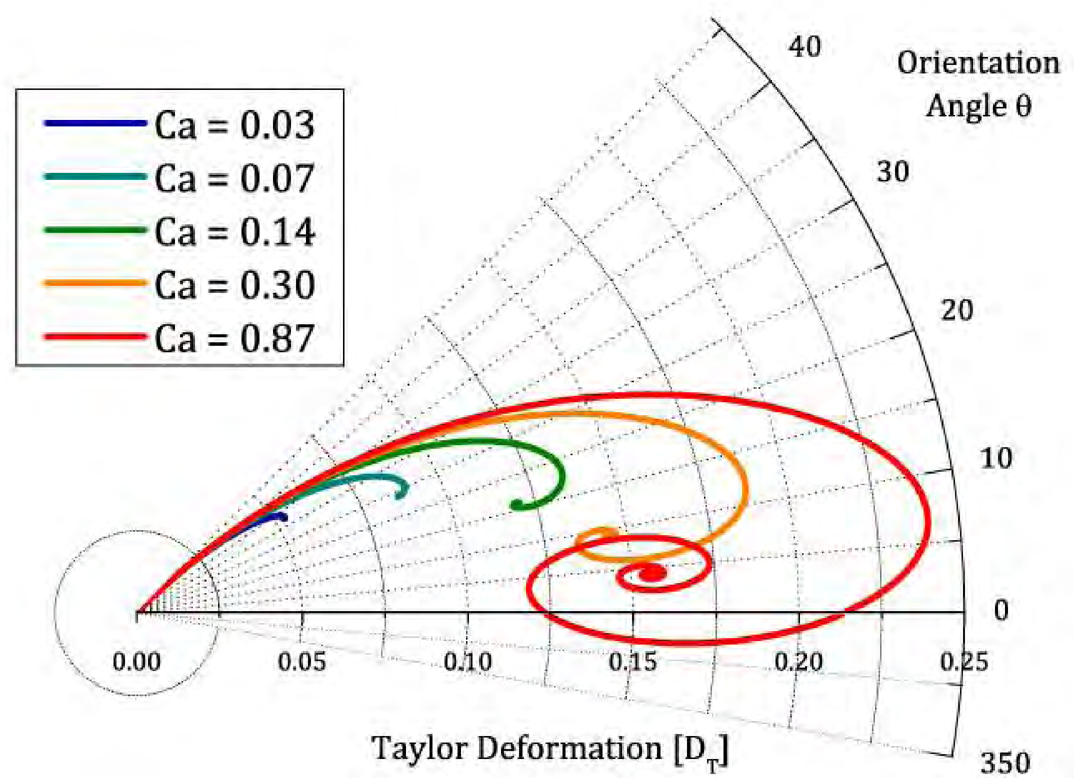

Figure 4.8 *Polar plot of Taylor deformation vs. orientation angle for different capillary numbers with* $\alpha = 0.03$, $\lambda_{\mu} = 16$.

CHAPTER 5.

Characteristic times of drop deformation under an elongational flow with vorticity

The drop deformation in strong flows shown in Chapter 3 and 4 indicate a not so obvious different behavior of the principal axes of the drop from the initial spherical shape up to the attained stationary values. Fig. 3.2 and Fig. 4.3 present *the time evolution of the principal axes* of the drop, showing that (a) the behavior of the principal axes parallel to the flow (*L-axis* and *B-axis*), reaches its steady state values with a similar characteristic time-scale, while (b) the *W-axis* has an appreciable delay to attain the steady state. This chapter focuses in the study of these characteristic times of the principal axes of the drop. The first part of Chapter 5 present the time-scales associated to the stationary shape of the drop. In this part, there is a similar analysis of the time evolution of the axes as Fig. 3.2 or Fig. 4.3, although with different the capillary numbers. However, in all cases the capillary number is lower than the critical capillary number Ca_{cr}.

For this reason, as a qualitatively comparison other major differences will be studied as well. The second significant effect corresponds to the observed oscillations (a consequence of the race to attain stationary shape). The time to attain stationary shape (in Fig. 4.3 stationary shape needs at least 40 [Gt] of time, in Fig. 3.2 only 15 [Gt]) occurs in different scales. Finally, differences of the characteristic times for

all axes are very similar in Figure 4.2. In Chapter 5 the analysis of the constants of time of the axes in the case of Chapter 3 will show the different scales of times in the axes of the plane of the flow and the cross section. In the regime of $\lambda_\mu = 16$, the scales are very similar. For this reason, only the analysis of constants of time in the impulsive flows were explore in the regime of $\lambda_\mu = 0.012$.

The second part of this Chapter analyze the characteristic times of retraction. If a drop is deformed without reaching its critical deformation, the drop will always attain a steady shape. If the flow is stopped subsequently, the drop returns to the equilibrium shape (spherical shape) due to the interfacial tension stresses. In this manner, the interfacial tension value could be measured, by simply analyzing the dynamics of retraction of the drop (Guido & Villone, 1999). This last part of this Chapter estimates the accuracy of the interfacial tension techniques base on the retraction analysis.

5.1 Characteristic times obtained in drop deformation in an impulsive flow applied with small viscosity ratio

Figures 5.1, 5.2 and 5.3 show the evolution of the principal axes of the drops with different capillary numbers for a flow with $\alpha = 0.13$. The evolution of the principal axes of the drop changes depending of the capillary number. The same behavior happens for the other cases $\alpha = 0.03$ and $\alpha = 0.05$.

The characteristic time-scale of evolution of the principal axes of the drop have a similar behavior at the onset of a weak steady flow. Then, as the capillary number increases, the axes attain the stationary states in different time-scales, *i.e.*, in Fig. 5.1 the *L-axis*, under a capillary number Ca = 0.05, attains the stationary states at 0.2 [Gt], while time goes to 1.5 [Gt] for capillary numbers about Ca = 0.40. The analysis on *B-axis* and *W-axis* show a similar behavior; as the capillary number increases, the time to attain the stationary state increases too; Figs. 5.2 and 5.3.

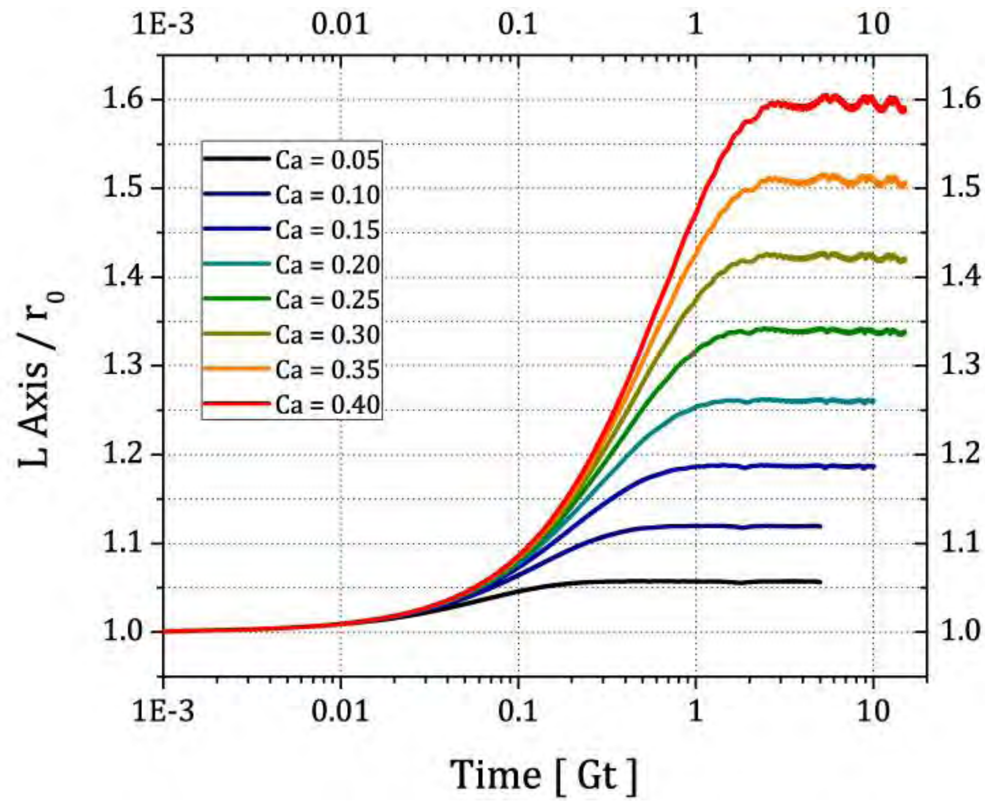

Figure 5.1 *Time evolution of principal L axis which* $\alpha = 0.13$ *for different values of Ca.*

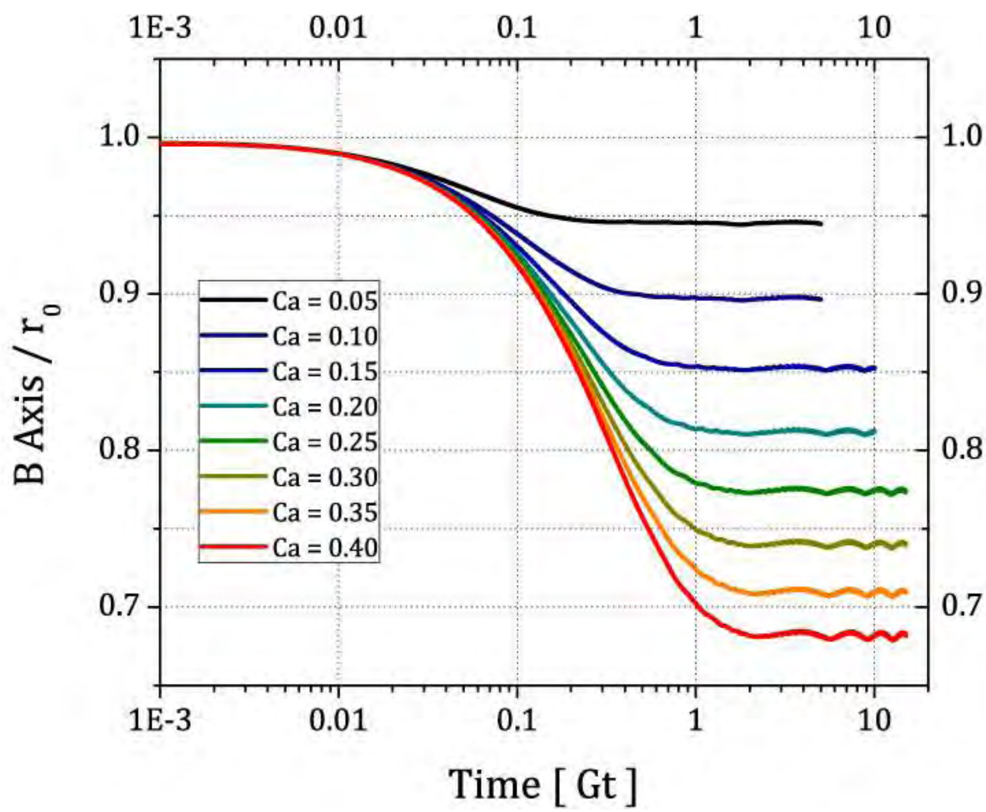

Figure 5.2 *Time evolution of principal B axis which* $\alpha = 0.13$ *for different values of Ca.*

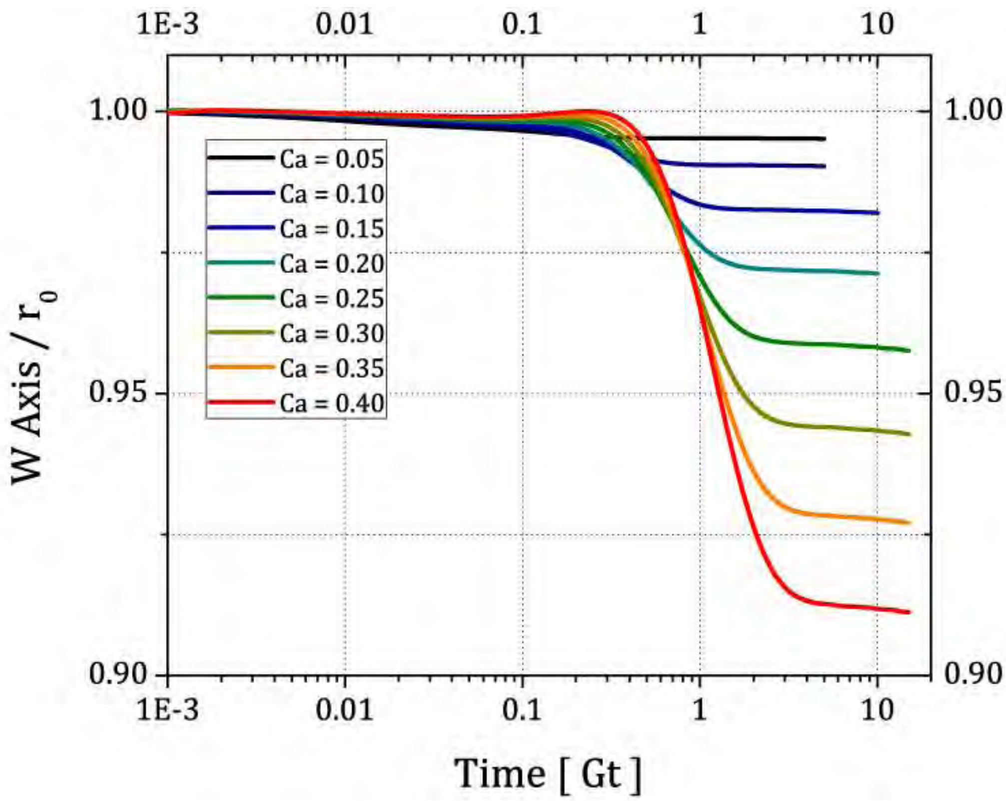

Figure 5.3 *Time evolution of principal W axis which* $\alpha = 0.13$ *for different values of Ca.*

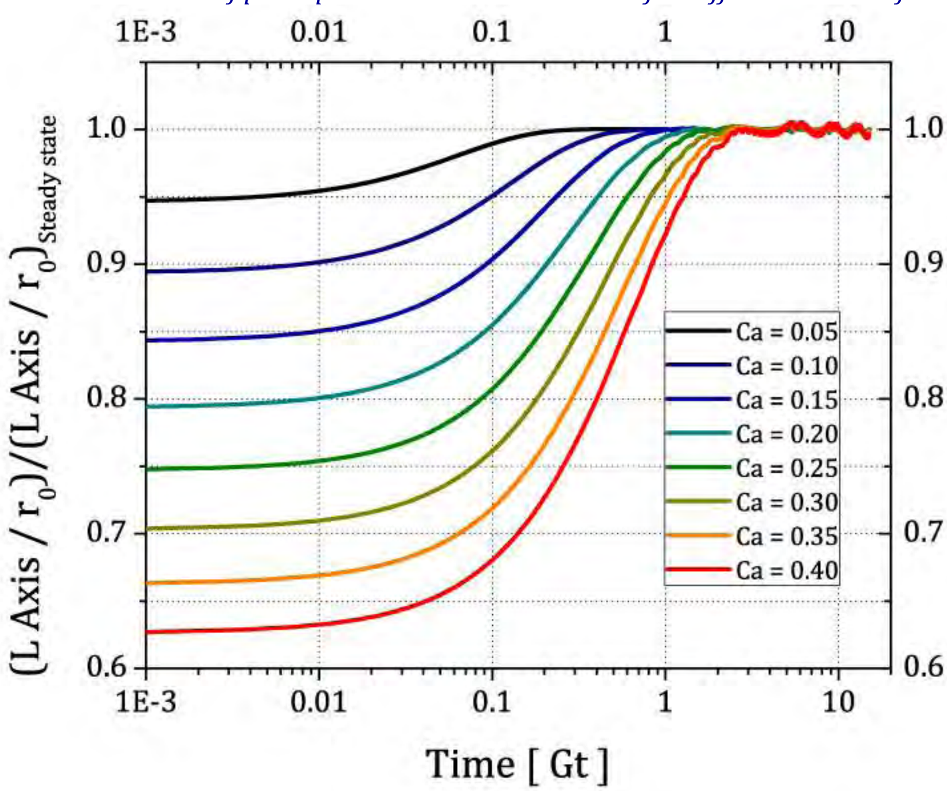

Figure 5.4 *Time evolution of principal L axis normalized wrt. stationary value, which* $\alpha = 0.13$ *for different values of Ca.*

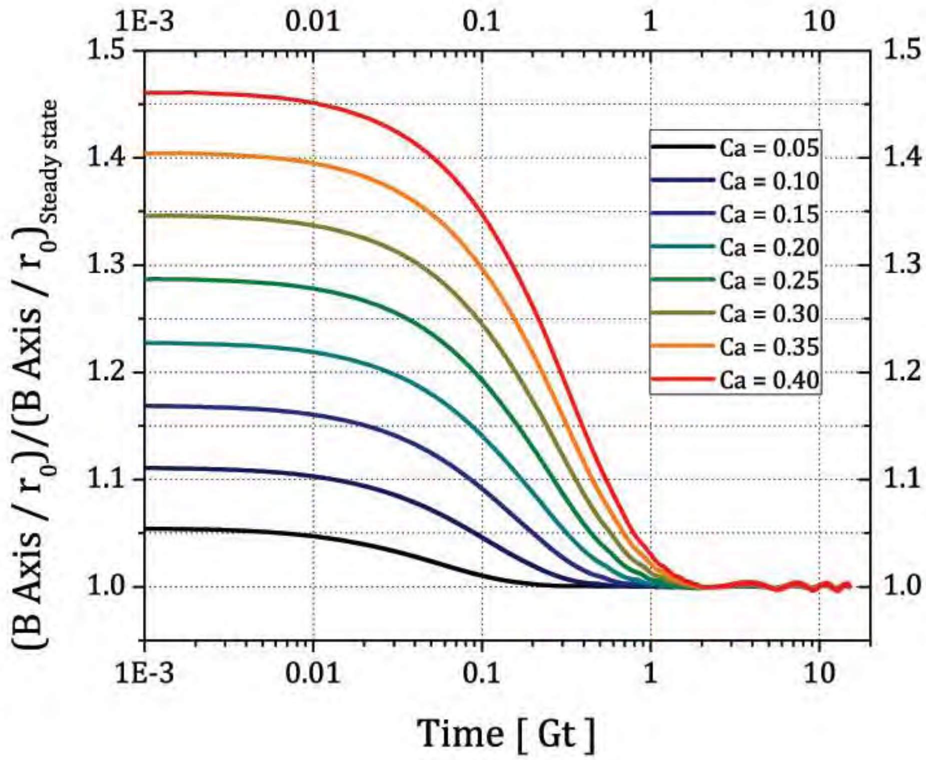

Figure 5.5 *Time evolution of principal B axis normalized wrt. stationary value, which* $\alpha = 0.13$ *for different values of Ca.*

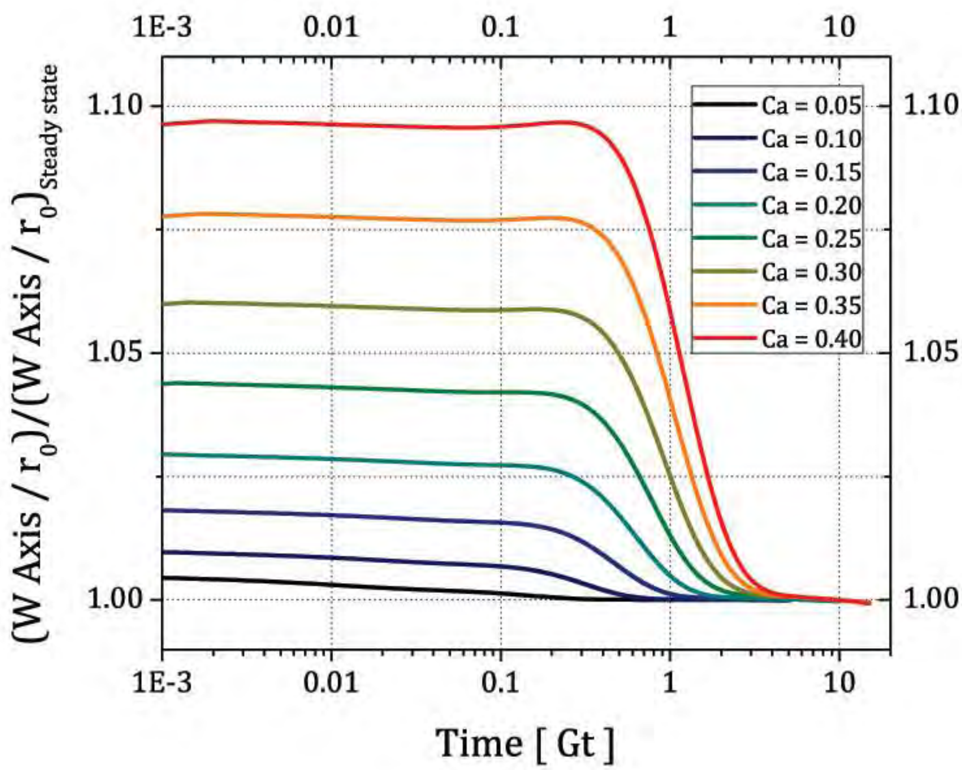

Figure 5.6 *Time evolution of principal W axis normalized wrt. stationary value, which* $\alpha = 0.13$ *for different values of Ca.*

In order to focus on the transient behavior and its time-scales, the first step was to normalize the change of the longitude of the principal axes with respect to the stationary state, as Figs. 5.4, 5.5 and 5.6 show. Figure 5.6 shows the evolution of the *W-axis* with a small change in the initial values of the evolution time due to the overshoots showed in Fig. 5.2. The normalized axis evolution were fitted with a family of exponential functions, attempting to establish the characteristic time to attain the final stationary state, in a similar way to methods for the charge and discharge of a capacitor (Greenberg, 1998) and (Resnick, 1980).

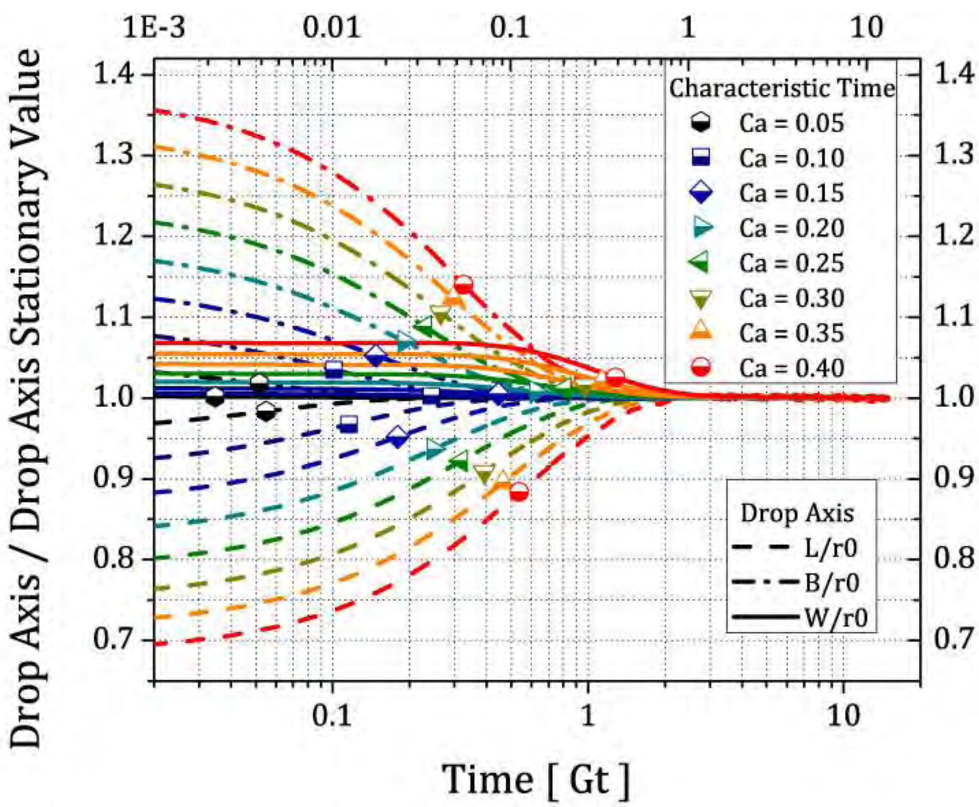

Figure 5.7 *Time evolution of principal axes normalized wrt. stationary value, which α=0.03 for different values of Ca. Characteristic times in symbols.*

The time of evolution of the principal axes were taken using the value of $\tau = 1; e^{-\tau} = e^{-1} = 36.79\%$. The dynamics of drop deformation changes as the capillary number increases in the same type of flow. The analysis of the characteristic times exposed the different axes dynamics under the same flow, *i.e.* for the same drop experiment, the steady deformation is attained at different times for each principal axes. This situation appears to be a consequence of the planar flow effects: the drop (*3D-surface*) rotates faster in the *xy-plane* (Fig 1.1 b), compared to the other direction.

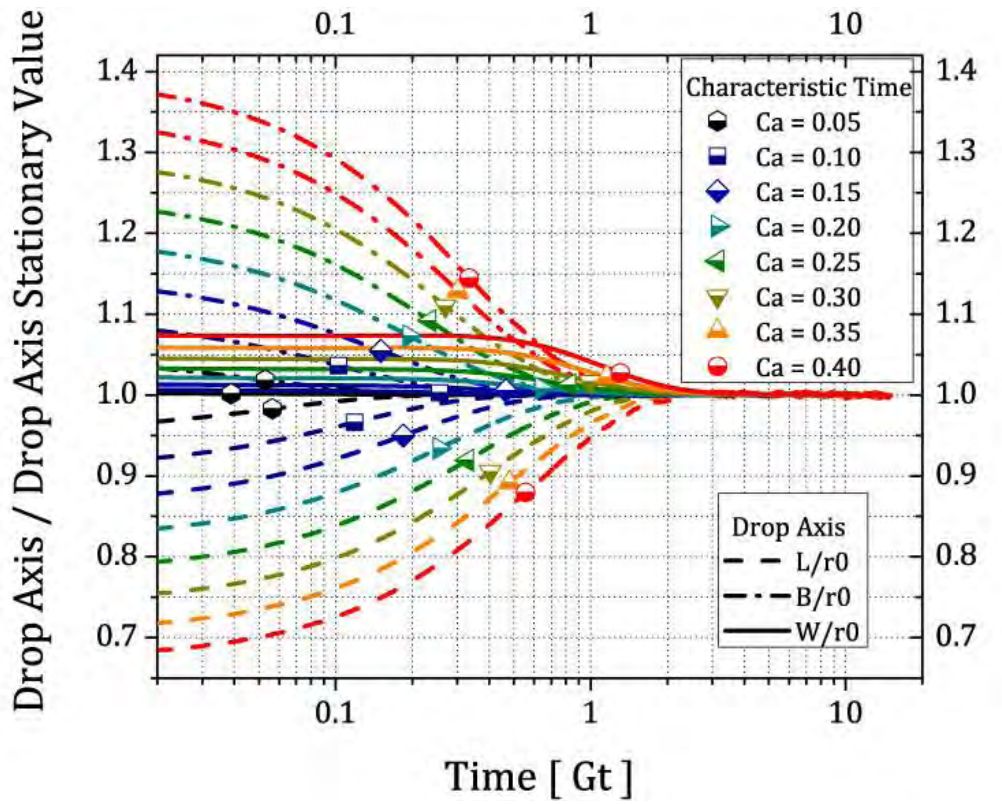

Figure 5.8 *Time evolution of principal axes normalized wrt. stationary value, which α=0.05 for different values of Ca. Characteristic times in symbols.*

Figures 5.7, 5.8 and 5.9 show, for different types of flow, the evolution of the principal axes of the drop, and the value of their characteristic time. As the capillary number increases in all numerical experiments, the time-scale of the axes becomes quite different. The *B-axis* appears to be the fastest. In contrast, the *W-axis* present the longest lag. In Figure 5.9, for *Ca* = 0.40, the *W-axis* evolution starts to change when the evolution of *B-axis* is near its stationary value. Add to this, the characteristic time of *W-axis* is 1.54 [*G*t] when the *B-axis* attains its stationary value and the *L-axis* is near the 80% of its steady value.

The characteristic times, evaluated with the numerical experiments, are plotted in Fig. 5.10. Here there appear to exist a similar behavior for the growth rate of the characteristic time values. All time-scales have equal values when $Ca \lesssim 0.05$. When $Ca = 0.40$, the *B-axis* time is one third of the value of the *L-axis* time value, and one fifth of *W-axis* time value. Figure 5.10 shows the delay of the dynamics of drop deformation in the normal plane of the applied flow.

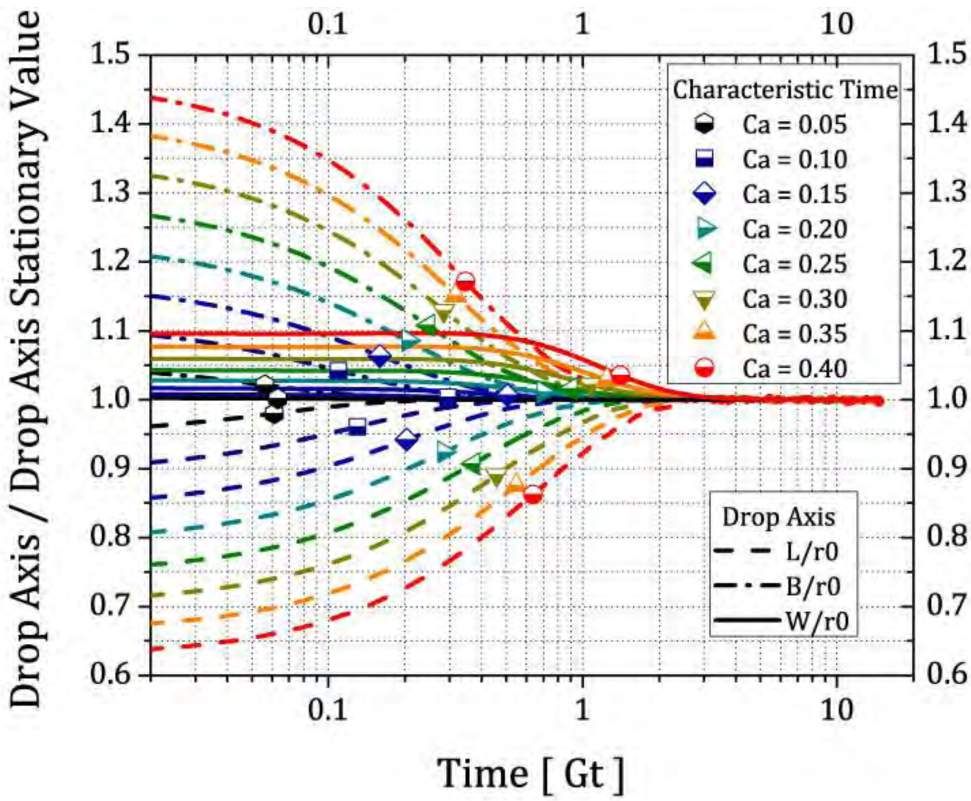

Figure 5.9 *Time evolution of principal axes normalized wrt. stationary value, which α=0.13 for different values of Ca. Characteristic times in symbols.*

The *W-axis* presents the smaller changes of deformation —less than 10% of the initial value and there is a delay to start to change. Please note that the time used for the complete numerical simulations is enough to attain the steady state in the *xy-plane*. This time was calibrated *wrt* the experiments of Rosas (Rosas, Reyes, Minzoni, & Geffroy, 2014). However, Figure 5.3 shows that the *W-axis* is still evolving toward the steady state; *i.e.* to evaluate the time lapse to attain the stationary shape in a drop —for a capillary number less than the critical capillary number $Ca < Ca_{Cr}$— the *xy-plane* data appear not to be enough. Figure 5.9 and Fig. 5.10 show clearly delays in the *W-axis*; in other words, the *W-axis* is still changing when the *xy-plane* form looks to have attained a stationary state.

Figure 5.10 reveals the fact that drops deformed by a *2D-flow* is still a *3D-object*, because the dynamic of its three-principal directions evolve in different time-scales. The dynamics imposed by the flow starts to deform the drop on the plane parallel to the flow. However, the delay on the third axis, *W-axis*, seems to be decouple and is a consequence of a *2D-flow*. Based on this information, the study *of steady states* should consider as well as the characteristic time of the *W-axis*, because the steady state ought to be determined by the slowest axis. It is only then that we can be sure that the stationary state was obtained.

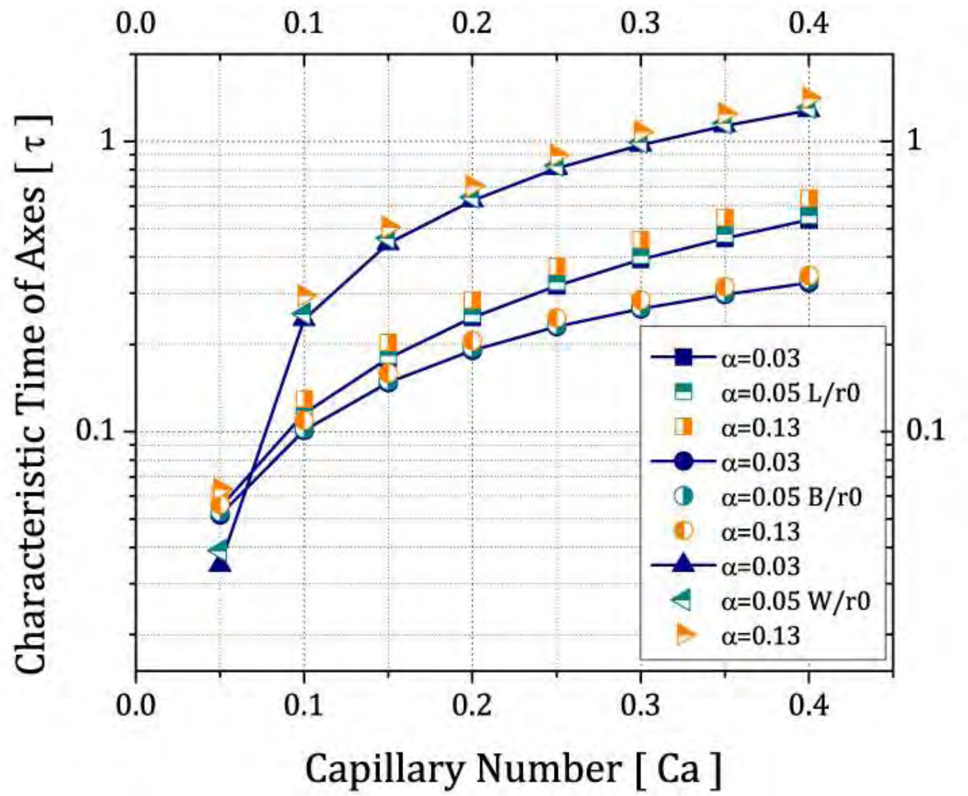

Figure 5.10 *Characteristic Times vs Capillary Number.*

Finally, Figure 5.11 shows the evolution of the principal axes of drops to reach the stationary state of deformation with a flow $\alpha = 0.13$. In this plot, when the drop is at rest, the values of the principal axes are zero; when the drops go to the stationary shape, the values of the axes goes to one. Here, all traces of the time of evolution shown in Fig. 5.10 are normalized by the value of its capillary number. In this Figure 5.10 a range of characteristic times of the principal axes due to the kind of flow imposed are also shown as a single dot. The time-scales of the *L-* and *B-axis* are similar, while the *W-axis* systematically lags the others, except for $Ca = 0.05$.

With this information, it is possible to define a characteristic time associated to the flow imposed to the drop. For the *2D-flow* with $\alpha = 0.13$, and using *W-axis* values of time observed in Fig. 5.11, an estimate of the minimum time to attain the stationary deformation of a drop is $Gt/Ca = 10$ which is the time obtained by the *W-axis*.

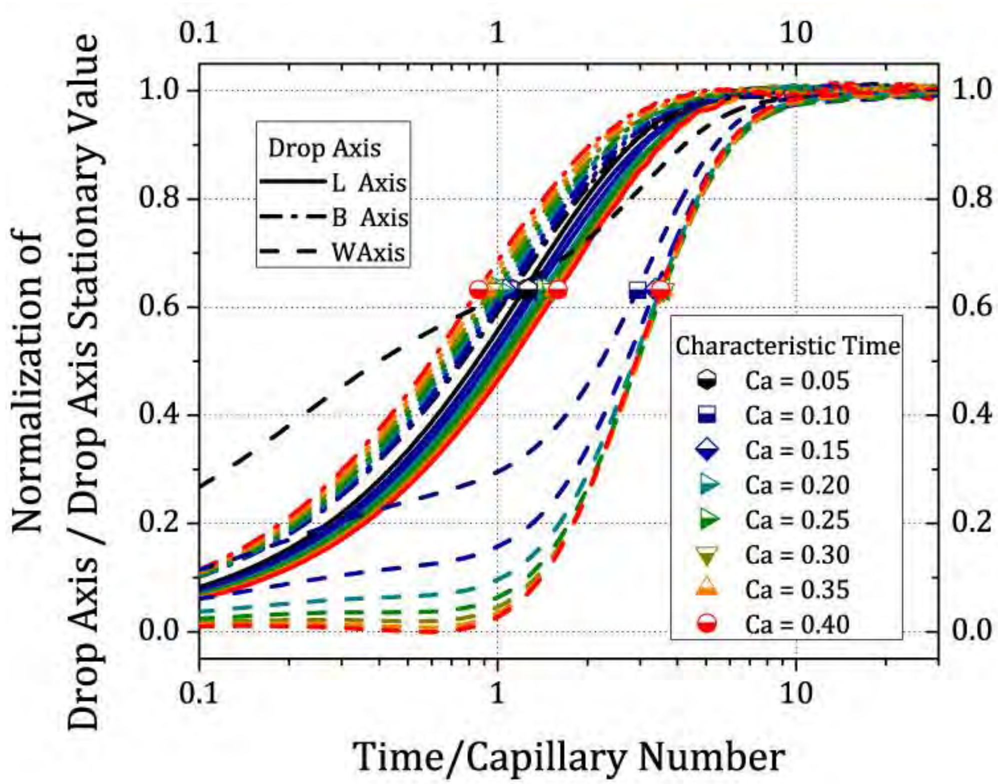

Figure 5.11 *Analysis of the Principal Axes of the drop wrt. the characteristic time due capillary value, with* $\alpha = 0.13$ *and* $\lambda_\mu = 0.012$*. Symbols are the characteristic times of the Principal Axes of the drop.*

For cases when the ratio of viscosity is large, $\lambda_\mu = 16$, characteristic times are not easily observed, because the drop evolution on the *xy-plane* presents multiple rotations, and each axis evolution is not like an simple exponential. As annotation, the evolution of *W-axis* is neither a true exponential decay. Because it clearly presents an overshoot, observed in Fig. 5.3; this spurious effect is shown in Fig. 5.11, and the dash curves associated to the *W-axis* do not have the same behavior as the other principal axes. For its time scale analysis, is not severe enough as to modify the exponential behavior as Fig. 5.6 shows. In Chapter 6, I present an analysis of this overshoot when the capillary number increases; Fig. 5.3 already shows that the overshoot increases as the capillary number rises.

5.2 Characteristic time in the retraction of a drop

The experimental and theoretical studies of dynamics of drops show that a deformed drop always comes back to its spherical form once the external flow stresses cease to act. If the drop deformation is lower than a critical elongation in steady flow, the retraction only generates a single drop, and may produce multiple (albeit smaller) drops for elongation greater that this critical value. The next analysis shows the retraction of the drop to a single equilibrium shape (near spherical). The data obtained was used to validate the *Deformed Drop Retraction (DDR)* method, used for obtaining experimental values of the interfacial tension, γ. This method has the advantage of measuring γ even when the buoyancy forces are very weak —a condition required for most other techniques— and has been used extensively with shear-flow-devices deformation. However, in strong flows the method was not been used a lot.

The *DDR* model assumes a simple exponential decay behavior dependent on the interfacial tension. The interfacial tension value for a drop interface, determines the rate of evolution of the deformation of the drop from the initial process of retraction (when the flow in the continuum phase was turned off and the drop reached the steady shape for a constant flow) D_0 to the final equilibrium shape (spherical drop), see Fig 5.12 and Fig 5.13. For this analysis, the values of deformation are normalized *wrt* the initial value of deformation: $D(t)/D_0$. The logarithm of the ratios of deformation are plotted versus time of retraction: Fig. 5.13. Finally, the interfacial tension is obtained assuming a linear dependency of the slope by the expression:

$$\gamma = -\mu_1 r_0 \left[\frac{(2\lambda_\mu + 3)(19\lambda_\mu + 16)}{40(\lambda_\mu + 1)}\right] K_{Slope}. \tag{5.1}$$

The constants μ_1 and r_0 are the viscosity of the continuum phase and the radius of the spherical shape drop. The expression inside the bracket corresponds to Hadamard and Rybszynski resistance due to a drop with different viscosity μ_2, but here due to the analytical (asymptotic expansion) correction between the capillary number and the Taylor Deformation of the drop using an ellipsoidal model -near spherical shape- of the drop (Rallison, 1984), (Taylor, 1932),

$$(\boldsymbol{r}\cdot\boldsymbol{r})^{1/2} = 1 + \varepsilon\, \boldsymbol{r}\cdot\boldsymbol{A}(t)\cdot\boldsymbol{r} + \mathcal{O}(\varepsilon^2) \text{ with } \varepsilon \ll 1,$$

and ε being the small parameter (quasi-spherical drop), and $\boldsymbol{A}(t)$ measures the drop deformation. The time evolution of its distortion is calculated by

$$\varepsilon \frac{D\boldsymbol{A}(t)}{Dt} \equiv \varepsilon \frac{\partial \boldsymbol{A}(t)}{\partial t} + \boldsymbol{u} \cdot \nabla \boldsymbol{A}(t) = \varepsilon \frac{\partial \boldsymbol{A}(t)}{\partial t} - \varepsilon\, Ca\, \boldsymbol{\omega} \cdot \boldsymbol{A}(t) + \varepsilon\, Ca\, \boldsymbol{A}(t) \cdot \boldsymbol{\omega}$$

$$= \frac{5Ca\, \boldsymbol{e}}{2\lambda_\mu + 3} - \frac{40\left(\lambda_\mu + 1\right) \varepsilon\, \boldsymbol{A}(t)}{\left(2\lambda_\mu + 3\right)\left(19\lambda_\mu + 16\right)} + \mathcal{O}(\varepsilon\, Ca, \varepsilon^2). \tag{5.2}$$

Though complicated in appearance, the simple physical interpretation of Equation (5.2) is that the rate of change of the distortion $\boldsymbol{A}(t)$ rotates with the local fluid angular velocity —measured on a rotating frame of reference for the drop, and couple-free—, is due mainly to the effect of the ambient strain-rate field $\boldsymbol{e}$, and secondly to the retraction effect of the surface tension (proportional to $\boldsymbol{A}(t)$). Neglected terms are, of order $\varepsilon\, Ca$, arising from the straining flow acting on the perturbed shape, and $\mathcal{O}(\varepsilon^2)$ terms from harmonics higher than the second.

If $Ca \ll 1$, Eq. (5.2) can be solved analytically for $\boldsymbol{A}(t)$ at steady state, obtaining

$$\varepsilon\, \boldsymbol{A}(t_{ss}) = \frac{19\lambda_\mu + 16}{8\lambda_\mu + 1} Ca\, \boldsymbol{e}\,.$$

The well-known Taylor equation (From Equation (10) to Equation (13), (Taylor, 1934)) follows:

$$D_T = \frac{19\lambda_\mu + 16}{16\lambda_\mu + 16} Ca, \tag{5.3}$$

for the deformation parameter in *simple shear flow*.

Since during retraction there is no applied flow field, the evolution of $\boldsymbol{A}(t)$ —in Eq. (5.2)— is defined only for the *rhs* expression of Eq. (5.2). So, the rate of change of the distortion $\boldsymbol{A}$ could be model by a single exponential decay, Fig. 5.11, from which the following equation for $D(t)$ is obtained

$$D_T = D_{T-SS}\ \exp\left(-\frac{40\left(\lambda_\mu + 1\right)\gamma}{\left(2\lambda_\mu + 3\right)\left(19\lambda_\mu + 16\right)\mu_1 r_0} t\right). \tag{5.4}$$

Finally, using Eq. (5.4), Eq. (5.1) is derived.

The *DDR method* employs the Taylor deformation Eq. (1.11). However, there are other parameters to define the deformation of the drop. In this section, another parameter for deformation is used, the *Mo shape parameter*. The model of *Mo*, (Mo, 2000), assumes that the drop deformed is a regular ellipsoid. Then, the *eigen*-values of the matrix that represents the shape of the drop correspond to the principal axes of the ellipsoid. *Mo* shape parameter is represented by the following equation,

$$D_{Mo} = L^2 - B^2 \tag{5.5}$$

Taylor deformation and the Mo shape parameter assume a drop shape nearly spherical and a regular ellipsoid, respectively. However, as we saw in Chapter 2, Chapter 3 and Chapter 4, deformed drops by an *extensional flow* with vorticity have not a circular cross-section, and the relationship between capillary number and Taylor Deformation is not linear. Also, the *3D-effects* of a drop due to a *2D-flow* were shown, in the previous section, to have different characteristic times for principal axes. It is then safe to assume that for the phenomena of retraction, the evolution of the drop principal axes is not guaranteed to have an symmetric cross-section.

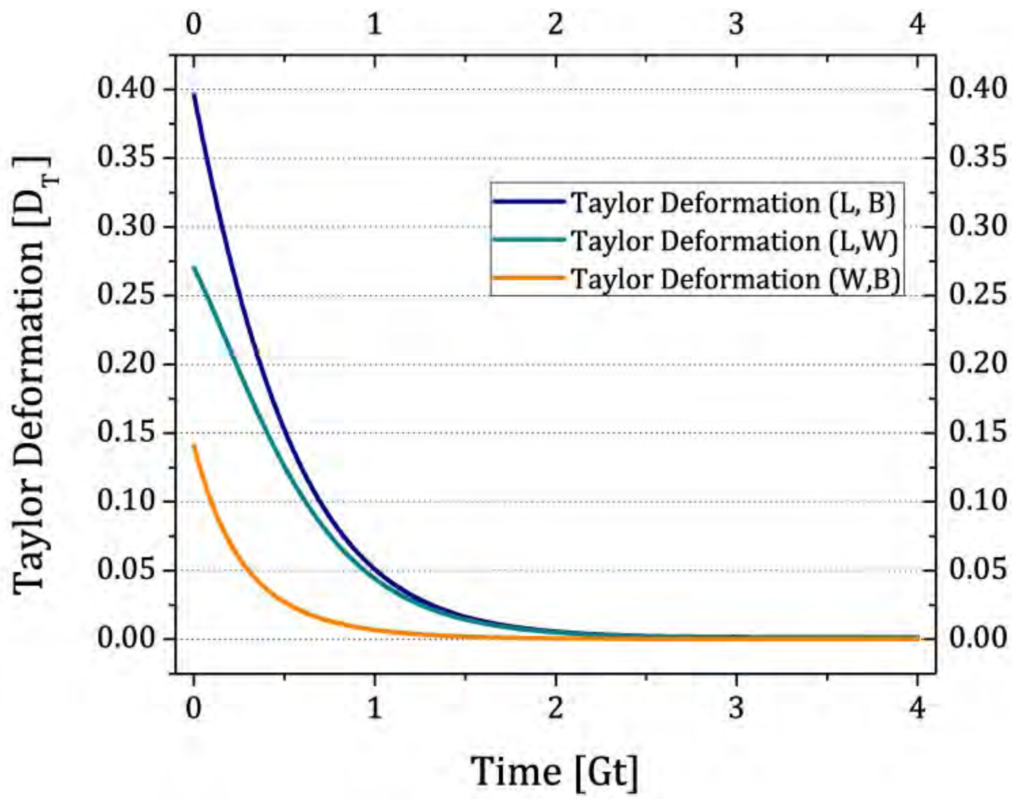

Figure 5.12 *Taylor Deformation of the drop vs time.* $\alpha = 0.13$, $Ca = 0.40$.

This observation motivates this Section about the DDR model to evaluate retraction, because it is essential to observed if the *xy-plane* is enough to obtain the value of interfacial tension with the precision required for strong flows. Equation (5.4) involves the Taylor deformation evaluated in the plane of flow. However, the retraction of the drop occurs under flow conditions similar to a *uniaxial extensional flow* so, the analysis of retraction could be observed from different planes of the drop. In this study, the plane generated by the principal axes were taken to observed how Eq. (5.1) estimates the value of interfacial tension with respect to the nominal value employed in the numerical code.

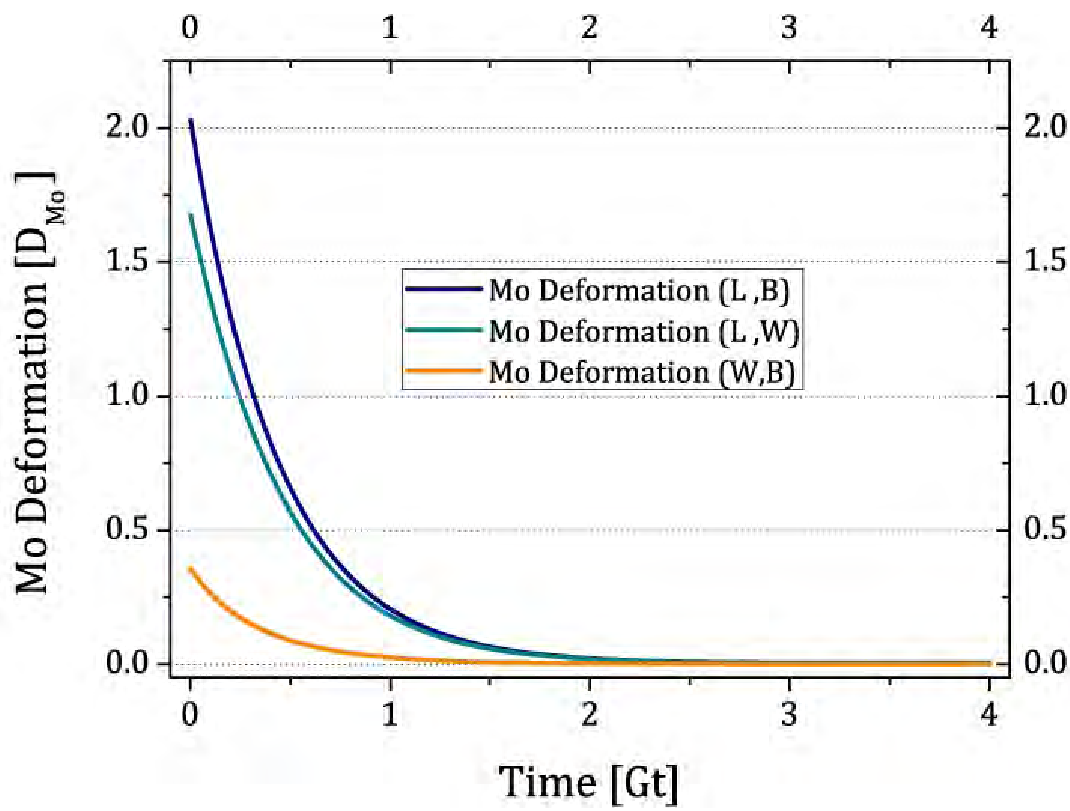

Figure 5.13 *Mo Shape Parameter vs Time.* $\alpha = 0.13$, $Ca = 0.40$.

Figure 5.12 shows the Taylor deformation of the drop using the principal axes of the drop with $Ca = 0.40$ and $\alpha = 0.13$. The decay of the Taylor deformation in all the cases could be modelled as an exponential decay. As well, a similar behavior is shown in Fig. 5.13 for the analysis of the drop retraction using the Mo shape parameter. As Figure 5.12, Fig. 5.13 show, the drop goes to a spherical shape in less than the middle of the time observed.

Figure 5.14 and 5.15 shows the comparison of the curves of decay of deformation with respect to an exponential decay, adjusted with the numerical data.

The exponential decay uses the characteristic time τ in a model of Taylor Deformation and of the *Mo* shape parameter. Fig. 5.14 shows that the retraction evolution model as an exponential decay, is a good approximation for cases when the Taylor deformation involves the *L*- and *B-axes* (*xy-plane*) or the deformation in the cross-section *W*- and *B-axis*. A measure of Taylor deformation between *L-axis* and *W-axis* shows a behavior a little bit different. Another observation is the difference of the characteristic time τ in the different planes of Taylor Deformation parameter. As the former Section shows, the characteristic times during retraction should be similar to the time required for reaching the stationary state (although different for each axis). The time τ for the deformation between the *L*- and *B*-axes is comparable to the time between *L*- and *W*-axes mainly because the *L-axis* deformation is larger than the *B-axis* or *W-axis* values. The characteristic time for the cross-section is different, because of the enormous difference in the values of *B- wrt W-axis*. Also, the change of cross-section manifest another dynamical behavior that the model of retraction cannot estimate in the early stages. So, obtaining the interfacial tension with the cross-section parameters may not be guaranteed.

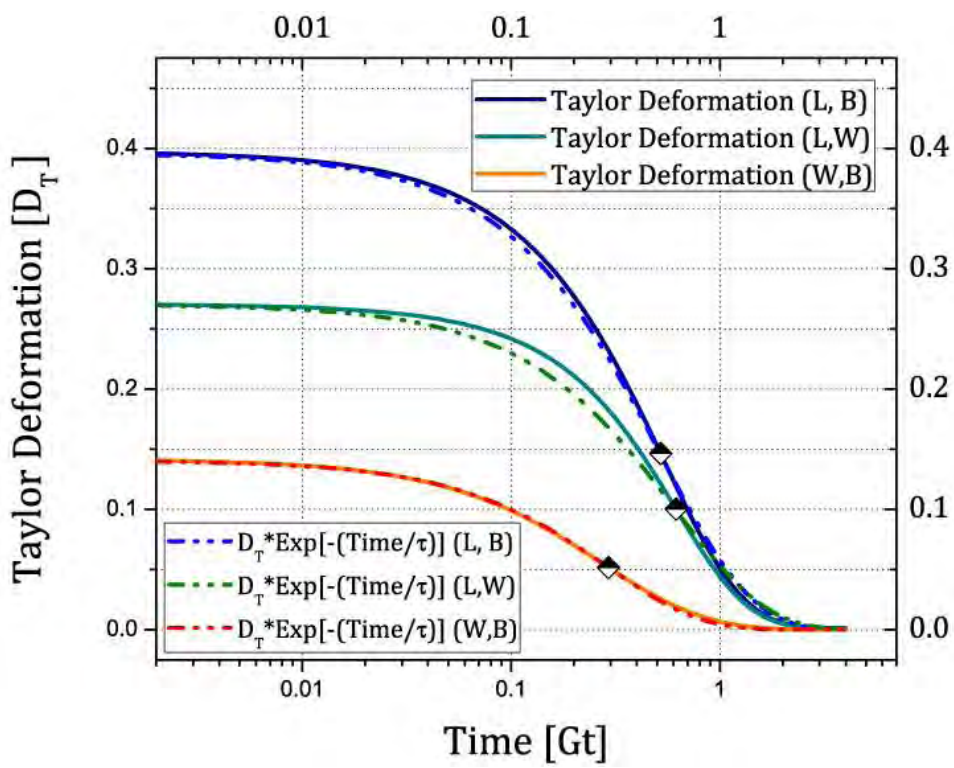

Figure 5.14 *Taylor Deformation vs Time (lines). Exponential Decay vs Time (dash-dot-dot). Characteristic time τ (diamonds).* $\alpha = 0.13$, $Ca = 0.40$.

The analysis for the *Mo* shape parameter was made as the case before, using the principal axes of the drop to estimate the deformations. The result of the characteristic time for a drop with $Ca = 0.40$ and a parameter of flow $\alpha = 0.13$, is shown in Fig 5.15. The match of the evolution of the retraction and the model of exponential decay is better than the analysis using the Taylor Deformation. Again, the characteristic times of cross section is evidently different than the other planes of observation.

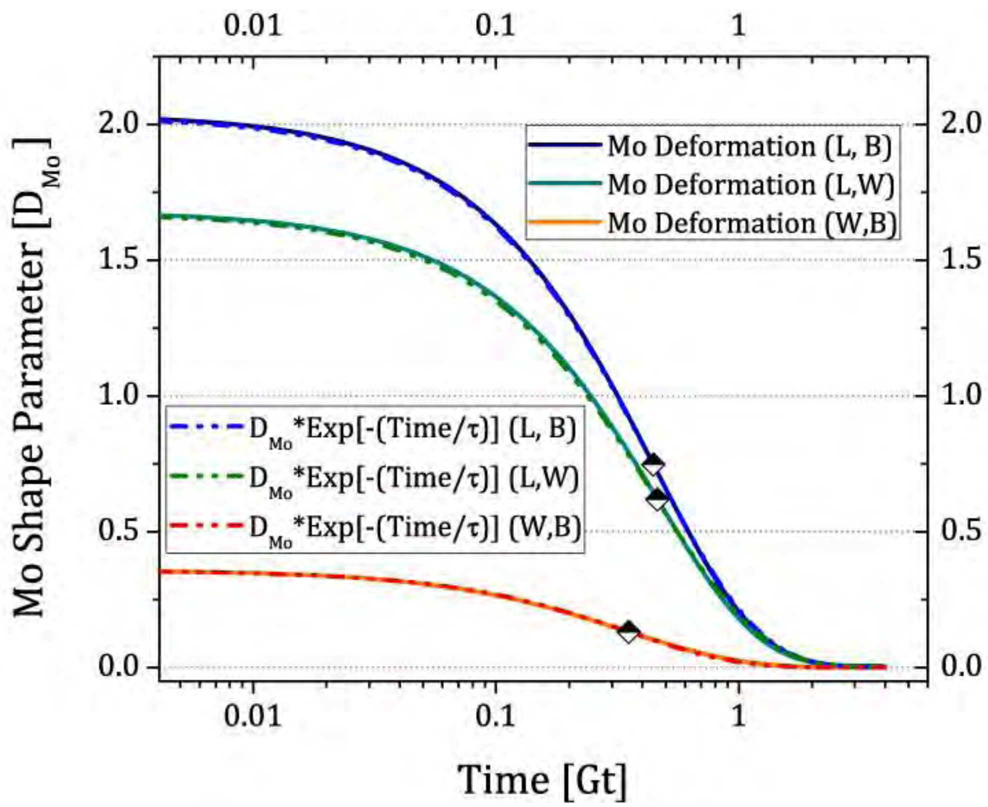

Figure 5.15 *Taylor Deformation vs Time (lines). Exponential Decay vs Time (dash-dot-dot). Characteristic time τ (diamonds). α=0.13, Ca=0.40.*

The characteristic time of deformation on the principal plane is plotted in Fig. 5.16. All values of the characteristic time using the *Mo*-shape parameter and Taylor Deformation appear to be different in each plane of observation. The characteristic time increases too when the capillary number grows. However, comparing cases of planes *L*- and *B*-axes versus *L*- and *W*-axes all appear to have similar values. Time-scales for the cross-section are shorter than for the other planes of the drop because the *W-axis* deformation is the smallest and the time to return to spherical shape, initial radius, is the shortest time.

These results are similar, as Fig. 5.10 shows, despite the former using the *information of the principal axes of the drop,* while Fig. 5.16 employing the *information of the deformation on the plane of observation*. In both figures, the *3D-effects* of the drop in a *2D-flow* are evident in Fig. 5.10, because the principal axes have different characteristic times and the perpendicular axis to the flow, the *W-axis*, shows a different time-scale to attain the steady shape. In the other case, Fig. 5.16 shows the behavior on different planes defined by the principal axes, so, time-scales associated to the deformation of the drop represent *competition of times-scales on the principal axes* of the drop. Figure 5.16, implies that the characteristic time of retraction of the parallel plane of the flow retracts faster than the normal planes. In Fig. 5.15 the Taylor deformation or the *Mo*-shape parameter of the *WB*−plane have different scales of time with respect the other two planes. The characteristic times of the *Mo*-shape parameter have values among themselves, closer than when using Taylor deformation. At this point, the *Mo*-shape parameter appears to be the best (tighter) simple prediction of the retraction of the drop.

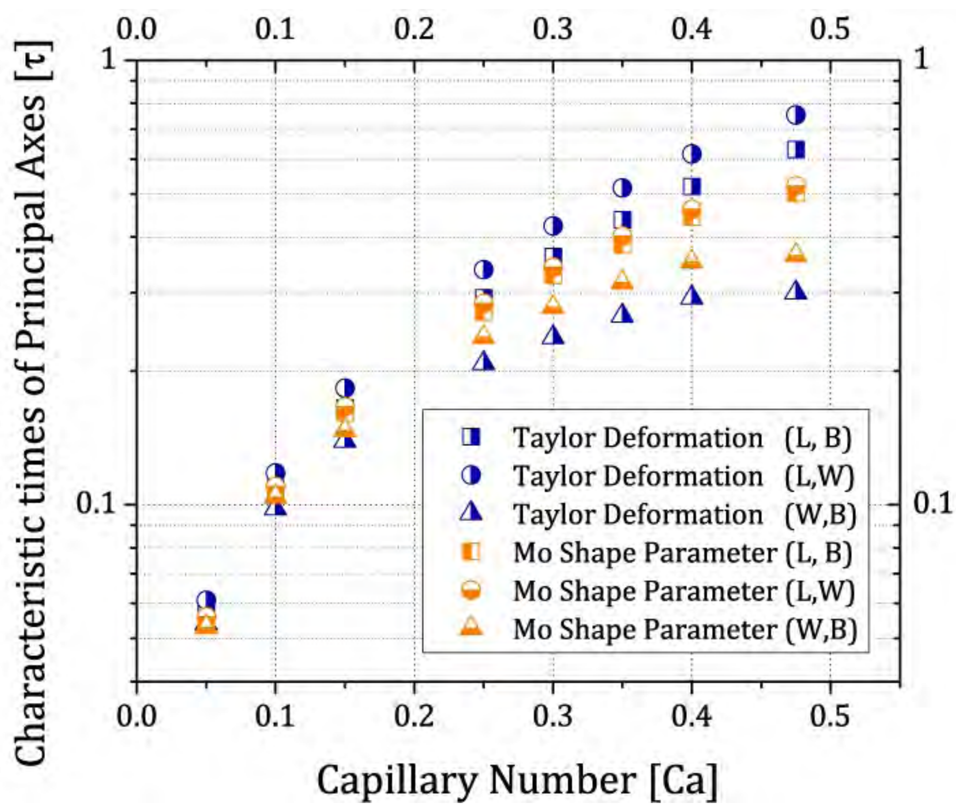

Figure 5.16 *Characteristic times t vs Capillary Number which parameter* $\alpha = 0.13$.

The next analysis addresses data differences between the conventional methods and information on the other planes, which are not observed by *2D*-numerical methods or experimental devices as Two Roll Mill, (Reyes, 2005), (Rosas I. Y., 2013), and (Rojas, 2016). The information on the different planes will be discussed after the explanation of the technique for obtaining the interfacial tension.

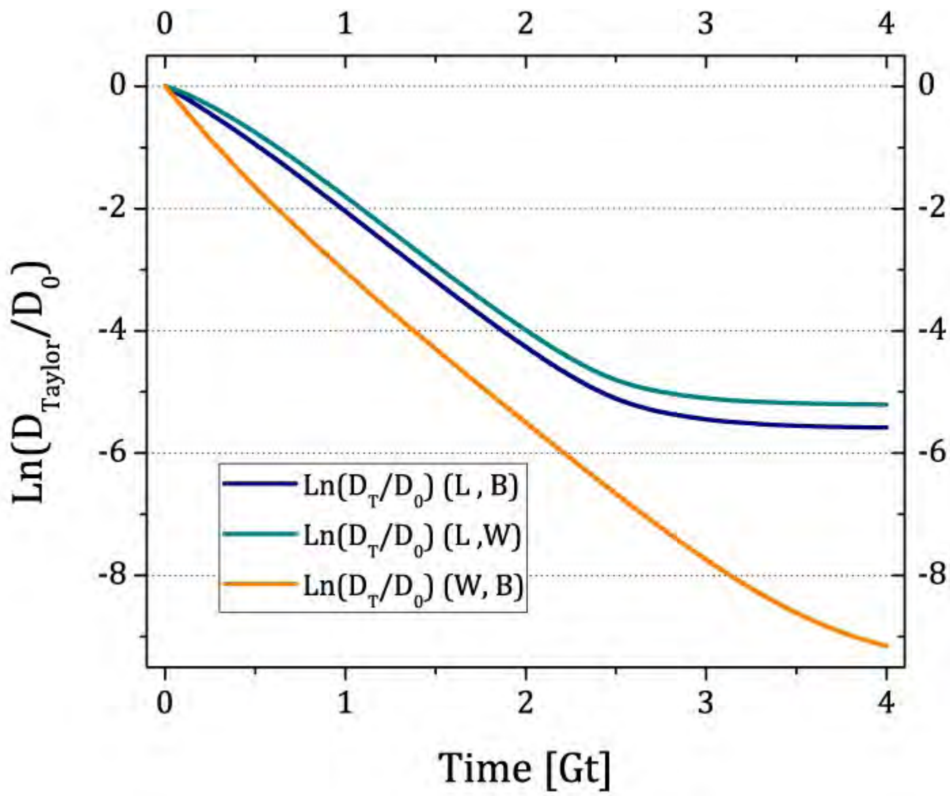

Figure 5.17 *$Ln(D_T/D_0)$ vs. time. Drop retraction was analyzed with Taylor deformation which $Ca = 40$ and $\alpha = 0.13$.*

At the end of drop retraction, there is a residual deformation with values of $0.001 < D_T < 0.005$ for the capillary numbers used. This residual deformation is clearly observable in the $ln(D_T/D_0)$ *vs. time plots*, as shown Fig. 5.17 for a drop subjected to $Ca = 0.40$ and $\alpha = 0.13$. Residual deformations are essentially a measure of the uncertainties of the numerical code. The values of Taylor deformation mentioned before imply a difference of about 0.2% to 0.3% of the length of the principal axes. In this work, the evaluation the length of the axes always take the largest distance from the drop surface to the center of the drop as the *L-axis*, afterwards the *B- and W-axis* are estimated. When the resolution is less than 0.3%, obtaining the Taylor deformation close to zero becomes very difficult. A similar case of residual deformations is obtained in the *Mo*-shape parameter. The values are $0.0015 < D_{Mo} < 0.006$, and the behavior is similar as the Fig. 5.17.

Therefore, the DDR model must assume a clean linear slope due to the logarithmic dependency, with no consideration to residual deformations (either experimental or theoretical). With this fact, the optimum evaluation of the interfacial tension should be based on deformations before the drop attains its smallest residual values.

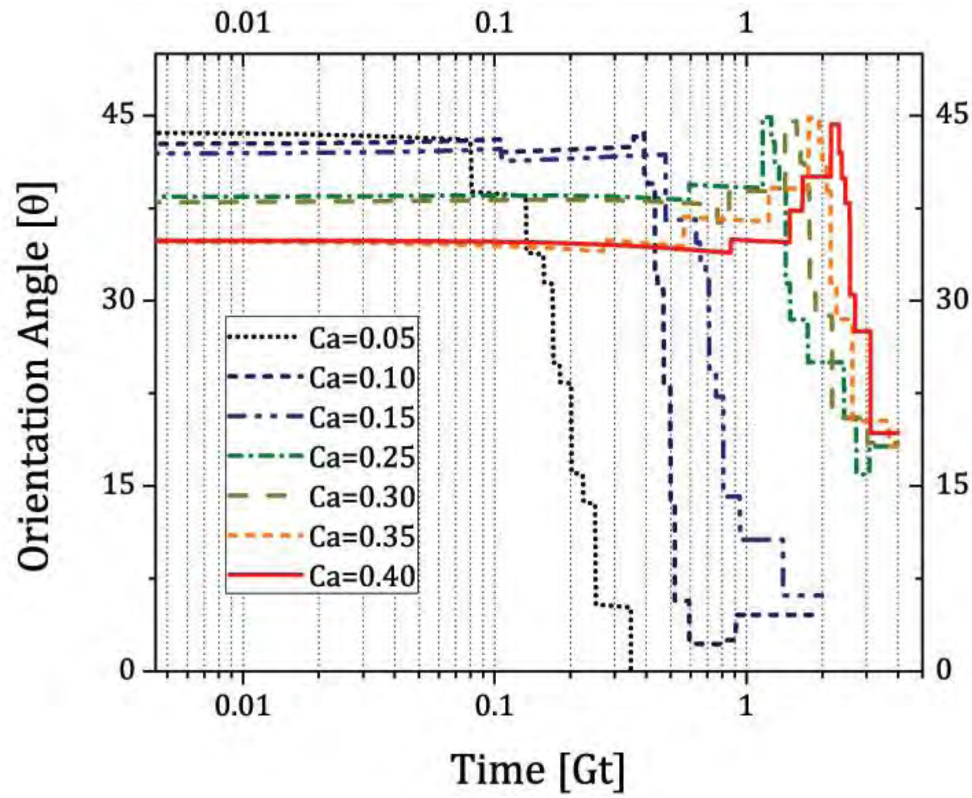

Figure 5.18 *Angle of Orientation vs. time in drop retraction which parameter* $\alpha = 0.13$.

Experiments of retraction of a drop, deformed by a *2D-flow*, must maintain the angle of orientation of the drop fixed, for the flow is due only to the elasticity of the interface, which is based on a symmetric shape for the drop. Figure 5.18 shows the orientation of the drops during the retraction process, which changes when the drop is nearly spherical. In other words, the angle of orientation can be used to define the longest useful time of retraction, just before observing the residual deformations.

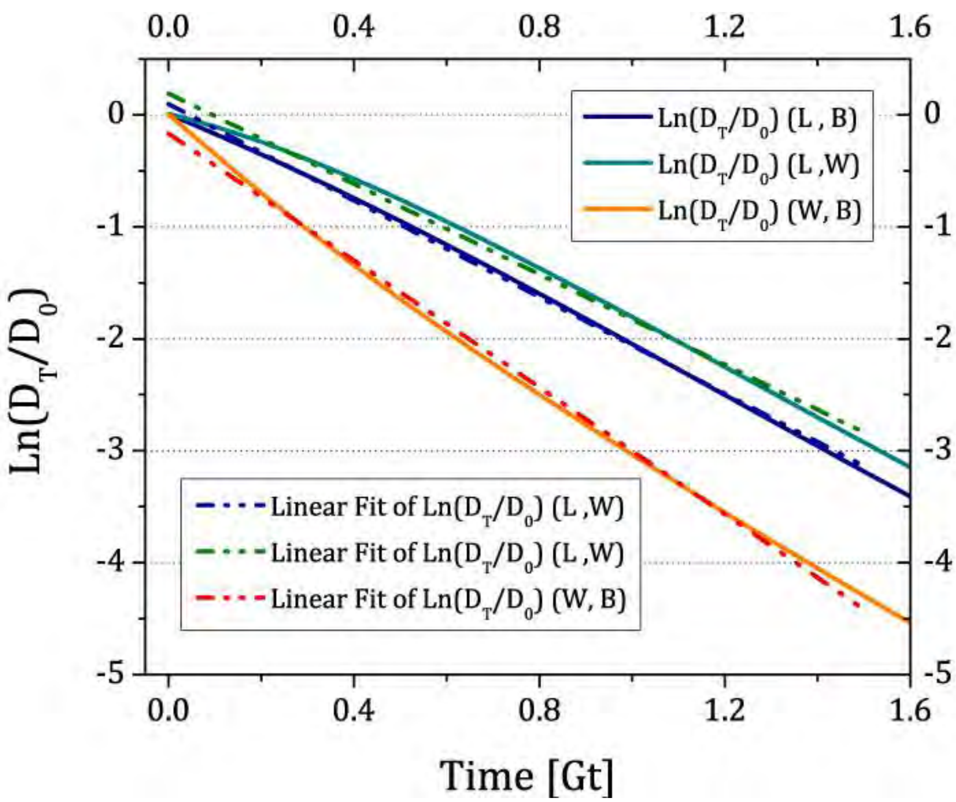

Figure 5.19 *Linear fit of* $-Ln(D_T/D_0)$ *for* 5τ *of time. Drop retraction was analyzed with Taylor deformation which* $Ca = 40$ *and* $\alpha = 0.13$.

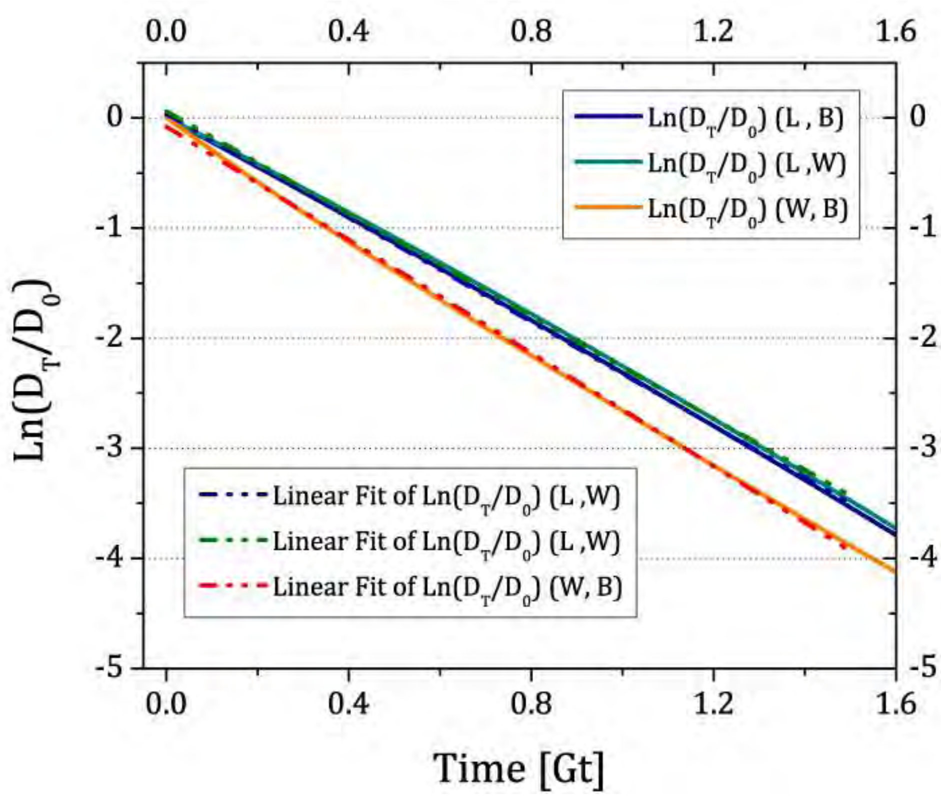

Figure 5.20 *Linear fit of* $-Ln(D_{Mo}/D_0)$ *for* 5τ *of time. Drop retraction was analyzed with Mo shape parameter which* $Ca = 40$ *and* $\alpha = 0.13$.

In order to simulate numerically a wide range of interfacial tension values, the capillary numbers were fixed but the interfacial tension varied, while the other variables of Ca were adjusted accordingly. So, the capillary number used in this analysis varies as the reciprocal of the interfacial tension.

Figures 5.21 and 5.22 indicate with lines the value of the interfacial tension introduced in the numerical scheme. The values obtained with *DDR method* correspond to the symbols. Using multiples of the characteristic time, which was obtained in the same form than in Section 5.1 for the retraction phenomenon on drops, the interfacial tension was evaluated by the slope of the traces as shown in Fig. 5.19 and 5.20, and using Eq. 5.1. These results of interfacial tension were plotted in Fig. 5.21 and 5.22.

Figure 5.21 shows the values of the interfacial tension, obtained for different capillary numbers, using Taylor deformation. For small capillary values, between $0.05 < Ca < 0.15$, the interfacial tension is large, or $20 > \gamma > 6.66$ [mN/m], and the interfacial tension values predicted by the *DDR method* appear to be poor. The best prediction using Taylor deformation occurs in the range of $0.20 < Ca < 0.35$ or $5 > \gamma > 2.86$ [mN/m]. When the capillary number is $Ca = 40$, the prediction start to fail again. Figure 5.21 shown how the prediction is affected using different values of time τ. Analyzing the best prediction, the interfacial tension prediction approximates better the interfacial value when the characteristic times is between $2 < \tau < 2.5$.

Figure 5.22 present the same results but now using the *Mo*-shape parameter. These latter predictions appear to be worse than those using the Taylor deformation. Figure 5.22 shows the best range when $0.20 < Ca < 0.35$ or $5 > \gamma > 2.86$ [mN/m], close to the simulated value. However, Figures 5.21 and 5.22 may indicate the failure of the two methods when attempting to calculate the interfacial tension using the *DDR method*. The best match for the two methods corresponds to a zone between 2τ and 3τ. Figures 5.23 and 5.24 present the systematic error on the interfacial tension values when using different intervals of the characteristic time τ. These errors are presented with respect to the Taylor Deformation and the *Mo*-shape parameter in the *DDR method*.

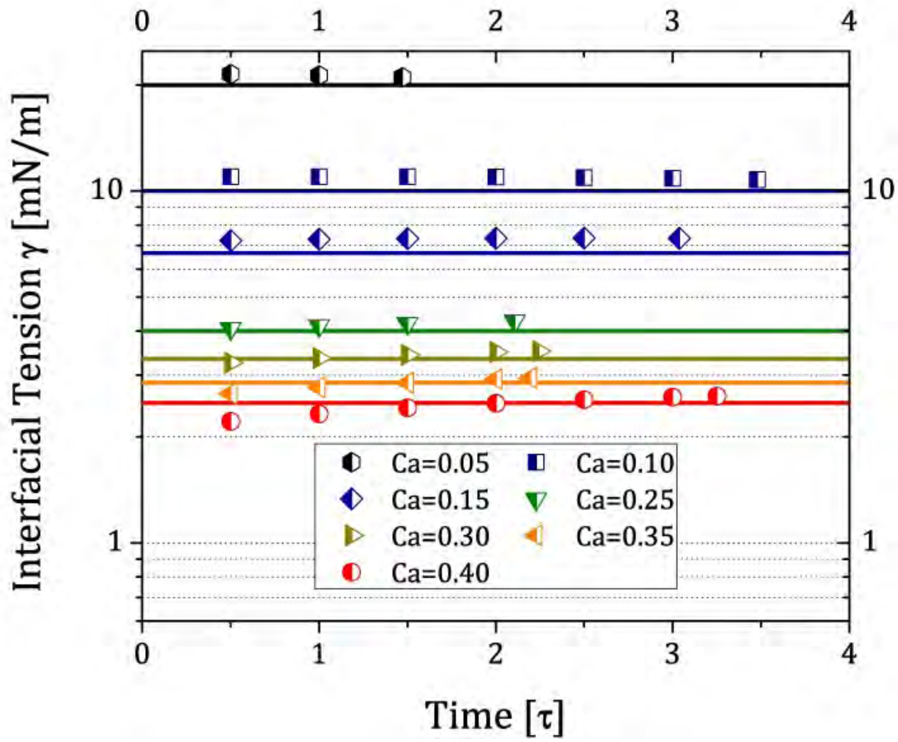

Figure 5.21 *Comparison between the interfacial tension used in numerical simulations (lines) and the interfacial tension obtained with DDR method (symbols). The analysis was made with Taylor deformation, the slopes used were based on multiples of τ.*

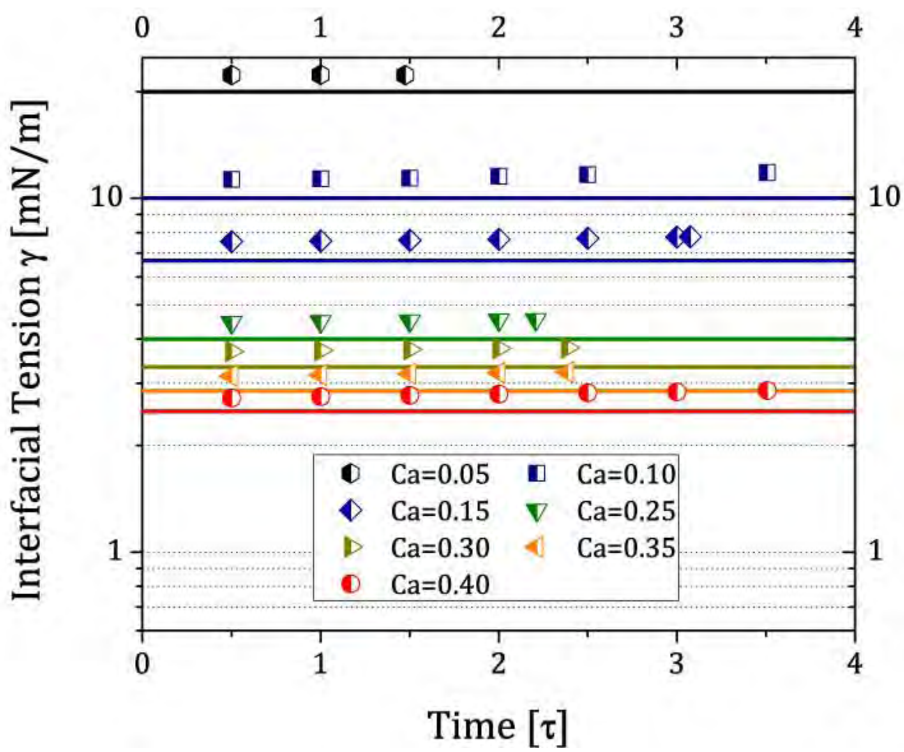

Figure 5.22 *Comparison between the interfacial tension used in numerical simulations (lines) and the interfacial tension obtained with DDR method (symbols). The analysis was made with Mo shape parameter, the slopes used were based on multiples of τ.*

Based on the numerical simulation, the optimal time to obtain the best slope for estimation of the interfacial tension is $Time \simeq 2\tau$. Plots shown in Figs. 5.23 and 5.25 are now linear and the error is shown to be less than $+10\%$. When values of interfacial tension are large -black color- errors are less than 5% for 1.5τ. The interfacial tension best value could be obtained at 2τ, but the orientation angle has changed. If values of γ are taken avoiding the information about the orientation angle, errors could be reduced to less than 3%. The next range -blue marks- have a difference of 10%. For capillary numbers $0.25 < Ca < 0.35$, the error decreases to less than 7% in the best case: $Ca = 0.10$. Finally, for the large capillary numbers used in this work, an error less than 2% would be possible for γ. A similar accurate prediction is feasible at 2τ. However, the estimation of the interfacial tension for low capillary numbers —bigger values of γ— have an accuracy of less than 10%; the difference in this case is the over-estimation of γ. For the other values, errors vary from 5% to 0.1%. If for the analyses more data for $-ln(D_T/D_0)$ is taken, *i.e.*, $Time > 2.5\tau$, the curves deviate more strongly from the linear dependency and the uncertainty increases, Fig. 5.17.

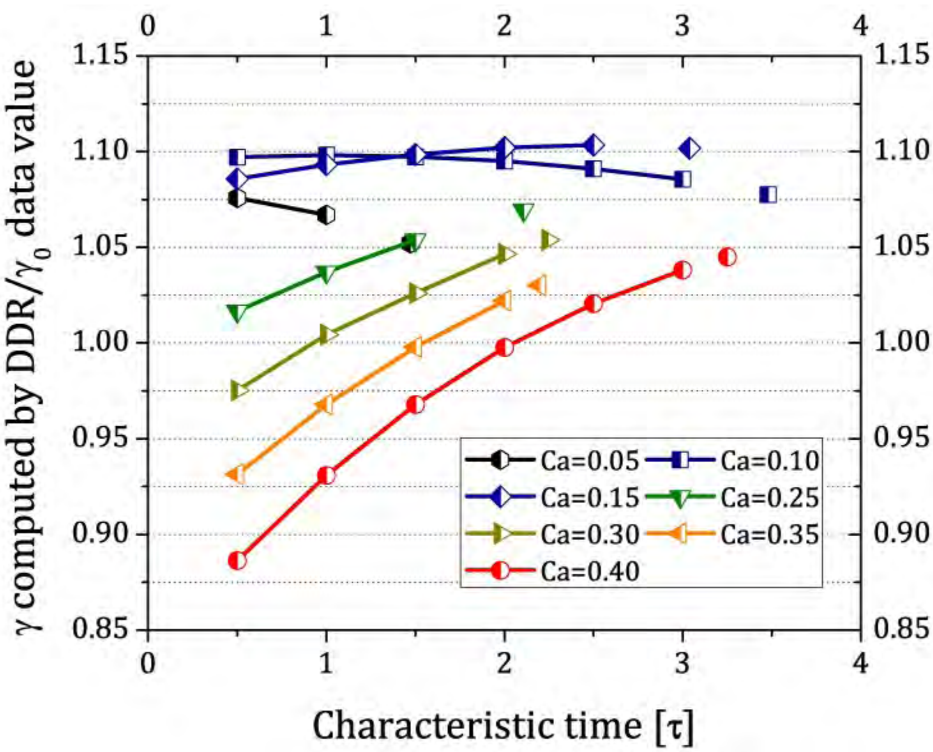

Figure 5.23 *Error between the interfacial tension used in numerical simulations and the interfacial tension obtained with DDR method (symbols). The analysis was made with Taylor deformation, the slopes used were based on multiples of τ.*

Using the *Mo*-shape parameter presents other important features. Comparing Figs. 5.14 and 5.15, The *Mo*-shape model has a very good exponential decay behavior, actually better than when using the Taylor deformation. In Figure 5.15 the analyses on the planes (*L*-*B*) and (*L*-*W*) are similar, with a good exponential decay, and the calculated slopes present also the same behavior. For the same drop shown in Fig. 5.20, the behavior of $-ln(D_{\mathrm{Mo}}/D_0)$ appears better than that of $-ln(D_{\mathrm{T}}/D_0)$. However, values of γ appear to be less accurate than the case of Taylor Deformation.

In Figure 5.22, is evident the difference between the interfacial tension used and the interfacial tension estimated. Figure 5.24 indicates that the error is always bigger than 7.5%. In the cases of $0.15 \leq Ca \leq 0.40$, the optimal time to obtain interfacial tension is in $0.5\tau \leq Time \leq 2\tau$. However, the accuracy is less than 10%. When the interfacial tension was 20[mN/m], the optimal slope was in the same interval $1.5\tau \leq Time \leq 2\tau$. The approximation in this case was less than 10%. Then as the study of DDR with Taylor deformation, the value has less accuracy if the time is $2\tau \leq Time$.

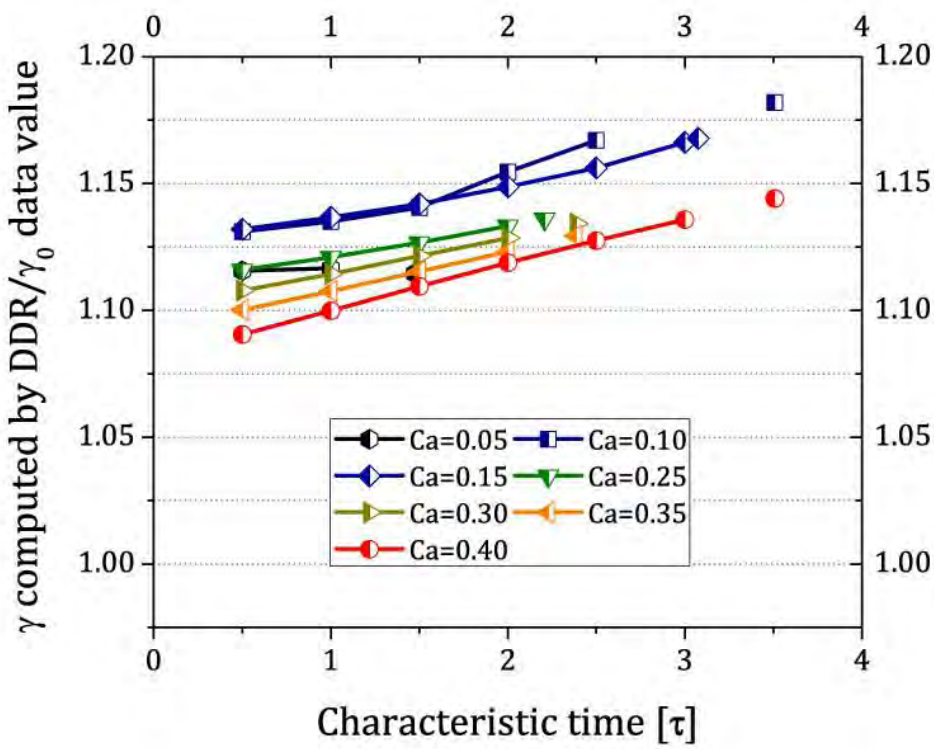

Figure 5.24 *Error between the interfacial tension used in numerical simulations and the interfacial tension obtained with DDR method (symbols). The analysis was made with Mo shape parameter, the slopes used were based on multiples of* τ*.*

Figures 5.23 and 5.24 indicate that the use of Taylor deformation appear to predict better values of interfacial tension when γ is less than 5 [mN/m]. The method does not seem to be sufficiently accurate for values from 5 to 15 [mN/m].

There are other *DDR method*s, based on *the deformation of the W-axis and the projection of the L-axis on the xz-plane*; that is, the experimentally determine length is $L^{\prime} = \boldsymbol{P} \cdot L$, on the *xz-plane*; $\boldsymbol{P}$ is simply a projection operator. Remembering Fig. 1.1, the drop suffers an orientation due to the kind of flow applied. Experiment as Guido, (Guido & Greco, 2001), have access to all views of the drop deformed during the retraction phenomenon. However, experiments by (Yu, Bousmina, & Zhou, 2004) or (Mo, 2000), only have access to the top-view projection of the drop (on the *xz-plane*). In these cases, the actual value of the W-length is observed, but the L-measured is a projection of the real L-length: $L^{\prime}$. With this information, comparisons between theory or numerical results with experimental information that use the *xz-plane* projection can only approach the accuracies here calculated.

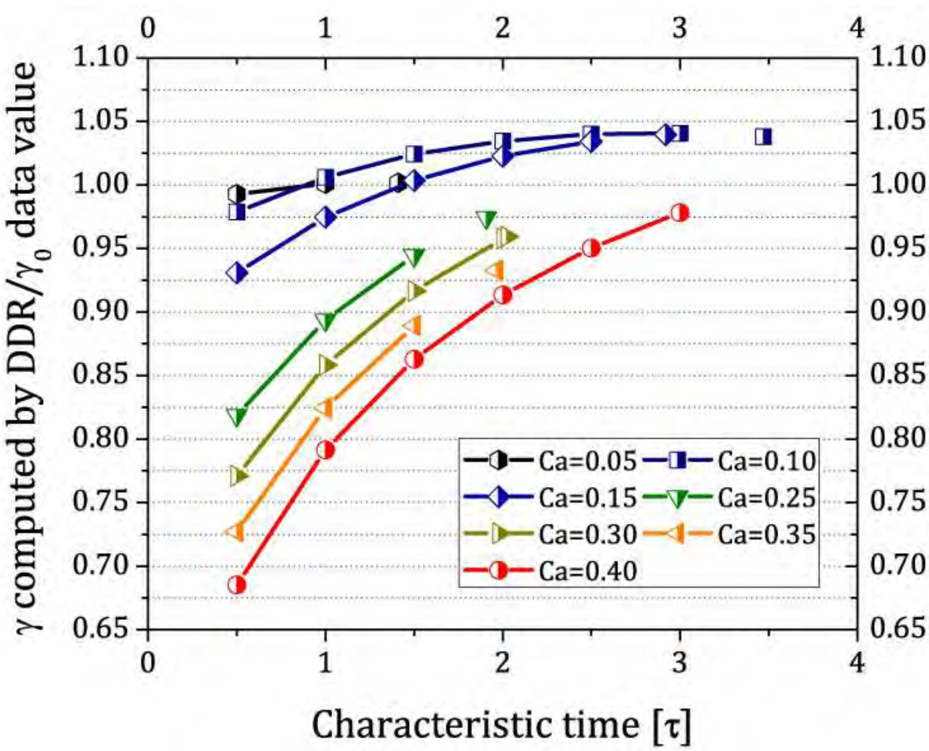

Figure 5.25 *Error between the interfacial tension used in numerical simulations and the interfacial tension obtained with DDR method (symbols). The analysis was made with Taylor deformation in (L, W), the slopes used were based on multiples of τ.*

Figure 5.25 shows errors when using the *LW-plane* in the *DDR method*. Here, Taylor-deformation measure shows a better approximation in $2\tau \leq Time \leq 3\tau$ that the analysis with *L-* and *B-axis*: the error is $\leq 6\%$; see Fig. 5.23. In Figure 5.25, the error is $5\% \leq error \leq 10\%$ when $2\tau \leq Time \leq 3\tau$ and $0.10 < Ca < 0.15$. When the interfacial tension is approximately 2.5 [mN/m], the error is always above 10% at $Time < 2\tau$, and the error decays to less than 3% when $Time = 3\tau$. Finally, when $0.15 < Ca < 0.35$, and using the change of the orientation angle, the *DDR method* can be applied for $Time \leq 2\tau$. Prediction for these cases imply errors less than 10%. Comparing the errors of Figs. 5.23 and 5.25, the use of the *LW-plane* predicts better values of the interfacial tension than when using the plane parallel to the flow, *LB-plane*.

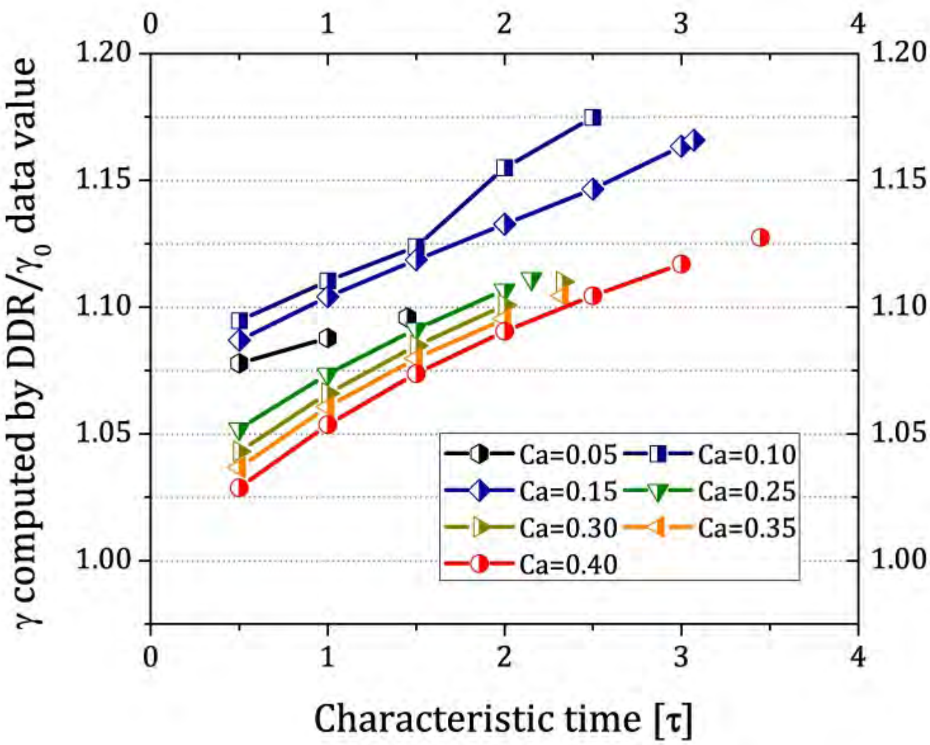

Figure 5.26 *Error between the interfacial tension used in numerical simulations and the interfacial tension obtained with DDR method (symbols). The analysis was made with Mo shape deformation in (L, W), the slopes used were based on multiples of τ.*

Figure 5.26 shows the error using the *LW-plane* in the *DDR method* using the *Mo*-shape parameter. Figure 5.26 indicates a better approximation than the analysis with the *L-* and *B-axis*. The error is $5\% \leq error \leq 10\%$ in the case of 0.5τ for $0.25 < Ca < 0.35$. When the interfacial tension is $20 \geq \gamma \geq 6.66$ [mN/m], the error is always above 5% and less than 11%, when $0.5\tau < Time < \tau$. The error increases for times above $Time = \tau$. When the interfacial tension is 2.5 [mN/m], the error is always below 10%, when $Time < 3\tau$; then, the error increases. The error is $> 3\%$ in the best of predictions for this interfacial tension value.

Analyses using the Taylor deformation or the *Mo*-shape parameter applied on the cross section of the drop, *W*- and *B-axis*, provide the worst predictions, except for $Ca = 0.05$ and 1.5τ. For these cases the approximation is between $10\% < \ error\ <$ 15%; for the other cases, the error was more than 15%. When $Ca = 0.05$, the model with the *Mo*-shape parameter provides the weakest estimation with the error being larger than 15%.

The best match to estimate interfacial tension for a drop that was deformed in a flow with $\alpha = 0.13$ and then was relaxed to a near spherical shape is the *DDR method* using the Taylor deformation. However, this match was obtained analyzing *LW-plane*. Reviewing Fig. 5.14, this plane has a decay which differs more than the *LB-plane* compared as decay exponential behavior. In fact, the best adjusts to a decay exponential (Mo shape parameter) have the worst approximation. Another remark is that Mo shape parameter adjust an ellipsoidal drop, the interfacial tension that was well estimated with this parameter was $Ca = 0.45$, *i.e.*; the deformation of the drop was the largest, in theory, the best adjust must be the case when the drop has $Ca =$ 0.05, when the drop is like a spheroidal shape object.

Based on results presented in Chapter 3 and 4 it is now clear that deformations along the three axes of the drop are different. At the end, *models of prediction as ellipsoidal forms are not appropriate*, because the retraction process of a drop appears to be the worse approximation when attempting to simulate an elliptical shape with an exponential decay. As well, taking the numerical data of Chapter 3 and 4, the projection of the drops on the *xy-plane* is clearly not an ellipse. Thus, all processes of retraction of ellipsoid drops will incur the errors observed. Using *LW-plane* data will present essentially the same problem, albeit with errors a whit smaller. We may say that the drop in the *LW-plane* is closer to an elliptical form than the conventional *xy-plane* projection: *LB-plane*.

Another possible interpretation of these observations is that theories that assume an exponential decay may be the correct ones, as shown in Fig. 5.14 and 5.15, but those theories clearly require improvements for an accurate estimation of the deformation shape as function of Ca, λ_{μ} and the flow parameter α. The existing linear models used to date are not good enough to describe accurately the retraction in time, with a poor prediction of the interfacial tension value. For this reason, a new theory is needed, even for the simplest liquids, such as a Newtonian fluid.

The interfacial tension values determined by a dynamical experiment indicate that a better prediction of γ comes from the third direction that is not analyzed in conventional experimental devices. This observation comes about the numerical calibrations presented here. But having a view of the third dimension is a difficult experimental task (Guido, Greco, & Villone, 1999). For cases (Yu, Bousmina, & Zhou, 2004) and (Mo, 2000), when the *W-axis* is well determined but the *L-axis* is only its projection, the obtained deformations are not the correct ones. Another complication is to get the data of precise drop dimensions: there are always uncertainties of the experimental device due to illumination, or the image algorithms to analyzed the shapes of the drops (Rojas 2014), (Rosas 2012), et cetera. For these reasons, I hope that numerical experiments will help to develop better estimations of the drops parameters and consequently better determination of the interfacial tension. As an example, is the fact of optimization of characteristic time it is possible using numerical data. With this information, the estimation of the interfacial tension will be improve with the less possible error intrinsic by the method

To estimate interfacial tension, it will be necessary to employ numerical simulations with experiments in the laboratory to have the information in the LW-*plane* to predict the interfacial tension using *DDR* techniques.

CHAPTER 6.

Shear rate vs. Capillary number.
Effect on the steady flow form of drops

In Chapter 3 and 4, the analysis of evolution of the drop-form is given in terms of the behavior for the orientation and the deformation of the drop once the stationary state is reached. In this Chapter, I present and interesting variant based on the analysis for the drop deformation histories when the capillary number is fixed, but the shear rate is varied. That is, here all numerical experiments presented were carried out at constant Ca, but effects induced by *the intensity of the flow* G are systematically studied. G corresponds to the intensity of the rate of deformation tensor (Makosco, 1994), Eq. 1.6. If the case is *simple shear flow* G is known as shear rate, $\dot{\gamma}$; and in the case of *extensional 2D-flow*, G is the *strain rate*, $\dot{\varepsilon}$. The non-dimensional time Gt is the same for all experiments. As in Chapter 3 and 4, the observed behaviors are the same for different values of flow parameter α, thus here the Figures employed to illustrate the relevant results correspond to only the regime for $\alpha = 0.13$.

6.1 Consequences of shear rate *G* in the stationary state of the drop for small value of rate of viscosity

In Chapter 3, the evolution of the shape of the drop is shown in Fig 3.1. In all figures plotted, the trace presents tiny oscillations at long times, with similar fluctuations on the trace for the orientation angle. At the beginning of this study, those oscillations were assumed to be due to a poor resolution of the mesh applied in these numerical simulations. However, when the mesh was plotted to make movies of the history of deformation of the drop, the resolution of the mesh performed very well, in all images; *i.e.*, the elements' shapes and the median curvatures were consistent with the data obtained. The resolution for all times was good enough to discard the presence of an abnormal deformation of the mesh.

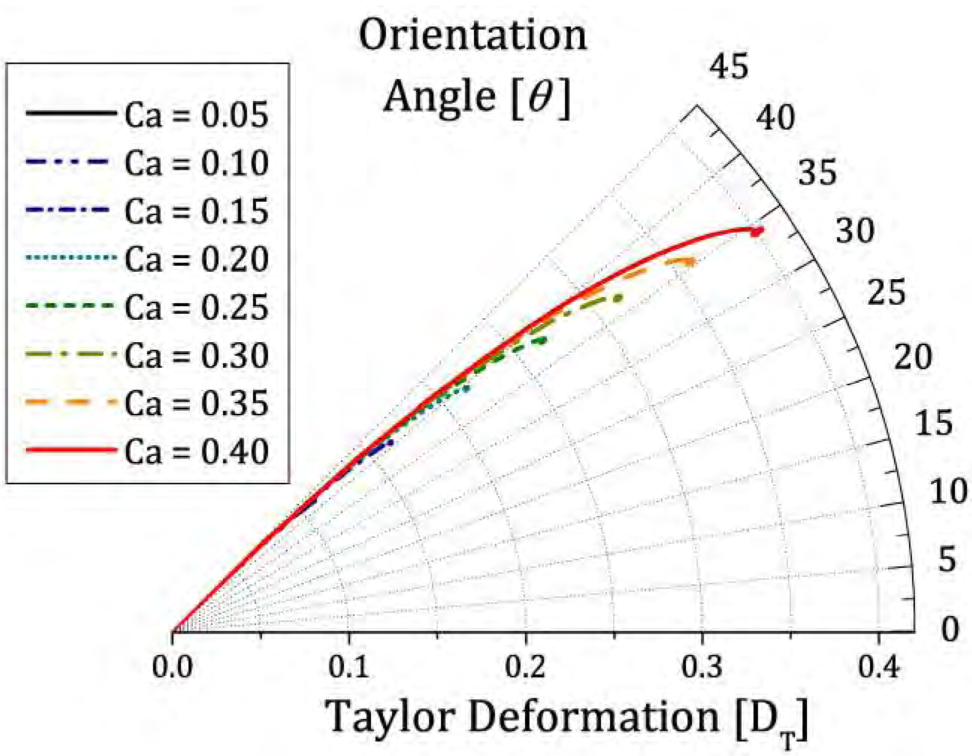

Figure 6.1 *Polar representation of Angle of Orientation vs. Taylor Deformation with* $\alpha = 0.13$ *and* $\lambda_\mu = 0.012$.

The movies of the dynamics of drop deformation show a sliding motion of the mesh on the interface: The final shape of the stationary state and the tiny oscillations were quite obvious. These oscillations of the drop actually change the deformation and the orientation of the drop. However, the observed behavior of these tiny oscillations was not a function of mesh size resolution. The orientation of the drop plotted versus deformation of the drop is presented in Figure 6.1, for the long-time form under a flow with $\alpha = 0.13$ and $\lambda_{\mu} = 0.012$, as the data given in Chapter 3.

At the onset, there is no flow applied in the continuum phase, and the drop is spherical —the deformation is zero and there is no angle of orientation. Then, when the strong flow is applied, the drop starts to deform and suffers a re-orientation proportional with —along the direction of— the *eigen*-direction of the stress deformation tensor. At the end, when the long-time shape is attained, the deformation and the angle of orientation come to their final values. However, the "steady state of deformation" is characterized by an interesting behavior: there are oscillations. That is, curves in polar plots should finish in a single point when a stationary shape is attained. However, if these plots are observed with care, the stationary shape is not a point; in fact, there is a little curved closed trajectory at long times. This aspect motivated to analyze again this figure.

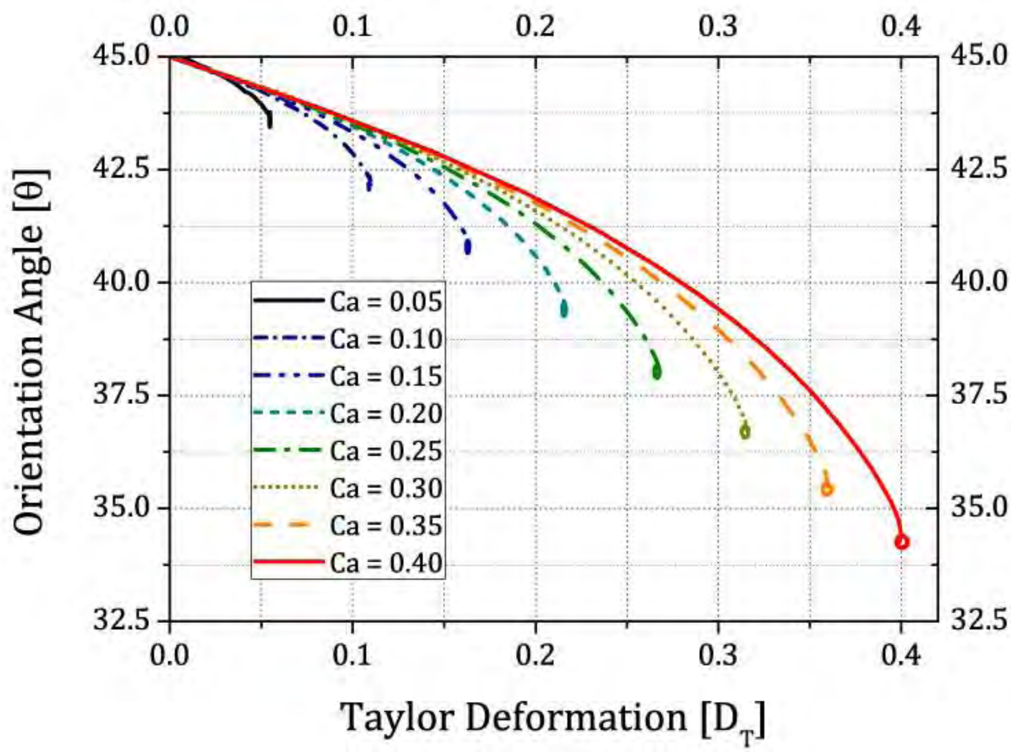

Figure 6.2 *Angle of Orientation vs. Taylor Deformation with $\alpha = 0.13$ and $\lambda_{\mu} = 0.012$.*

Figure 6.2 shows the same information of Fig. 6.1. However, the representation is not a polar plot, it is in a Cartesian form. Here, the final trajectory around the end point is more evident than the previous plots. The information about these drops with low capillary number is clearer than the polar plot figure. The end "point" of the steady shape of deformation in polar plot are actually tight trajectories around a point. This final close trajectory corresponds to the oscillations mentioned above. In other words, the stationary shape in the numerical oscillations goes around searching for a stationary point.

Comparing the polar plots data of Reyes (Reyes, 2005) and Rosas (Rosas, Reyes, Minzoni, & Geffroy, 2014) the *2D-numerical* and the experimental data referred have a similar behavior (trajectories around a point). However, the analysis of these curved trajectories around a point was omitted because the relevant behavior was due to supposed effects of the resolution of the numerical method (Reyes, 2005). In the experimental case, these oscillations were considered as normal variation of uncertainties of the control mechanism.

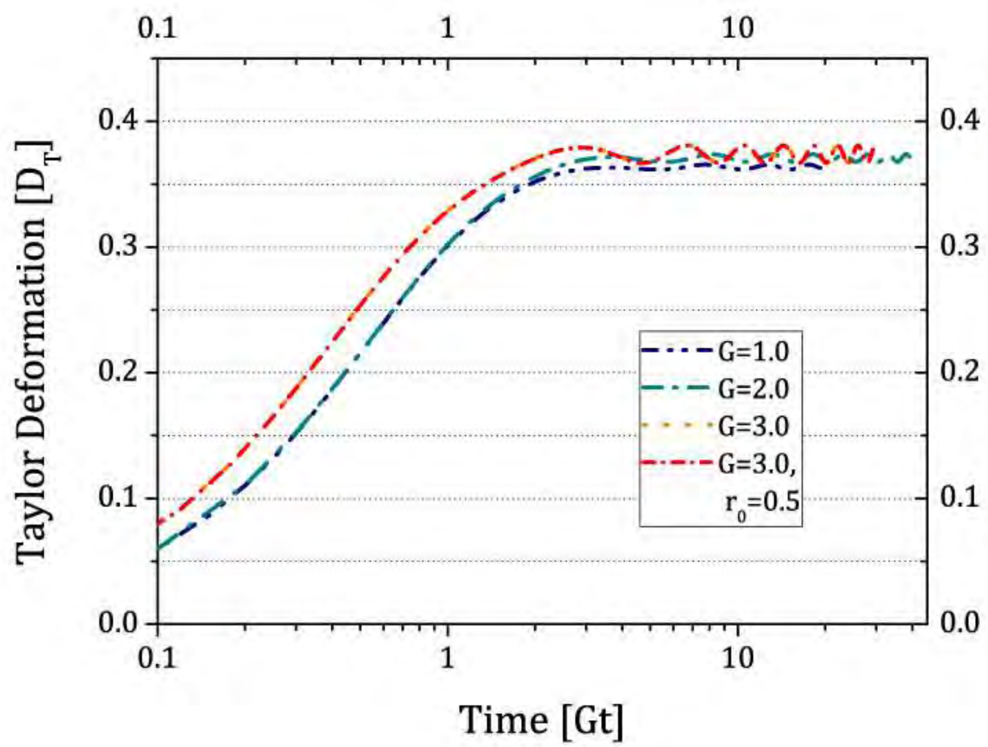

Figure 6.3 *Evolution of Taylor Deformation for the same capillary number with different values of G.*

To better understand the observed behavior, a large set of numerical simulations were needed. It was necessary to design new algorithms to achieve long times of simulation, without a heavy computational cost. Initially, to optimize CPU-Time and to carry out the required number of numerical simulations, the G value (shear rate) was increased maintaining the capillary number fixed. For example, to perform a simulation for a capillary number $Ca = 0.35$, the time-length of every step was set at $h = 0.001\,Gt$. Then, the stationary shape and the retraction time were observed only after 20,000 time steps. The idea was to set a larger value for G, in order to produce a simulation faster. For the case of $Ca = 0.35$, $G = 2.0$, the final number of steps were 10,000; in this manner, in less than 16hr the simulations were completed. The results for capillary number of $Ca = 0.35$ are shown in Fig. 6.3; the deformation histories of the drop, observed in 4 different experiments, were different.

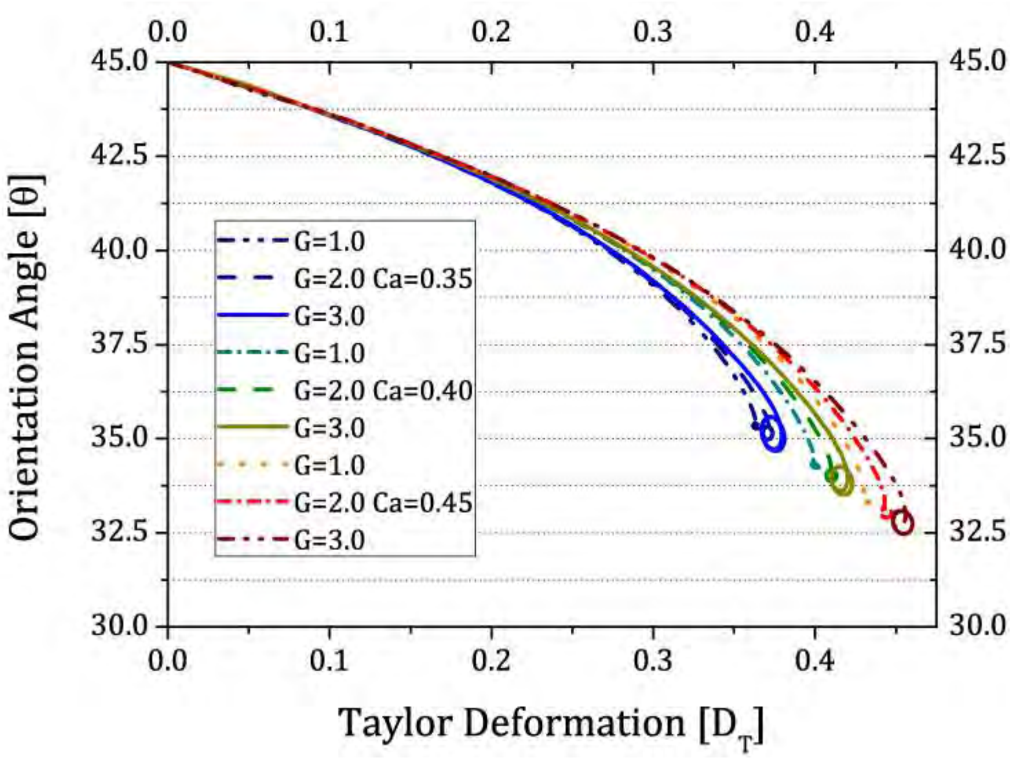

Figure 6.4 *Angle of Orientation vs Taylor Deformation for drops with the same Capillary Number with different values of G*

In Figure 6.3, the long-time oscillations increased proportionally in size to the value of G. Also, the deformation in the steady state has a different final value (larger but less than 7% in most cases). Figure 6.3 shows an experiment where the parameter G and the initial radius of the drop were change while maintaining constant the value of the capillary number: $Ca = 0.35$. The final oscillations were similar as those numerical results when the parameter G was the only change; *i.e.*, it appears that it does not matter if the radius of the drop or the interfacial tension is modified. In essence, the phenomenon of rotation of *the steady shape may be dominated by changes of the intensity of the applied flow*.

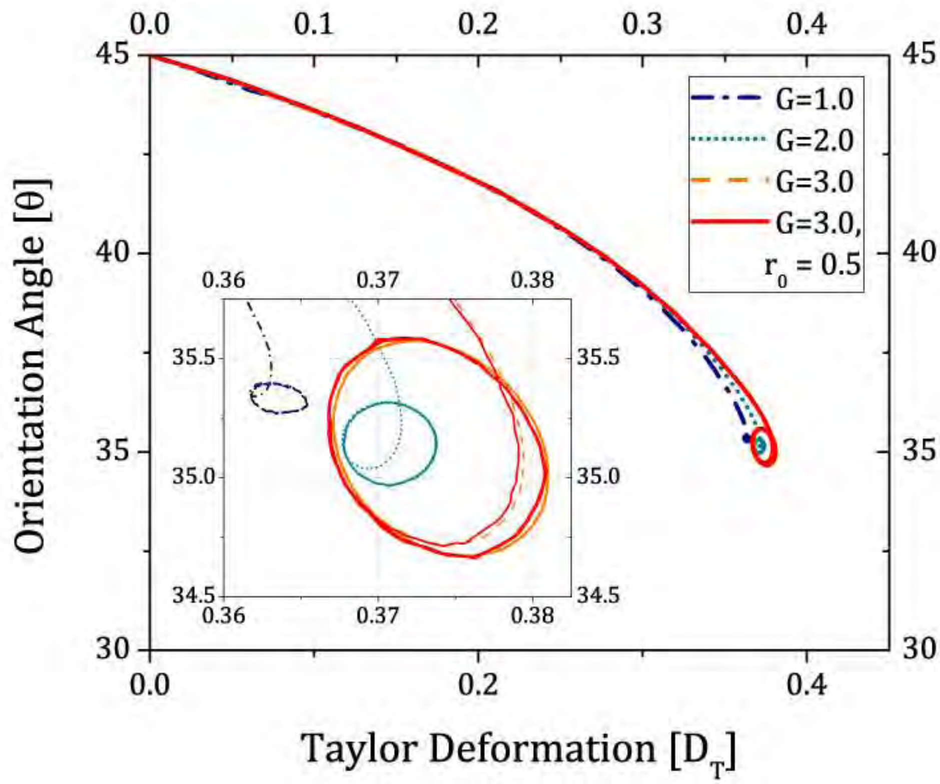

Figure 6.5 *Angle of Orientation vs. Capillary Number for* $Ca = 0.35$ *and different values of* G.

Figure 6.4 shows the trajectories of the orientation angle versus deformation of the drop, for capillary numbers below the critical capillary number $Ca < Ca_{cr}$, but with values large enough to observe appreciable states of deformation. These values are: $0.35 \leq Ca \leq 0.45$. It can be observed in Fig. 6.4 that close trajectories increase in size for the larger values of G.

Oscillations around a stationary state suggest the possibility of a Hopf bifurcation. During the last 7 years, several research groups (Zhao & Shaqfeh, 2011), (Lalanne, Tanguy, & Risso, 2013), and (Spann, Zhao, & Shaqfeh, 2014) have found similar phenomena for *vesicles in shear flow*. They found that the oscillation increase in amplitude as a result of larger shear rates in the *shear flow*. That effect is very likely equivalent to the one presented above. However, there is no detailed analysis of drop deformations in neither strong flows nor the least for *simple shear flow*.

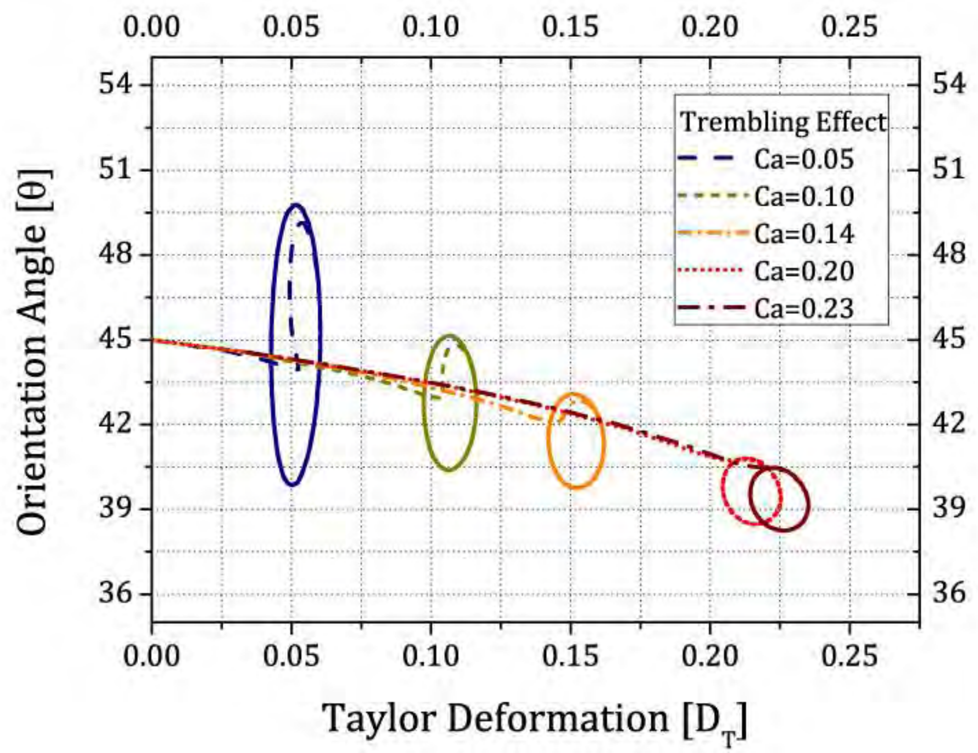

Figure 6.6 *Trembling effect in drop deformation for different capillary number with $G >> 1$, $\alpha = 0.13$, $\lambda_\mu = 0.01$.*

The numerical studies referred above indicate the sliding motion of the vesicle interface without altering details of its shape. These vesicles reach a steady orientation and deformation but also *slide* in the plane of the flow, in a motion similar to a sliding skin on a rigid body, and this effect is called *tank-treading*. In this case, the vesicle describes a similar trajectory as drops in Fig. 6.4 with $G = 1$. For plots of orientation angle vs. deformation, the *tank-treading* rotation is observed as a close tight trajectory at long times. The phenomenon of *tank-treading is observed in this simulations as the slipping motion of mesh elements on the interface*, while maintaining the orientation and deformation of the drop with very tight quasi-constant values.

The other solution of the Hopf bifurcation observed in vesicles corresponds to the "*trembling effect*". For this second solution, vesicles suffer a continuous and simultaneous oscillatory reorientation as well as an oscillatory degree of deformation. That is, a small contraction-expansion of the vesicle, as well as a continuous jiggling of reorientation are observed; *i.e.*, *the deformed vesicle trembles*. The plot in time of angle vs. deformation of this effect in vesicles is like the trajectories observed in Figs. 6.4, when $G > 1$. Figure 6.5 shows an analysis in the steady regime when $Ca = 0.35$, but different values of G. Again, there is a similar effect to *tank-treading* for the case when $G = 1$. However, by zooming in the last part of the evolution, the figure shows that there is a case like *trembling*. The other trajectories present clearer evidence of the *trembling* behavior. Finally, the last two trajectories are similar. Changing the radius and the interfacial tension and maintaining the same capillary number does not affect the trajectory as Fig. 6.3 shows.

Until the data shown in Fig. 6.5, all cases observed correspond mainly to *trembling effects*. The second solution referred to as "*tank-treading*" appears when values of the intensity of the applied flow G are small and $0.03 < Ca < Ca_{cr}$. These dynamics are characterized by an interesting behavior. Looking again the close trajectories of Fig. 6.5, the *trembling* condition corresponds to an ellipsoidal trajectory in the orientation angle vs. deformation diagram; in contrast to *tank-treading* can still be best characterized by a tight close trajectory.

A third possible solution, which corresponds to the last case observed in vesicles in *simple shear flow*, is the *tumbling* solution. The "*tumbling effect*" (or solution) corresponds to a closed trajectory with a *bell shape*, carefully described in the work of Shaqfeh and coworkers (Spann, Zhao, & Shaqfeh, 2014) and observed while studying vesicles. The changes of the orientation angle and deformation are the largest for these trajectories, with a rather unusual form: a bell shaped. To observe this phenomenon in vesicles, the shear rate values must be very large; the *tumbling effect* was not observed.

Figure 6.7 shows different drop simulations for cases when values of G are larger than 1, and the *trembling* solution is observed, in all cases. As the capillary number is reduced, the observed closed trajectories become more elongated. In other words, for a small capillary number and for very large G values, we observe, in Figure 6.6, the transition between *trembling* and *tumbling*. Figure 6.8 shows the comparison of the trajectories for similar capillary values with a notable difference in the value of G. The Trajectories are similar, only the parameters characterizing steady states change.

Furthermore, evidence of the Hopf bifurcation is observed in vesicles. This evidence implies the necessity to review the drop deformation in strong flows by mapping these transitions of the configurations with the purpose of understanding these bifurcations.

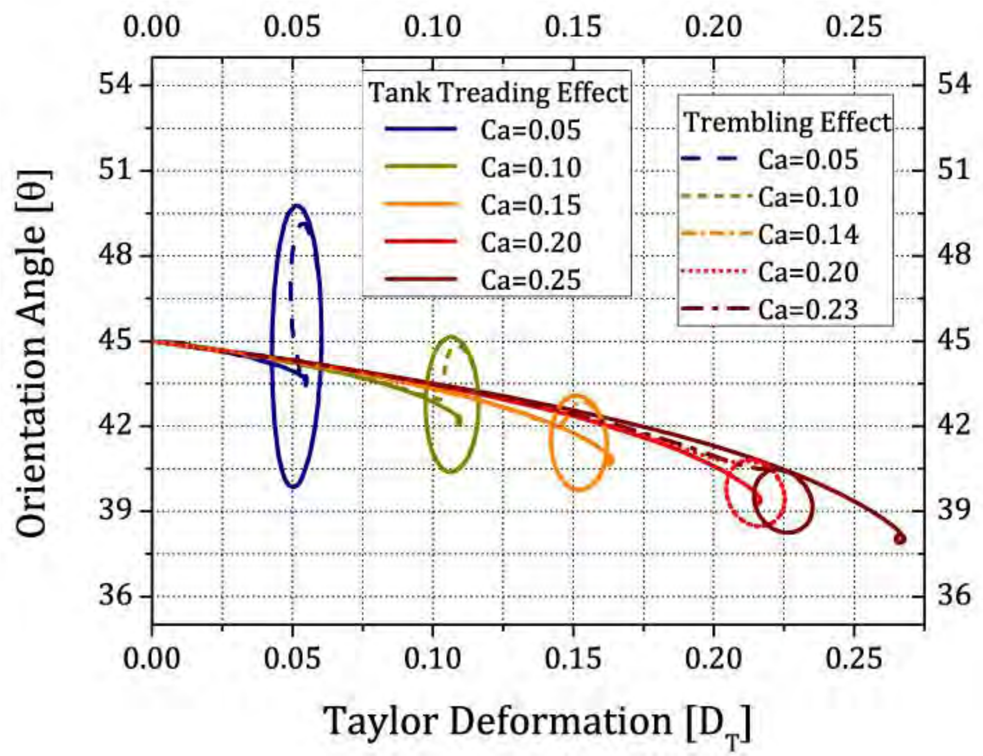

Figure 6.7 *Comparison of the trajectories of steady states for Angle of Orientation vs. Taylor Deformation of different Capillary Number, but different values of G*

The tiny oscillations described in Chapter 2, and observed in all subsequent Figures, are due to increments of the value of the shear rate *G*. When the intensity *G* increases, drops simultaneously elongate and rotate towards the outflow axis until a final orientation and deformation is reached; this is essentially the *tank-treading phenomenon* (Please recall that the final steady state orientation of the drop under small *Ca* is not necessarily closely aligned with the outflow axis). In contrast, a final state with finite oscillatory changes in the deformation and orientation corresponds to the *trembling effect*, and was reported too; so, *trembling* refers to jiggling about the equilibrium orientation.

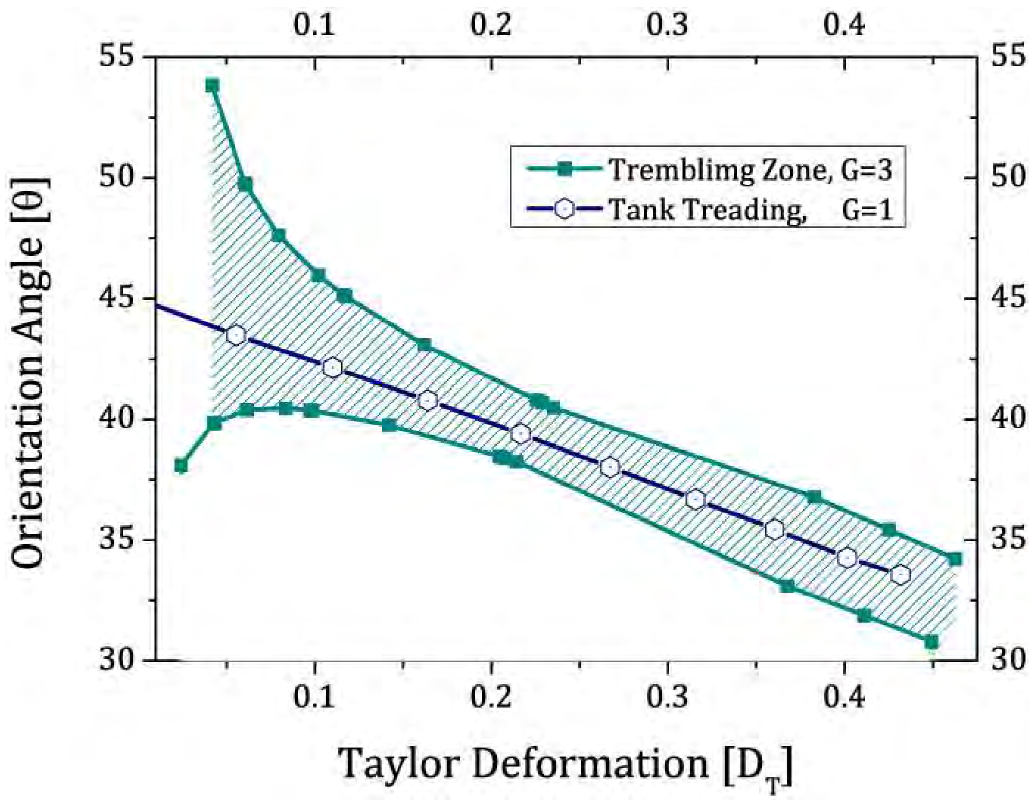

Figure 6.8 *Comparison of the trajectories of steady states for Angle of Orientation vs. Taylor Deformation of different Capillary Number, but different values of G,with* $\alpha = 0.13$ *and* $\lambda_\mu = 0.012$. *The blue symbols are the steady states of drop deformation for capillary numbers* $(Ca = 0.05, 0.10, 0.15, 0.20, 0.25, 0.30, 0.35, 0.40, and\ 0.45)$ *with* $G = 1$. *The green symbols are the steady states of deformation obtained by* $G = 3$.

The concepts of *tank-treading* and *trembling* have been observed for capsules. However, there are no reports of these effects in the literature of drop deformation. One possible explanation for this lapse of understanding may be due primarily to the use of the normal component of the velocity (Pozrikidis, 1992) for previous numerical studies. When the drop deforms, many of the numerical codes use the Eulerian representation of the mesh and reconfigure it, so, they could not observe the steady displacement of the mesh in the plane of flow by the effect of the flow vorticity, in the Lagrangian representation.

The numerical code employed here does not refine the initial mesh, so, a Lagrangian representation is used to avoid distortions of the elements due the deformation of the mesh. For this reason, to observe specific features of the dynamics of deformation, the tangential component of the velocity of the mesh is determined and is employed to advance the time evolution while being useful as well to evaluate possible surface slipping. This additional information better exposes, in an unambiguous manner, the *tank threading* phenomenon described. Figure 6.9 shows the effect on plotting the drop deformation vs. time, when using different frames of reference, the Eulerian frame of reference of the drop versus the Lagrangian frame — specifically, the normal and the tangential components of the velocity field.

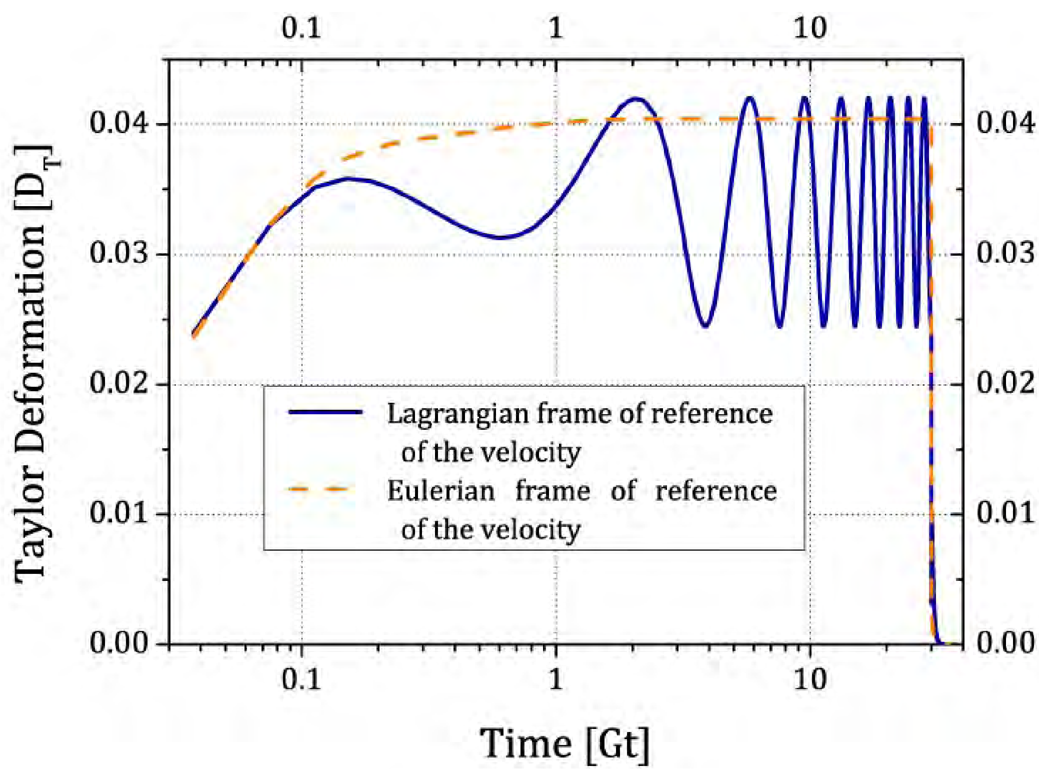

Figure 6.9 *Comparison of evolution of drop deformation in the trembling effect when the velocity is calculated using only the normal component of the velocity (dash line). The velocity calculated using the normal and tangential component employed is observed in the blue line. $Ca = 0.04$, $\alpha = 0.13$ and $\lambda_\mu = 0.012$.*

The values of the drop deformation versus time, shown in Fig. 6.9, correspond to the trajectories of Fig. 6.8, indicating the translation of the mesh along the tangential direction of the applied local flow. A similar behavior is reported by Spann (Spann, Zhao, & Shaqfeh, 2014), the tangential motion of the mesh, like that of a conveyor belt, is associated to *the phenomenon called tank-treading*.

6.2 Consequences of shear rate G on the stationary state of the drop for large capillary number values.

All numerical experiments presented here attempt to simulate an equivalent flow condition, based upon the dimensionless numbers, to previous *experimental studies* of drops in flows used several times the same drop (that is, size, viscosity and surface tension being fixed) while the parameter that changes in experiments is correspondingly the shear rate, in *simple shear flow*; the *strain rate*, in *extensional flows*; or in strong flows, the parameter G (Bentley & Leal, 1986b), (Kennedy, Pozrikidis, & Skalak, 1994), (Rosas I. Y., 2013).

For the experiments presented in Chapter 3, the numerical method predicts similar deformations as those observed in the experimental data. Thus, the initial idea was to validate the previous experimental data. Motivated by these results, many numerical experiments were carried out under the regime of $\alpha = 0.03$, $\alpha = 0.05$ and $\alpha = 0.13$. For these runs, the capillary number used constant values for the surface tension, the viscosity of the continuum phase and the radius of the drop. Only the parameter G was varied in order to vary the capillary number. The experimental data, took values of the parameter G within the range of: $0.5 \leq G \leq 1$. Therefore, my numerical experiments were performed with these data inputs.

In the previous Section, I emphasize the importance of *the intensity of the flow* for the selection of the solution describing the drop deformation elucidating the *tank-treading*, *trembling* and other possible solutions. That is, the numerical experiments maintained the values of the capillary number fixed, but the parameter G varied.

In contrast, as Figures 6.10, 6.11 and 6.12 show, the numerical and experimentally evaluated steady shapes of drop deformation diverge; most importantly, for different capillary numbers and for a large set of different flows: $\alpha = 0.03, 0.05$, and 0.13, respectively. In all cases, the numerical predictions matched well the experimental values, but only for small capillary numbers. However, as the value of the capillary number increases, differences between numerical and experimental data grow; up to the critical value of the capillary number (that is, when the drop can no longer attain a stationary shape).

Figure 6.10 shows the numerical vs. experimental difference for the deformation parameter caused by a flow similar to simple shear: $\alpha = 0.03$. Differences are small when $\mathrm{Ca} < 0.2$: less than 20%; then, they increase up to 60% near the critical capillary number, when numerical data predicts a value below $D_T = 0.5$, while the experimental data is close to $D_T = 0.8$. Figure 6.11 and 6.12 show similar discrepancies under more elongational type of flows; *i.e.*, for low values of the capillary number, the numerical method correctly predicts steady Taylor deformations, in contrast with larger capillary numbers. However, as the parameter α increases, differences decrease from 60% (for $\alpha = 0.03$) to 25% (for $\alpha = 0.13$).

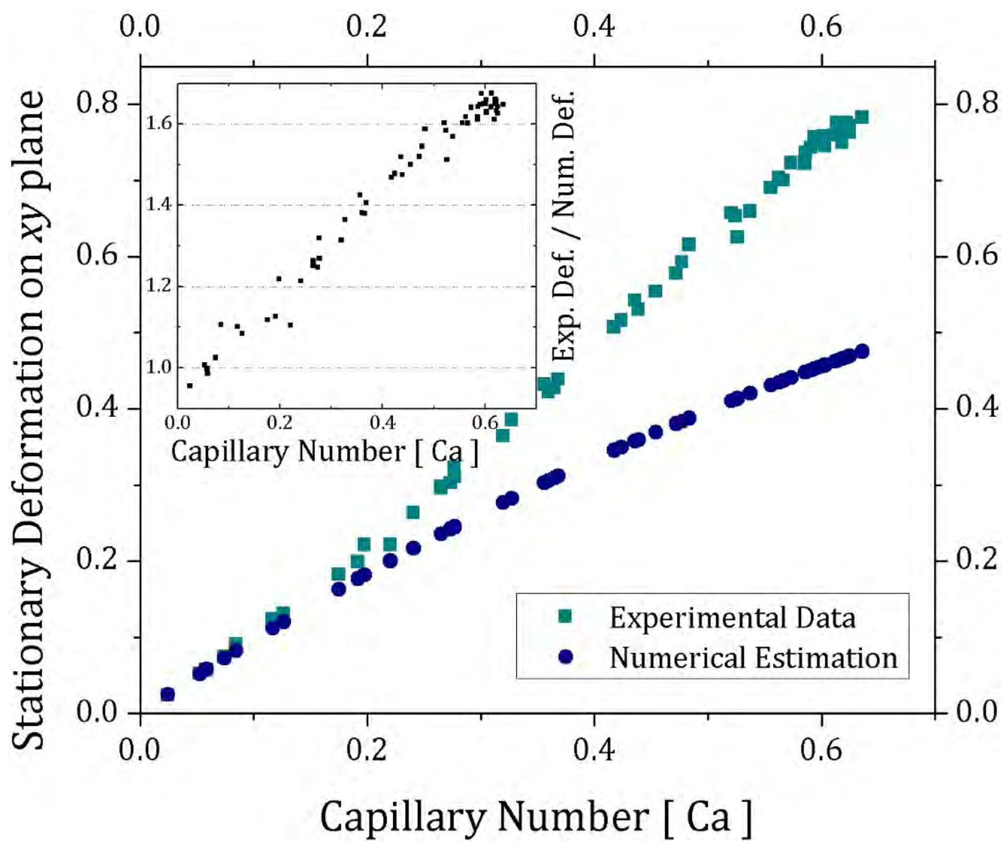

Figure 6.10 *Comparison between experimental and numerical data of steady deformation on xy plane vs capillary numbers with $\alpha = 0.03$, $\lambda_\mu = 0.012$. The insert Figure is the numerical error w.r.t. the experimental data.*

The input data for numerical experiments, employed in previous Chapters, were selected based on the best match between numerical vs. experimental data. The next *question to address* was to study and understand possible reasons for the failure of the numerical predictions of the drop deformation during the steady state phase and for the larger capillary numbers. The main motivation was the possibility of having an accurate numerical method to simulate large steady deformation, near to the break up point (rupture) of the drop

Because rupture-of-drop conditions is a very relevant topic of study in multiple applications, understanding the physical processes around the events before rupture are essential to a detailed modeling of the whole process. Modeling those pre-rupture processes with the BEM3D algorithm could give a complete panorama about large and critical deformations. However, as Figures 6.10, 6.11 and 6.12 indicate, reaching the capillary critical values may not guarantee attaining the associated critical state of deformation as was expected.

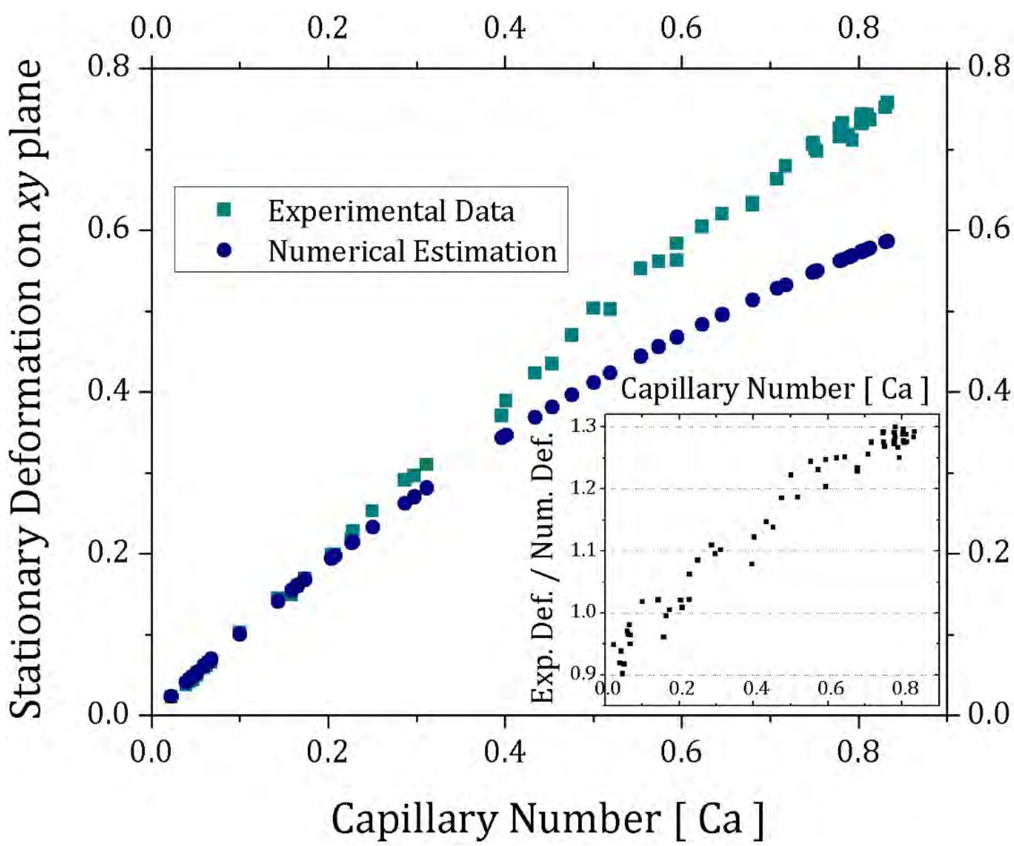

Figure 6.11 *Comparison between experimental and numerical data of steady deformation on xy plane vs capillary numbers with* $\alpha = 0.05$, $\lambda_\mu = 0.012$. *The inset Figure is the numerical error w.r.t. the experimental data.*

At the beginning of this Chapter, the study of a drop in a flow emphasized the states of deformation when the parameter G strengthened. Also, these results indicated that the shape attained for long times (commonly referred as) the stationary state can, in fact, be *a region about* a given value. Trajectories about a point are shown in Fig. 6.5. If increments of G for the flow around a drop could cause a new state of deformation as in Fig. 6.10, 6.11, and 6.12, then there are at least two types of flow kinematics providing two solutions: one with a very tight trajectory about a point and the other a trajectory about an ampler region about a point. The first one corresponds to the *tank threading* conditions while the latter to the *trembling* solution.

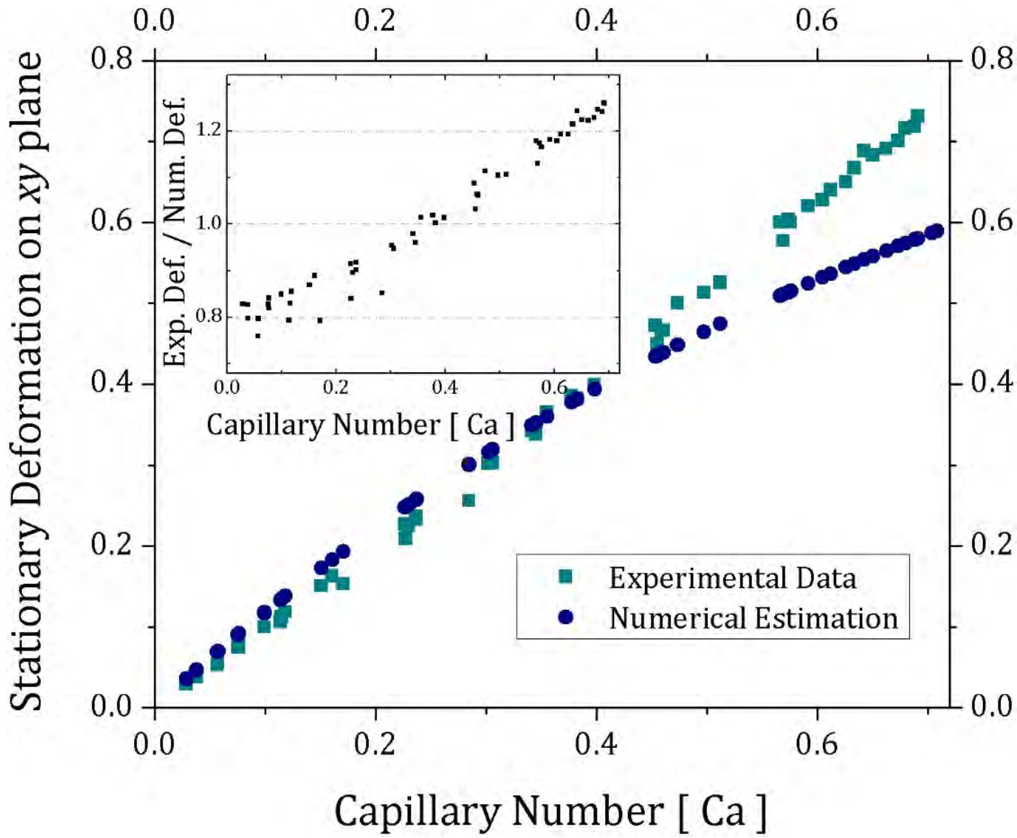

Figure 6.12 *Comparison between experimental and numerical data of steady deformation on xy plane vs capillary numbers with* $\alpha = 0.13$, $\lambda_\mu = 0.012$. *The inset Figure is the numerical error w.r.t. the experimental data.*

The question now is to determine which of these two possible solutions was observed experimentally by Rosas *et al.* Carrying out BEM3D simulations about the critical capillary number, while varying (increasing) the *parameter G* of the flow, in a similar manner to the experiments of Rosas, then the "correct" solution, observed experimentally, could be determined. So, if multiple solutions exist, then these simulations may imply that there are at least two Hopf bifurcation solutions. Simultaneously, the parameter space for the other branch (solution) and its critical values could be investigated numerically for future comparison with experimental studies of these phenomena. Furthermore, one, or both solutions may lead to rupture; but which ones and under what conditions is still a question not addressed experimentally.

The next analysis considers effects due to large values of the G parameter. Here, I mean G values thrice and four times larger than the values used in Fig. 6.10 to 6.12. As in Chapter 3, the viscosity ratio was $\lambda_{\mu} = 0.012$, and $\alpha = 0.13$. The analysis presented in this section will be focus near the capillary critical number for $\alpha = 0.13$.

Figure 6.13 shows the deformation of the principal *L-axis* of the drops for different capillary numbers. Amplitudes of oscillation increase as G increases. The largest value of G implies less observation time, because the time resolution (time step for evolution) and the distortion of the mesh (characteristic length-scale for the numerical code become critical. This distortion causes that, numerical simulations finish with obvious numerical errors.

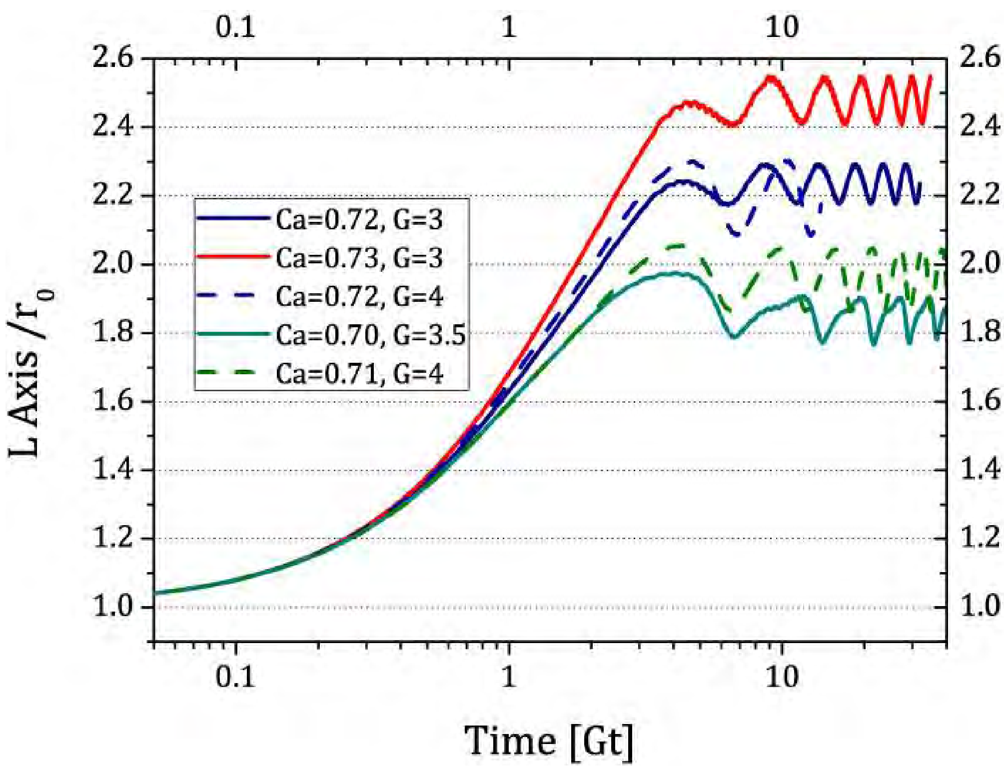

Figure 6.13 *Comparison of the L-axis when capillary number is increased as function of Parameter G, with $\lambda_{\mu} = 0.012$, $\alpha = 0.13$.*

In order to avoid a poor resolution (a distorted mesh) due to a large time step, for these simulations the number of time-steps was set larger, maintaining constant the total time of simulation. These new simulations are carried out with a higher cost of the CPU-time. The new predictions of the steady states of deformations do not show significant differences: A slightly larger deformation, but essentially the same frequency and amplitude for the long-time oscillations. The Hopf bifurcation is observed as was predicted in the latter Section, but a critical deformation is not attained, as was suggested. The steady state is the same in general terms.

A critical deformation is attained when the drop form is no longer able to sustain a steady deformation of the drop, and it keeps on deforming. Unless an unexpected shape becomes dominant, which this is the case, and may provide insight about a Hopf solution not considered earlier. Thus, Figure 6.14 shows the evolution of the W-axis (along the neutral direction of the flow field) for the same experiments. Here, there is a small change in the early overshoot of the W-axis evolution. At long times, it is quite obvious that the length of W-axis remains significant without an obvious change in the steady deformation attained. The drop mass distributes along the z-axis more heavily, with a higher average curvature on the drop, while restraining the elongation of the drop. In contrast, there is only a change of the frequency of the oscillations, as was shown in the latter Section.

The data of Figure 6.14 indicate that the numerical *critical* capillary number may not the same as the experimental value indicates. In this case, I can assume that the numerical method does not represent the same dynamical behavior of previous laboratory experiments. However, if the numerical simulations do not reach the critical deformation, then what will happen if the capillary number is increased by numerical simulation that advance very slowly the simulation time to avoid the distortion of the mesh? These simulations will attain the critical deformation? What is the drop shape at this critical state? To elucidate these questions, a new set of experiments was developed.

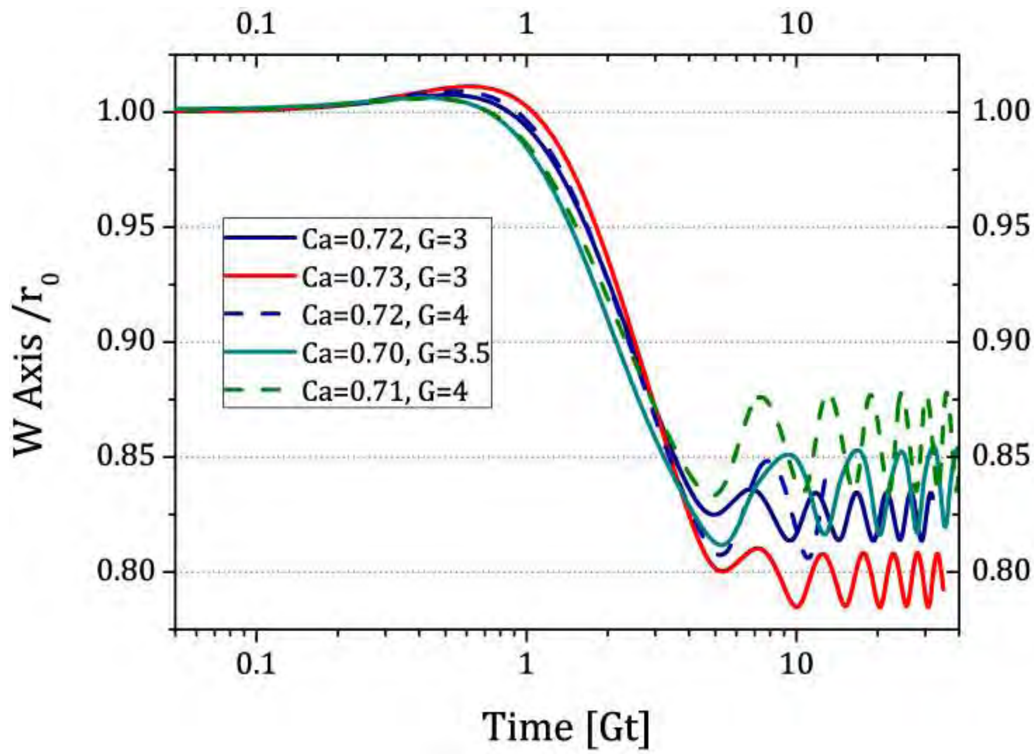

Figure 6.14 *Comparison of the W-axes when capillary number is increased as function of Parameter G, which* $\lambda_{\mu} = 0.012$, $\alpha = 0.13$.

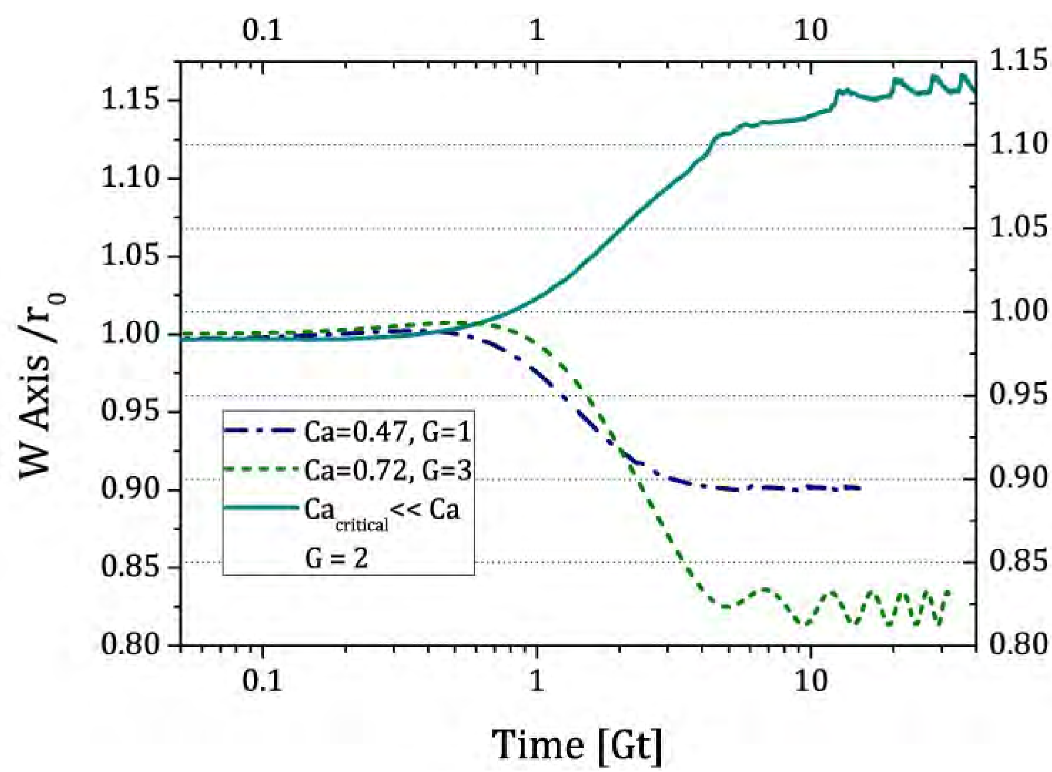

Figure 6.15 *Comparison of the W-axes when* $Ca_{cr} \ll Ca$, *which* $\lambda_{\mu} = 0.012$, $\alpha = 0.13$.

The numerical experiments were carried out by mainly visualizing the evolution of the *W-axis* as a new, relevant behavior. Remembering Fig. 5.3, the evolution of the *W-axis* in a strong flow, $\alpha = 0.13$, and different values of the capillary numbers show small early overshoots in every experiment. These overshoots increase as the capillary number grows. Figure 6.15 shows plots of the *W-axis*, when the capillary number value increases as function of the G parameter; with $\alpha = 0.13$ and $\lambda_\mu = 0.012$. In these experiments, capillary numbers attain values above the critical value. The time-step is smaller than all cases before, to avoid the abrupt distortion of the mesh.

The results indicate an evolution in the overshoot of the *W-axis*. The overshoot always starts at one (spherical equilibrium), then there is an increment in the value of *W-axis* which is above one. At the end of the numerical simulations, the value of the *W-axis* decreased to the final state. However, when the capillary number increases markedly, $Ca \gtrsim Ca_{cr}$, then the overshoot disappears and the normalized values of *W-axis* are always larger than one. In other words, the *cross section of the drop* goes to a new state which is flatter than in previous simulations. Figure 6.16 shows the three principal axes evolution of a drop when $Ca_{cr} \lesssim Ca$, $\alpha = 0.13$ and $\lambda_\mu = 0.012$. Compared with plots in Figure 3.2, the main difference is the behavior of the *W-axis*. The characteristic times are similar to all cases observed in Chapter 5; *L*- and *B-axis* have a similar time-scale, however the *W-axis* presents always a delay. The little oscillations are a consequence of the Hopf-bifurcation mentioned earlier.

Figure 6.17 shows the *Taylor deformation values for all planes of the drop*; for a drop with $Ca = 0.40$, $\alpha = 0.13$ and $\lambda_\mu = 0.012$. The deformation in the (B, W) plane presents an overshoot caused by the evolution of the *W-axis*. Figure 6.18 shows the evolution of Taylor deformations when $Ca_{cr} \lesssim Ca$, $\alpha = 0.13$ and $\lambda_\mu = 0.012$. The deformation on the principal plane, *xy-plane* or (L, B) plane, presents the largest value of deformation. The cross section shows a rather flat deformation, along the neutral direction of the flow; *i.e.*, the (B, W) plane shape presents a very flattened form.

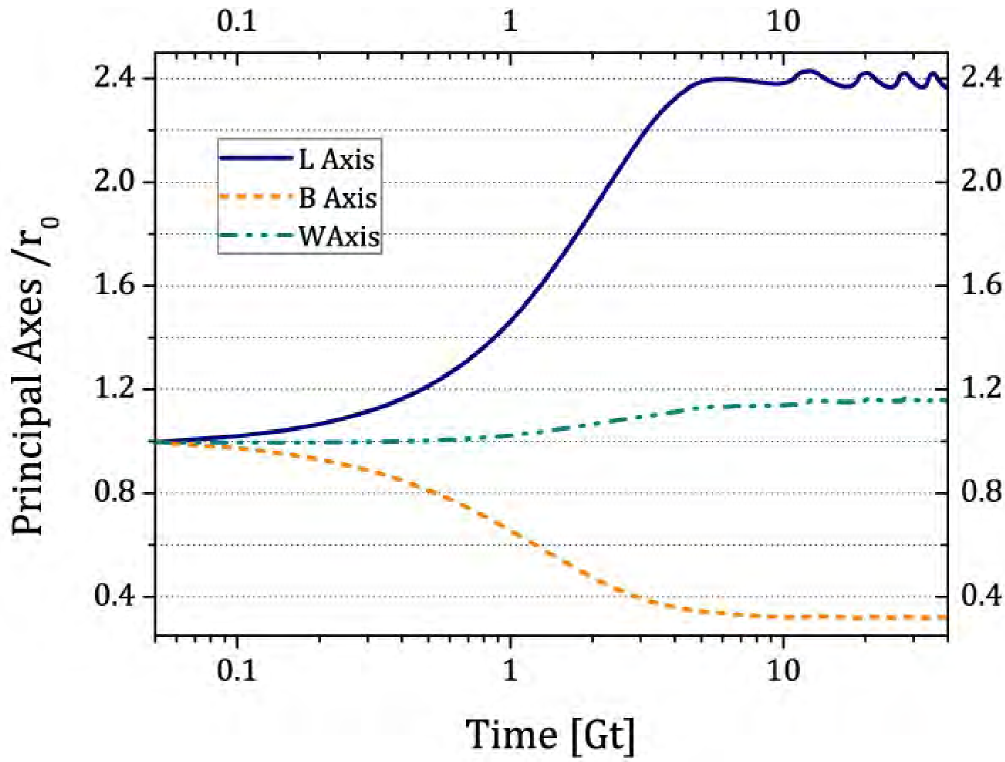

Figure 6.16 *Principal Axes evolution in the time for a drop whit* $Ca_{cr} \ll Ca$, $\lambda_\mu = 0.012$, $\alpha = 0.13$.

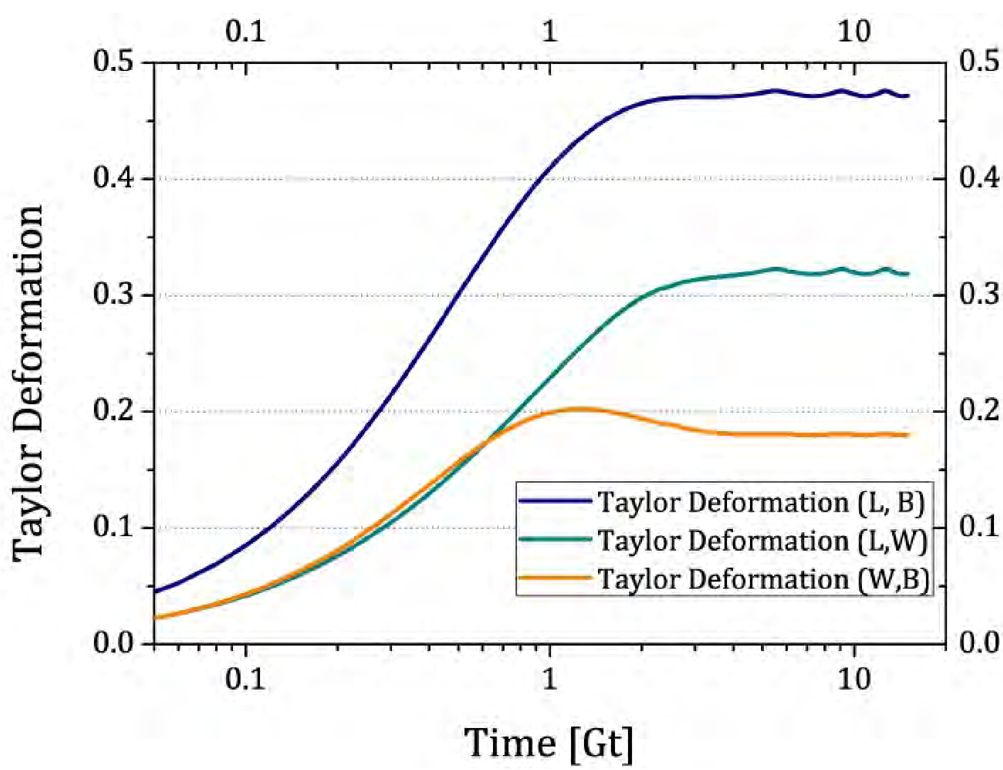

Figure 6.17 *Taylor Deformation in different planes:* $Ca = 0.40$, $\lambda_\mu = 0.012$, $\alpha = 0.13$.

In Chapter 4 and 5, the cross section of deformed drops, by strong flows show a non-circular shape. The shapes were clearly more elliptical than circular, with a cross sectional area less than r_0^2. Thus, drops in steady state for $Ca \leq Ca_{cr}$ are ellipsoids. However, as the value of the capillary number increases, the shape of drops distorts more, away from the ellipsoidal shapes observed for small Ca values.

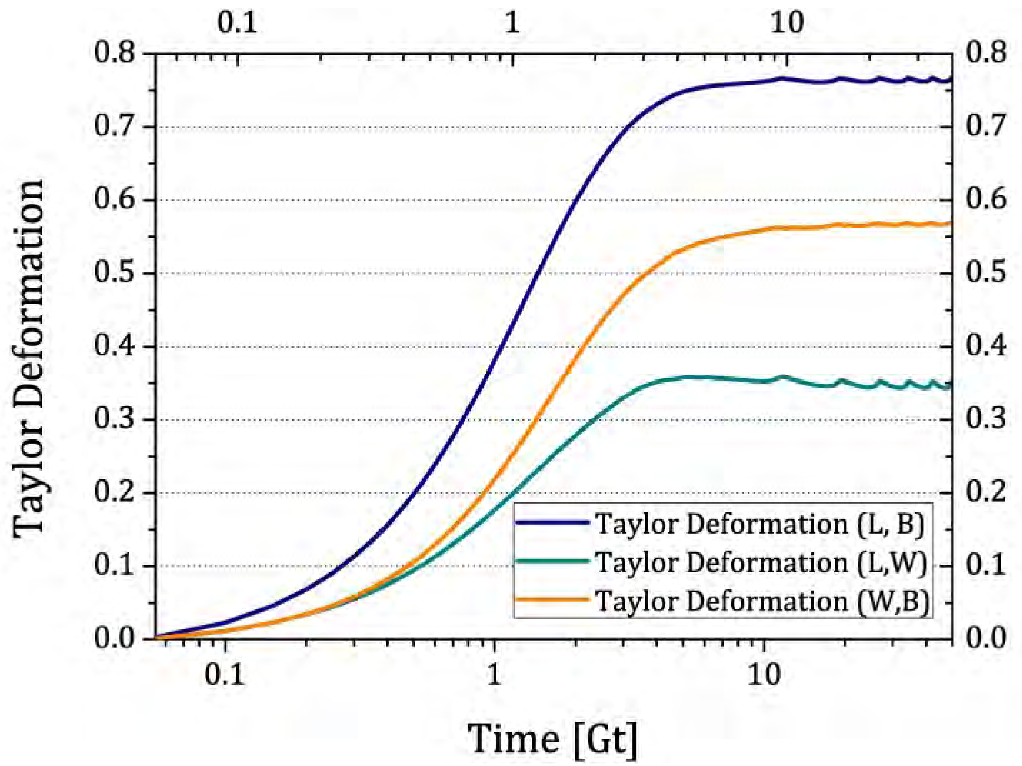

Figure 6.18 *Taylor Deformation whit $Ca_{cr} \ll Ca$, $\lambda_\mu = 0.012$, $\alpha = 0.13$.*

Finally, when the capillary number is large as a consequence of the parameter G, as shown in Fig. 6.18, this kind of deformed drops become to "super *guarache*" — *Guaraches* represent deformed drops with the cross section flattened, the principal axes W- is larger than B-, and the size of W-*axis* may be larger than the initial radius of the drop. Figure 6.19 shows the *super guarache* obtained with $Ca_{cr} \leq Ca$, $\alpha = 0.13$ and $\lambda_\mu = 0.012$. The different views are orthogonal to the principal axes of the drop; the dimensions observed are dimensionless using the initial radius as characteristic length.

For a drop in the regime of $\alpha = 0.13$ and $\lambda_\mu = 0.012$, deformation of *super guaraches* attain approximately the same values as the experimental data reported about the critical deformation (the maximum deformation for a steady state shape for the L- and B-*axis*). However, the capillary numbers are not the same in experimental and numerical experiments. Therefore, there is a need to understand in detail the behavior of critical deformation of drops. However, even though, there is a good approximation by using the BEM3D code for capillary numbers below critical values, there is no good answer to address discrepancies, mainly because experimental data about the *W-axis* is not yet available. This is the crucial piece of information missing as predicted by these numerical results, information that becomes indispensable for a detailed comparison of the numerical and experimental data when the capillary number of a drop is near the critical value.

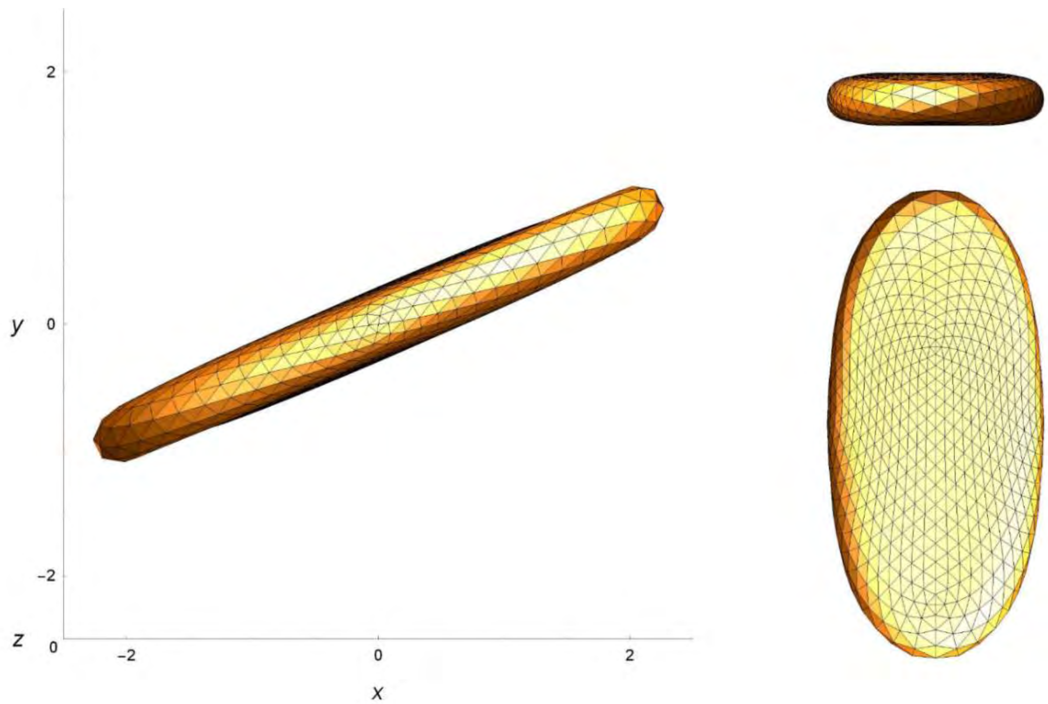

Figure 6.19 *Guarache shape obtained in a drop with $Ca_{cr} \ll Ca$, $\lambda_\mu = 0.012$, $\alpha = 0.13$. There is the different perspectives orthogonal to the principal axes. Orientation Angle $\theta = 23.85°$.*

However, there is a possibly useful perspective on the problem outline above. Considering $\alpha = 0.03$ and $\lambda_\mu = 16$, the drop deformation near critical capillary number exhibits different shapes. In general, for drops with $\lambda_\mu = 16$, all shaper are closer to ellipsoidal shapes and rupture is not observed for most cases, up to the Ca_{cr} . Thus, a set of numerical experiments were made for viscous drops, shown in Fig. 6.20, as possible solutions of a Hopf bifurcation

The difference with the experiments of Chapter 4 was in the values of G. In Chapter 4, G had a value near unity, $G \sim 1$. Now, the experiments were performed with $G \sim 3$. Figure 6.20 shows the polar diagram of orientation angle vs. Taylor deformation. The evolution of the drop has a behavior seen in the experimental results of Rosas *et al*., (Rosas I. Y., 2013). This comparison is for a drop with $Ca = 0.87$. The *trembling effect* provokes a history of deformation of the drop with more oscillations as reported by Rosas. This effect was discovered recently. However, the numerical simulations of these experiments need a lot of CPU-Time, and a detailed analysis of Hopf bifurcation will be part of another study.

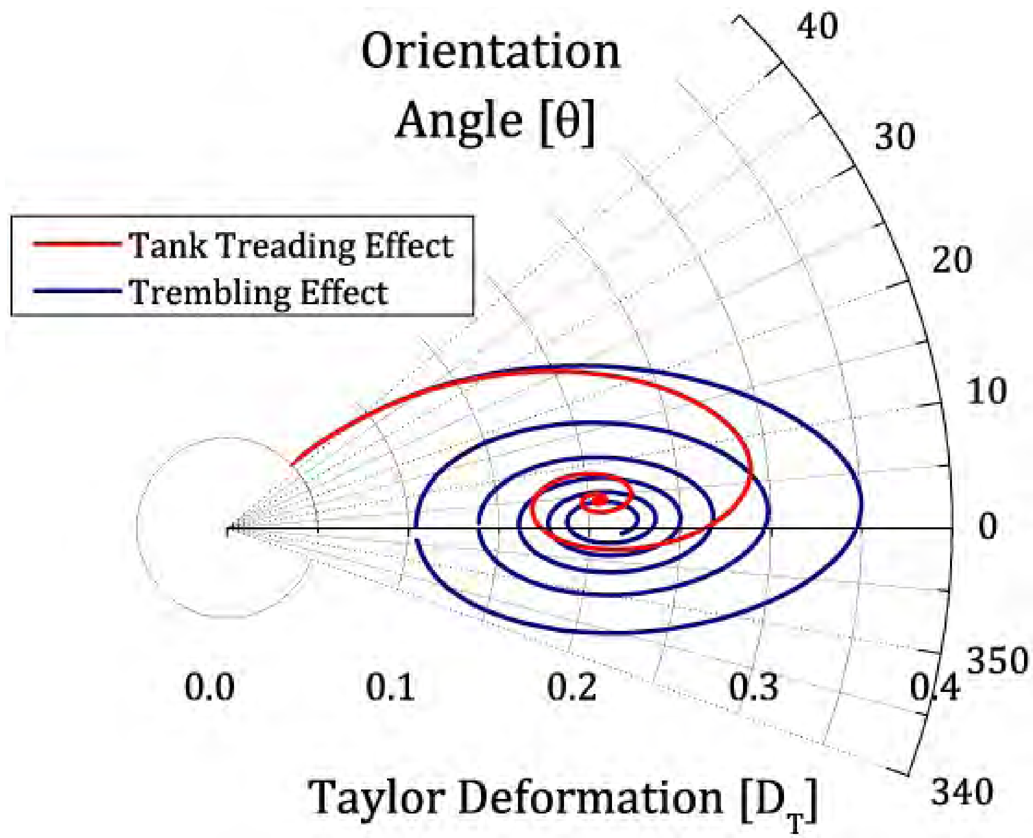

Figure 6.20 *Tank-treading and Trembling effect of a drop with* $Ca = 87$, $\lambda_\mu = 16$ *and* $\alpha = 0.03$.

In this Chapter I present the dependency of the drop deformation as a function of the parameter G (intensity of the rate of deformation tensor (Makosco, 1994) Eq. (1.6)). The numerical vs. the experimental data differences near the critical capillary number was not possible to resolve. However, the evidence for the existence of a Hopf bifurcation in drops should motivate a new type of experimental and numerical studies. In fact, the analysis around the *trembling effect* explains part of the dynamical of the oscillations when the rate of viscosity is high, Fig. 6.20. In general, the oscillatory behavior was previously explained only by the difference of the viscosity ratios of the fluids. It was a surprise to find out that this dependency (of the oscillatory behavior) may depend on G as well.

CHAPTER 7.

The form of a drop immersed in an extensional flow

Extensional fluids are an important branch in fluid mechanics because its kinematics does not include any vorticity. Historically since Taylor's work, (Taylor, 1932) and (Taylor, 1934), studies of drop deformation immersed in another fluid has used mostly *simple shear flows* with a more limited archive of results on elongational flows. In this Chapter, I present results for the simplest case, *the 3D elongational flow*, *i.e.*; the drop will be compressed in two directions while being elongated in the third axis. To study these flows, there is an important advantage for the corresponding theoretical analysis published by Acrivos and Lo, (Acrivos & Lo, 1978). Thus, in this Chapter, a family of steady states of deformation for this flow is shown, with comparisons to Acrivos & Lo model.

In this way, pursuing the above objective, the analysis of *extensional 2D-flows* can now be carried out addressing a different perspective, similar to what Acrivos and Hinch did, (Hinch & Acrivos, 1979). For reasons that will become obvious in the next Chapter, the elongational *2D-* and *3D-flows* together add to a more detailed and consistent understanding of solution for the deformation behavior of drops. However, it is essential to understand first the simpler *3D-flow* for subsequently proceeding to the *2D-flow* presented in Chapter 8.

7.1 Drop deformation in uniaxial flow

Drop deformation in uniaxial (*3D*-) flow is the simplest case of drop deformation to imagine because the drop only elongates. Idealized, as all theoretical and numerical approximations do (Taylor, 1932), (Acrivos & Lo, 1978) and (Spann, Zhao, & Shaqfeh, 2014), the drop remains always fixed in the flow field. Thus, the drop does not rotate, is always in the center of the flow, and the drop simply elongates in one direction as is pushed in from the other two directions. Figure 7.1 shows the evolution of the axes of deformation for a drop in uniaxial flow. The *B- and W-axes* are fully equivalent and show the same behavior (axisymmetric deformation). Figure 7.1, shows that the drop attains a stationary shape, and when the flow is stopped, the drop returns to a spherical shape; *i.e.*, this phenomenon is always observed when Ca is less than the critical capillary number $Ca = 0.31$, for $\lambda_\mu = 0.01$.

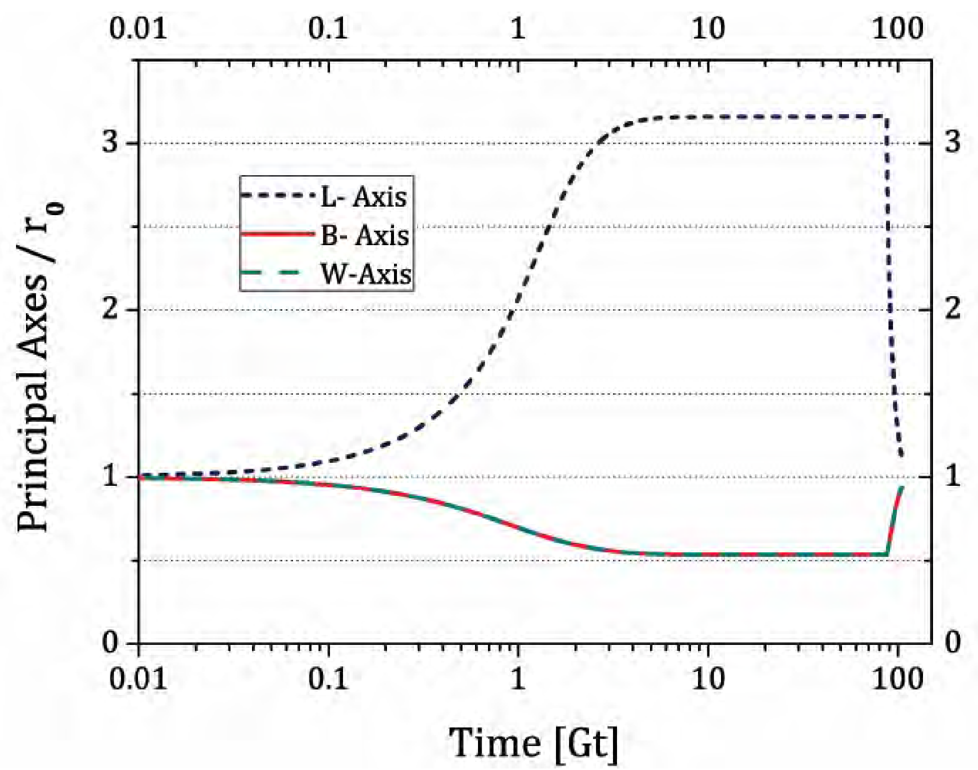

Figure 7.1 *Evolution of principal axes of a drop in uniaxial flow.* $\lambda_\mu = 0.01, Ca = 0.31$

To imagine the deformation of the drop is easy, however trying to solve the problem in theoretical form is not. The next part is a summary of (Acrivos & Lo, 1978) model and its analytical results. To simplify the fluid dynamics of drop deformation in uniaxial flow, their working assumptions are based on an axisymmetric shape for the drop, as well as assuming a known solution for the flow inside, outside and on the surface of the drop. As most numerical methods do, Acrivos and Lo solved first the form of the surface of the drop. The flow outside the drop at infinity is the uniaxial flow. Inside the drop, Acrivos and Lo set the viscosity of the fluid being lower than the continuum phase, so $\lambda_\mu << 1$; assuming the opposite does not seem reasonable for the possible deformation will be weak.

Acrivos and Lo insight is based upon the value of these approximations to determine a family of solutions of the steady state of deformation of drops when a uniaxial flow was applied. This assumption solves the problem inside the drop because the flow inside the drop is essentially due to the hyperbolas generated by the imposed flow. With this simplification, Acrivos and Lo obtained the configuration of the shape of the surface of the drop.

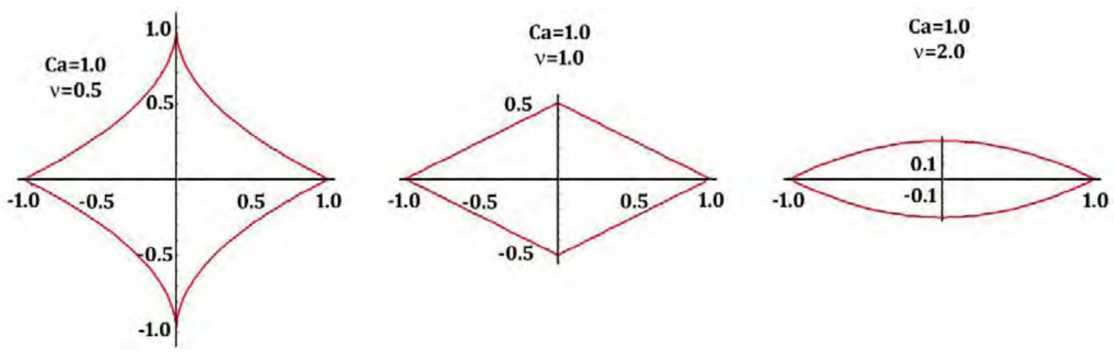

***Figure 7.2** Shapes of steady drop deformation, football balloon predicted by Acrivos and Lo.*

This analytical drop-form solution is plotted in Figure 7.2. The analytical solution is a function of the length of the axis of elongation (z in this case) and the ν parameter (parameter which is a function of pressure of the drop, capillary number and intensity of the flow). The interface shape is given by

$$R(z) = (2\nu)^{-1}(1 - |z|^\nu). \tag{7.1}$$

If the value of ν is small, $0 < \nu < 1$, the configuration of the drop is similar to the hyperbolic configuration of the flow. If $\nu = 2$, the shape form becomes a paraboloid by the term $|z|^\nu$. For these forms, steady states of deformation of drops are like a common football. There are many solutions if $2 < \nu$, but the parabolic shape obtained changes very little. Then with the shape of the stationary state the flow inside and outside the drop are obtained by an analogous algorithm to that of numerical methods.

Finally, the assumption of $\lambda_\mu = 0$ is taken, and a diagram of the deformation of the drop and the capillary number is obtained. The difference between the diagrams shown in Fig. 3.5, 3.7 and 4.5 is the employed value of λ_μ in those diagrams. My numerical method uses values of the ratio of viscosity $\lambda_\mu = 0.01$. Here, λ_μ is low, but finite; *i.e.*, the assumption of Acrivos and Lo $\lambda_\mu = 0$ will provoke a different behavior in the same diagram. With this idea, the curve predicted by Acrivos and Lo of deformation was compared with the numerical data. Both diagrams are shown in Figure 7.3.

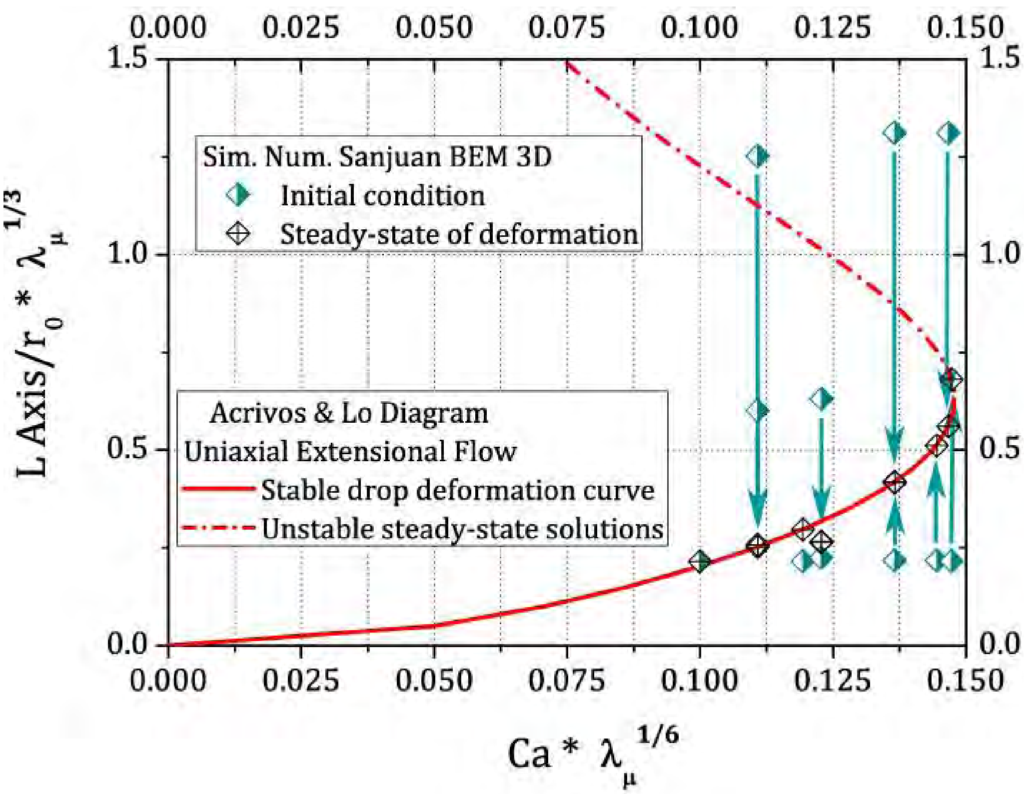

Figure 7.3 *Diagram of Steady State of deformation in extensional 3D-flow made by Acrivos and Lo (lines, $\lambda_\mu = 0$) and the numerical experiments (marks $\lambda_\mu = 0.01$).*

The theoretical solution of Acrivos & Lo was obtained applying an *asymptotic analysis* around the elongated axis of the drop in the *slender body theory*. Their results are plotted in Fig. 7.3 and presents two branches of solutions The first, *the continuous line* corresponds to the steady state of deformation for a drop in a steady uniaxial flow, with a drop shape predicted to be stable. The second, *dashed line*, is an unstable solution. Physically, for an inviscid drop immersed in a uniaxial flow, the shapes that drops will attain resemble a continuous line: an infinitely slender body. If there is no flow, the ratio $L - \text{axis}/r_0$ is 1. If the conditions of equilibrium of stresses are attained, the ratio $L - \text{axis}/r_0$ must have values greater than 1, because it is an indication of the deformation of the drop. Employing the relation between $(L - \text{axis}/r_0) * \lambda_\mu{}^{1/3}$ and $\text{Ca}* \lambda_\mu{}^{1/6}$ it is possible to obtain the equilibrium conditions for a drop with $L - \text{axis}/r_0$ of value near 1 and a small but finite strength of the flow applied. When the viscosity is small, i. e., of $\lambda_\mu = 0.01$, the equilibrium deformation is the first mark in Fig. 7.3; corresponding to $\text{Ca}* \lambda_\mu{}^{1/6} = 0.1$. That is, when $\lambda_\mu = 0.01$, stable steady solution of drop deformation will be observable from this value to the limit at 0.148, i. e., $0.1 \leq Ca * \lambda_\mu{}^{1/6} < 0.148$.

With this information, a series of numerical simulations were carried out. In Figure 7.3, green markers correspond to the initial elongation (arbitrary) conditions of the simulations. Black markers are the long term steady state of deformation that the numerical simulation attained. In the numerical experiments here reported, the initial conditions considered two cases. The first, the more natural experiment, in the sense that the initial conditions were those of the drop at rest (spherical shape with $(L - \text{axis}) \,/\, r_0 \; = 1$) evolving until reaching a steady state of deformation. The second type of experiments were analogous to those used by Stone when studying *extensional 2D-flows*, (Stone & Leal, 1989b), beginning with an elongated drop.

Stone drop shapes are those reached after a steady flow with a capillary number near the critical value. Afterwards, a second phase is applied (with a flow with half of the previous *Ca* value), and monitors how the drop goes to a second steady state or one of critical behavior or breakup (even though the capillary number is lower, the initial elongation for this second phase of the flow history is larger, with respect to the new capillary number). The idea here is using initial values of the drop elongation that correspond to positions on the plot of Fig. 7.3 away and above the stable line, and observe if drops elongation evolves to a steady state of deformation described by Acrivos and Lo, or if there are drops that evolve toward an unstable state —the second branch solution, with an eventual rupture. Figure 7.3 shows all initial deformations as green markers predicting that it does not matters the initial value, drops will always go to the steady state condition (black markers, directly below).

Figure 7.4 is a comparison of the steady shape of a drop in uniaxial flow. The left image is the theoretical results of Acrivos & Lo, employing Eq. 7.1 to obtain the shape of the drop with $\nu = 2$, $Ca = 0.26$, and $L - a$xis $= 1.375$. The predicted shape by numerical computations is presented in the right-hand part, with the same capillary number and $\lambda_\mu = 0.01$.

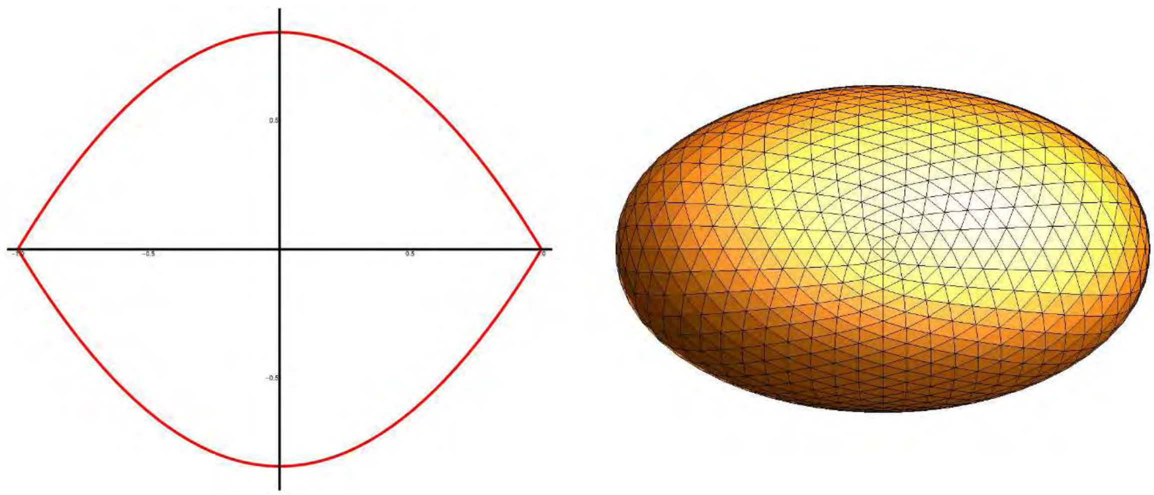

__Figure 7.4__ Comparison of a drop in uniaxial flow in steady shape. Theoretical shape by Acrivos & Lo, left; and BEM3D, rigth. $Ca = 0.26$ and $\lambda_\mu = 0.01$.

Figure 7.4 shows that drop shape differences are obvious, in particular Taylor's deformation measures. Both predictions are similar in the length of L-axis, however, the local curvature of the drop is not. The deformation attained with BEM3D models is always higher than predictions by Acrivos & Lo. Also, the theoretical shape has always an end pinch. This pinch configuration is unstable because its local curvature tends to zero. Acrivos and Lo made that reference. In the case of a finite λ_μ, the end pinch must disappear and we recover the solution as numerical data indicates. These pointed ends in the theoretical shape are never present in the numerical shape. As the ratio of viscosity increases, differences also increase. For example, by changing the parameter ν to $\nu = 2.5$, the new shape obtained is similar in global form to that of the BEM3D model. HoweverThe drop forms shown in Fig. 7.4 correspond to the shapes of the second black mark from left to right in Fig. 7.3. The pointed drop shape has been observed in many experiments where the local interfacial tension is dependent on the surface position. So, when a flow is applied, the high mobility of the tensoactive agents cause a concentration of surfactants mostly at the ends of the drop, reducing drastically the local surface tension, thus, inducing high curvature tips. Here, the surface tension value is fixed in these numerical simulations, and there is no gradient of concentration of surfactant (no variation of the value of surface tension) on the surface of the drop. Hence, a smother variation of curvatures is the only possible shape solution.

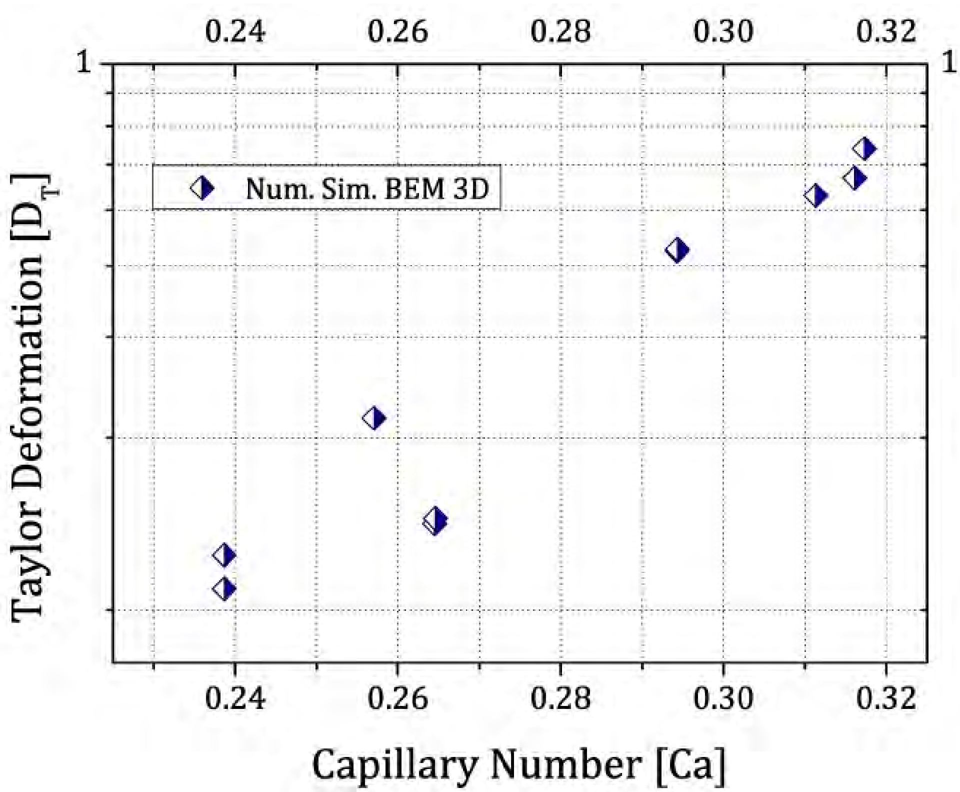

Figure 7.5 *Taylor Deformation vs Capillary number for a drop in extensional flow 3D.*

Figure 7.5 shows the conventional diagram of Taylor Deformation-*vs*-Capillary number. However, simulations show that Taylor deformation is very small for a weak value of the capillary number; that is, simulations indicate that a remnant flow —with a nonzero value of *G*, which appears in the capillary value— predict a quasi-spherical drop. As the *Ca* increases beyond the value of 0.32, the steady state shape deformation is higher. The relationship of deformation vs. flow is not linear and similar to the curved shape when the critical deformation is attained, as shown in Fig. 7.3.

An important observation of Figure 7.3 and 7.5 corresponds to the last two capillary number shown. In Fig. 7.5, the last value has the biggest deformation attained, with a steady shape. In Fig. 7.3, the steady state solution for this capillary number, corresponds to the first marker from right to left. The steady state attained is on the unstable theoretical solution. This result is consistent with the fact that theoretical solution is for inviscid drops, $\lambda_\mu = 0$, and the solution may be valid as well for small values of λ_μ. However, if λ_μ increases, the approximation will tends to fail. The analysis in Fig. 7.3 indicates that for weak deformations the theoretical prediction is only approximated. However, for higher ratios of viscosity, the drop shapes at the extremes ought to differ. Regarding again Fig. 7.3, the same analysis for uniaxial flow implies that, up to $\lambda_\mu = 0.7$ steady shapes do not exist —for $Ca * {\lambda_\mu}^{1/3} > 0.148$, and having assumed $((L - \text{axis}) \,/\, r_0) = 1)$. With these data, an equivalent Acrivos and Lo diagram for large viscosities ratios would indicate that there is no equilibrium shape for a drop; it does not matter if the drop is a sphere, an analytical solution does not exist. That idea appears to contradict the fact that there exists a steady state of deformation in *2D-extensional flows* as was shown in Chapter 2. *2D-extensional flow* is a special kind of *strong flow*, but for low capillary numbers and leaving out effects due to noncircular cross-section of the drop, the existence of (numerical) steady shapes may imply, in general, the existence of steady states for *uniaxial flows*. Figure 7.6 shows the predicted (numerical) Taylor deformation in *3D-extensional flows*, for $\lambda_\mu = 1.0$ and $Ca = 0.1$. The plot shows how the steady state is attained in its principal axes.

Chapter 4 shows as well there exists the possibility of steady shapes for high ratios of viscosity. So, here I study the evolution of diagram Acrivos & Lo for systems with a large ratio of viscosities. Theoretically, the study for viscous drops immersed in a *uniaxial extensional flow* can be treated analogously to that of Acrivos and Lo. However, in this new approximation the inside flow must resemble that observed when using the numerical method. Thus, the inner flow can no longer be mainly shaped by hyperbola streamlines, as predicated by Acrivos analysis, where the important underlying assumption to obtain their diagram is an inviscid drop: $\lambda_\mu \ll 1$. So, for steady state flows, there cannot exist inner-flow hyperbolas. Even more, there is a strong likelihood that the inner flow may be characterized by two toroidal vortices, aligned both with the axis of deformation, and with a skin flow from the waist toward the ends of the drop. These streamlines are a consequence of balancing stresses generated by the outer-flow and the interfacial stresses of the drop that maintain an equilibrium shape.

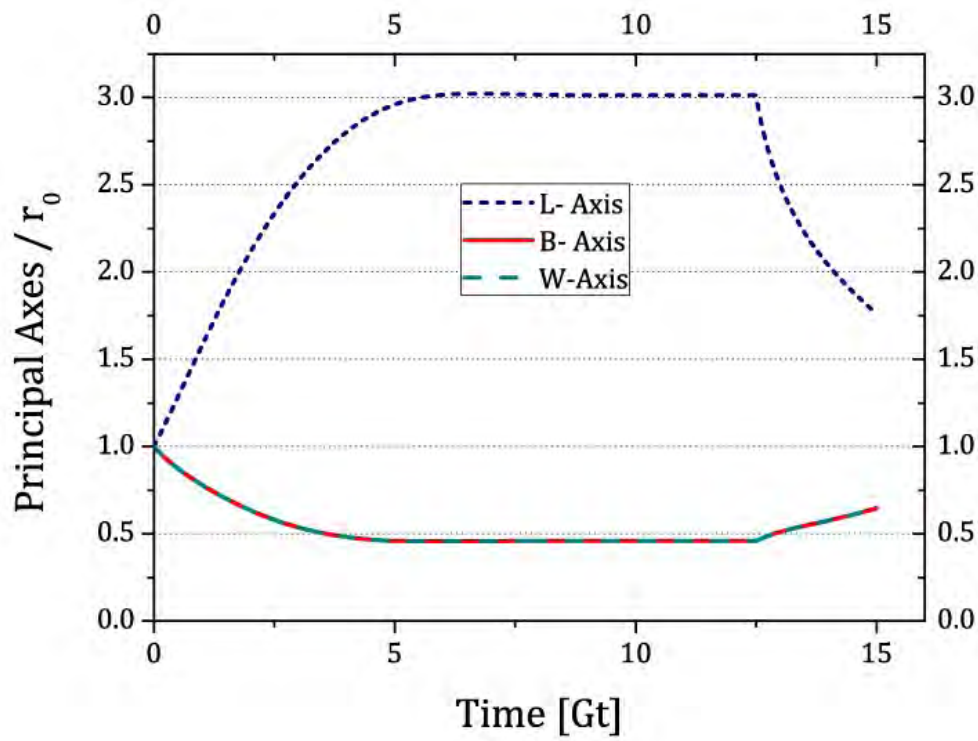

Figure 7.6 *Steady state attain by a drop in extensional 3D-flow with* $\lambda_\mu = 1.0$ *and* $Ca = 0.1$*.*

Today, streamlines of the flow inside the drop are a mystery; there are no published theoretical nor experimental data to visualize the velocity field inside the drop. Nor there are numerical data, because it is necessary to develop specific extension of the numerical codes to evaluate the pressure fields and subsequently the velocity field of the inner fluid: Eq. 1.22 c). In the future, I will implement this extension of the method to show the inner flow of the drop. With this information, the Acrivos and Lo analysis may present a new, or multiple branches, in their diagram, and *uniaxial extensional flows* will be understood in greater detail.

7.2 Drop deformation in extensional 2-D flow

The archive of experiments of drop deformations under regimes of *extensional flows* is at an early stage, mostly limited to *2D-extensional flows*. Since Taylor work (Taylor, 1934), the cross-section of the drop characteristics has been poorly studied. The optical line of sight for most of the available experimental devices did not permit a thorough study neither of the evolution of the cross-section of the drop, nor the flow characteristics inside the drop. And there is not a clear idea of the consequences of this lack of information in the phenomenon of drop deformation.

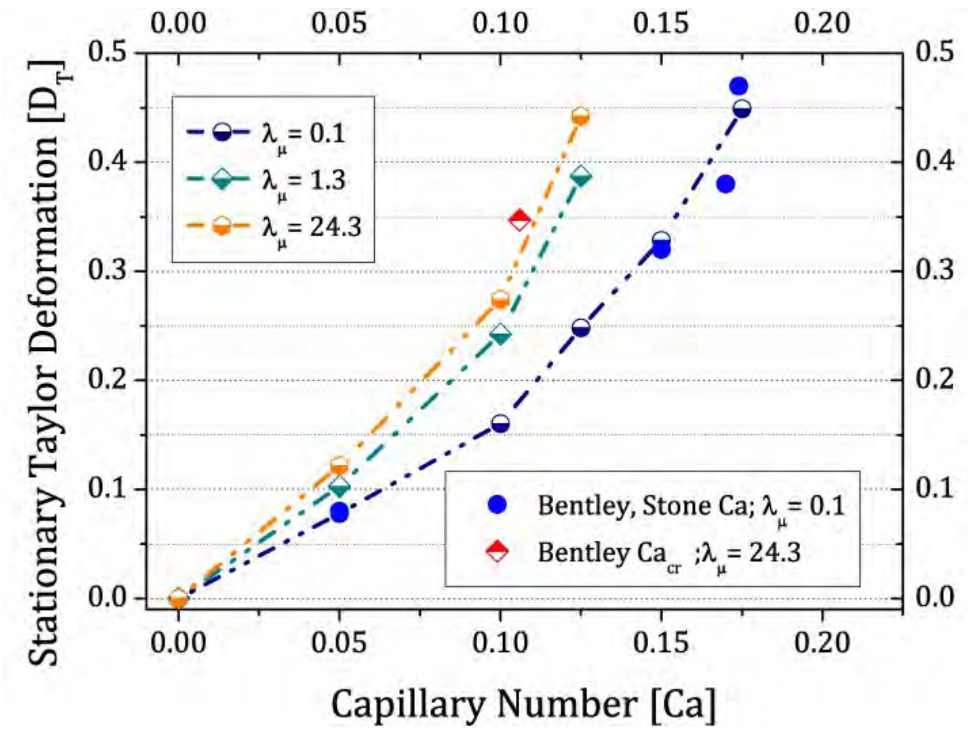

Figure 7.7 *Stationary deformation vs capillary numbers with different values of λ_μ for extensional 2D-flow. Comparison with exp. data of Ca_{cr}*

Here I attempt expanding our knowledge of the now ever-more-clear *3D*-character of drop deformations, even for the simplest of flows. Since the calibration of the method is done using data obtained with *simple shear flows*, the numerical method unambiguously predicts a non-circular shape in the cross section in the steady state attained by the drop for non-zero *Ca*. In Chapter 3, I show how the ellipsoidal shape of the cross-section of the drop drifts away from the circular shape as the parameter α increases, even for the same (small) capillary number; see Fig. 3.4. For these cases, the ratio between the *B- and W-axis* decreases more for the cases of

$\alpha = 0.13$. In this Chapter, $\alpha = 1.0$ in order to study the cross-section in *extensional 2D-flow*.

Figure 7.7 shows the comparison of numerical data vs. experimental data for the ratio of *B- vs. W-axis*, and for different values of λ_μ, in *extensional 2D-flows*. The experiments of drop deformation correspond to data reported by Stone and Leal, (Stone & Leal, 1989b) and Ha and Leal, (Ha & Leal, 2001). As was commented in Chapter 2, Ca_{cr} values are small in every case.

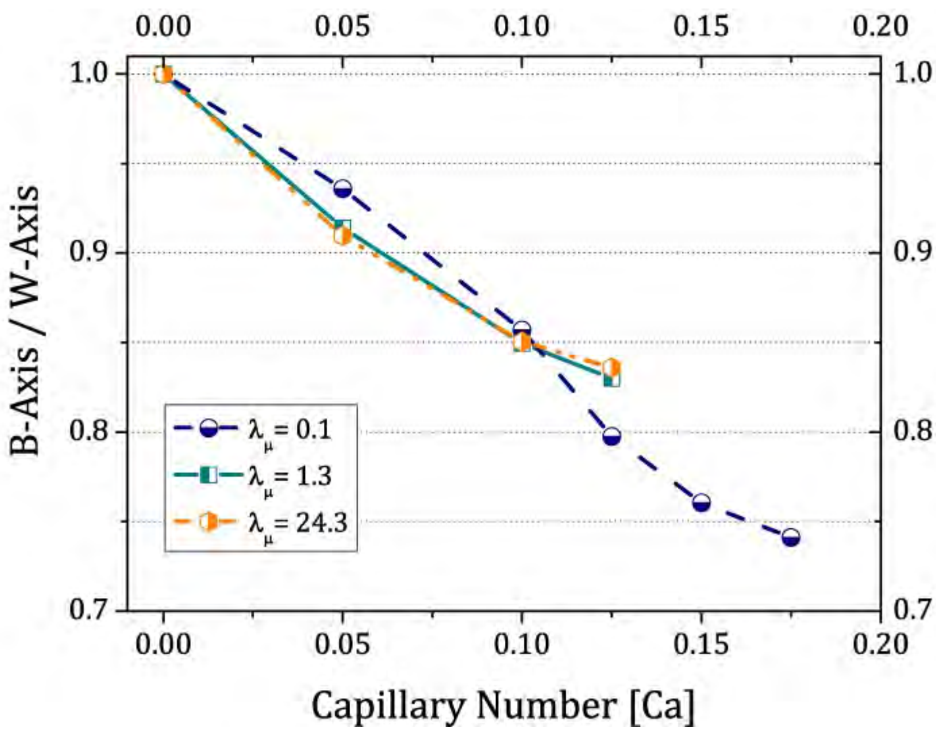

Figure 7.8 *Ratio of B- wrt. W-axis for different values of λ_μ in pure extensional 2D-flow axis.*

Figure 7.8 shows the cross-section dimensions vs. *Ca* number when the drop deformation *is attained near the critical capillary number*. Albeit the limited experimental data for these conditions —it is not vast, as compared to that of *simple shear flows*, as was commented in Chapter 2— the dependence of the cross-section dimensions vs. *Ca* number— firstly indicate that values of the ratio of these axes lengths are similar, regardless of the ratio of viscosity of the drop. As shown in the previous Section, the analytical approximation assumes an axisymmetric shape, equivalent to those induced by *uniaxial flows*. However, as the value of the flow parameter α increases (the flow character becomes more elongational, while simultaneously losing vorticity) the ratio between the *B- and W-axis* becomes smaller, away from one; *i.e.*, the cross-section is less circular as the flow becomes more extensional, as long as it maintains the *2D-flow* character.

Figure 7.9 shows the behavior of the principal axes of the drop for different viscosities ratios in *extensional 2D-flow*. In Chapter 2 and 3, the plots of the principal axes in a *2D-flow* were linear for small capillary numbers: see Figs. 2.2 and 3.8. In this case, Fig. 7.9, trajectories are clearly non-linear.

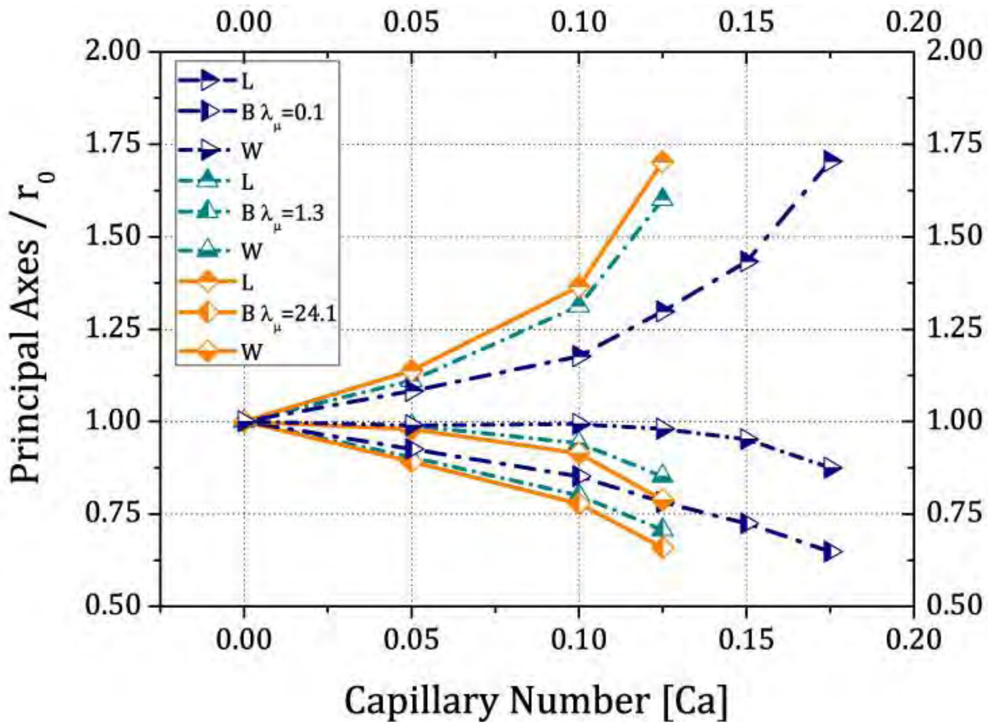

Figure 7.9 *Lengths (normalized) of the principal axes of drops under stationary extensional 2D-flow with* $\alpha = 1.0$*, and* $\lambda_\mu = 0.1$*, 1.3 and* 24.1*, for different values of Ca.*

Using Figure 7.8 and Fig. 7.9, a comparison can be made between deformations attained for a drop with essentially the same flow conditions, -*uniaxial extensional flow*, Fig. 7.6 and the *extensional 2D-flows*. Even for the case with most similar deformation lengths: $Ca = 0.1$, $\lambda_\mu = 1.0, 1.3$; differences appear to be meaningful. Here the stationary state deformations attained are completely different for the same capillary number. Clearly, the rate of deformation is the same, but the deformation attained is not. That result may indicate as well the differences due to an applied *2D-flow* versus the dynamics under a *3D-flow*. In Chapter 8, I present a series of numerical experiments attempting to elucidate the drop deformation differences observed when the flow kinematics goes from a *2D-extensional flow* to *uniaxial flow*.

CHAPTER 8.

Hysteresis behavior in drop deformation

In Chapter 7, I present a study of drop deformation in *purely (3-dimensional) extensional flows,* given emphasis to a comparison of theoretical ideas (those of Acrivos and Lo) vs. the numerical experiments here presented. Even though the model is quite simple, it is powerful enough to predict multiple solutions for the deformation of drops subjected to this flow history. But this is not the only elongational flow without vorticity. When studying planar elongational flows, the vorticity is contained along the third direction, varying from the maximum observed with the *simple shear flow* case to a fully *hyperbolic flow without vorticity*. It is using these planar flows that the non-symmetric drop shapes are observed in a systematic manner. Thus, the question addressed in this Chapter is how possible transitions of the solution occur from a purely axis-symmetric highly elongated shape that occurs in a *3D*-elongational flow to a non-symmetric drop form prevalent in all *2D-extensional flow*.

After the study of Acrivos and Lo, Hinch and Acrivos (Hinch & Acrivos, 1980) published a theoretical study of the drop deformation induced by *2D*-elongational flows using techniques similar to that previously used for purely *extensional flow*. They find that there are multiple solutions for the degree of deformation (induced by a steady flow) with a drop shape vs. deformation behavior similar to that displayed by Fig. 7.3, which predicts a *S-shape solution (a stable and an unstable branches)*. The method used by Hinch and Acrivos to prove this space is based on changing in time

the flow structure (the values that define the flow character) from a *uniaxial extensional* flow to a *2D-elongational flow*, and monitoring the evolution of the steady shapes of drop deformations.

For the *extensional 3D-flows* case, the basic assumption made by Acrivos and Lo was that of an inviscid drop. For the case of *simple shear flows*, the analysis is for inviscid drops and the assumption of axis-symmetric behavior of the drop are made. As shown in Chapter 2, drops in *simple shear flow* are not axis-symmetric. In Chapter 3, 5 and 6, the cross-section of the drop was not circular, for cases with $\lambda_\mu = 0.01$ and for flows near simple shear flow. With these facts, the numerical model will hardly be symmetric, thus should be different from the theoretical model proposed by Hinch and Acrivos. Now, a larger set of possible conditions (assumptions) for the numerical experiments can be used, which may show a complementary set of steady state shapes for the drop not predicted by the theoretical model. In particular, a large range of viscosity values implies a larger set of possible behaviors for the curves predicted theoretically.

Making a comparison as done in Chapter 7 for extensional *3D-flows* is not yet possible. However, the transition among the curves which describe the deformation of drop in different flows must exist. In this Chapter, a simplified model of this transition is presented. The main difficulty with addressing the full range of ideas associated to this problem is due to the multi-parameters involved: the type of flow parameter α, the ratio of viscosities, or *the intensity of the flow G*, all causes different effects on the dynamics of the drop deformation —as shown in Chapter 6. So, the family of curves describing all possible steady state drop shapes in a flow should be studied carefully.

The first step to analyze the transition of these curves is to consider the simplest case, as Hinch and Acrivos did, using a *2D-extensional flow*, (Hinch & Acrivos, 1979). So, they studied the transition from uniaxial flow (*3D-flow*) until reaching the kinematics of a planar flow (*extensional 2D-flow*). Their main idea is supplemented here by considering now numerically a larger class of flows: swepping —varying the planar flow *parameter α*— from *simple shear flow* (maximum vorticity) to a pure *2D-elongational* flow (without vorticity along the neutral direction) and subsequently evolving toward a pure elongational *3D-flow* (later, a new *elongational flow parameter* is used: β, which augments the compressive character of the flow along the neutral direction). In this way, I expect to study possible drop deformations solutions that may occur when varying the kinematics from shearing flows to a purely hyperbolic *3D-flow*.

The family of flows prescribed by Eq. (1.9), covers the full range of *plane flows* and the full class of drop deformation induced by strong *2D-flows*. However, there exists a much larger class of *elongational flows*, in particular, here I emphasize flows without vorticity that go from an *extensional 2D-* to *uniaxial 3D-flow*. This is a completely new class of deformation and kinematics of deformation and therefore is wise to consider these results as preliminary, mainly because a lot of information is lacking and many more studies may be needed to better understand this topic in drop deformations. However, observations in drop deformation show effects, which were not taken into count in neither the earlier portion of this work nor any other published results. With this caveat, I present my results that I consider the most solid qualitatively.

8.1 Hysteresis in strong flows

When multiple solutions are present, such as those presented in Fig. 7.3, a special type of hysteretic effect can be observed as a result of a *S*-shape solution space. That is, whenever a deformation is induced beyond the critical value, a branch jump is possible. Subsequently, attempting the reduce the drop deformation by reducing the flow field strength, what most frequently occurs is a return trajectory (along the second branch) different from the initial one, in a similar way to the hysteresis loops observed in ferro-magnetism.

As mentioned before, the study of the hysteresis in strong flows requires the use of a family of flows, such as those applied before, with different values of α. Thus, in order to prove different solution branches, the class of flow must be modified by changing the intensity of the rate of deformation tensor as a function of the type parameter α: using Eq. 1.9. So, the parameter G can only be a function of α; *i.e.*, normalizing the intensity of the rate of deformation tensor is done by varying the magnitude of its second invariant: the expression of Eq. 1.5. With this idea, a drop shape reached with a given capillary number may be subsequently deformed by varying the flow parameter from zero to one (from *simple shear flow* to fully *extensional 2D-flow*) within a class of flows.

The next numerical experiments were performed using three stages. By a first phase, applying a *simple shear flow* to the continuum phase, and waiting for the drop to attain the steady state of deformation. The next phase, characterized by a continuous sequence of flows, requires changing the parameter α from zero to one while holding *Ca* constant. Subsequently, the third phase, requires α to be varied from one to zero. The evolution of α in time is shown in Fig. 8.1. The simulation time is plotted in the abscissa axis. In order to guarantee attaining the steady shape in *shear flow*, 20% of total time was used to attain the steady state.

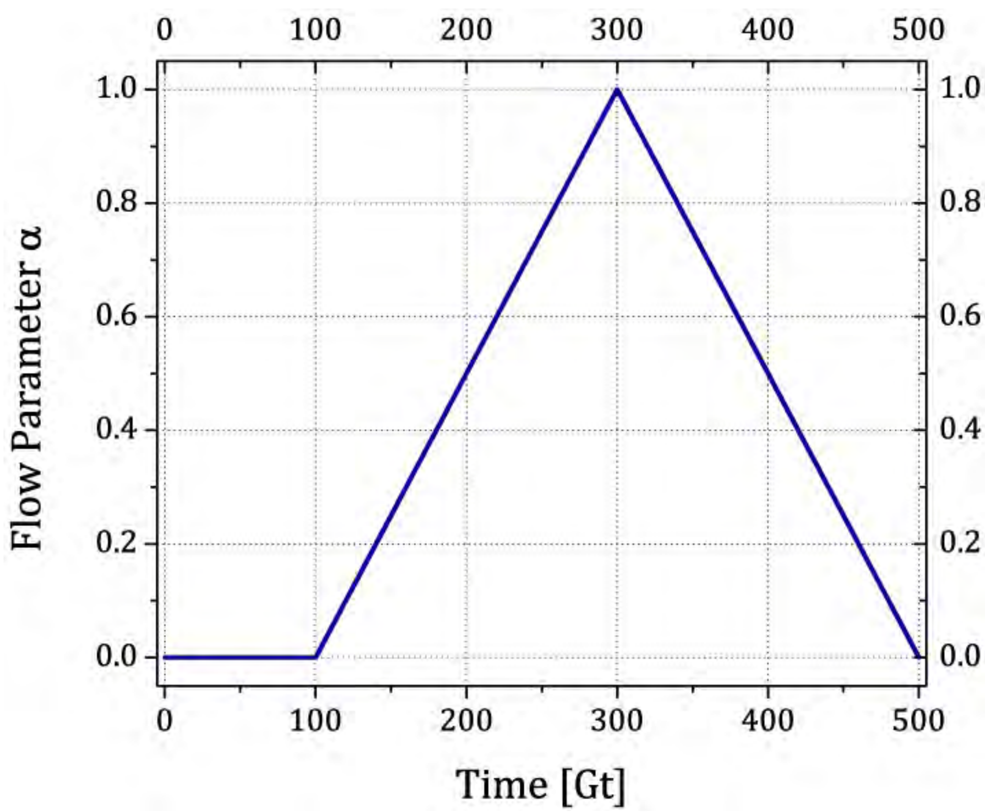

Figure 8.1 *Evolution of the flow parameter α in time for a numerical simulation of a flow induced hysteretic loop. The first portion of the flow corresponds to a simple shear flow. At Gt = 100, the parameter α evolves as a ramp up and down; pure elongational flow occurs at $\alpha = 1$..*

After the initial phase, the evolution of the drop deformation is monitored. Figure 8.2 presents the evolution in time for all three axes of the drop, with a capillary number $Ca = 0.20$ and $\lambda_\mu = 16$. Figure 8.2 shows the values of lengths scales for the principal axes attained at the steady states similar to the results presented in Chap. 4.

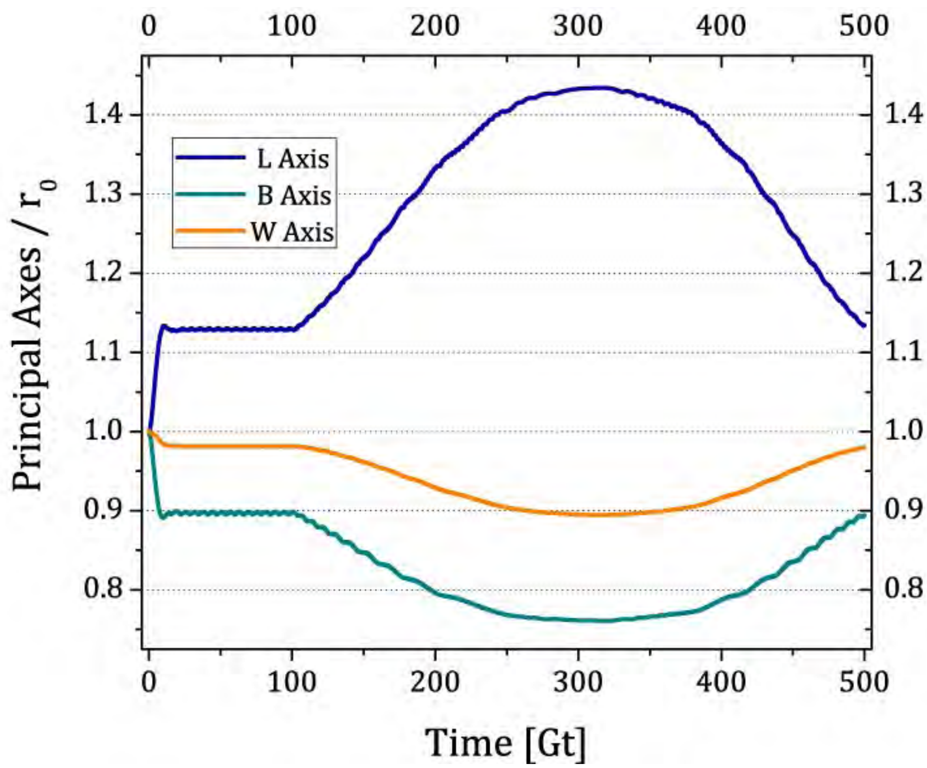

Figure 8.2 *Evolution of principal axes length scales vs time; i.e., vs the parameter α. Time $=$ 500 [Gt], $\lambda_\mu = 16$, from simple shear flow $\alpha = 0$ to extensional 2D-flow $\alpha = 1$. Time:* $100 < Gt < 500$.

Remembering the fact that the drop has a larger viscosity than the continuum fluid, an oscillatory behavior is observed before attaining the steady state. At time $Gt = 100$, the rumping of the *parameter* α begins, with the drop deformation and orientations reached while $\alpha = 0$. Fig. 8.2 shows that after $Gt = 100$ the time evolution of the principal axes is characterized by a parabolic trajectory, until the end of the simulation. The initial value of the principal axes is the same at the beginning and the end of the ramp.

The effect of the parameter α on drop deformations is analyzed in Fig. 8.3 by plotting the lengths of the principal axes versus the parameter α. Assuming the parabolic behavior shown the Fig. 8.2, the expected behavior would be a symmetric curve, describing the deformation along the axes when the α flow parameter changed. However, the data in Fig. 8.3 indicates a curved more complicated because for each value of α, there are two possible steady shapes of the drop, one is due to the first part of the ramp and the second when the flows go from *extensional 2D-flow* to *simple shear flow*. In all the cases, the axes developed two different states for the same value of α.

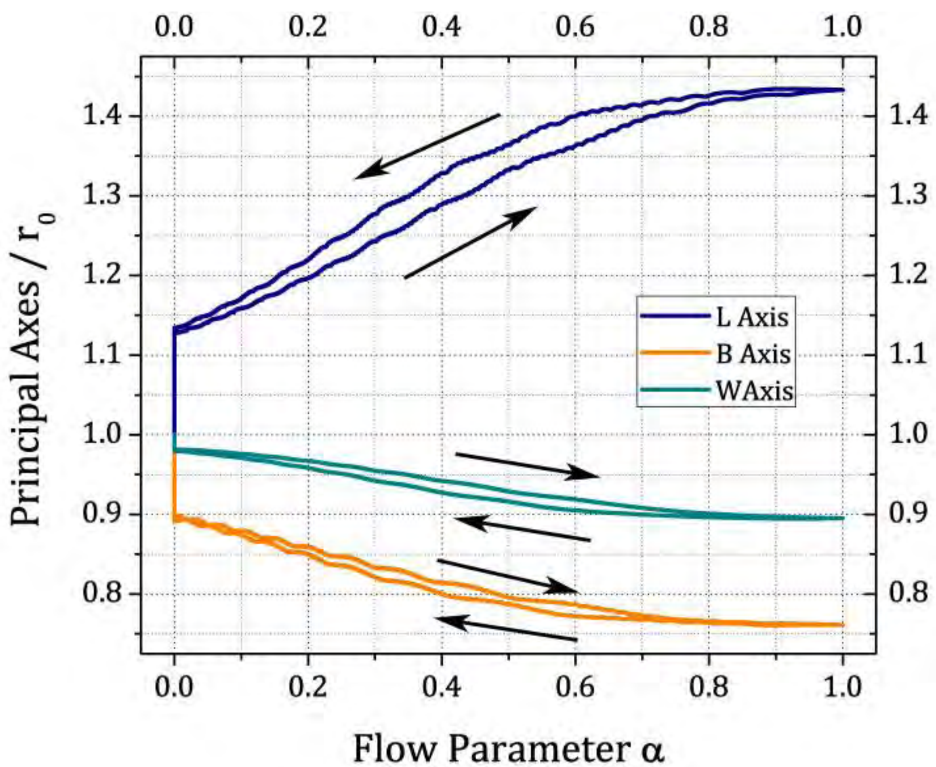

Figure 8.3 *Principal axes vs flow parameter alpha* α*. With* $\lambda_\mu = 16$*, from simple shear flow* $\alpha = 0$ *to extensional 2D-flow* $\alpha = 1$*. Time:* $100 < Gt < 500$*.*

Figure 8.3 shows the behavior of all axes with respect to the flow parameter. Arrows of this Figure indicate the applied ramp direction used and the deformation attain with; see Fig. 8.1. In the third phase flows, decreasing the ramp until the flow reaches *simple shear flow* kinematics, the value of the *L-axis* is larger than the value attain in first portion of the ramp. For the other axes, the situation is the opposite; *i.e.*, the value with the bigger value is observed in the early part of the second phase (increasing ramp).

Regarding the first portion of the simulation shown in Fig. 8.2, the drop comes to a steady shape at $Gt < 100$, and the shape describes a behavior observed in Chapter 6. It is clear that the principal *L*- and *B*-Axis present small oscillations like the *trembling effect*, which is a consequence of the huge value of the *parameter G*. Then, as the flow changes to a more elongational character (α increases) the oscillations tend to disappear. When α is one, —fully *extensional 2D-flow*— there are no oscillations. Finally, when the flow becomes again α equal to zero, the oscillations reappear.

In Figures 8.2 and 8.3, oscillations of the lengths as the value of the α parameter increases appear to *decrease in frequency until* $\alpha = 1$. When the evolution of the flow is reversed, the sequence of oscillations is inverted until the *simple shear flow* regime is attained. That is, these oscillations come in packs; e.g., in the L-axis, when, $L - \text{axis}/r_0 = 1.25$ and $\alpha = 0.3$, a small undershoot is present that repeats when $\alpha = 0.39$ and $\alpha = 0.46$. The undershoot persists until $\alpha = 0.93$, then oscillations end. When $1 = \alpha \rightarrow 0$ flow, undershoots are observed at similar values of α (those values are not the same that the first cases). When the flow is *less strong*, undershoots cannot be as clearly appreciated as the mentioned undershoots, but for the L-axis evolution —shown in Fig. 8.3— it is possible to see the oscillation packs. The undershoot behavior in the oscillations in the B-axis also occurs. There, the effects of undershoots are larger than the L-axis case. For the W-axis, the oscillations are the tiniest and hardest to appreciate.

With this information, the shape of the drop can be inferred. When the value of α increases, the *L-axis* length is smaller than during the third phase flows. However, the other axes present the opposite situation. So, the drop is more elongated during the second part of the ramping cycle. This evidence is confirmed in Fig. 8.4. In this Figure, it is the Taylor deformation of the drop plotted vs. the α value. Again, the arrows indicate the direction of the ramp in Fig. 8.1. The plane of the applied flow presents the larger deformation shapes. Considering this *LB-plane*, with $\lambda_\mu = 16$ and the $Ca = 0.20$, and the values of α used in Chapter 4, Taylor deformations between those of Chapter 4 and the data obtained in this Hysteretic simulation agree.

The features of the drop dynamics mentioned above present a stationary state in the *2D-extensional regime*. However, there are no experiments published in this regime with similar values of the ratio of viscosity. In order to have an idea of the feasibility of the data results, the benchmarks used are those of Bentley and Leal (Bentley & Leal, 1986b) and (Taylor, 1934). For the former case, the ratio of viscosity, critical capillary number and Taylor deformation obtained were: for $\lambda_\mu = 13.8$, $Ca_{cr} = 0.103$ and $D_{cr} = 0.362$. The second case is $\lambda_\mu = 24.5$, $Ca_{cr} = 0.106$ and $D_{cr} = 0.347$. Finally, Taylor considers $\lambda_\mu = 20$, $Ca_{cr} = 0.28$ and $D_{cr} = 0.45$.

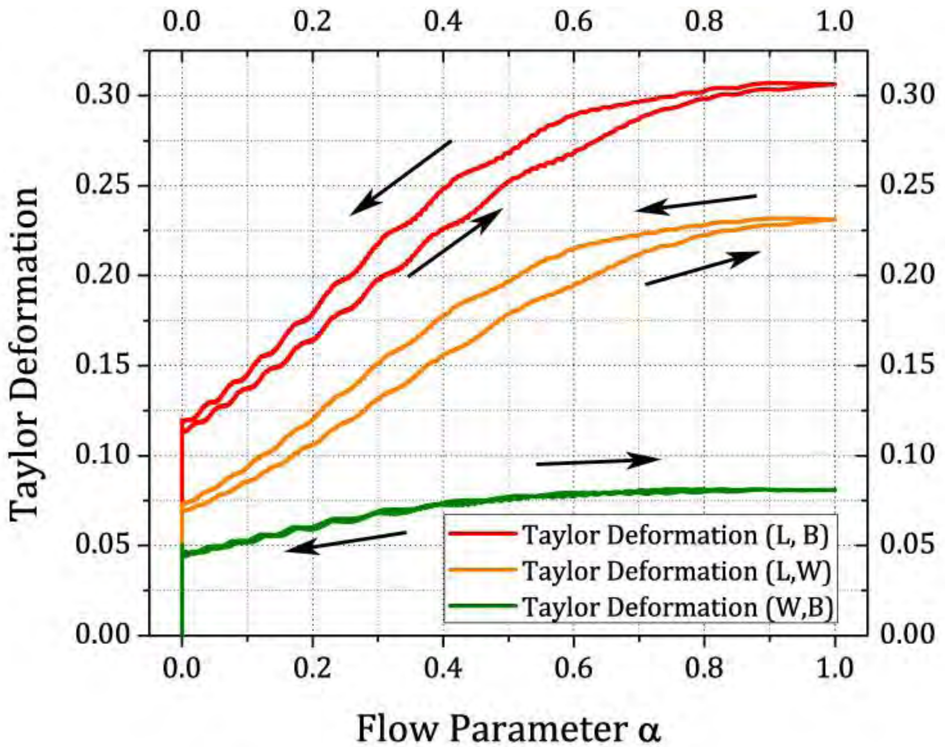

Figure 8.4 *Taylor Deformations vs. flow parameter α. With $\lambda_\mu = 16$, from simple shear flow $\alpha = 0$ to extensional 2D-flow $\alpha = 1$. Time period: $100 < Gt < 500$.*

With these values for the experimental parameters, the conception of stationary shapes by numerical simulation could be wrong. However, by looking the experimental data carefully, the data provides the next solid evidence. First, the deformation obtained by the numerical simulations is always less than the critical deformation reported in all experimental cases. So, if the circumstances of the flow provoke Taylor deformations smaller than the critical Taylor deformation, the drops will remain in a stationary deformation. With this information, the possibility to obtaining a stationary shape for the simulated conditions is feasible. Second, the *parameter G* values used are smaller, (the *trembling effect* is observed in the first part of the hysteretic numerical simulation; however, the oscillation values are less than 5% of the stationary value, and so, the curve around the point in *tank-treading* will be small). Thus, the effect observed in the second part of Chapter 6 could occur in this regimen too. If the applied *strain rates* in those experiments provoke an effect like *trembling effect*, the state of deformation could go to another state of drop deformation (critical deformation). This observation may explain the apparently contradictory differences between the data of Bentley and that of G. I. Taylor.

There is another numerical experiment of hysteretic phenomenon reported by Young *et al.*, (Young, Blawzdziewicz, Cristini, & Goodman, 2008). In this work, the range of drop ratio of viscosities is $\lambda_\mu = 50, 100$ and 200. The capillary numbers are less or equal to 0.2: $Ca \leq 0.20$. The simulated flow behavior is similar except that the flow history goes from *2D-extensional flow* to simpler shear, and comeback. The analysis focuses only in the *L-axis* length, and there is no comparison with experimental data. Also, there is no information about the Taylor deformation or the changes of the *B-axis*. The hysteretic phenomenon observed by Young *et al.*, appear to be different than the results presented in this Chapter, mainly because the region where the drop has two shapes is clearly smaller, for the same conditions of the flow type parameter.

Using the information of Figure 3.2, the stationary state of the L-axis, for a capillary number of $Ca = 0.40$, is the same as the maximum measure observed by Young *et al*. Reviewing the diagram of Taylor deformation *vs. Ca* in Fig. 3.5, the steady state deformation attained for this capillary number is $D_T = 0.4$, a similar value than that by Young *et al*. —their deformation is close to $D_T = 0.4$. If this situation is correct, the Taylor deformation attained by this numerical simulation is near the critical case.

The work of Young *et al.* uses the same idea of the flow type evolution. However, their experiment starts in the regime of *2D-extensional flow*. In this work, steady states employ 5% of the total simulation time, while the total time of the ramp is $T = 2000\lambda_\mu r_0 \mu\gamma^{-1} \approx 80Gt$ of our time of simulation. In other words, the simulation of the ramp of Young *et al.* is about three times faster than that shown in Fig. 8.1.

The results in Figure 8.5 are for the same capillary number than the previous Figure. However, the total time of the simulation is less than that of Figs. 8.3 and 8.4. The total time used is shown in the inset Graph. For the same conditions of the drop,

$Ca = 0.20$ and $\lambda_\mu = 16$, the dynamics of the principal axes of the drop are quite different. That is, the main features of the hysteresis depend as well on the ramping speed, which implies the necessity to analyze this kind of experiments to represent the quasi-static deformation dynamics. Here, the lengths of the principal axes vary a little in the extremes of the regimen of flow parameter, (in *simple shear flow* and *extensional 2D-flow*). The hysteresis in Fig. 8.5 is bigger than the case in Fig. 8.3.

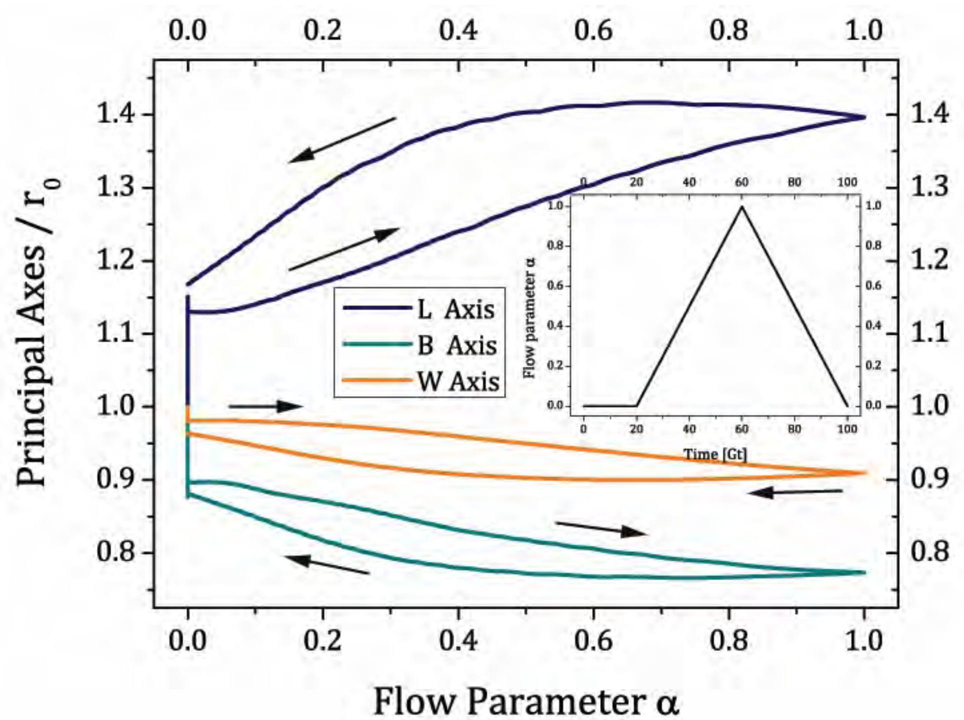

Figure 8.5 *Principal axes vs. flow parameter* α. *With* $\lambda_\mu = 16$, *from simple shear flow* $\alpha = 0$ *to extensional 2D-flow* $\alpha = 1$. *Time:* $20 < Gt < 100$.

8.2 Hysteresis in *extensional flows*

The previous Section addresses the hysteretic behavior of drop deformation as a consequence of the type of *two-dimensional flow* applied in the continuum phase. Also, in Chapter 7 the asymmetric drop deformation is studied when caused by (1) *two-dimensional extensional* and (2) *uniaxial flow*. In this Section, a new set of questions to resolve are addressed, among them: (a) what happens if the parameter *elongational type of flow* is varied from *uniaxial flow* to *extensional 2D-flow*? And (b) is there a hysteretic behavior in *extensional flows*?

In order to be able to simulate such a *2D*- to *3D*-variation of the flow type, a new parameter is introduced: the *elongational type of flow parameter* β. Thus, by varying β, drops can be subjected to a full range of flows without vorticity: from the *2D-linear incident flow* to a *pure elongational flow*, where the compression axis (of the former flow) becomes a compression plane perpendicular to the elongation axis. These simple kinematic conditions can be modeled by a velocity field given by

$$\boldsymbol{u}(x, y, z) = \frac{G}{2\sqrt{(4 - 2\beta + \beta^2)}}\big((\beta - 2)x, 2y, -\beta z\big), \tag{8.1}$$

where G is the intensity of the rate of deformation tensor (Makosco, 1994). Now, β is the parameter characterizing the strength flow along the symmetric (third) axis. Values of β goes from zero (*extensional 2D-flow*), to one (uniaxial flow).

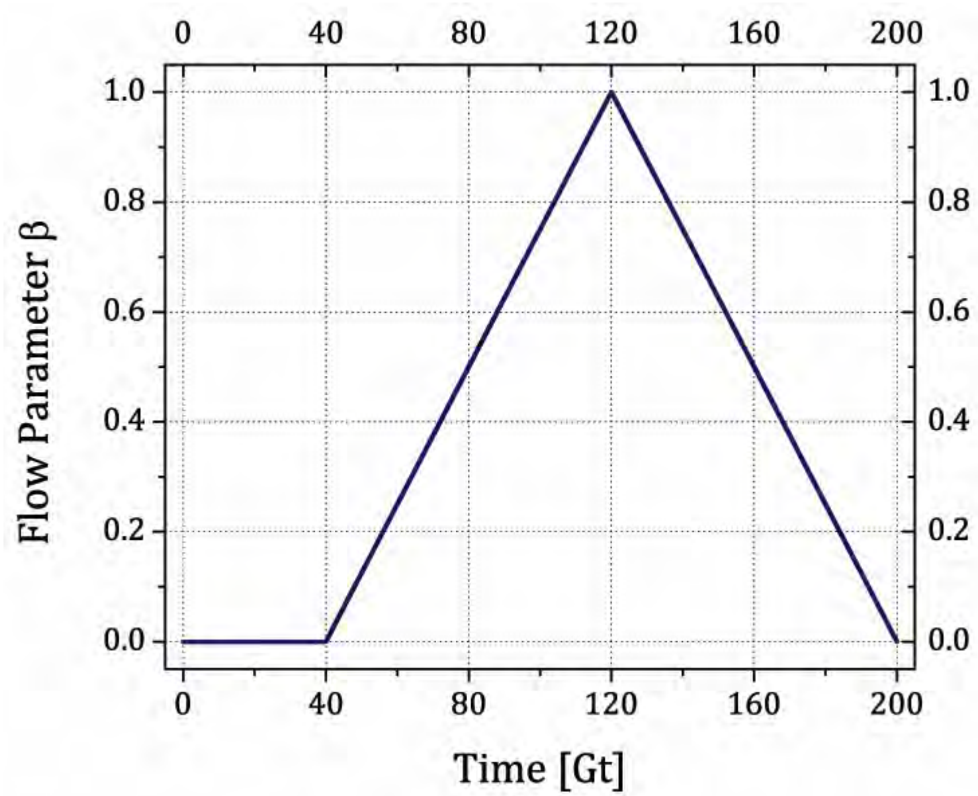

Figure 8.6 *Evolution of β in the time of numerical simulation. The first part was the value in extensional 2D-flow. Then the parameter betta was modified as a ramp.*

The *parameter* β offers the possibility to study another class of *elongational flow*, which occur when the value of $\beta = 1$ goes to 2. In this latter case, the *extensional* *flow* elongates in two directions and pushes in along the third axis. For this study, the family of *extensional flows* employed were in the regime of $0 \leq \beta \leq 1$.

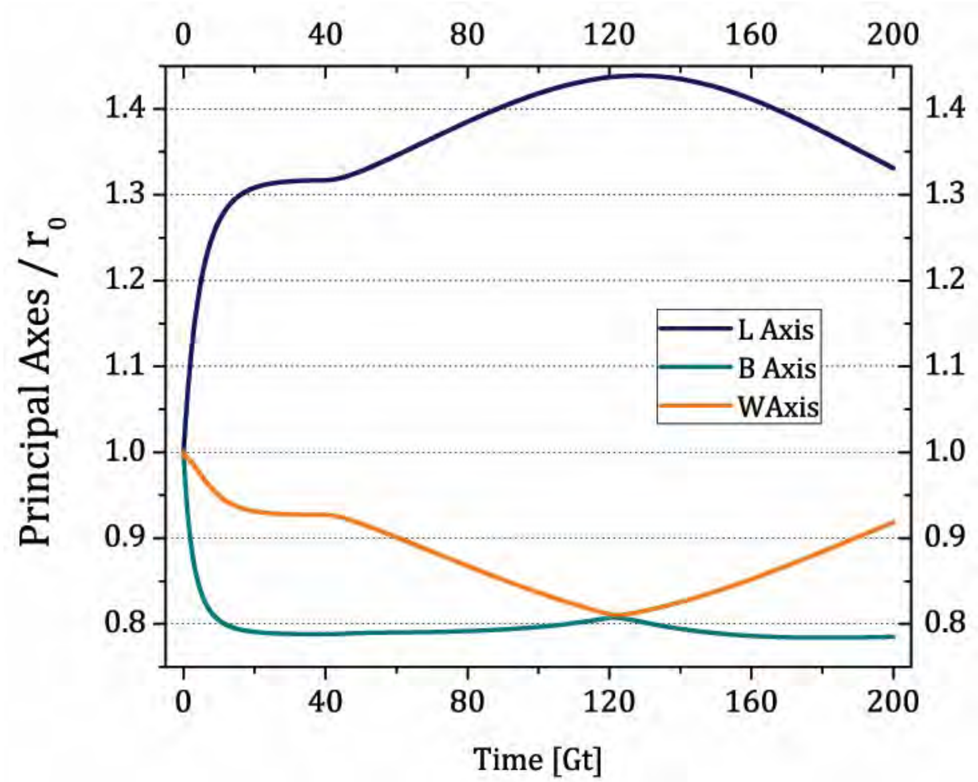

Figure 8.7 *Evolution of principal axes vs time. Time* $= 200Gt$*,* $\lambda_\mu = 16$*, from extensional 2D-flow* $\beta = 0$ *to uniaxial flow* $\beta = 1$*. Time:* $40 < Gt < 200$

The experiment begins applying a *2D-extensional flow* in the continuum phase. Then when the drops attains the steady shape for this flow, the value of the β parameter changes from zero to one, as Fig. 8.6 shows. As was commented in the previous Section, the critical capillary value in *extensional flows* are small. For this reason, the hysteretic numerical experiment was performed with a drop with $Ca = 0.10$ and $\lambda_\mu = 16$ to avoid the critical deformation and to guarantee attaining the steady shape of deformation.

Figure 8.7 shows the evolution of the principal axes during this experiment. After the application of *uniaxial extensional flow* under conditions of $\beta = 1$ and $T = 120\ Gt$, the drop achieves a circular cross-section. As a result of a smaller value of β, the drop evolves toward a modified cross-section: an elliptical shape. The evolution of *L-axis* is shown in Fig. 8.8 (equivalent to Fig. 8.3 for the variation of the *parameter* α), with the behavior of the others principal axes, all showing the existence of hysteresis in purely *extensional flows*. Here however, the difference between the two states of deformation is weaker than for the cases in strong α flows.

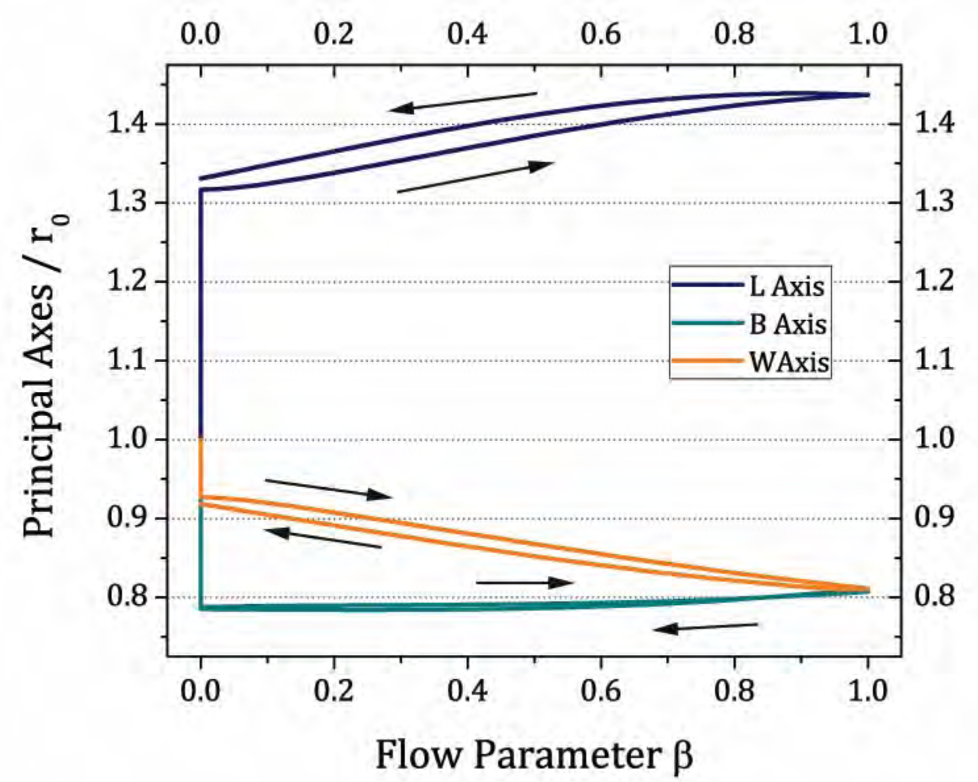

Figure 8.8 *Principal axes vs flow parameter betta β. With $\lambda_\mu = 16$, from extensional 2D-flow $\beta = 0$ to uniaxial flow $\beta = 1$. Time: $40 < Gt < 200$.*

The *B- and W-axes* start in the steady shape for *extensional 2D-flow* with different values. As the value of β changes to the value of one, the symmetric behavior of the uniaxial flow begins to dominate the size of those axes. This result is consistent with the data in uniaxial flow. Furthermore, the hysteretic behavior may depend of the ramp speed used in those experiments. However, if the total time used during the extensional regimen increases, the hysteresis of the principal axes of the drop will probably be imperceptible.

CHAPTER 9.

General conclusions:
drop deformation in strong flows

Until about 2005, the study of deformations of drops induced by strong type of flows has a significant contribution of both experimental and theoretical studies. These results indicate a large set of unsettle questions regarding the fluid hydrodynamics of these rather simple bi-phase flows. For example, there is no clear idea as to whether the main features of shape of drops change when the flow character varies from simple shear to a *2D*-elongational flow. Or whether for *2D*- and *3D*-elongational flows, shapes ought to be essentially the same.

On the drop dynamics front, if drops have different deformation along the preferential axes of the applied flow, the open question is what are the characteristic time-scales involve for the time evolution of the drop? or how different axes evolve in time, or evolve as the flow type changes?. There are as well various indicators that the different observed drop forms may indicate multiple Hopf bifurcation solutions for the governing equations. Here I attempted to elucidate on one hand some of the imply-cation on the shapes of drops and on the other hand the implications for the flow fields of the different solutions.

Thus, the initial objective of this study was to study not only these questions but as well possible implications for a more detailed understanding of the

implications opened by answers to these questions. In particular, very little is known about the flow fields inside and outside of the drops when different shapes of the drop occur. As well, which are the flow patterns inside the drop as a result of these possible solutions. These and possibly many other question-answers may influence signify-cantly the scope of amenable applications due to a better understanding of the dynamics of drop deformation.

As a first task, data is presented of observable stationary deformations for different ratios of viscosities and for different types of *2D*-strong flows. As many experimental studies have reported, the critical capillary numbers for drop rupture for each case of λ_μ have not been uniquely resolved for most cases, nor for different flow types. Hence, one of the first points addressed here is to properly calibrate the numerical code by finding the steady shape of drops near Ca_{cr} conditions. Subsequently, it was necessary to repeat the same calibration for every *2D-flow*. Finally, a diagram describing the critical capillary value for different flows, from *simple shear flow* to *extensional 2D-flow* is obtained.

As a second task, the three-dimensional behavior of the drop in these flows was carefully evaluated, because early theoretical works assumed asymmetric drops. However, the deviation from the axisymmetric case always occurs. In Fact, in *simple shear flow*, when the differences are less, the ratio of *B*- vs. *W-axis* is always less than 0.85. As the flow goes to the *extensional 2D-flow* —when the *α parameter* increases— the ratio goes up. The only flow which induces the asymmetric behavior of the cross section of the drop corresponds to the *3D-extensional flow*, an even then the drop ends may have a smoother variation of curvature than those predicted by theoretical models.

The prevalence of the non-circular cross-section that was observed while using the BEM3D code casts initial doubts about the codes used, due in part because theoretical works or many other numerical studies do not observe deviations from the circular cross-section case, as I noted. The code used here was rebuilt at least 4 times, but the results were the same, the cross-section is not circular, is more like an ellipse, but not quite. However, Guido *et al.*, work, published about the same time as the annotations of Acrivos and Lo, changed this work expectations, because their predictions made it possible to calibrate the code. Experimental data does provide sufficient information to disregard circular cross-section as the only possible form, but does not provide enough information to resolve uncertainties as to the correct shape for the ellipsoid. These results are presented in Chapter 2. The cross-sections are most likely non-circular, with the code predicting very well the small archive of experimental data.

In Chapter 3 and 4, the predicted deformed drop in stationary flows is shown to be like an asymmetric ellipsoid even for small capillary numbers—implying shapes close to spherical, but deformed. However, as the capillary number increases, the shape differs from a perfect ellipse, as well. In the past, these shapes were the *initial form* of drops used to evaluate the interfacial surface tension. Thus, in Chapter 5 possible methods for measuring the interfacial tension of the interface during the retraction phase of an elongated drop are reviewed, and with them I attempted a high-resolution calibration of the method, by comparisons of experimental, theoretical and numerical results. The theoretical analysis using ellipsoidal shapes appears to provide an initial good prediction of interfacial tension values in drops in strong flows. However, possible differences determined while obtaining the interfacial tension implies the necessity to adjust an optimized shape model to better approximate the interfacial tension measurements in the laboratory: that is, the technique works best when drops have an elongated, with symmetric cross-section shape. However, if these numerical results (capable of predicting the three-dimensional length scales of a drop) are compared the experimental data, predictions of interfacial tension can be more accurate than using only the analytical methods evaluated.

A second factor, affecting the surface tension measuring technique are the characteristic time-scales associated with the evolution of each principal axis of the drop, which were studied as well. When a *2D*-flow is applied in the continuum phase, the response of the drop in the same plane of the flow is faster than the third principal axis, orthogonal to the flow. However, the principal axes in the plane of flow have similar characteristic time-scales, but are not quite the same. Finally, a significant departure from previous works is that the numerical predictions —for the time needed to attain the stationary shape— seem to imply that *the characteristic time perpendicular to the flow* is the most appropriate relaxation time-scale, for it is *the slowest time of the drop retraction*. With this result, the analysis of stationary shapes of deformation should change the observation time, because in most experiments the stationary shape employed corresponds to that of the plane of the flow, even though, the third direction of the drop needs more time to attain the stationary state.

It is quite obvious that a diagram of critical capillary values is desirable to generate a detailed understanding of maximum drop deformations, especially for very elongated drops. However, as Chapter 6 indicates, the code appears not to be capable of predicting the large deformations previously reported by experimental data. As Chapter 3 and 4 showed, the prediction of the numerical code may be accurate only for small capillary numbers. Even more, for the case of highly viscous drops $\lambda_{\mu} = 16$, the code seems to fail about predictions of the stationary deformation and the number of oscillations while the drop tumbles. Add to this, the numerical data presents tiny regular oscillations which change in amplitude and frequency. At that moment, the possible cause of discrepancies was a mesh of poor resolution that was discarded after some extra simulations. Finally, as was demonstrated in Chapter 6, the dynamics of drop deformations in strong flows may have a cause associated with *the intensity of the flow, and not the mesh size*. The dynamics of drop deformation presents a bifurcation which depends on G, the shear rate. So, to find a simple stationary state of deformation is not possible, at least not when the shear rate is very small, $G \ll 1$.

There are a few published papers that establish the existence of Hopf bifurcation solutions for drops deformed by flows. However, there still exists a need in fluid mechanics to understand possible branches of these solutions, mainly because different type of shapes may imply different inner and outer flow fields. This is the first work that attempts to address possible consequences of Hopf bifurcations and hysteresis effects in drops and their dependence on $\lambda_{\mu}, \alpha, \beta, G$, and Ca as *complementary independent parameters*; mainly because, most experiments carried out in the laboratory employ variation of G, but linearly modify as the value of the capillary number.

Since Cox's work, oscillations observed in deformation and orientation of drops traces of experimental data or numerical simulations were explained as due to the competition of stresses imposed by the flow against interfacial stresses of the drop. Add to this idea that weak deformation of a drop observed when $\lambda_\mu \gg 1$, which predicts a *tumbling* behavior where the drops achieves negative orientation angles. However, as was commented in the last part of Chapter 6, the bifurcation of Hopf may contribute as well with the oscillatory behavior. Unfortunately, I arrive to this observation toward the end of this project. Therefore, to analyze carefully this behavior there is a need for more time because those numerical simulations take a lot of CPU-Time. The bifurcation diagram is necessary to understand the contributions of the shear rate on the dynamics of drop deformation. This line of work remains to be addressed.

In Chapter 7 there is a new contribution in fluid mechanics in the sense that the theoretical Diagram for deformation of drops in *extensional flows* was reviewed. The asymptotic approximation solution of Acrivos & Lo permits predicting the deformation of drops in *2D-flows* when the ratio of viscosity is small. However, as λ_μ attains values different from zero, this theoretical approximation starts to deviate from predictions, because λ_μ plays an essential role in drop deformation. In these cases, the proposed inner flow plays an essential participation, too. However, given the difficulty in the equations of fluids mechanics, the approximation made by Acrivos & Lo using slender body theory is a good step in the knowledge of drop deformation. Today with the numerical code here presented, it is possible to rebuild this diagram for different values of λ_μ. The long-term advantage would be that it may be possible to upgrade the code to have the inner and outer flow of the drop.

The analysis of drop deformation in *extensional 2D-flow* conveys the proper behavior of cross-sections, which is important to predict correctly the state of deformation. In Chapter 7, the evolution of behavior of cross-sections is extended, now between the *uniaxial flow* and the *extensional 2D-flow*. Understanding possible deformations under these regimes help us in making an analogous diagram to the one previously published by Acrivos & Lo.

In the last part of this study, the main focus is on the hysteretic nature observed when the flow applied in the continuum phase changes its main features; i.e., from *simple shear flow* to *extensional 2D*-flow, and subsequently from *extensional 2D-flow* to *uniaxial flow*. These experiments are presented in Chapter 8, and may represent the beginning of another branch of study in drop deformations. These numerical experiments need more time than those cases presented in Chapter 4: $\lambda_\mu \ll 1$. These latter numerical simulations take months to produce the data presented. Even more, there is a behavior that was not taken in account due to time constraints: the required time-lapse to attain the steady state shape when the parameter α is changing in the external flow. This observation is essential in this experiment. Thus, it is extremely important to guarantee that a steady shape is attained in every flow experiment. However, to reach this stage was out of the question for this project.

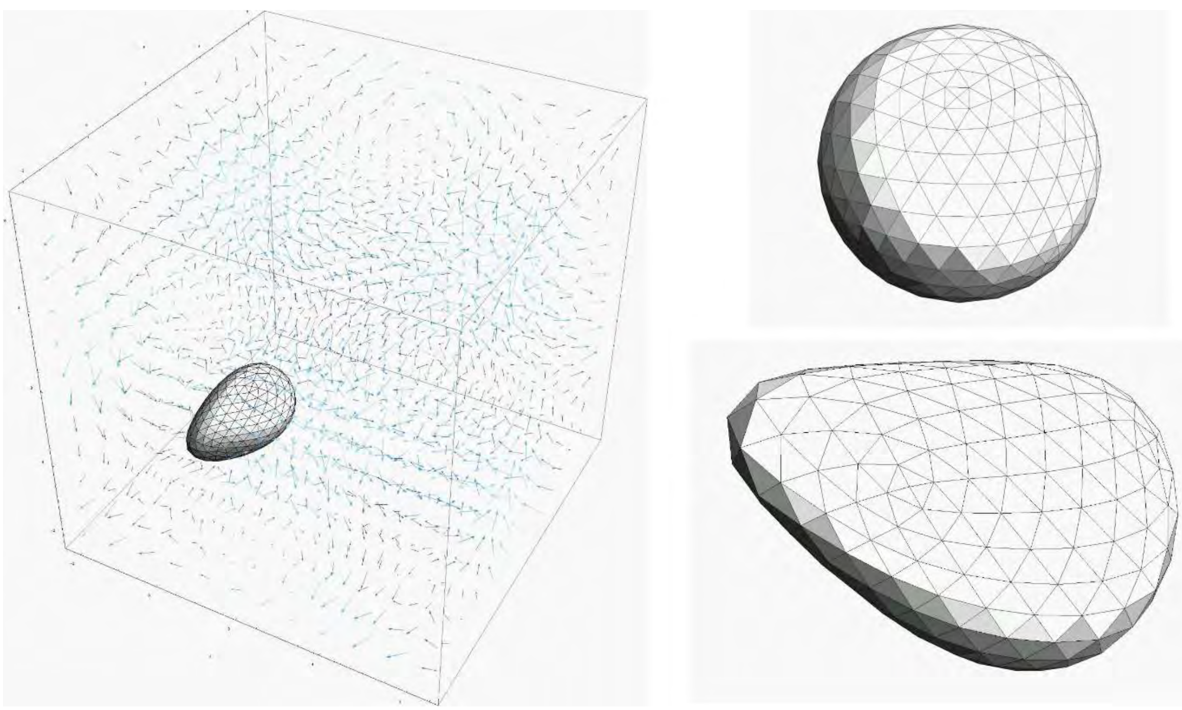

Figure 9.1 *Drop deformed in an ABC flow.* $\lambda_\mu = 1$. *View of the drop inside the flow, left. Initial shape of the drop and view of the state of deformation right-hand.*

In summary, in this study I was able to construct a new diagram. However, this diagram is still limited mainly because I discovered that drop deformation has not finished yet (reached its steady state form), which is due to a bifurcation generated by the value of the applied shear rate G. This result implies that drops are not perfect ellipses under flows with a large capillary number, regardless of the axis of observation of the drop shape. Also for these cases, shape models to predict the interfacial tension by retraction observations need to be polished. The theoretical diagram in uniaxial flow needs a new contribution for λ_μ different to zero.

The numerical code is not confined to *2D-flows*. The possibility to study complex deformations in three dimensions is possible. When performing flows that evolve in flow type character —such as with the hysteretic simulations—, there is a stable two-state of deformation for the same value of the flow parameter, β. As was observed, those states of deformations depend of λ_μ and the flow parameter.

As an example, the deformation of a drop in a theoretical flow: the Arnold-Beltrami-Childress flow (ABC flow) was made. Figure 9.1 shows the steady of deformation of a (initially spherical) drop. Please note the highly deformed cross-section. There, the *guarache* shape is now non-symmetric along the fore-aft axis, plus a curved shape similar to that of a potato chip. These forms are highly planar, with a non-symmetric fore-aft form, and with two main global curvatures. Under this scenario, these drops may present rather different relaxation mechanisms that have not been studied yet. Finally this code provides further avenues of research: for example, other natural phenomena may be amenable to study which will be extremely relevant in coming years due to climate change such as could be the locomotion of small animals in the sea, with movements being modeled as creeping flows. These flora and fauna contributions to the transport in the upper layer of the oceanic waters may influence the rates of absorption of CO_2 by the oceans but has been barely studied to date. Thus, this work is simply the beginning of a lot of new projects; the conclusions of this project are only in the sense of this book. However, the fluid mechanics in strong flows will continue.

BIBLIOGRAPHY

Acrivos, A., & Lo, L. S. (1978). Deformation and breakup of a single slender drop in an *extensional flow*. *J. Fluid Mech., 86*(4), 641-672.

Akin, E. (2003). *Object-Oriented Programming Via Fortran 90/95.* Cambridge: Cambridge.

Astarita, G. (1979). Objective and generally applicable criteria for flow classification. *Journal of Non-Newtonian Fluid Mechanics, 6*(1), 69-76.

Barthès-Biesel, D. a. (1973). Deformation and burst of a liquid droplet freely suspended in a linear shear field. *J. Fluid Mech., 61*, 1.

Batchelor, G. K. (1967). *An Introduction to Fluid Dynamics.* New York: Cambridge Mathematical Library.

Bazhlekov, B. P. (2004). Nonsingular boundary integral method for deformable drops. *Physics of Fluids, 16*(4), 1064-1081.

Bentley, B. J., & Leal, L. G. (1986a). A Computer-Controlled Four-Roll Mill for Investigations of Particle and Drop Dynamics in Two-Dimensional Linear *Shear flows*. *J. Fluid Mech., 167*, 219-240.

Bentley, B. J., & Leal, L. G. (1986b). An Experimental Investigation of Drop Deformation and Breakup in Steady, Two-Dimensional Linear Flows. *J. Fluid Mech., 167*, 241-283.

Blawzdziewicz, J., Cristini, V., & Loewenberg, M. (2003). Multiple stationary states for deformable drops in linear Stokes fows. *Physics of Fluids, 15*(5), L37-L40.

Brady, J. F., & Acrivos, A. (1982). The deformation and breakup of a slender drop in an *extensional flow*: inertial effects. *J. fluid Mech., 115*, 443-451.

Brebbia, C. A. (1978). *The Boundary element method for Engineers.* London: Pentech.

Chapman, S. J. (1998). *Fortran 90/95.* Boston: McGraw-Hill.

Cools, R., & Rabinowitz, P. (1993). Monominal cubature rules since "Stroud": a compilation. *Journal of Computational and Applied Mathematics, 48*, 309-326.

Coulliette, C., & Pozrikidis, C. (1998). Motion of an array of drops through a cylindrical tube. *J. Fluid Mech., 358*, 1-28.

Cox, R. G. (1969). The Deformation of a Drop in a General Time-Dependent Fluid Flow. *J. Fluid Mech., 37, part 3*, 601-623.

Cristini, V., Blawzdziewicz, J., & Loewenberg, M. (2000). An Adaptative Mesh Algorithm for Evolving Surfaces: Simulations of Drop Breakup and Coalescence. *Journal of Computational Physcis, 168*, 445-463.

Einstein, A. (1906). Eine neue Bestimmung der Moleküldimensionen. *Ann. Physik, 324*(2), 289.

Escalante, C. A., Reyes, M. A., Rosas, I. Y., & Geffroy, E. (2015). Drop deformation in two-roll mills considering wall effects. *Journal of Physics: Conferences Series, 582*(1).

Frankel, N. A., & Acrivos, A. (1970). The constitutive equation for a dilute emulsion. *J. fluid Mech., 44*, 65-78.

Grace, H. P. (1982). Dispersion phenomena in high viscosity immiscible fluid systems and application of static mixers as dispersion devices in such systems. *Chem. Engng. Commun., 14*, 225.

Greenberg, M. D. (1998). *Advanced Engineering Mathematics* (Second ed.). London, Syney, Toronto: Prenticed Hall.

Grinfeld, P. (2010). *Introduction to Tensor Analysis and the calculus of Moving Surfaces.* New york, Heidelberg, Dordrecht, London.: Springer.

Guido, S., & Greco, F. (2001). Drop Shape Under Slow Steady *Shear flow* and During Relaxation. Experimental Results and Comparison with Theory. *Rheol Acta, 40*, 176-184.

Guido, S., & Villone, M. (1997). Three-dimensional shape of a drop under *simple shear flow*. *Journal of Rheology, 42*, 395-415.

Guido, S., & Villone, M. (1999). Measurement of Interfacial Tension by Drop Retraction Analysis. *Journal of Colloid and Interface Science, 209*, 247-250.

Guido, S., Greco, F., & Villone, M. (1999). Experimental Determination of Drop Shape in Slow Steady *Shear flow*. *Journal of Colloid and Interface science, 219*, 298-309.

Ha, J.-W., & Leal, L. G. (2001). An Experimental Study of Drop Deformation and Breakup in *Extensional flow* at High Capillary Number. *Phys. Fluids, 13*(6), 1568-1576.

Happel, J., & Brenner, H. (1963). *Low Reynolds Number Hydrodynaics.* The Hague: Kluwer.

Hinch, E. J., & Acrivos, A. (1979). Steady long slender droplets in two-dimensional straining motion. *Journal of Fluid Mechanics, 91*(3), 401-414.

Hinch, E. J., & Acrivos, A. (1980). Long slender drops in a *simple shear flow*. *J. Fluid Mech., 98*(part 2), 305-328.

Huo, T. Y., Lowengrub, J. S., & Shelley, M. J. (2001). Boundary Integral Mhetods dor Multicomponent Fluids and Multiphase Materials. *J. Comput. Phys., 169*, 302-362.

Kennedy, M. R., Pozrikidis, C., & Skalak, R. (1994). Motion and deformation of liquid drops, and the rheology of dilute emulsions in *simple shear flow*. *Computers & Fluids, 23*(2), 251-278.

Khayat, R. E. (2000). Three-dimensional boundary element analysis of drop deformation in confined flow for Newtonian and viscoelastic systems. *International Journal for numerical methods in fluids., 34*, 241-275.

Khismatullin, D., Renardy, Y., & Renardy, M. (2006). Development and implementation of VOF-PROST for 3D viscoelasticic liquid-liquid simulation. *J. Non-Newtonian Fluid Mech., 140*(1), 120-131.

Kim, J., & Lowengrub, J. (2004). *Interfaces and multicomponent fluids.* Irvine, USA: University of California.

Kwak, S., & Pozrikidis, C. (1998). Adaptative Triangulation of Evolving, Closed, or Open Surfaces by the Advancing-Front Method. *Journal of Computational Physics, 145*, 61-88.

Ladyzhenskaya, O. A. (1963). *The Mathematical Theory of Viscous Incompressible Flow.* New York: Gordon and Breach.

Lalanne, B., Tanguy, S., & Risso, F. (2013). Effects of rising motion on the damped shape oscillations of drops and bubbles. *Physics of Fluids, 25*, 1120107 1-22.

Lamb, H. (1945). *Hydrodinamics.* New York: Dover.

Lambert, J. D. (1973). *Computational Methods in Ordinary Differential Equations.* Chichester, New York.: John Wiley & Sons.

Leal, L. G. (2007). *Advanced Transport Phenomena.* Cambridge: Cambridge University Press.

Levi, E. (1980). *Teorías y Métodos de las Matemáticas Aplicadas.* Ciudad de México: Facultad de Ingeniería, UNAM.

Loewenberg, M., & Hinch, E. J. (1996). Numerical simulation of a concentrated emulsion in *shear flow*. *J. Fluid Mech., 321*, 395-419.

Lorentz, H. A. (1907). Ein allgemeiner Satz, die Bewegung einer reibenden Flüssigkeit betreffend, nebst einigen Anwendungen desselben. *Abhand. theor. Phys.*, 13-42.

Mählmann, S., & Papageorgiou, D. T. (2009). Numerical study of electric field effects on the deformation of two-dimensional liquid drops in *simple shear flow* at arbitrary Reynolds number. *J. Fluid Mech., 626*, 367-393.

Makosco, C. (1994). *Rheology Principles, Measurements, and Applications.* New York, Chichester: Wiley-VCH.

Metcalf, M., Reid, J., & Cohen, M. (2004). *Fortran 95/2003 Explained.* New York: Oxford.

Miller, R. L. (2000). *Algorithms Secuential & Parallel. An Unified Appriach.* London, Sydney, Toronto, Mexico, New Delhi.: Prentice Hall.

Mo, H. Z. (2000). A new method to determine interfacial tension from the retraction of ellipsoidals drops. *J. Non-Newtonian Fluid Mech., 91*, 221-232.

Power, H., & Wrobel, L. C. (1995). *Boundary Integral Methods in Fluid Mechanics.* Southampton Boston: Computational Mechanics Publications.

Pozrikidis, C. (1992). *Boundary Integral and Singularity methods for Linearized Viscous Flow.* New York: Cambridge University Press.

Pozrikidis, C. (1997). *Introduction to Theorethical and Computational Fluid Dynamics.* Ney York: Oxford.

Pozrikidis, C. (1997). Numerical studies of singularity formation at free surfaces and fluid interfaces in two-dimensional Stokes flow. *J. Fluid Mech., 331*, 145-167.

Pozrikidis, C. (1998). *Numerical Computation in Science and Engineering.* New York, New York, United States: Oxford.

Pozrikidis, C. (2000). Interfacial Dynamics for Stokes Flow. *Journal of Computational Physics, 169*, 350-301.

Pozrikidis, C. (2002). *A Practical Guide to Boundary element methods with the Software Library BMLIB.* Boca Raton: Chapman & Hall.

Primo, A. R., Wrobel, L. C., & H., P. (2000). Boundary integral formulation for slow viscous flow in a deforming region containing a solid inclusion. . *Engineering Analysis with Boundary Elments , 24*, 53-63.

Prosperetti, A., & Tryggvason, G. (2007). *Computational Methods for Multiphase Flow.* Cambridge: Cambridge.

Rallison, J. M. (1978). Note on the Faxén Relations for a Particle in Stokes Flow. *j. Fluid Mech., 88*, 529-533.

Rallison, J. M. (1980). Note on the time-dependent deformation of a viscous drop which is almost spherical. *J. Fluid Mech., 98*, 625-633.

Rallison, J. M. (1981). A Numerical Study of the Deformation and Burst of a Viscous Drop in General *Shear flows*. *J. Fluid Mech., 109*, 465-482.

Rallison, J. M. (1984). The deformation of small viscous drops and bubbles in *shear flow*. *Ann. Rev.Fluid Mech., 16*, 45-66.

Rallison, J. M., & Acrivos, A. (1978). A Numerical Study of the Deformation and Burst of a Viscous Drop in an *Extensional flow*. *J. Fluid Mech., 89, Part 1*, 191-200.

Ramalingam, S., Ramkrishna, D., & Basaran, O. A. (2012). Free vibrations of a spherical drop constrined at an azimuth. *Physics of Fluids, 24*, 082102 1-20.

Rayleigh, F. R. (1879). On the Capillary Phenomena of Jets. *Proc. Roy. Soc., 29*, 71-97.

Resnick, R. a. (1980). *Física* (Vol. 2). España, Argentina, México: Compañia Editorial Continental.

Reyes, M. A. (2005). Hydrodynamics of Deformable Objects in Creeping Flows. *Tesis de Doctorado en Ciencia e Ingeniería de Materiales*.

Reyes, M. A., Minzoni, A. A., & Geffroy, E. (2011). Numerical study of the effectof nonlinear control on the behavior of a liquid drop in elengational flow with vorticity. *Journal Engineering Mathematic, 71*(2), 185-203.

Rojas, J. G. (2016). Medición de la tensión superficial de una gota con deformaciones generadas en un molino de dos rodillos. Su relevancia para el estudio de la reología de emulsiones en flujos fuertes. *Thesis de Maestría en Ciencia e Ingeniería de Materiales*.

Rosas, I. Y. (2013). *Experimental study of the deformation of a drop immersed in a elongational flow with vorticity.* Mexico.: UNAM, Thesis.

Rosas, I. Y., Reyes, M. A., Minzoni, A. A., & Geffroy, E. (2014). Computer-controlled Two-Roll Mill flow cell for the experimental study of particle and drop dynamics. *Experimental Thermal and Fluid Science, 60*, 54-65.

Rumscheid, F. D., & Mason, S. G. (1961). Particle motions in sheared suspencions xii. deformation and burst of fluid drops in shear and h. *J. Colloid Sci., 16*, 238.

Sohn, J. S., Yu-Hau, T., Li, S., Voigt, A., & Lowengrub, J. S. (2010). Dynamics of multicomponent vesicles in a viscous fluid. *J. Comput Phys., 229*(1), 119-144.

Spann, A. P., Zhao, H., & Shaqfeh, E. S. (2014). Loop subdivision surface boundary integral method simulations of vesicles at low reduced volume ratio in shear and *extensional flow*. *Physics of Fluids, 26*, 031902-1-26.

Stoker, J. J. (1989). *Differential Geometry.* New York,London, Sydney, Toronto: Wiley Classics Library, Wiley Interscience.

Stone, H. A., & Leal, L. G. (1989a). Relaxation and breakup of an initially extended drop in an otherwise quiescent fluid. *J. Fluid. Mech., 198*, 399-427.

Stone, H. A., & Leal, L. G. (1989b). The influence of initial deformation on drop breakup in subcritical time-dependent flows at low Reynolds numbers. *J. Fluid Mech, 206*, 223-263.

Stone, H. A., Bentley, B. J., & Leal, L. G. (1986). An experimental study of transient effects in the breakup of viscous drops. *J. Fluid Mech., 173*, 131-158.

Subramanian, G., & Koch, D. L. (2006). Centrifugal Forces Alter Streamline Topology and Greatly Enhance the Rate of Heat and Mass Transfer drom Neutrally Bouyan Particles to a *Shear flow*. *Physical Review Letters, 96*, 134803 1-4.

Taylor, G. I. (1932). The Viscosity of a Fluid Containing Small Drops of Another Fluid. *Proceedings of the Royal Society A, 138*(834), 41-48.

Taylor, G. I. (1934). The Formation of Emulsions in Definable Fields of Flow. *Proc. R. Soc. Lond. A, 146*, 501-523.

Taylor, G. I. (1964). Conical free surfaces and fluid interfaces. *Proc. 11th Int. Cong. Appl. Mech.* Munich.

Torza, S., Cox, R. G., & Mason, S. G. (1972). Particle motions in sheared suspensions. xxvii. transient and esteady deformation and burstof liquid drops. *J- Colloid Interface Sci., 38*, 395.

Unverdi, S. O., & Tryggvason, G. (1991). A Front-Tracking Method for Viscous, Incompressible, Multiphase-fluid Flows. *Jorunal of Computational Physics, 100*, 25-37.

Wannaborworn, S., Mackley, M. R., & Renardy, Y. (2002). Experimental observation and matching numerical simulation for the deformation amd breakup of immiscible drops in oscillatory shear. *J. Rheol., 46*(5), 1279-1293.

Young, Y. N., Blawzdziewicz, J., Cristini, V., & Goodman, R. H. (2008). Hysteretic and chaotic dynamics of viscous drops in creeping flows with rotation. *J. Fluid Mech., 607*, 209-234.

Youngren, G. K., & Acrivos, A. (1975). Stokes flow past a particle of arbitrary shape: a numerical method of solution. *J. Fluid Mech., 69*, 377-403.

Yu, W., Bousmina, M., & Zhou, C. (2004). Determination of interfacial tension by the retracction method of highly deformed drop. *Rheol Acta, 43*, 342-349.

Yuriko, J. L., Renardy, Y., & Renardy, M. (2000). Numerical Simulation of Breakup of a Viscous Drop in *Simple shear flow* Through a Volume-of-Fluid Method. *Phys. of Fluids, 12*(2), 269-282.

Zhao, H., & Shaqfeh, E. S. (2011). The dynamics of a vesicle in *simple shear flow*. *J. Fluid Mech., 674*, 578-604.

Zinchenko, A. Z. (1997). A Novel Boundary-Integral Algorithm for Viscous Interaction of Deformable Drops. *Phys, Fluids, 9*(6), 1493-1511.

APPENDIX A

VIII International Congress of Engineering Physics IOP Publishing
IOP Conf. Series: Journal of Physics: Conf. Series **792** (2017) 012005 doi:10.1088/1742-6596/792/1/012005

Numerical study of the *3D*-shape of a drop immersed in a fluid under an elongational flow with vorticity

A S Sanjuan[1], M A H Reyes [2], A A Minzoni[3] and E Geffroy[1]
[1]Instituto de Investigaciones en Materiales, Universidad Nacional Autónoma de México, Mexico City.
[2]Departamento de Termofluidos, Facultad de Ingeniería. Universidad Nacional Autónoma de México, Mexico City.
[3]Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas., Universidad Nacional Autónoma de México, Mexico City.

E-mail: alfrsj07@gmail.com

Abstract. This work focuses on a three-dimensional analysis of the deformation of a drop —immersed in a Newtonian fluid— generated by a 2D elongational flow with vorticity. The study of steady-state deformations of the cross-section of the drop shows a prevalent non-circular shape. The axisymmetric idealization of the ellipsoid is not observed nor the linear dependency between capillary number and deformation of the drop, as Taylor and Cox theory predicted. Our numerical results are consistent with experiments and other numerical simulations. However, in the latter cases, measurements of the cross section of the drop are few while a limited class of flows is applied. In this work, deformations induced by general two-dimensional flows upon the 3D drop shape are presented with special emphasis about the length scale along the third axis —perpendicular to the plane of the applied flow field.

1. Introduction

Flow-induced drop deformation is an important topic in Fluid Mechanics [1,2] due to applications in industry as emulsion processing, micro-fluidics technologies [3-5], and others. Since the work of Taylor [1,2] the study of drop deformation has focused in the effects induced by the flow applied on the continuum phase. As Rallison explains [3], studies of drop deformations have focused principally when it is subjected to shear flows or extensional flows. The analysis of these two-dimensional flows permitted the theoretical analysis of small-drop deformation [1,3,5,6], assuming an spheroidal shape with the cross-section remaining axisymmetric. However, Guido et all [7] found that cross-sections in simple shear flow are non circular. Numerical studies of drop deformation showed this conditions [5,8], but these studies did not analyze the drop cross-section.

The study of drop deformation in two-dimensional flows is still an open question because an ample family of 2D-flows is not easily carried out in the laboratory [9-11]. As we will see in the next Section, a family of 2D-flows ranging from simple shear to pure extensional flow can be obtained as a mixture of rotation and elongation parts [1,12]. With this family of flows, it is possible to study the drop deformations induced from simple shear to extensional flow, while observing the contribution of added vorticity in the flow. The numerical experiments here presented can supplement the experimental data to understand better the dynamics of drop deformations.

This work focuses in flows with kinematics close to those of simple shear flow but, with less vorticity —more elongational effects. To calibrate the numerical experiments, the experimental data of Rosas et al., [10] are used.

VIII International Congress of Engineering Physics IOP Publishing
IOP Conf. Series: Journal of Physics: Conf. Series **792** (2017) 012005 doi:10.1088/1742-6596/792/1/012005

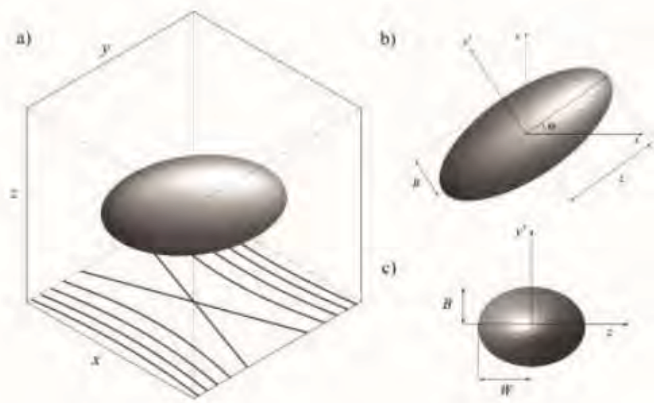

Figure 1. Scheme of a two-phase fluid. Drop in stationary shape. $\lambda_\mu = 0.012$, $\alpha = 0.13$, $Ca = 0.4$.
a) 3D-view view of the drop within a 2D-flow applied.
b) Conventional perspective on the *xy* plane.
c) Cross section of the drop.

2. Drop deformation in 2-D linear flow

The analysis of drop deformation assumes two Newtonian fluids, figure 1. The first (inner) phase is the drop with viscosity μ_0 and surface tension σ. The continuum phase has a viscosity μ_1. The viscosity ratio is $\lambda_\mu = \mu_0/\mu_1$. The two fluids have the same density and both are immiscible. There is not surfactant and there are not Marangoni stresses present. The creeping-flow conditions are assumed. In the creeping-flow regime the fluid motion is governed by the Stokes equations [12]

$$\mu \nabla^2 u = \nabla p ,$$
$$\nabla \cdot u = 0 . \qquad (1)$$

The velocity $\boldsymbol{u}$ is continuous at the drop interface *S*. The tractions exerted on the two sides of the interface between the two fluids have two different values, with a corresponding discontinuity:

$$\Delta f = f_1 - f_0 = (\Pi_1 - \Pi_0) \cdot \hat{n} , \qquad (2)$$

where $\hat{\boldsymbol{n}}$ is the normal vector pointing out of *S*. In this work, a constant value surface tension [13] is assumed; i.e., $\nabla \cdot \hat{\boldsymbol{n}}$ is equal to twice the mean curvature κ_m at that point on the interface,

$$\Delta f = \sigma \hat{n} \nabla \cdot \hat{n} = 2\sigma \kappa_m \hat{n} \qquad (3)$$

The drop is subject to a two dimensional linear incident flow

$$u(x, y) = \frac{G}{(1+\alpha)}(\alpha y, x) , \qquad (4)$$

where G is the intensity of the rate of deformation tensor [15] and the flow parameter is α [9-11, 14].

$$G = \left(\left|II_{2D}\right|\right)^{1/2} . \qquad (5)$$

Equation (4) assumes that the α parameter goes from zero to one. The case $\alpha = 0$ corresponds to simple shear flow with the velocity gradient in the *y* direction, the value of G is equal to shear rate $\dot{\gamma}$. For $\alpha = 1$ we have an extensional flow in two-dimension with the principal axes of deformation at $x = y$ —the compressional axes being $x = -y$; the intensity of the flow is the strain rate $\dot{\varepsilon}$.

The dynamic of drop deformation is characterized by three dimensionless numbers. The viscosity ratio λ_μ, the capillary number *Ca* equation (6) and The flow parameter α equation (7). The capillary number characterizes the ratio between viscous stresses —imposed by the flow— and capillary forces that resist the deformation and drive the drop towards the equilibrium shape, were r_0 is the non-deformed radius of the drop.

$$Ca = \frac{r_0 \mu_1 G}{\sigma} , \qquad (6)$$

Finally, the flow parameter α characterizes the magnitude of the rotational component relative to the extensional component of the external flow. Here, results are presented for a range of values of the flow-parameter close to simple shear flow, i.e., $\alpha = 0.03$, $\alpha = 0.05$ and $\alpha = 0.13$. The applied flow

VIII International Congress of Engineering Physics IOP Publishing
IOP Conf. Series: Journal of Physics: Conf. Series **792** (2017) 012005 doi:10.1088/1742-6596/792/1/012005

covers a range of the Capillary number from *0.05* to *0.40*. For values less than the minimum, the drop shape does not present different values of its principal conditions; for values of *0.4* or less, the deformation is not significant to cause drop breakup. The continuum phase is more viscous than the drop viscosity: $\lambda_\mu = 0.012$; the value is chosen so that numerical predictions match the values of the experiment results of Rosas [10].

The measure of drop deformation used corresponds to Taylor Deformation:

$$D_T = \frac{L - B}{L + B}, \tag{7}$$

were L and B are the axes of the drop shown in figure 1. Additionally, we present the time evolution of the lengths of all principal axes of the drop *vs.* the initial radius; see figure 2.

3. Numerical Method

The numerical method used is the 3D collocation boundary element method [13]. Using the Lorentz reciprocal theorem for Stokes regime [12,13], particular solutions can be expressed in terms of known solutions as point sources on the interface where $\boldsymbol{u}_0$ is the flow on the surface of the drop and f_0 is the force on the surface of the drop.

$$\int_S (\mu_1 u_1 f_0 - \mu_0 u_0 f_1) dS = 0, \tag{8}$$

were $f = \boldsymbol{\sigma} \cdot \hat{n}$, considering on the surface of the drop point sources whose velocity and stresses are the stokeslet and the stresslet respectively [5,12,13]. Then, the numerical scheme is represented as

$$u^\infty(x_0) - \frac{1}{8\pi\mu_1}\int_S \Delta f(x) \cdot G(x, x_0) dS(x) + \frac{1-\lambda_\mu}{8\pi}\int_S u_1(x_0) \cdot T(x, x_0) \cdot \hat{n}(x) dS(x) = \begin{cases} u_1(x_0) & \text{a}), \\ \dfrac{1+\lambda_\mu}{8\pi} u_1(x_0) & \text{b}), \text{ or} \\ \lambda_\mu u_2(x_0) & \text{c}). \end{cases} \tag{9}$$

In this equation there are three cases. The case, (b) Solves the velocity field when x_0 is on the surface of the drop. Then with this information the case (a) calculates the velocity field when x_0 is outside of the drop —exterior flow. Finally, (c) calculates the velocity field when x_0 is inside the drop.

The flow imposed on the continuum phase is determined by $\boldsymbol{u}^\infty(x_0)$. The second term on the left side of equation (9) is known as the Single Layer Potential and evaluates stresses across the interface. The third term is known as the Double Layer Potential and is used to evaluate the velocity field at the interface. The boundary element method has the advantage that it reduces the 3D computation problem into a 2D-evaluation, i.e., the method requires to solve the velocity field on the surface of the drop, and (b) with this information is possible to have the velocity field outside or inside the drop.

3.1 Numerical implementation of the computation of stresses and velocity fields

The numerical scheme solves the Stokes flow equation at an instant of time, equation (9 b). The evolution of the drop shape is carried out using the velocities obtained at the collocation points. By means of an interpolation subroutine the velocity at all nodes is calculated and then the new position of the drop is evaluated. For the evaluation of the Single Layer Potential, an approximation of the local curvature of the drop is used, using equation 3 with curved triangles. A quadratic interpolation throughout these triangles is performed to obtain the mean curvature of the elements of the drop. In this manner, the mesh of the interface and the algebraic system of equations of the boundary elements is smaller than those of other more conventional numerical schemes, such as e.g., finite differences.

3.2 Numerical Accuracy

The numerical simulation presented in this paper uses a surface of *2048* elements. To obtain the *time evolution* of the drop deformation, time is advanced using a fourth-order Adams-Bashforth-Moulton method [16]. The total time required to reach the stationary shape of the drop is $T = Gt$, reaching the final stationary deformation in all numerical experiments. These numerical results were compared with the data reported by Rosas [10] under the same flow regime.

VIII International Congress of Engineering Physics IOP Publishing
IOP Conf. Series: Journal of Physics: Conf. Series **792** (2017) 012005 doi:10.1088/1742-6596/792/1/012005

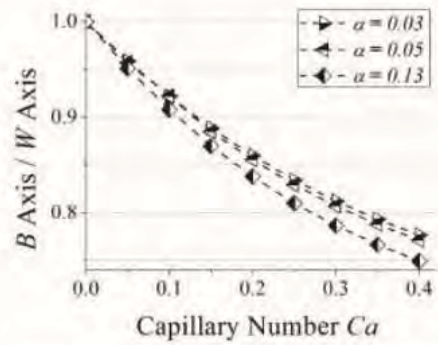

Figure 4. Stationary ratio of B wrt W axes in numerical simulations for a flow with $\alpha = 0.03$, $\alpha = 0.05$ and $\alpha = 0.13$ for different values of Ca.

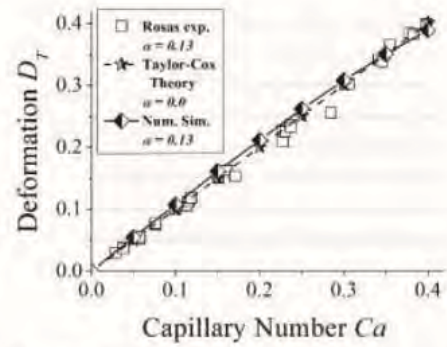

Figure 5. Stationary Taylor Deformation values for numerical simulations and experimental data for $\alpha = 0.13$ and analytical results of Taylor-Cox at $\alpha = 0.0$ for different values of Ca.

In figure 4 the ratio between axes of the cross section of the drop are given for three different flow strengths. As the capillary number increases, the eccentricity of the ellipsoid of the cross section increases as well. This ratio also depends on the flow-type parameter α becoming more eccentric for flows close to simple shear flow. For simple shear flows, these simulations produce values for the ratios of the cross section that are similar to the numerical results reported in References [7,8], having as well the same general behavior, even though the viscosity ratio reported is larger. Qualitatively it is possible to see the similarity [8]. Figure 5 presents a comparison of experimental results [10] and the simulated values for $\alpha = 0.13$, and the analytical results of Taylor-Cox for $\alpha = 0.0$ for the stationary Taylor Deformation value in the xy-plane; see figure 1(b).

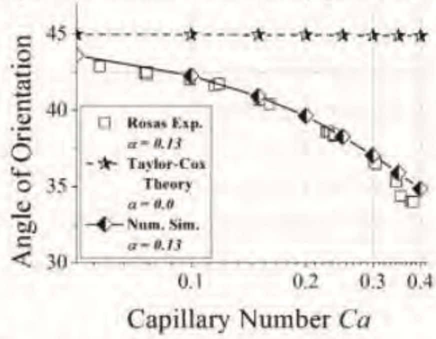

Figure 6. Orientation of the drop for numerical simulations and experimental data for $\alpha = 0.13$ and analytical results of Taylor-Cox $\alpha = 0.0$ for different values of Ca.

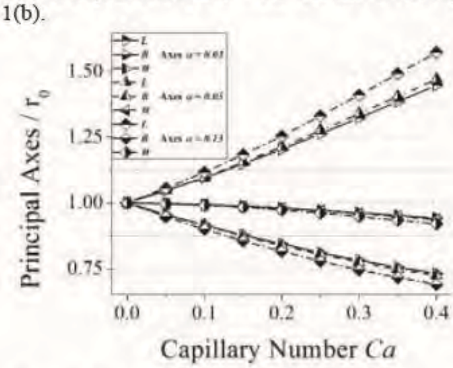

Figure 7. Stationary values of the principal axes of the drop for a flow with $\alpha = 0.03$, $\alpha = 0.05$ and $\alpha = 0.13$ for different values of Ca.

The stationary deformation of the drop depends upon the capillary number as well as the flow type. When $Ca = 0.4$, and for flows with $\alpha = 0.03$, the deformation achieved is less than 85% of the values reached for flows of $\alpha = 0.13$. Cox [5] and Taylor [1] theory for simple shear flows establishes a linear relationship between deformation *vs*. the capillary number. However, the dependence observed with simulations of stronger flows is no longer linear, the later being in agreement with the experiments of Rosas [10, 11].

Figure 6 shows the rotation of the principal axis of the drop; see figure 1 (b). In figure 6, the orientation is comparable with the experimental data for similar capillary numbers. The theory based in

VIII International Congress of Engineering Physics IOP Publishing
IOP Conf. Series: Journal of Physics: Conf. Series **792** (2017) 012005 doi:10.1088/1742-6596/792/1/012005

simple shear flow does not match with the experimental data. Finally, the measurements of the principal axes of the drop are shown in figure 7. These values were obtained in the stationary shape of the drop.

5. Conclusions

The simulations here presented for drop forms induced in 2D-flows do not appear to match the axisymmetric case established by G. I. Taylor and Cox, as neither did Kennedy et al., and Guido et al., for the simple shear flows. This is likely the case for capillary number less than *0.4*.

For $Ca > 0.4$, the deformation in the *xy*-plane increases, but, neither as numerical simulation show, there is a correlation with the Taylor-Cox model. On the other hand, the simulated behavior appears consistent with the experimental data of Rosas.

The difference between the simple shear flow, $\alpha = 0.0$, and flows with stronger degree of elongation induced larger drop deformations while the angle of orientation (principal axis) rotates away from *45* degrees.

The analysis of the *L*, *B* and *W*-axes of the ellipsoidal drop, in steady state, clearly show deviations from the axisymmetric shape as the flow type parameter α increases. However, the change of *W* remains unchanged for the values $\alpha = 0.03 - 0.13$, as a result of the vorticity of the applied 2D-flow.

6. Acknowledgments

The authors acknowledge funding provided by Project No. IG100714 of PAPIIT-UNAM Program. ASS gratefully acknowledges the financial support provided by CONACyT and PCeIM-UNAM during his PhD studies.

7. References

[1] Taylor G I 1932 *Proc. Roy. Soc.* A **138** 41
[2] Taylor G I 193. *Proc. Roy. Soc,* A **146** 501
[3] Rallison J M 1984 *Ann. Rev. Fluid Mech.* **16** 45
[4] Prosperetti A and Tryggvasson G 2009 *Computational Methods for Multiphase Flow* (Cambridge, New York, Melbourne: Cambridge University Press)
[5] Cox R G 1969 *J. Fluid Mech.* **37** 601
[6] Hinch E J and Acrivos A 1980 *J. Fluid Mech.* **98** 305
[7] Guido S and Villone M 1999 *J. Rheol.* **42**(2) 395
[8] Kennedy M R, Pozrikidis C and Skalak R 1994 *Computers Fluids.* **23**(2) 251
[9] Bentley B J and Leal L G 1986 *J. Fluid Mech.* **167** 241
[10] Rosas I Y, Reyes M A H, Minzoni A A and Geffroy E 2015 *Exp. Therm. Fluid Sci.* **60** 54
[11] Escalante C A, Reyes M A H, Rosas I Y and Geffroy E 2015 *J. Phys.: Conf. Ser.* **582** 012014
[12] Leal L G 2007 *Advanced Transport Phenomena* (Cambridge, New York: Cambridge University Press)
[13] Pozrikidis C 1997 *Introduction to Theoretical and Computational Fluid dynamics* (New York, Oxford: Oxford University Press)
[14] Reyes M A H, Minzoni A A and Geffroy E 2011 *J. Eng. Math.* **71**(2) 185
[15] Macosko C W 1994 *Rheology Principles, Measurements, and Applications* ed Wiley-VCH (Chichester New York) p21-23 68-74
[16] Lambert J D 1973 *Computational Methods in Ordinary Differential Equations* ed Wiley & Sons (Chichester New York)

APPENDIX B

In this appendix B, I will present the complete code made to study the drop deformation in strong flows. Unfortunately, for the extension of the code (around of 15000 lines, 240 pages) the appendix B is digital.

The code is attached in CD-ROM. If there are doubts, or comments about the code, do not hesitate to contact me: alfrsj07@gmail.com.

The code is divided in the principal program and 17 modules. Every module has its subroutines. Next, I present a list of the modules of the code.

- Prtcl_3D is the main program.
- Mod_SharedVars: Module with shared variables.
- Mod_Trgl_Octa: Module which made the mesh of the drop.
- Mod_Prtcl_Geo: Modules with subroutines to computes the drop's geometry.
- Mod_Nodal_Interpolation: Module with subroutines to interpolate the mesh in the time.
- Mod_Gauss_Coefs: Module with the numerical weights employed to integrate.
- Mod_Builder_Matrix_Arrays: Module which made the algebraic system to solve the Eq. (1.22b).
- Mod_Prtcl_slp.: Module which computes the Single Layer Potential.
- Mod_sgf_3d_fs: Module which computes the Stokeslet.
- Mod_Prtcl_DLP: Module which computes the Double Layer Potential.
- Mod_sgf_3d_sfs: Module which computes the Stresslet.
- Mod_Velocity_Menu: Module with the different option of flows.

- Mod_VelFieldTRM_13: Module which computes the analytical velocity field estimated for the Two-Roll-Mill device.
- Mod_SNEDOS: Module with subroutines to solve the ordinary differential equations.
- Mod_Correction: Module which preserves the condition of conservation of volume.
- Mod_Data_Files: Module which indicates how to save the data.
- Mod_axb: Module which solves the algebraic system equations.

```
PROGRAM Prtcl_3D
!=====================================================================================================
!
!           Version 0.5                                                               C. POZRIKIDIS
!-----------------------------------------------------------------------------------------------------
!                       Version 0.7                                MARCO ANTONIO REYES HUESCA
!-----------------------------------------------------------------------------------------------------
!                                   Versión 1.0            ALFREDO SANJUAN SANJUAN
!
!                                   **Mexico D.F., in the Year of Our Lord of 2012**
!
!=====================================================================================================
!=====================================================================================================
!   This code is a numerical proyect about a drop deformation in Stokes' Flows. You would see this versio was
!obtained of the wor of Pozrikidis as a begining. However, this code has oher aplication becouse it shows what's
!happen in a drop inmerse in anoher fluid when someone applies on continue phase a strong flwo.
!=====================================================================================================
!   This program computes the following Stokes flows:
!
!       1) Flow due to the motion of a 3-dimensional drop immerse in another immiscible fluid. In this case, the
!          particle is an ellipsoid with:
!
!           x semi-axis equal to a
!           y semi-axis equal to b
!           z semi-axis equal to c
!=====================================================================================================
! SYMBOLS:
!=====================================================================================================
!  p(i,j) ......................... coordinates of the ith node (j=1,2,3)
!  ne (k,j) ....................... ne(k,1) is the number of elements adjacent to point k
!                                   ne(k,2), ... are the elements numbers, j = 2, ..., 7 for this triangulation
!                                   up to six
!  n(k,i) ......................... connectivity table: points for element k, i = 1,...,6
!  nbe(k,j) ....................... the three neighboring elements of element k (j=1,2,3)
!  arel(i) ........................ surface area of ith element
!  npts ........................... total number of points
!  nelm ........................... total number of elements
!  x0, y0, z0 ..................... coordinates of collocation points
!  vnx0, vny0, vnz0 ............... unit normal vector at collocation points
!  alphaQ, betaQ, gammaQ .......... parameters for quadratic xi-eta isoparametric mapping
!=====================================================================================================
!=====================================================================================================
!The inicial thing is to USE all MODULES
   USE Mod_SharedVars                                                  !
   USE Mod_TRMCoefs                                                    !
   USE Mod_Trgl_Octa                                                   !
   USE Mod_Gauss_Coefs                                                 !
   USE Solucion_Ax_b                                                   !
   USE Mod_SNEDOS                                                      !
   USE Mod_Data_Files
   USE Mod_Builder_Matrix_Arrays                                       !
   USE Mod_Prtcl_3D_Geo !, ONLY: Printel, abc, elm_geom, elm_geom2    !
   USE Mod_Correction
   USE Mod_Nodal_Interp
   USE Mod_Velocity_Menu
   USE Mod_VelFieldTRM                                                 !
!=====================================================================================================
!=====================================================================================================
!   In this par we call the libraries used in this code
!-----------------------------------------------------------------------------------------------------
!   MKL libraries
    USE mkl95_lapack
    USE mkl95_precision
    USE omp_lib
!=====================================================================================================
!   IMSL libreries.
!   USE IMSL_LIBRARIES
!   USE CS1GD_INT
!   USE lin_sol_gen_int
!   IMSL Library header file
!   INCLUDE 'link_f90_static.h'
!=====================================================================================================
```



```
!=====================================================================================================
   IMPLICIT NONE
!=====================================================================================================
!=====================================================================================================
!   Initialization of Varieables
!
!In this part of the code the variables has been clasified becouse there are a lot of them. Other reason is
!there's a necesity of modify in future, so is necesary to know the new variables and what are the obsolet or
!old variables to delete them.
!Inside these clasification there are  subcategories, fist the character variables, then integer variables and
!finally the real variables. This manner of variable initialization comes from CHAPMAN FORTRAN BOOK'S
!=====================================================================================================
!   Input Data Variables
!These variables are the variables obtained from IMPUTDATA file.
    INTEGER  :: IFlow                               !Type of flow
    INTEGER  :: ndiv                                !Level of triangulation
    INTEGER  :: mint, NGL                           !Number mint is Gauss_triangle base point number, it's used to
    INTEGER  :: steps                               !make the regular integral. NGL is used to make singular
    INTEGER  :: stoper                              !step when stop the flow
    INTEGER  :: reducer                             !step when reduces the flow to half value
    INTEGER  :: Iprec
    INTEGER  :: Ireg
    REAL (KIND = DBL)  :: boa,coa                   !these variables are used in subroutine scale_drop_adjust.
    REAL (KIND = DBL)  :: req                       !
    REAL (KIND = DBL)  :: cxp,  cyp,  czp           !
    REAL (KIND = DBL)  :: phi1, phi2, phi3          !
    REAL (KIND = DBL)  :: visc, visc2               !Viscosity of contiunuos phase and drop
    REAL (KIND = DBL)  :: gamma                     !Surface tensioon
    REAL (KIND = DBL)  :: h                         !stride
    REAL (KIND = DBL)  :: alpha, gi, alphantom      ! parameter of flow and intensity of field flow. Code appendix
    REAL (KIND = DBL)  :: omicron                   !parameter of transition between eulerian and lagrangian advance
!-----------------------------------------------------------------------------------------------------
    LOGICAL :: adjc                                 !!Adjust of center

!-----------------------------------------------------------------------------------------------------
!   Variables of Mod_Trgl_Octa
    INTEGER  :: npts, nelm                         !number of elements and points
!=====================================================================================================
!   In this part apears the variables which fix the frame coordenates. This part could be changed to other module.
!It was in 21 / Augost / 2012
!-----------------------------------------------------------------------------------------------------
!   Variables of Mod_Geo
!   In this par, there are theold variables, until this module will change
    REAL (KIND = DBL)  :: srf_area, srf_area_n
    REAL (KIND = DBL)  :: prt_vlm, prt_vlm_n
    REAL (KIND = DBL)  :: cx, cy, cz
    REAL (KIND = DBL)  :: DT, DTm, Dtmm, lmajor, bminor, wminor, Tiempo, angulo
    REAL (KIND = DBL)  :: xi, eta
    REAL (KIND = DBL)  :: Dxdxi, Dydxi, Dzdxi
    REAL (KIND = DBL)  :: Dxdet, Dydet, Dzdet
    REAL (KIND = DBL)  :: hs
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: xmom, ymom, zmom
!=====================================================================================================
!   Variables of Green's Functions  **** review the Module: Mod_Builder_Matrix_Arrays
    REAL (KIND = DBL)  :: cdg, cdt, fc           !cdg, cdt are coefficients for the dimensionallity of the flow
!   Variables of matrix array
   INTEGER,           ALLOCATABLE, DIMENSION(:)   :: ipiv
   REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: GM, RM, Uoo, Vel0, Vel1
   REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) ::    K1, K2, K3 !, K4
   REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: TM
   REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: rhs
   REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: sln
   REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: Resist
   REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: Bl, Sol
!=====================================================================================================
!   Indices and constants in the code.
    CHARACTER ( 8) :: DATE
    CHARACTER (12) :: TIME
    CHARACTER (190):: msg
    INTEGER  :: null, Nfour, Nseven
    INTEGER  :: i, j, k, i1, i2, i3, i4, i5, i6, m
```

```
    INTEGER  :: ichoose
    INTEGER  :: inelm, inelm2, jnelm, jnelm2
    INTEGER  :: mdim
    INTEGER  :: stride
    INTEGER  :: isymg, iwlpvt, istop
    INTEGER  :: md, mode, neigh_elm, lelm, index    !md I use to control the stability of mesh
    INTEGER :: ifile = 0, fstatus = 0, astatus = 0, Istp = 0, lberror = 0
    INTEGER :: sstatus=0                            !second index status
    REAL ::  CPUTime
    REAL (KIND = DBL)  :: pih, pi2, pi4, pi6, pi8
    REAL (KIND = DBL)  :: phi, shIFt, rr, rrr
    REAL (KIND = DBL)  :: thet0, thetz, thet
    REAL (KIND = DBL)  :: lmu
!-----------------------------------------------------------------------------------------------------------------
!   Other variables, in the future, there wont be nothing here. :)
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: Vel0x, Vel0y, Vel0z
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: Vel1x, Vel1y, Vel1z
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: Vel1x1, Vel1y1, Vel1z1
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: Vel1x2, Vel1y2, Vel1z2
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: Vel1x3, Vel1y3, Vel1z3
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: Vel1x4, Vel1y4, Vel1z4
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: fx_node, fy_node, fz_node
!   Influence matrix
!   REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: dfl_rm, dfl_rhs
!   REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: residual
    !REAL (KIND = DBL)  :: sum1, sum2, sum3, sum4, sum5, &
    !&                     sum6, sum7, sum8, sum9
    !REAL (KIND = DBL)  :: xhat, yhat, zhat
    !REAL (KIND = DBL)  :: fnormal
    !REAL (KIND = DBL)  :: frcx, frcy, frcz
    !REAL (KIND = DBL)  :: trqx, trqy, trqz
    !REAL (KIND = DBL)  :: trcx, trcy, trcz
    !REAL (KIND = DBL),  DIMENSION(2) :: P0, U0TRM
    !REAL (KIND = DBL)  :: wPrt, Frt1, Frt2
!=================================================================================================================
!=================================================================================================================
!   Name list are here. Today (14 / 08 / 2012), there are only 2 namelist, but only one's worked :(
!-----------------------------------------------------------------------------------------------------------------
    NAMELIST / InputData /  IFlow,                  &
    &   ndiv, boa, coa, req, cxp, cyp, czp,         &
    &   phi1, phi2, phi3,                           &
    &   mint, NGL, Ns, Np,                          &
    &   visc, visc2, gamma,                         &
    &   alpha, gi,                                  &
    &   h, steps, stoper, reducer,                  &
    &   adjc, omicron, doomsday,                    &
    &   eps, Iprec, Ireg
!   end of namelist
!=================================================================================================================
!   Name list are here. It will work :)
!=================================================================================================================
    !NAMELIST / TRMCoefs / R, w, vR, ORy, StPnt,    &
    !&                     a, de, g, M1, M2,        &
    !&                     A0, B0, C0, D0, Kc,      &
    !&                     An, Bn, Cn, Dn,          &
    !&                     lambda, shrate
!=================================================================================================================
    NAMELIST / TRMCoefs / StPnt, shrate, lambda,   &
    &                    R, ORy, de, g, a, vR,    &
    &                    w, M1, Kc,               &
    &                    A0, B0, C0, D0, M2,      &
    &                    An, Bn, Cn, Dn
!=================================================================================================================
!   Initialitation of constants
!   This part has the values of constants like, pi, null, etc.
!-----------------------------------------------------------------------------------------------------------------
    pih = 0.500*pi
    pi2 = 2.0D0*pi
    pi4 = 4.0D0*pi
    pi6 = 6.0D0*pi
    pi8 = 8.0D0*pi
!-----------------------------------------------------------------------------------------------------------------
```



```
    null = 0
    Nfour = 4
    Nseven = 7
!-----------------------------------------------------------------------------------------------------------------
    CPUTime = 0
!=================================================================================================================
!=================================================================================================================
!   Open files statments.
!   This part has the new feature in OPEN statement. IOMSG is very important.
!*****************************************************************************************************************
    OPEN(UNIT = 10, FILE = 'Prtcl_3D.nml', STATUS = 'OLD', ACTION = 'READ',&    !there is IOSMG in all open
    &    DELIM = 'QUOTE', IOSTAT = fstatus, IOMSG = msg)                       !statements
!*****************************************************************************************************************
!The firs statement is obtain the data and time.
    CALL DATE_AND_TIME(DATE, TIME)
!=================================================================================================================
!   In this part, the file Prtcl_3D.nml is opened and red. The code has two warning statements, the first is
!about the existence of file, and the second is about the file has the correct values in their variables.
!The use of this conditions is necesary to find problems with the imput data.
    IF (fstatus == 0) then
        READ(UNIT = 10, NML=InputData, IOSTAT = sstatus, IOMSG = msg)
        !-------------------------------------------------------------------------------------------------------------
        !This is the second condition, here there is a warning about the structure of file Prtcl_3D
        IF (sstatus == 0) then
            CLOSE(UNIT = 10)
        ELSE
            OPEN(UNIT = ULogi, FILE = 'Prtcl3D_Logi.log',STATUS = 'REPLACE', ACTION = 'WRITE')
            WRITE (ULogi,*) ' DATE:  ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
            WRITE (ULogi,*)'Error, little Padawan, InputData has a mistake!'
            WRITE (ULogi, *) 'First Error index = ', fstatus
            WRITE (ULogi,*)'number of error'
            WRITE (ULogi, *) 'Second Error index = ', sstatus
            WRITE (ULogi,*) ' message of error:'
            WRITE (ULogi,FMT=*) TRIM(msg)
            WRITE (ULogi,*) ' message of error ended'
            WRITE (ULogi,*)'Now, go to find the mistake, Do not see this more time!'
            WRITE (ULogi,FMT=*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
            WRITE(ULogi,*)'The Force will be with you'
            CLOSE(UNIT = 10)
            lberror = 1
        END IF
        !-------------------------------------------------------------------------------------------------------------
    ELSE
        OPEN(UNIT = ULogi, FILE = 'Prtcl3D_Logi.log',STATUS = 'REPLACE', ACTION = 'WRITE')
        WRITE (ULogi,*) ' DATE:  ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
        WRITE (ULogi,*)'Error, little Padawan, InputData has a mistake!'
        WRITE (ULogi,*)'number of error'
        WRITE (ULogi, *) 'First Error index = ', fstatus
        WRITE (ULogi,*) ' message of error:'
        WRITE (ULogi,FMT=*) TRIM(msg)
        WRITE (ULogi,*) ' message of error ended'
        WRITE (ULogi,*)'Now, go to find the mistake, Do not see this more time!'
        WRITE (ULogi,FMT=*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
        WRITE(ULogi,*)'The Force will be with you'
        CLOSE(UNIT = 10)
        WRITE (ULogi,*) ' DATE:  ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
        lberror = 1
    END IF
!=================================================================================================================
!=================================================================================================================
!   Open TRMill files statments.
!   This part has the new feature in OPEN statement. IOMSG is very important.
!*****************************************************************************************************************
    OPEN(UNIT = 10, FILE = 'TRMCoefs.nml', STATUS = 'OLD', ACTION = 'READ',&    !there is IOSMG in all open
    &    DELIM = 'QUOTE', IOSTAT = fstatus, IOMSG = msg)                       !statements
!*****************************************************************************************************************
!The firs statement is obtain the data and time.
    CALL DATE_AND_TIME(DATE, TIME)
!=================================================================================================================
!   In this part, the file Prtcl_3D.nml is opened and red. The code has two warning statements, the first is
!about the existence of file, and the second is about the file has the correct values in their variables.
!The use of this conditions is necesary to find problems with the imput data.
```

```
    IF (fstatus == 0) then
        READ(UNIT = 10, NML=TRMCoefs, IOSTAT = sstatus, IOMSG = msg)
        !---------------------------------------------------------------------------------------------------
        !This is the second condition, here there is a warning about the structure of file Prtcl_3D
        IF (sstatus == 0) then
            CLOSE(UNIT = 10)
        ELSE
            OPEN(UNIT = ULogi, FILE = 'Prtcl3D_Logi.log',STATUS = 'REPLACE', ACTION = 'WRITE')
            WRITE (ULogi,*) ' DATE:  ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
            WRITE (ULogi,*)'Error, little Padawan, TRMCoefs has a mistake!'
            WRITE (ULogi, *) 'First Error index = ', fstatus
            WRITE (ULogi,*)'number of error'
            WRITE (ULogi, *) 'Second Error index = ', sstatus
            WRITE (ULogi,*) ' message of error:'
            WRITE (ULogi,FMT=*) TRIM(msg)
            WRITE (ULogi,*) ' message of error ended'
            WRITE (ULogi,*)'Now, go to find the mistake, Do not see this more time!'
            WRITE (ULogi,FMT=*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
            WRITE(ULogi,*)'The Force will be with you'
            CLOSE(UNIT = 10)
            lberror = 1
        END IF
        !---------------------------------------------------------------------------------------------------
    ELSE
        OPEN(UNIT = ULogi, FILE = 'Prtcl3D_Logi.log',STATUS = 'REPLACE', ACTION = 'WRITE')
        WRITE (ULogi,*) ' DATE:  ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
        WRITE (ULogi,*)'Error, little Padawan, TRMCoefs has a mistake!'
        WRITE (ULogi,*)'number of error'
        WRITE (ULogi, *) 'First Error index = ', fstatus
        WRITE (ULogi,*) ' message of error:'
        WRITE (ULogi,FMT=*) TRIM(msg)
        WRITE (ULogi,*) ' message of error ended'
        WRITE (ULogi,*)'Now, go to find the mistake, Do not see this more time!'
        WRITE (ULogi,FMT=*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
        WRITE(ULogi,*)'The Force will be with you'
        CLOSE(UNIT = 10)
        WRITE (ULogi,*) ' DATE:  ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
        lberror = 1
    END IF
!=====================================================================================================
!   The next step is to have an IF structure to decide and to continue the code. The name was init, it was
!changed by bolt, because is a part of a door and is a statement for coming inside the code.
    Bolt: IF (sstatus == 0) THEN
!=====================================================================================================
!   The initial code had the condition on fstatus, because that code don´t have IOMSG option. This new code has
!the condition arround second error index called sstatus. This statement stablish a second look if ther is a
!mistake in the data input file.
!=====================================================================================================
!   The nex step is to open the file Prtcl_Dat which has the values of drop's geometry.
        OPEN(UNIT = UDat, FILE = 'Prtcl3D_Dat.dat',              &
        &        STATUS = 'REPLACE', ACTION = 'WRITE')
        WRITE (UDat,*) ' Prtcl3D_Dat.dat  '
        WRITE (UDat,*) ' DATE:  ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
        WRITE (UDat,*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
        WRITE (UDat,*) ' '
!=====================================================================================================
! prepare the file Prtcl3D_log.
        OPEN(UNIT = ULog, FILE = 'Prtcl3D_Log.log',              &
        &        STATUS = 'REPLACE', ACTION = 'WRITE')

        WRITE (ULog,*) ' DATE:  ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
        WRITE (ULog,*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
        WRITE (ULog,*) ' '
        WRITE (ULog,*) ndiv
!=====================================================================================================
! prepare the file Prtcl3D_Geo for visualization in Mathemathica.
        OPEN (UNIT = UGeo, FILE = "Prtcl3D_Geo.dat",              &
        &     STATUS = 'REPLACE', ACTION = 'WRITE')
!=====================================================================================================
! Read TRM Coefficients. Tooday (16 / 08 /2012) there is not used.
!        IF (Iflow == 3) THEN
!            OPEN(UNIT = 10, FILE = 'TRMCoefs.nml', STATUS = 'OLD',   &
```



```
!            &    ACTION = 'READ', DELIM = 'QUOTE', IOSTAT = astatus)
!            IF (astatus == 0) THEN
!                !    'Procede to read data from file'
!                READ  (UNIT = 10, NML=TRMCoefs, IOSTAT = fstatus)
!                CLOSE (UNIT = 10)
!            END IF
!        END IF
!=====================================================================================================
!=====================================================================================================
!   Triangulate the unit sphere
!   In this par there is a method todiscretize shape and generate the connectivity table
!=====================================================================================================
    CALL trgl_octa(ndiv, npts, nelm)
!    Allocation of p, n, ne and nbe occurs in this subroutine.
!=====================================================================================================
!=====================================================================================================
!   Triangulate the unit sphere
!   In this par there is a file with the geometry, the values of
!-----------------------------------------------------------------------------------------------------
!   Call the subroutine trgl_octa
!        CALL trgl_octa_read(ndiv, npts, nelm)
!    Allocation of p, n, ne and nbe occurs in this subroutine.
!=====================================================================================================
!=====================================================================================================
!   The next part computes the expantion to specified shape and equivalent radius. The idea is introduce this
!statements in a new subroutine.
        CALL drop_scale_adjust(p,     npts,   nelm, &
                              &boa,   coa,    req,  &
                              &eps,                 &
                              &cxp,   cyp,    czp, &
                              &phi1,  phi2,   phi3 )
!=====================================================================================================
!=====================================================================================================
!   The follow acction is to updated the quadratures
!-----------------------------------------------------------------------------------------------------
!   Read the quadratures. First there are an allocate statement for arrays, then the values of coefficients are
!red and finally there is the compute of quadratures over each element.
        ALLOCATE (alphaQ(nelm), betaQ(nelm), gammaQ(nelm))
        alphaQ = 0.0D0
        betaQ  = 0.0D0
        gammaQ = 0.0D0
        CALL Gauss_Legendre(NGL)                                !   Allocation of WW and ZZ
        CALL Gauss_Trgl(mint)                                   !   Allocation of xiq, etq, wq
!=====================================================================================================
!   Here we develop the advance in one step of time
!-----------------------------------------------------------------------------------------------------
!       Compute the coefficients alphaQ, betaQ, gammaQ, for the quadratic xi-eta mapping of each element
!$omp parallel
!$omp do
        DO k = 1, nelm
            i1 = n(k,1)
            i2 = n(k,2)
            i3 = n(k,3)
            i4 = n(k,4)
            i5 = n(k,5)
            i6 = n(k,6)
            CALL abc(p(i1,1), p(i1,2), p(i1,3),     &
            &       p(i2,1), p(i2,2), p(i2,3),     &
            &       p(i3,1), p(i3,2), p(i3,3),     &
            &       p(i4,1), p(i4,2), p(i4,3),     &
            &       p(i5,1), p(i5,2), p(i5,3),     &
            &       p(i6,1), p(i6,2), p(i6,3),     &
            &       alphaQ(k), betaQ(k), gammaQ(k))
        END DO
!$omp end do
!$omp end parallel
!=====================================================================================================
!*****************************************************************************************************
!   Collocation points will be placed at the element centroids
! Compute:
!  1) coordinates of collocation points
!  2) normal vector at collocation points
```

```
!-------------------------------------------------------------------------------------------------------------
!   There are the ALLOCATE statement and the initialization of values  in the arrays used to have the
!features of collocation points.
        ALLOCATE ( & x0(nelm),   y0(nelm),   z0(nelm),  &
        &          vnx0(nelm), vny0(nelm), vnz0(nelm), &
        &          vna1(6*nelm,4))
        x0   = 0.0D0
        y0   = 0.0D0
        z0   = 0.0D0
        vnx0 = 0.0D0
        vny0 = 0.0D0
        vnz0 = 0.0D0
        vna1 = 0.0D0
        xi  = 1.0D0/3.0D0
        eta = 1.0D0/3.0D0
!$omp parallel
!$omp do
!=============================================================================================================
        DO i = 1, nelm
!   will be interpolate the normal vector and features of collocation points.
            i1 = n(i,1)
            i2 = n(i,2)
            i3 = n(i,3)
            i4 = n(i,4)
            i5 = n(i,5)
            i6 = n(i,6)
!-------------------------------------------------------------------------------------------------------------
            CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
            &             p(i2,1),   p(i2,2),  p(i2,3),    &
            &             p(i3,1),   p(i3,2),  p(i3,3),    &
            &             p(i4,1),   p(i4,2),  p(i4,3),    &
            &             p(i5,1),   p(i5,2),  p(i5,3),    &
            &             p(i6,1),   p(i6,2),  p(i6,3),    &
            &             alphaQ(i), betaQ(i), gammaQ(i),  &
            &             xi,        eta,                  &
            &             x0(i),     y0(i),    z0(i),      &
            &             DxDxi,     DyDxi,    DzDxi,      &
            &             DxDet,     DyDet,    DzDet,      &
            &             vnx0(i),   vny0(i),  vnz0(i),    &
            &             hs)
!-------------------------------------------------------------------------------------------------------------
        END DO
!$omp end do
!$omp end parallel
!=============================================================================================================
! compute:
!   The surface area of the individual elements  x, y, and z moments over each element
!   The particle surface area and volume
!   The mean curvature of each element: crvmel
!   The averaged element normal vector
!-------------------------------------------------------------------------------------------------------------
        ALLOCATE (arel(nelm), crvmel(nelm), farel(nelm), &
        &         xmom(nelm), ymom(nelm), zmom(nelm))
        stride = 0
!$omp parallel
        CALL elm_geom(nelm, npts, mint,     &
        &             xmom, ymom, zmom,     &
        &             srf_area, prt_vlm,    &
        &             cx,   cy,   cz, stride)
!$omp end parallel
!        Call interp_curv( nelm,  ne,   nbe,&
!                         & crvmel)
!=============================================================================================================
        !CALL elm_geom4(nelm, npts, mint,     &
        !&             xmom, ymom, zmom,     &
        !&             srf_area, prt_vlm,    &
        !&             cx,   cy,   cz)
!=============================================================================================================
! normalize surface area and volume
      srf_area_n = srf_area/(pi4*req*req)
      prt_vlm_n  = prt_vlm/(pi4*req*req*req/3.0D0)
!-------------------------------------------------------------------------------------------------------------
```



```
!   The obteined values are writing on UData  file.
        WRITE (UDat,*)
        WRITE (UDat,*)   stride
        WRITE (UDat,110) srf_area_n
        WRITE (UDat,111) prt_vlm_n
        WRITE (UDat,112) cx,cy,cz
        WRITE (UDat,*)
        thet0 = alpha
!=============================================================================================================
!$omp parallel
        CALL D_taylor(p, npts, DT, DTm, DTmm, lmajor, bminor, wminor, angulo, cx, cy, cz, req)
        Tiempo = DBLE(stride)*h*gi
!$omp end parallel
        WRITE (UGeo,103) Tiempo, lmajor, bminor, wminor, DT, DTm, DTmm, angulo, bminor/wminor, 0.0D0
!=============================================================================================================
        CALL write_datan(nelm, stride)
        CALL write_datac(nelm, stride)
        CALL write_datak(nelm, stride)
!         CALL write_datar(nelm, stride)
        md = IFlow
        alphantom = alpha
!=============================================================================================================
        CALL interp_evn( npts,    ne,   nbe,&
        & vnx0,  vny0,  vnz0,&
        & vna)
!=============================================================================================================
!   three rows at a time,
! corresponding to the x, y, z
! components of the integral equation
!-------------------------------------------------------------------------------------------------------------
!   System size
!The system size is obtained in the next sentences, the sistem is tridimentional, therefore, the size is
!threefold element size. That becouse we have coordinates in x, y and z.
!=============================================================================================================
        Mdim = 3*nelm
        ALLOCATE(GM(Mdim), TM(Mdim,Mdim), Uoo(Mdim), Vel0(Mdim), Vel1(Mdim))!, rhs(Mdim,3), sln(Mdim,3), &
        ! &        ipiv(Mdim))
        ! initialize
        GM  = 0.0D0
        Uoo = 0.0D0
        Vel0= 0.0D0
        CALL dissociate(nelm,  Vel0,       &
                        & Vel0x, Vel0y, Vel0z)
        Vel1= 0.0D0
        TM  = 0.0D0
        CALL write_datavl(nelm, stride, Vel0x, Vel0y, Vel0z)
!=============================================================================================================
!   The coeficient lammda mu is obtained here.
        cdg = 1.0D0/(pi8*visc)
        lmu= visc2/visc
        cdt= (1.0D0-lmu)/(pi8)
!=============================================================================================================
!   Running over the matrix GM and TM
        CALL Builder_GM(GM, cdg, nelm, gamma, mint, NGL)
        CALL Builder_TM(TM, cdt, lmu, nelm, mint, NGL)
!         CALL write_datakFG(nelm, stride, GM)
!=============================================================================================================
        CALL Infinity_Flow(Iflow, alpha, gi, Uoo, stride, steps, nelm, h)
!-------------------------------------------------------------------------------------------------------------
        Vel0 = GM - Uoo
!=============================================================================================================
        CALL answer_axb1(Mdim,TM,Vel0,Vel1)
!=============================================================================================================
        CALL dissociate(nelm,  Vel1,       &
                        & Vel1x, Vel1y, Vel1z)
!=============================================================================================================
        ALLOCATE ( xel(nelm),   yel(nelm),   zel(nelm),  &
        &          pel(npts, 3))
!        ALLOCATE(K1(npts,3), K2(npts,3), K3(npts,3))
        ALLOCATE(K1(nelm), K2(nelm), K3(nelm))
        xel = 0.0D0
        yel = 0.0D0
```

```
        zel = 0.0D0
        Pel = 0.0D0
        K1  = 0.0D0
        K2  = 0.0D0
        K3  = 0.0D0
!================================================================================================================
!        CALL Euler_Method3(x0, y0, z0,&
!                        & Vel1x, Vel1y, Vel1z, &
!                        &h, nelm, omicron)
!!================================================================================================================
!        CALL interp_e( npts,    ne,    nbe,&
!        & x0, y0, z0,&
!        & p                 )
!================================================================================================================
            CALL interp_ven( npts,  ne,   nbe,&
                         & Vel1x, Vel1y, Vel1z, Vel)
!================================================================================================================
!   The RK3 starts
!================================================================================================================
    xel=x0
    yel=y0
    zel=z0
    K1 = Vel1
    pel= p
!================================================================================================================
            !CALL write_datan(nelm, stride)
            !CALL write_datac(nelm, stride)
!            CALL write_datak(nelm, stride)
            istop = doomsday
            doomsday = 1
!================================================================================================================
!   This is the master loop to advance the time in dynamic drop deformation.
!The next lines are the same over here. However, the comands will be without comments unlees will be necesary
!The Force will be with you, The year of our Lord 2013.
!================================================================================================================
        DO m = 1,  steps !, 2
!================================================================================================================
            !CALL Euler_Met5(p, 0.5D0*h, K1, npts, omicron)
!================================================================================================================
        CALL RK31(x0, y0, z0, 0.5D0*h, K1, nelm, omicron)
!================================================================================================================
        CALL interp_e( npts,    ne,    nbe,&
        & x0, y0, z0,&
        & p                 )
!================================================================================================================
            select case (md)
            case (1:11)
                if (m > stoper) then
                    Iflow = 1
                else
                    Iflow=md
                end if
            case (12)
                if ( m > reducer) then
                    Iflow = 13
                else
                    Iflow = 12
                end if
            case (16)
                if ( m > reducer) then
                    Iflow = 17
                else
                    Iflow = 16
                end if
            case (15)
                IF( m > reducer) then
                    IFlow = 15
                else
                    Iflow = 10
                end if
            case (19)
                if (m > stoper) then
```



```
                    Iflow = 13
                    alpha = alphantom
                    thet0 = ((2.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 12
                    alpha = alphantom
                    thet0 = ((-0.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 10
                    alpha = 0.0D0
                    thet0 = alpha
                end if
            case (20)
                if (m > stoper) then
                    Iflow = 17
                    alpha = alphantom
                    thet0 = ((2.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 16
                    alpha = alphantom
                    thet0 = ((-0.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 10
                    alpha = 0.0D0
                    thet0 = alpha
                end if
            case (21)
                if (m > stoper) then
                    Iflow = 16
                    alpha = alphantom
                    thet0 = ((-1.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 17
                    alpha = alphantom
                    thet0 = ((1.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 10
                    alpha = 1.0D0
                    thet0 = alpha
                end if
            case (22)
                if (m > stoper) then
                    Iflow = 72
                    alpha = alphantom
                    thet0 = ((2.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 71
                    alpha = alphantom
                    thet0 = ((-0.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 7
                    alpha = 0.0D0
                    thet0 = alpha
                end if
            case (23)
                if (m > stoper) then
                    Iflow = 71
                    alpha = alphantom
                    thet0 = ((-1.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 72
                    alpha = alphantom
                    thet0 = ((1.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
```

```
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 7
                    alpha = 1.0D0
                    thet0 = alpha
                end if
            case default
                Iflow = 10
            end select
!===================================================================================================
            CALL Corrector_vol(p, npts, prt_vlm_n, cx, cy, cz)
!===================================================================================================
!$omp parallel
!$omp do
            DO k = 1, nelm
                i1 = n(k,1)
                i2 = n(k,2)
                i3 = n(k,3)
                i4 = n(k,4)
                i5 = n(k,5)
                i6 = n(k,6)
                CALL abc(p(i1,1), p(i1,2), p(i1,3),     &
                &        p(i2,1), p(i2,2), p(i2,3),     &
                &        p(i3,1), p(i3,2), p(i3,3),     &
                &        p(i4,1), p(i4,2), p(i4,3),     &
                &        p(i5,1), p(i5,2), p(i5,3),     &
                &        p(i6,1), p(i6,2), p(i6,3),     &
                &        alphaQ(k), betaQ(k), gammaQ(k))
            END DO
!$omp end do
!$omp end parallel
!---------------------------------------------------------------------------------------------------
            x0   = 0.0D0
            y0   = 0.0D0
            z0   = 0.0D0
            vnx0 = 0.0D0
            vny0 = 0.0D0
            vnz0 = 0.0D0
            xi  = 1.0D0/3.0D0
            eta = 1.0D0/3.0D0
!===================================================================================================
!$omp parallel
!$omp do
            DO i = 1, nelm
!   will be interpolate the normal vector and features of collocation points.
                i1 = n(i,1)
                i2 = n(i,2)
                i3 = n(i,3)
                i4 = n(i,4)
                i5 = n(i,5)
                i6 = n(i,6)
!---------------------------------------------------------------------------------------------------
                CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
                &             p(i2,1),   p(i2,2),  p(i2,3),    &
                &             p(i3,1),   p(i3,2),  p(i3,3),    &
                &             p(i4,1),   p(i4,2),  p(i4,3),    &
                &             p(i5,1),   p(i5,2),  p(i5,3),    &
                &             p(i6,1),   p(i6,2),  p(i6,3),    &
                &             alphaQ(i), betaQ(i), gammaQ(i),  &
                &             xi,        eta,                  &
                &             x0(i),     y0(i),    z0(i),      &
                &             DxDxi,     DyDxi,    DzDxi,      &
                &             DxDet,     DyDet,    DzDet,      &
                &             vnx0(i),   vny0(i),  vnz0(i),    &
                &             hs)
!---------------------------------------------------------------------------------------------------
            END DO
!$omp end do
!$omp end parallel
!===================================================================================================
            CALL elm_geom(nelm, npts, mint,     &
            &             xmom, ymom, zmom,     &
```



```
            &             srf_area, prt_vlm,    &
            &             cx,   cy,   cz, stride)
            !Call interp_curv( nelm,  ne,   nbe,&
            !                 & crvmel)
!===================================================================================================
! normalize surface area and volume
            srf_area_n = srf_area /(pi4*req*req)
            prt_vlm_n  = prt_vlm  /(pi4*req*req*req/3.0D0)
!===================================================================================================
            CALL interp_evn( npts,    ne,   nbe,&
            & vnx0,  vny0,  vnz0,&
            & vna)
!===================================================================================================
            GM  = 0.0D0
            Uoo = 0.0D0
            Vel0= 0.0D0
            Vel1= 0.0D0
            TM  = 0.0D0
!===================================================================================================
            CALL Builder_GM(GM, cdg, nelm, gamma, mint, NGL)
            CALL Builder_TM(TM, cdt, lmu, nelm, mint, NGL)
!===================================================================================================
            CALL Infinity_Flow(Iflow, thet0, gi, Uoo, m, steps, nelm, h)
            Vel0 = GM - Uoo
!===================================================================================================
            CALL answer_axb1(Mdim,TM,Vel0,Vel1)
!===================================================================================================
            CALL dissociate(nelm,  Vel1,       &
                        & Vel1x, Vel1y, Vel1z)
!===================================================================================================
            !CALL interp_ven( npts,  ne,   nbe,&
            !              & Vel1x, Vel1y, Vel1z, Vel)
!===================================================================================================
            K2 = Vel1
            p  = pel
            x0 = xel
            y0 = yel
            z0 = zel
!===================================================================================================
!            CALL Euler_Met6(p, h, K1, K2, npts, omicron)
!===================================================================================================
        CALL RK32(x0, y0, z0, h, K1, K2, nelm, omicron)
!===================================================================================================
        CALL interp_e( npts,    ne,    nbe,&
        & x0, y0, z0,&
        & p                 )
!===================================================================================================
            select case (md)
            case (1:11)
                if (m > stoper) then
                    Iflow = 1
                else
                    Iflow=md
                end if
            case (12)
                if ( m > reducer) then
                    Iflow = 13
                else
                    Iflow = 12
                end if
            case (16)
                if ( m > reducer) then
                    Iflow = 17
                else
                    Iflow = 16
                end if
            case (15)
                IF( m > reducer) then
                    IFlow = 15
                else
                    Iflow = 10
                end if
```

```
            case (19)
                if (m > stoper) then
                    Iflow = 13
                    alpha = alphantom
                    thet0 = ((2.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 12
                    alpha = alphantom
                    thet0 = ((-0.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 10
                    alpha = 0.0D0
                    thet0 = alpha
                end if
            case (20)
                if (m > stoper) then
                    Iflow = 17
                    alpha = alphantom
                    thet0 = ((2.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 16
                    alpha = alphantom
                    thet0 = ((-0.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 10
                    alpha = 0.0D0
                    thet0 = alpha
                end if
            case (21)
                if (m > stoper) then
                    Iflow = 16
                    alpha = alphantom
                    thet0 = ((-1.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 17
                    alpha = alphantom
                    thet0 = ((1.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 10
                    alpha = 1.0D0
                    thet0 = alpha
                end if
            case (22)
                if (m > stoper) then
                    Iflow = 72
                    alpha = alphantom
                    thet0 = ((2.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 71
                    alpha = alphantom
                    thet0 = ((-0.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 7
                    alpha = 0.0D0
                    thet0 = alpha
                end if
            case (23)
                if (m > stoper) then
                    Iflow = 71
                    alpha = alphantom
                    thet0 = ((-1.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 72
```



```
                    alpha = alphantom
                    thet0 = ((1.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 7
                    alpha = 1.0D0
                    thet0 = alpha
                end if
            case default
                Iflow = 10
            end select
!===============================================================================================================
            CALL Corrector_vol(p, npts, prt_vlm_n, cx, cy, cz)
!===============================================================================================================
!$omp parallel
!$omp do
            DO k = 1, nelm
                i1 = n(k,1)
                i2 = n(k,2)
                i3 = n(k,3)
                i4 = n(k,4)
                i5 = n(k,5)
                i6 = n(k,6)
                CALL abc(p(i1,1), p(i1,2), p(i1,3),     &
                &        p(i2,1), p(i2,2), p(i2,3),     &
                &        p(i3,1), p(i3,2), p(i3,3),     &
                &        p(i4,1), p(i4,2), p(i4,3),     &
                &        p(i5,1), p(i5,2), p(i5,3),     &
                &        p(i6,1), p(i6,2), p(i6,3),     &
                &        alphaQ(k), betaQ(k), gammaQ(k))
            END DO
!$omp end do
!$omp end parallel
!---------------------------------------------------------------------------------------------------------------
            x0   = 0.0D0
            y0   = 0.0D0
            z0   = 0.0D0
            vnx0 = 0.0D0
            vny0 = 0.0D0
            vnz0 = 0.0D0
            xi  = 1.0D0/3.0D0
            eta = 1.0D0/3.0D0
!===============================================================================================================
!$omp parallel
!$omp do
            DO i = 1, nelm
!   will be interpolate the normal vector and features of collocation points.
                i1 = n(i,1)
                i2 = n(i,2)
                i3 = n(i,3)
                i4 = n(i,4)
                i5 = n(i,5)
                i6 = n(i,6)
!---------------------------------------------------------------------------------------------------------------
                CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
                &             p(i2,1),   p(i2,2),  p(i2,3),    &
                &             p(i3,1),   p(i3,2),  p(i3,3),    &
                &             p(i4,1),   p(i4,2),  p(i4,3),    &
                &             p(i5,1),   p(i5,2),  p(i5,3),    &
                &             p(i6,1),   p(i6,2),  p(i6,3),    &
                &             alphaQ(i), betaQ(i), gammaQ(i),  &
                &             xi,        eta,                  &
                &             x0(i),     y0(i),    z0(i),      &
                &             DxDxi,     DyDxi,    DzDxi,      &
                &             DxDet,     DyDet,    DzDet,      &
                &             vnx0(i),   vny0(i),  vnz0(i),    &
                &             hs)
!---------------------------------------------------------------------------------------------------------------
            END DO
!$omp end do
!$omp end parallel
!===============================================================================================================
```

```
            CALL elm_geom(nelm, npts, mint,     &
            &             xmom, ymom, zmom,     &
            &             srf_area, prt_vlm,    &
            &             cx,   cy,   cz, stride)
            Call interp_curv( nelm,  ne,   nbe,&
                             & crvmel)
!==============================================================================================================
! normalize surface area and volume
            srf_area_n = srf_area /(pi4*req*req)
            prt_vlm_n  = prt_vlm  /(pi4*req*req*req/3.0D0)
!==============================================================================================================
            CALL interp_evn( npts,    ne,   nbe,&
            & vnx0,  vny0,  vnz0,&
            & vna)
!==============================================================================================================
            GM  = 0.0D0
            Uoo = 0.0D0
            Vel0= 0.0D0
            Vel1= 0.0D0
            TM  = 0.0D0
!==============================================================================================================
            CALL Builder_GM(GM, cdg, nelm, gamma, mint, NGL)
            CALL Builder_TM(TM, cdt, lmu, nelm, mint, NGL)
!==============================================================================================================
            CALL Infinity_Flow(Iflow, thet0, gi, Uoo, m, steps, nelm, h)
            Vel0 = GM - Uoo
!==============================================================================================================
            CALL answer_axb1(Mdim,TM,Vel0,Vel1)
!==============================================================================================================
            CALL dissociate(nelm,  Vel1,       &
                        & Vel1x, Vel1y, Vel1z)
!==============================================================================================================
            !CALL interp_ven( npts,  ne,   nbe,&
            !             & Vel1x, Vel1y, Vel1z, Vel)
!==============================================================================================================
            K3 = Vel1
            p  = pel
            x0 = xel
            y0 = yel
            z0 = zel
!==============================================================================================================
!            CALL KMK3(p, K1, K2, K3, npts, h)
            CALL RMK3(x0, y0, z0, &
                    & K1, K2, K3, nelm, h)
 !==============================================================================================================
        CALL interp_e( npts,    ne,    nbe,&
        & x0, y0, z0,&
        & p                  )
           pel = p
!==============================================================================================================
            select case (md)
            case (1:11)
                if (m > stoper) then
                    Iflow = 1
                else
                    Iflow=md
                end if
            case (12)
                if ( m > reducer) then
                    Iflow = 13
                else
                    Iflow = 12
                end if
            case (16)
                if ( m > reducer) then
                    Iflow = 17
                else
                    Iflow = 16
                end if
            case (15)
                IF( m > reducer) then
                    IFlow = 15
```



```
                else
                    Iflow = 10
                end if
            case (18)
                IF( m > stoper) then
                    IFlow = 1
                else
                    Iflow = 18
                end if
            case (19)
                if (m > stoper) then
                    Iflow = 13
                    alpha = alphantom
                    thet0 = ((2.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 12
                    alpha = alphantom
                    thet0 = ((-0.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 18
                    alpha = 0.0D0
                    thet0 = alpha
                end if
            case (20)
                if (m > stoper) then
                    Iflow = 17
                    alpha = alphantom
                    thet0 = ((2.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 16
                    alpha = alphantom
                    thet0 = ((-0.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 10
                    alpha = 0.0D0
                    thet0 = alpha
                end if
            case (21)
                if (m > stoper) then
                    Iflow = 16
                    alpha = alphantom
                    thet0 = ((-1.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 17
                    alpha = alphantom
                    thet0 = ((1.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 10
                    alpha = 1.0D0
                    thet0 = alpha
                end if
            case (22)
                if (m > stoper) then
                    Iflow = 72
                    alpha = alphantom
                    thet0 = ((2.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 71
                    alpha = alphantom
                    thet0 = ((-0.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 7
                    alpha = 0.0D0
                    thet0 = alpha
```

```
                end if
            case (23)
                if (m > stoper) then
                    Iflow = 71
                    alpha = alphantom
                    thet0 = ((-1.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                elseif (m > reducer) then
                    Iflow = 72
                    alpha = alphantom
                    thet0 = ((1.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))
                    !thet0 = ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride)))
                else
                    Iflow = 7
                    alpha = 1.0D0
                    thet0 = alpha
                end if
            case default
                Iflow = 10
            end select
!===============================================================================================================
            CALL Corrector_vol(p, npts, prt_vlm_n, cx, cy, cz)
!===============================================================================================================
!$omp parallel
!$omp do
            DO k = 1, nelm
                i1 = n(k,1)
                i2 = n(k,2)
                i3 = n(k,3)
                i4 = n(k,4)
                i5 = n(k,5)
                i6 = n(k,6)
                CALL abc(p(i1,1), p(i1,2), p(i1,3),     &
                &        p(i2,1), p(i2,2), p(i2,3),     &
                &        p(i3,1), p(i3,2), p(i3,3),     &
                &        p(i4,1), p(i4,2), p(i4,3),     &
                &        p(i5,1), p(i5,2), p(i5,3),     &
                &        p(i6,1), p(i6,2), p(i6,3),     &
                &        alphaQ(k), betaQ(k), gammaQ(k))
            END DO
!$omp end do
!$omp end parallel
!---------------------------------------------------------------------------------------------------------------
            x0   = 0.0D0
            y0   = 0.0D0
            z0   = 0.0D0
            vnx0 = 0.0D0
            vny0 = 0.0D0
            vnz0 = 0.0D0
            xi  = 1.0D0/3.0D0
            eta = 1.0D0/3.0D0
!===============================================================================================================
!$omp parallel
!$omp do
            DO i = 1, nelm
!   will be interpolate the normal vector and features of collocation points.
                i1 = n(i,1)
                i2 = n(i,2)
                i3 = n(i,3)
                i4 = n(i,4)
                i5 = n(i,5)
                i6 = n(i,6)
!---------------------------------------------------------------------------------------------------------------
                CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
                &             p(i2,1),   p(i2,2),  p(i2,3),    &
                &             p(i3,1),   p(i3,2),  p(i3,3),    &
                &             p(i4,1),   p(i4,2),  p(i4,3),    &
                &             p(i5,1),   p(i5,2),  p(i5,3),    &
                &             p(i6,1),   p(i6,2),  p(i6,3),    &
                &             alphaQ(i), betaQ(i), gammaQ(i),  &
                &             xi,        eta,                  &
                &             x0(i),     y0(i),    z0(i),      &
```



```
                &             DxDxi,     DyDxi,    DzDxi,      &
                &             DxDet,     DyDet,    DzDet,      &
                &             vnx0(i),   vny0(i),  vnz0(i),    &
                &             hs)
!---------------------------------------------------------------------------------------------------------------
            END DO
!$omp end do
!$omp end parallel
!===============================================================================================================
            CALL elm_geom(nelm, npts, mint,     &
            &             xmom, ymom, zmom,     &
            &             srf_area, prt_vlm,    &
            &             cx,   cy,   cz, stride)
            !Call interp_curv( nelm,  ne,   nbe,&
            !                 & crvmel)
!===============================================================================================================
! normalize surface area and volume
            srf_area_n = srf_area /(pi4*req*req)
            prt_vlm_n  = prt_vlm  /(pi4*req*req*req/3.0D0)
!===============================================================================================================
!   The obtained values are writing on UData  file.
            WRITE (UDat,*)
            WRITE (UDat,107) m
            WRITE (UDat,110) srf_area_n
            WRITE (UDat,111) prt_vlm_n
            WRITE (UDat,112) cx,cy,cz
            WRITE (UDat,*) md, Iflow
            WRITE (UDat,*)
!===============================================================================================================
            CALL D_taylor(p, npts, DT, DTm, DTmm, lmajor, bminor, wminor, angulo, cx, cy, cz, req)
            Tiempo = DBLE(m)*h*gi
            WRITE (UGeo,103) Tiempo, lmajor, bminor, wminor, DT, DTm, DTmm, angulo, bminor/wminor, thet0
!===============================================================================================================
            CALL interp_evn( npts,    ne,   nbe,&
            & vnx0,  vny0,  vnz0,&
            & vna)
!===============================================================================================================
            GM  = 0.0D0
            Uoo = 0.0D0
            Vel0= 0.0D0
            Vel1= 0.0D0
            TM  = 0.0D0
!===============================================================================================================
            CALL Builder_GM(GM, cdg, nelm, gamma, mint, NGL)
            CALL Builder_TM(TM, cdt, lmu, nelm, mint, NGL)
!===============================================================================================================
            CALL Infinity_Flow(Iflow, thet0, gi, Uoo, m, steps, nelm, h)
            Vel0 = GM - Uoo
!===============================================================================================================
            CALL answer_axb1(Mdim,TM,Vel0,Vel1)
!===============================================================================================================
            CALL dissociate(nelm,  Vel1,       &
                        & Vel1x, Vel1y, Vel1z)
!===============================================================================================================
            !CALL interp_ven( npts,  ne,   nbe,&
            !             & Vel1x, Vel1y, Vel1z, Vel)
!===============================================================================================================
            K1 = Vel1
            xel= x0
            yel= y0
            zel= z0
!===============================================================================================================
                stride = m
                if (doomsday == istop) then
                    CALL write_datan(nelm, stride)
                    CALL write_datac(nelm, stride)
                    CALL write_datak(nelm, stride)
                    CALL write_datavl(nelm, stride, Vel1x, Vel1y, Vel1z)
                    doomsday=1
                else
                    doomsday = doomsday + 1
                end if
```

```
!===================================================================================================
        END DO
!===================================================================================================
        CALL CPU_TIME(CPUTime)
        WRITE (UDat,*) ' '
        WRITE (UDat,*) ' CPU Time: ', CPUTime, 'seconds'
        WRITE (ULog,*) ' CPU Time: ', CPUTime, 'seconds'
        i1      = FLOOR(CPUTime/3600)
        CPUTime = CPUTime - i1*3600
        i2      = FLOOR(CPUTime/60)
        CPUTime = CPUTime - i2*60
        WRITE (UDat,120) i1, i2, NINT(CPUTime)
        WRITE (ULog,120) i1, i2, NINT(CPUTime)
        CALL DATE_AND_TIME(DATE, TIME)
        WRITE (UDat,*) ' DATE:  ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
        WRITE (UDat,*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
        WRITE (UDat,*) ' '
        WRITE (ULog,*) ' DATE:  ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
        WRITE (ULog,*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
        WRITE (UDat,*) ' '
        WRITE (UDat,*) ' stime = ', 0
        WRITE (UDat,*) ' '
        WRITE (UDat,*) ' Status = ', 0
        CLOSE (UGeo)
        CLOSE (UDat)
!===================================================================================================
!   There is the option if there is wrong whit the input data file.
    ELSE Bolt
!===================================================================================================
        OPEN(UNIT = ULog, FILE = 'Prtcl3D_Log.log',              &
        &        STATUS = 'REPLACE', ACTION = 'WRITE')
        WRITE (ULog, *) 'Warning, InputData File unavailable!'
        WRITE (ULog, *) 'Warning, InputData File unavailable!'
        WRITE (ULog, *) ' or another Input File is missing!'
        WRITE (ULog, *) ' Error1 = ', lberror
        WRITE (ULog, *) ' Error  = ', fstatus
        WRITE (ULog,*) ' message of error', TRIM(msg)
        OPEN(UNIT = ULogi, FILE = 'Prtcl3D_Logi.log',STATUS = 'REPLACE', ACTION = 'WRITE')
        WRITE (ULogi,*) ' DATE:  ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
        WRITE (ULogi,*)'Error, little Padawan, InputData has a mistake!'
        WRITE (ULogi, *) 'First Error index = ', fstatus
        WRITE (ULogi,*)'number of error'
        WRITE (ULogi, *) 'Second Error index = ', sstatus
        WRITE (ULogi,*) ' message of error:'
        WRITE (ULogi,FMT=*) TRIM(msg)
        WRITE (ULogi,*) ' message of error ended'
        WRITE (ULogi,*)'Now, go to find the mistake, Do not see this more time!'
        WRITE (ULogi,FMT=*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
        WRITE(ULogi,*)'The Force will be with you'
        CLOSE(UNIT = 10)
            lberror = 1
    END IF Bolt
!===================================================================================================
! The code finished here
!===================================================================================================
!   Arguments to different kind to save datas. There are formats to write.
    100 FORMAT(1x,i4,10(1x,ES24.16))
    101 FORMAT(10(1x,ES24.16))
    102 FORMAT(10(1x,ES24.16))
    103 FORMAT(10(1x,ES24.16))
    104 FORMAT(10(1x,ES24.16))
    105 FORMAT(10(1x,ES24.16))
    106 FORMAT(1x,ES24.16)
    107 FORMAT(8(1x,I10))
    108 FORMAT(4(1x,I10))
    110 FORMAT(" Surface Area :",ES24.16)
    111 FORMAT(" Volume       :",ES24.16)
    112 FORMAT(" Centroid     :",3(ES24.16))
    130 FORMAT(" Determinant =",ES24.16)
    200 FORMAT(100(1x,ES24.16))
    201 FORMAT(10(1x,ES24.16))
    205 FORMAT(10(1x,ES24.16))
```



```
!   210 FORMAT(1x,i4,10(1x,ES24.16))
    120 FORMAT (1X, ' CPU Time: ', I4,  ':', I2.2, ':',I2.2)
!===================================================================================================
END PROGRAM Prtcl_3D
```

```fortran
MODULE Mod_SharedVars
!==========================================================================================================
! : Version: 0.7 created on  -- / III /2010
!                                                                                 Marco Antonio Reyes Huesca
!----------------------------------------------------------------------------------------------------------
!   Version: 1.0 created on 09 / 08 / 2012
!                                                                                    Alfredo Sanjuan Sanjuan
!==========================================================================================================
!   This module was made to keep the variables which are common in this code. In this module, you wold see the
!features of DBL which is a value of real kind. The variables were established in order similar in main code.
!==========================================================================================================
   IMPLICIT NONE
!==========================================================================================================
!
!   As was mencioned above, in this place was kept the variable DBL. DBL is a variable which permit the code to
!have the same value in all real variables. This decision is very important because it gives versatility at the
!code, in other words, it's possible to use this code in diferents PC's and it doesn't matters the kind of that
!PC, the value will be the same in the space asigned to real variables.
!----------------------------------------------------------------------------------------------------------
!   The kind number is explained in Chapman books, the value 8 refers to a system of 64x
    INTEGER, PARAMETER :: DBL = 8                   !SELECTED_REAL_KIND(p=15, r = 300), alternative value
!----------------------------------------------------------------------------------------------------------
!   This places will be the value of pi number.
    REAL (KIND = DBL), PARAMETER :: Pi = 3.14159265358979323846264338327950D0
!----------------------------------------------------------------------------------------------------------
!   This place ther is the value of epsilon number. This number helps to discriminate small values and assigned
!zero to them
    REAL (KIND = DBL) :: epsilon, eps
!==========================================================================================================
!==========================================================================================================
!   Common blocks have differents values or as this case similar variables. Below this explanation you would
!find differents  common variables used in more than one modules. The variables appear in order of comes in the
!code.
!----------------------------------------------------------------------------------------------------------
!   Input Data
    INTEGER :: Ns, Np, doomsday
!==========================================================================================================
!   Variables of Two-Roll Mill
    INTEGER, PARAMETER :: NCs = 100                         !Number of coefficients to made the sumatories
    REAL (KIND = DBL), DIMENSION(2) :: R, w, vR             !Radii of Mills, w is the radial velocity?
    REAL (KIND = DBL), DIMENSION(2) :: ORy                  !Distance from origin to mill centers (direction y)
    REAL (KIND = DBL), DIMENSION(2) :: StPnt                !Stagnation point coordinates (x,y)
    REAL (KIND = DBL) :: a, de, g, M1, M2                   !Geometry Data: a is the distance between lines of
                                                            !foci, de is the mill ceter distance, g is the gap
    REAL (KIND = DBL) :: A0, B0, C0, D0, Kc                 !First term coefficients and parameter K
    REAL (KIND = DBL), DIMENSION(NCs) :: An, Bn, Cn, Dn     !Coefficients
    REAL (KIND = DBL) :: lambda, shrate                     ! Flow Type parameter and shear rate.
!==========================================================================================================
!   Module trgl_octa
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: p, pel
    INTEGER, ALLOCATABLE, DIMENSION(:,:) :: ne
    INTEGER, ALLOCATABLE, DIMENSION(:,:) :: n, nbe, nbe1
!----------------------------------------------------------------------------------------------------------
!   Module Gauss_Coefs
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: zz, ww !quadrature weigths of Gauss-Legendre quadrature
    REAL (KIND = DBL) :: wall
!----------------------------------------------------------------------------------------------------------
!   Module Prtcl_3D_Geo
!   Quadratic xi-eta isoparametric mapping
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: alphaQ, betaQ, gammaQ !quadrature weigths of quadrature
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: xiq, etq, wq          !parameters xi-eta mapping
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: arel, farel           !area
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: crvmel                !curvature media
!----------------------------------------------------------------------------------------------------------
!   Module Prtcl_3D_sf, sfs, slp and dlp
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: x0, y0, z0            !collocation point coordinates
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: xel, yel, zel         !collocation point coordinates
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: vnx0, vny0, vnz0      !normal vector point coordinates
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:)   :: vnx02, vny02, vnz02      !normal vector point coordinates
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: vna, Vel !, vna1      !normal vector point coordinates
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: vna1                  !normal vector point coordinates
!----------------------------------------------------------------------------------------------------------
```



```fortran
!   Text types
    INTEGER :: UDat = 80
    INTEGER :: UGeo = 85
    INTEGER :: UDef = 86
    INTEGER :: ULog = 89
    INTEGER :: ULogi= 90
    INTEGER :: UVel = 91
    INTEGER :: UCtr = 95
    INTEGER :: DelF = 92
    INTEGER :: UTmp = 93
    INTEGER :: UDf2 = 87
!==========================================================================================================
END MODULE Mod_SharedVars
```

```fortran
MODULE Mod_Trgl_Octa
!====================================================================================================
!   Version: 0.5 created on  26 / IX / 2007
!                                                                                      C. Pozrikidis
!----------------------------------------------------------------------------------------------------
!   Version: 0.7 created on  -- / III /2010
!                                                                          Marco Antonio Reyes Huesca
!----------------------------------------------------------------------------------------------------
!   Version: 1.0 created on 20 / 08 / 2012
!                                                                             Alfredo Sanjuan Sanjuan
!====================================================================================================
!====================================================================================================
!   This module makes the geometry of the drop. At the beginning, the dorp started as an octahedron, which has
!eight curved faces, then it is divided in many elements as you can see in the subroutine Trgl_Octa.
!====================================================================================================
    CONTAINS
!====================================================================================================
    SUBROUTINE trgl_octa(ndiv, npts, nelm)
!====================================================================================================
!   Triangulation of the unit sphere by subdividing a regular octahedron into six-node quadratic triangles.
!----------------------------------------------------------------------------------------------------
!   Variables
!----------------------------------------------------------------------------------------------------
!   nelm ........................ number of surface elements
!   npts ........................ number of nodal points
!   x(i,j), y(i,j), z(i,j) ...... Cartesian coords of point j on element i
!   p(i,j) ...................... (x,y,z) coords. of surface node labeled i  (j=1,2,3)
!                   x = p(i,1)
!                   y = p(i,2)
!                   z = p(i,3)
!   n,j(i) ...................... node number of point j on element i, where j=1,...,6
!   ne(i,j) ..................... ne(i,1) is the number of elements touching node i.
!                                 ne(i,2:ne(i,1)) are the corresponding element labels
!   nbe(i,j) .................... label of element sharing side j of element i where j = 1, 2, 3
!   ndiv ....  level of discretization of starting octahedron (0-ndiv)
!====================================================================================================
!====================================================================================================
!   Modules used in this subroutine.
    USE Mod_SharedVars, ONLY: DBL, ULog, p, ne, n, nbe, nbe1
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables IN/OUT
!----------------------------------------------------------------------------------------------------
    INTEGER,   INTENT(IN)   :: ndiv
    INTEGER,   INTENT(OUT)  :: npts, nelm
!----------------------------------------------------------------------------------------------------
!   Variables used inside the subroutine
    INTEGER :: IStat = 0, IFlag
    INTEGER :: fstatus
    INTEGER :: nsa
    INTEGER :: i, j, k, l, icount, jcount, kcount, num
    INTEGER :: istatp                                               ! Status: 0 for success
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) ::  x,  y,  z
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: xn, yn, zn
    REAL (KIND = DBL), PARAMETER :: eps=0.0000001
    REAL (KIND = DBL) :: r
    REAL (KIND = DBL), DIMENSION(3) :: ptm
    TYPE :: Ps
        REAL (KIND = DBL), DIMENSION(3) :: px
        TYPE (Ps), POINTER :: pp
    END TYPE Ps
    TYPE (Ps), POINTER :: head                                      ! Pointer to head of list
    TYPE (Ps), POINTER :: ptr                                       ! Temporary pointer
    TYPE (Ps), POINTER :: tail                                      ! Pointer to tail of list
!====================================================================================================
!====================================================================================================
!   Initial octahedron (0'th level discretization (8 elements)). In this part, nodes are set manually.
   nelm = 8
    IF (ndiv > 0) THEN
        nsa  = nelm*(4**ndiv)
    ELSE
```



```fortran
        nsa  = nelm
    END IF
!----------------------------------------------------------------------------------------------------
!   Allocation of all arrays wiht the information of drop's geometry
    ALLOCATE( x(nsa, 6),  y(nsa, 6),  z(nsa, 6),     &
    &        xn(nsa, 6), yn(nsa, 6), zn(nsa, 6), STAT = IStat)
!----------------------------------------------------------------------------------------------------
!   Conditional to have the errors, if there is any problem in allocation statement
    IF (IStat == 0) THEN
        x  = 0.0D0
        y  = 0.0D0
        z  = 0.0D0
        xn = 0.0D0
        yn = 0.0D0
        zn = 0.0D0
        p  = 0.0D0
        n   = 0
        ne  = 0
        nbe = 0
        fstatus = 0
!----------------------------------------------------------------------------------------------------
!   Corner points
!----------------------------------------------------------------------------------------------------
!   Upper half of xz plane
        x(1,1)= 0.0
        y(1,1)= 0.0
        z(1,1)= 1.0
!----------------------------------------------------------------------------------------------------
        x(1,2)= 1.0
        y(1,2)= 0.0
        z(1,2)= 0.0
!----------------------------------------------------------------------------------------------------
        x(1,3)= 0.0
        y(1,3)= 1.0
        z(1,3)= 0.0
!----------------------------------------------------------------------------------------------------
        x(5,1)= 1.0
        y(5,1)= 0.0
        z(5,1)= 0.0
!----------------------------------------------------------------------------------------------------
        x(5,2)= 0.0
        y(5,2)= 0.0
        z(5,2)=-1.0
!----------------------------------------------------------------------------------------------------
        x(5,3)= 0.0
        y(5,3)= 1.0
        z(5,3)= 0.0
!----------------------------------------------------------------------------------------------------
        x(6,1)= 0.0
        y(6,1)= 0.0
        z(6,1)=-1.0
!----------------------------------------------------------------------------------------------------
        x(6,2)=-1.0
        y(6,2)= 0.0
        z(6,2)= 0.0
!----------------------------------------------------------------------------------------------------
        x(6,3)= 0.0
        y(6,3)= 1.0
        z(6,3)= 0.0
!----------------------------------------------------------------------------------------------------
        x(2,1)=-1.0
        y(2,1)= 0.0
        z(2,1)= 0.0
!----------------------------------------------------------------------------------------------------
        x(2,2)= 0.0
        y(2,2)= 0.0
        z(2,2)= 1.0
!----------------------------------------------------------------------------------------------------
        x(2,3)= 0.0
        y(2,3)= 1.0
        z(2,3)= 0.0
!----------------------------------------------------------------------------------------------------
```

```fortran
!   Corner points lower half xz plane
        x(4,1)= 0.0
        y(4,1)= 0.0
        z(4,1)= 1.0
!----------------------------------------------------------------------------------------------------------
        x(4,2)= 0.0
        y(4,2)=-1.0
        z(4,2)= 0.0
!----------------------------------------------------------------------------------------------------------
        x(4,3)= 1.0
        y(4,3)= 0.0
        z(4,3)= 0.0
!----------------------------------------------------------------------------------------------------------
        x(8,1)= 1.0
        y(8,1)= 0.0
        z(8,1)= 0.0
!----------------------------------------------------------------------------------------------------------
        x(8,2)= 0.0
        y(8,2)=-1.0
        z(8,2)= 0.0
!----------------------------------------------------------------------------------------------------------
        x(8,3)= 0.0
        y(8,3)= 0.0
        z(8,3)=-1.0
!----------------------------------------------------------------------------------------------------------
        x(7,1)= 0.0
        y(7,1)= 0.0
        z(7,1)=-1.0
!----------------------------------------------------------------------------------------------------------
        x(7,2)= 0.0
        y(7,2)=-1.0
        z(7,2)= 0.0
!----------------------------------------------------------------------------------------------------------
        x(7,3)=-1.0
        y(7,3)= 0.0
        z(7,3)= 0.0
!----------------------------------------------------------------------------------------------------------
        x(3,1)=-1.0
        y(3,1)= 0.0
        z(3,1)= 0.0
!----------------------------------------------------------------------------------------------------------
        x(3,2)= 0.0
        y(3,2)=-1.0
        z(3,2)= 0.0
!----------------------------------------------------------------------------------------------------------
        x(3,3)= 0.0
        y(3,3)= 0.0
        z(3,3)= 1.0
!----------------------------------------------------------------------------------------------------------
!   compute the mid-points of the sides numbered 4, 5, 6
        FORALL (i = 1:nelm)
!----------------------------------------------------------------------------------------------------------
            x(i,4)= 0.5*(x(i,1)+x(i,2))
            y(i,4)= 0.5*(y(i,1)+y(i,2))
            z(i,4)= 0.5*(z(i,1)+z(i,2))
!----------------------------------------------------------------------------------------------------------
            x(i,5)= 0.5*(x(i,2)+x(i,3))
            y(i,5)= 0.5*(y(i,2)+y(i,3))
            z(i,5)= 0.5*(z(i,2)+z(i,3))
!----------------------------------------------------------------------------------------------------------
            x(i,6)= 0.5*(x(i,3)+x(i,1))
            y(i,6)= 0.5*(y(i,3)+y(i,1))
            z(i,6)= 0.5*(z(i,3)+z(i,1))
!----------------------------------------------------------------------------------------------------------
        END FORALL
!----------------------------------------------------------------------------------------------------------
!   Compute node coordinates on each element for discretization levels 1 through ndiv
!----------------------------------------------------------------------------------------------------------
!   Condition to divide the initial octahedron from 8 elements to (8*4**ndiv) elements
        IF (ndiv > 0) THEN
            DO i = 1, ndiv
                num = 1
```



```fortran
!----------------------------------------------------------------------------------------------------------
!   Loop over old elements to divide every elemento in four new elements
                DO j = 1, nelm
!----------------------------------------------------------------------------------------------------------
!   Assign corner points to sub-elements
!----------------------------------------------------------------------------------------------------------
!   First sub-element
                    xn(num,1)= x(j,1)
                    yn(num,1)= y(j,1)
                    zn(num,1)= z(j,1)
!----------------------------------------------------------------------------------------------------------
                    xn(num,2)= x(j,4)
                    yn(num,2)= y(j,4)
                    zn(num,2)= z(j,4)
!----------------------------------------------------------------------------------------------------------
                    xn(num,3)= x(j,6)
                    yn(num,3)= y(j,6)
                    zn(num,3)= z(j,6)
!----------------------------------------------------------------------------------------------------------
                    xn(num,4)= 0.5*(xn(num,1)+xn(num,2))
                    yn(num,4)= 0.5*(yn(num,1)+yn(num,2))
                    zn(num,4)= 0.5*(zn(num,1)+zn(num,2))
!----------------------------------------------------------------------------------------------------------
                    xn(num,5)= 0.5*(xn(num,2)+xn(num,3))
                    yn(num,5)= 0.5*(yn(num,2)+yn(num,3))
                    zn(num,5)= 0.5*(zn(num,2)+zn(num,3))
!----------------------------------------------------------------------------------------------------------
                    xn(num,6)= 0.5*(xn(num,3)+xn(num,1))
                    yn(num,6)= 0.5*(yn(num,3)+yn(num,1))
                    zn(num,6)= 0.5*(zn(num,3)+zn(num,1))
!----------------------------------------------------------------------------------------------------------
!   Second sub-element
                    xn(num+1,1)= x(j,4)
                    yn(num+1,1)= y(j,4)
                    zn(num+1,1)= z(j,4)
!----------------------------------------------------------------------------------------------------------
                    xn(num+1,2)= x(j,2)
                    yn(num+1,2)= y(j,2)
                    zn(num+1,2)= z(j,2)
!----------------------------------------------------------------------------------------------------------
                    xn(num+1,3)= x(j,5)
                    yn(num+1,3)= y(j,5)
                    zn(num+1,3)= z(j,5)
!----------------------------------------------------------------------------------------------------------
                    xn(num+1,4)= 0.5*(xn(num+1,1)+xn(num+1,2))
                    yn(num+1,4)= 0.5*(yn(num+1,1)+yn(num+1,2))
                    zn(num+1,4)= 0.5*(zn(num+1,1)+zn(num+1,2))
!----------------------------------------------------------------------------------------------------------
                    xn(num+1,5)= 0.5*(xn(num+1,2)+xn(num+1,3))
                    yn(num+1,5)= 0.5*(yn(num+1,2)+yn(num+1,3))
                    zn(num+1,5)= 0.5*(zn(num+1,2)+zn(num+1,3))
!----------------------------------------------------------------------------------------------------------
                    xn(num+1,6)= 0.5*(xn(num+1,3)+xn(num+1,1))
                    yn(num+1,6)= 0.5*(yn(num+1,3)+yn(num+1,1))
                    zn(num+1,6)= 0.5*(zn(num+1,3)+zn(num+1,1))
!----------------------------------------------------------------------------------------------------------
!   Third sub-element
                    xn(num+2,1)= x(j,6)
                    yn(num+2,1)= y(j,6)
                    zn(num+2,1)= z(j,6)
!----------------------------------------------------------------------------------------------------------
                    xn(num+2,2)= x(j,5)
                    yn(num+2,2)= y(j,5)
                    zn(num+2,2)= z(j,5)
!----------------------------------------------------------------------------------------------------------
                    xn(num+2,3)= x(j,3)
                    yn(num+2,3)= y(j,3)
                    zn(num+2,3)= z(j,3)
!----------------------------------------------------------------------------------------------------------
                    xn(num+2,4)= 0.5*(xn(num+2,1)+xn(num+2,2))
                    yn(num+2,4)= 0.5*(yn(num+2,1)+yn(num+2,2))
                    zn(num+2,4)= 0.5*(zn(num+2,1)+zn(num+2,2))
```

```fortran
!----------------------------------------------------------------------------------------------------
                    xn(num+2,5)= 0.5*(xn(num+2,2)+xn(num+2,3))
                    yn(num+2,5)= 0.5*(yn(num+2,2)+yn(num+2,3))
                    zn(num+2,5)= 0.5*(zn(num+2,2)+zn(num+2,3))
!----------------------------------------------------------------------------------------------------
                    xn(num+2,6)= 0.5*(xn(num+2,3)+xn(num+2,1))
                    yn(num+2,6)= 0.5*(yn(num+2,3)+yn(num+2,1))
                    zn(num+2,6)= 0.5*(zn(num+2,3)+zn(num+2,1))
!----------------------------------------------------------------------------------------------------
!   Fourth sub-element
                    xn(num+3,1)= x(j,4)
                    yn(num+3,1)= y(j,4)
                    zn(num+3,1)= z(j,4)
!----------------------------------------------------------------------------------------------------
                    xn(num+3,2)= x(j,5)
                    yn(num+3,2)= y(j,5)
                    zn(num+3,2)= z(j,5)
!----------------------------------------------------------------------------------------------------
                    xn(num+3,3)= x(j,6)
                    yn(num+3,3)= y(j,6)
                    zn(num+3,3)= z(j,6)
!----------------------------------------------------------------------------------------------------
                    xn(num+3,4)= 0.5*(xn(num+3,1)+xn(num+3,2))
                    yn(num+3,4)= 0.5*(yn(num+3,1)+yn(num+3,2))
                    zn(num+3,4)= 0.5*(zn(num+3,1)+zn(num+3,2))
!----------------------------------------------------------------------------------------------------
                    xn(num+3,5)= 0.5*(xn(num+3,2)+xn(num+3,3))
                    yn(num+3,5)= 0.5*(yn(num+3,2)+yn(num+3,3))
                    zn(num+3,5)= 0.5*(zn(num+3,2)+zn(num+3,3))
!----------------------------------------------------------------------------------------------------
                    xn(num+3,6)= 0.5*(xn(num+3,3)+xn(num+3,1))
                    yn(num+3,6)= 0.5*(yn(num+3,3)+yn(num+3,1))
                    zn(num+3,6)= 0.5*(zn(num+3,3)+zn(num+3,1))
!----------------------------------------------------------------------------------------------------
!   Four new elements were generated
                    num = num+4
!----------------------------------------------------------------------------------------------------
!   END of old element loop
                END DO
                nelm=nelm*4
!----------------------------------------------------------------------------------------------------
!   Rename the new points and place them in the master list
                DO k=1,nelm
                    DO l=1,6
                        x(k,l) = xn(k,l)
                        y(k,l) = yn(k,l)
                        z(k,l) = zn(k,l)
!----------------------------------------------------------------------------------------------------
!   Zero just in case
                        xn(k,l) = 0.0
                        yn(k,l) = 0.0
                        zn(k,l) = 0.0
                    END DO
                END DO
!----------------------------------------------------------------------------------------------------
!   END of level loop of i
            END DO
        END IF
!----------------------------------------------------------------------------------------------------
!   Create a list of surface nodes by looping over all elements and adding nodes not alREADy found in the list.
!Fill the connectivity table n(i,j) node numbers of element points 1-6
        ALLOCATE(n(nelm, 6), nbe(nelm, 3), nbe1(nelm, 3), STAT = IStat)
!----------------------------------------------------------------------------------------------------
!   First element is set mannualy
!Initialization of the linked list
        ALLOCATE (head, STAT=istatp)   ! Allocate new value
        tail => head                   ! Tail pts to new value
        NULLIFY (tail%pp)              ! Nullify pp in new value
        k = 1
        tail%Px = (/ x(1,k), y(1,k), z(1,k) /) ! Store point
!----------------------------------------------------------------------------------------------------
!   Loop over nodes in first element
```



```fortran
        DO k = 2, 6
            ALLOCATE (tail%pp, STAT=istatp)   ! Allocate new value
            tail => tail%pp                   ! Tail pts to new value
            NULLIFY (tail%pp)              ! Nullify pp in new value
            tail%Px = (/ x(1,k), y(1,k), z(1,k) /) ! Store point
            !         tail%Px(1) = x(1,k)           ! Store point
            !         tail%Px(2) = y(1,k)           ! Store point
            !         tail%Px(3) = z(1,k)           ! Store point
        END DO
!----------------------------------------------------------------------------------------------------
        n(1,1) = 1
        n(1,2) = 2
        n(1,3) = 3
        n(1,4) = 4
        n(1,5) = 5
        n(1,6) = 6
!----------------------------------------------------------------------------------------------------
        npts = 6
!----------------------------------------------------------------------------------------------------
!   Loop over elements, to know if the node had been associated before.
        DO i = 2, nelm
!----------------------------------------------------------------------------------------------------
! Loop over element nodes
            DO j = 1, 6
                IFlag = 0
                ptr => head
!----------------------------------------------------------------------------------------------------
!   Cycle DO WHILE
                DO k = 1, npts
                    IF ( .NOT. ASSOCIATED(ptr) ) EXIT           ! Pointer valid?
                    ptm(:) = ptr%Px(:)                          ! Yes: Write value
                    ptr => ptr%pp                               ! Get next pointer
!----------------------------------------------------------------------------------------------------
!   Condition which checks if a node had been assosiated before.
                    IF((ABS(x(i,j)-ptm(1)) <= eps) .AND. &
                    &  (ABS(y(i,j)-ptm(2)) <= eps) .AND. &
                    &  (ABS(z(i,j)-ptm(3)) <= eps)) THEN
                        ! the node has been previously recorded
                        IFlag = 1
                        ! the jth local node of element i
                        n(i,j) = k
                        ! is the kth global node
                    END IF
                END DO
!----------------------------------------------------------------------------------------------------
!   Record the node
                IF(IFlag == 0) THEN
                    npts = npts + 1
                    ALLOCATE (tail%pp, STAT=istatp)   ! Allocate new value
                    tail => tail%pp                   ! Tail pts to new value
                    NULLIFY (tail%pp)              ! Nullify pp in new value
                    tail%Px = (/ x(i,j), y(i,j), z(i,j) /)   ! Store point
                    !               tail%Px(1) = x(i,j)           ! Store point
                    !               tail%Px(2) = y(i,j)           ! Store point
                    !               tail%Px(3) = z(i,j)           ! Store point

                    n(i,j) = npts
                END IF
            END DO
!----------------------------------------------------------------------------------------------------
!  END of loop over elements
        END DO
!----------------------------------------------------------------------------------------------------
        ALLOCATE(p(npts, 3), ne(npts, 7), STAT = IStat)
        k = 1
        ptr => head
!----------------------------------------------------------------------------------------------------
        output2: DO
            IF ( .NOT. ASSOCIATED(ptr) ) EXIT   ! Pointer valid?
            p(k,:) = ptr%Px(:)                  ! Yes: Write value
            ptr => ptr%pp                       ! Get next pointer
            k = k + 1
```

```
        END DO output2
!--------------------------------------------------------------------------------------------------------------
        NULLIFY (head, ptr, tail)
!--------------------------------------------------------------------------------------------------------------
!   Generate connectivity table ne(i,j) for elements touching node i
        ne = 0
!   Loop over global nodes
        DO i=1,npts
            !       ne(i,1) = 0
            icount = 1
!--------------------------------------------------------------------------------------------------------------
!   Loop over local nodes
            DO j = 1, nelm
!--------------------------------------------------------------------------------------------------------------
!   Loop over nodes
                DO k = 1, 6
!--------------------------------------------------------------------------------------------------------------
!   Condition to see the conectivity
                    IF((ABS(p(i,1)-x(j,k)) <= eps) .AND. &
                        &  (ABS(p(i,2)-y(j,k)) <= eps) .AND. &
                        &  (ABS(p(i,3)-z(j,k)) <= eps)) THEN
                        icount=icount+1
                        ne(i,1)=ne(i,1)+1
                        ne(i,icount)=j
                        END IF
                END DO
!--------------------------------------------------------------------------------------------------------------
            END DO
!--------------------------------------------------------------------------------------------------------------
!   END of loop over surface points
        END DO
!--------------------------------------------------------------------------------------------------------------
!   Create connectivity table nbe(i,j) for neighboring elements j of element i. Testing is DOne with respect to
!the mid-points (for boundary elements with only 2 neighbors, the array entry will be zero)
!--------------------------------------------------------------------------------------------------------------
        nbe1 = 0
!--------------------------------------------------------------------------------------------------------------
!   Loop over elements
        DO i=1,nelm
            jcount=1
!--------------------------------------------------------------------------------------------------------------
!  Loop over mid-points
            DO j=4,6
!--------------------------------------------------------------------------------------------------------------
!  Test element
                DO k=1,nelm
!--------------------------------------------------------------------------------------------------------------
!  Loop over mid-points
                    IF(k == i) THEN
!--------------------------------------------------------------------------------------------------------------
                        DO l=4,6
                            IF((ABS(x(i,j)-x(k,l)) <= eps) .AND. &
                            &  (ABS(y(i,j)-y(k,l)) <= eps) .AND. &
                            &  (ABS(z(i,j)-z(k,l)) <= eps))  &
                            &   nbe1(i, jcount) = k
                        END DO
!--------------------------------------------------------------------------------------------------------------
                    END IF
                    !  END of test element
!--------------------------------------------------------------------------------------------------------------
                END DO
!--------------------------------------------------------------------------------------------------------------
                IF(nbe1(i,jcount) /= 0) jcount=jcount+1
!--------------------------------------------------------------------------------------------------------------
            END DO
!--------------------------------------------------------------------------------------------------------------
!   END of loop over elements
        END DO
!--------------------------------------------------------------------------------------------------------------
!   Create connectivity table nbe(i,j) for neighboring elements j of element i. Testing is DOne with respect to
!the mid-points (for boundary elements with only 2 neighbors, the array entry will be zero)
!--------------------------------------------------------------------------------------------------------------
```



```
        nbe = 0
        DO k=1,nelm
            kcount = 1
            DO j=1,nelm
                IF (k/=j) THEN
                    DO i=4,6
                        IF((ABS(n(k,4)-n(j,i)) < 1)) THEN
                        nbe(k, kcount) = j
                        kcount = kcount+1
                    ELSE IF((ABS(n(k,5)-n(j,i)) < 1)) THEN
                         nbe(k, kcount) = j
                        kcount = kcount+1
                    ELSE
                        IF((ABS(n(k,6)-n(j,i)) < 1)) THEN
                        nbe(k, kcount) = j
                        kcount = kcount+1
                        END IF
                    END IF
                    END DO
                END IF
            END DO
        END DO

!--------------------------------------------------------------------------------------------------------------
!   Poject points p(i,j) onto the unit sphere
!--------------------------------------------------------------------------------------------------------------
        DO i=1,npts
            r=DSQRT(p(i,1)**2+p(i,2)**2+p(i,3)**2)
            p(i,1) = p(i,1)/r
            p(i,2) = p(i,2)/r
            p(i,3) = p(i,3)/r
        END DO
!==============================================================================================================
        OPEN(UNIT = 500, FILE = 'n.dat' ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
    IF (fstatus == 0) THEN
        WRITE(500,*) npts
        WRITE(500,*) nelm
!--------------------------------------------------------------------------------------------------------------
        DO i = 1, nelm
            WRITE (500,107) n(i,1),n(i,2),n(i,3),n(i,4),n(i,5),n(i,6)
        END DO
    END IF
!--------------------------------------------------------------------------------------------------------------
    CLOSE(UNIT = 500)
!==============================================================================================================
        OPEN(UNIT = 500, FILE = 'nbe.dat' ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
    IF (fstatus == 0) THEN
!--------------------------------------------------------------------------------------------------------------
        DO i = 1, nelm
            WRITE (500,108) nbe(i,1),nbe(i,2),nbe(i,3)
        END DO
    END IF
!--------------------------------------------------------------------------------------------------------------
    CLOSE(UNIT = 500)
!==============================================================================================================
        OPEN(UNIT = 500, FILE = 'ne.dat' ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
    IF (fstatus == 0) THEN
!--------------------------------------------------------------------------------------------------------------
        DO i = 1, npts
            WRITE (500,107) ne(i,1),ne(i,2),ne(i,3),ne(i,4),ne(i,5),ne(i,6),ne(i,7)
        END DO
    END IF
!--------------------------------------------------------------------------------------------------------------
    CLOSE(UNIT = 500)
!==============================================================================================================
    107 FORMAT(8(1x,I10))
    108 FORMAT(4(1x,I10))
!--------------------------------------------------------------------------------------------------------------
!   DOne.
!                                                    La comedia e finita :)
!==============================================================================================================
    ELSE
```

```fortran
        WRITE(ULog,*) ' Error in allocation'
        WRITE(ULog,*) 'Little Padawan, you must review the code again'
        WRITE(ULog,*) ' Subroutine trgl_octa'
    END IF
!==========================================================================================================
    END SUBROUTINE trgl_octa
!==========================================================================================================
!==========================================================================================================
    SUBROUTINE drop_scale_adjust(p,     npts,   nelm,&
                                 &boa,   coa,    req, &
                                 &eps,   &
                                 &cxp,   cyp,    czp, &
                                 &phi1,  phi2,   phi3 )
!==========================================================================================================
!   This is a new subroutine to adjust the right size of the drop.
!----------------------------------------------------------------------------------------------------------
!   Variables
!----------------------------------------------------------------------------------------------------------
!   nelm ....................... number of surface elements
!   scale....................... scale of the drop
!   x_axis, y_axis, z_axis....... rate of drop'size in every coordinate axis
!   cs, sn......................
!   tmpx, tmpy, tmpz............
!   p(i,j) ..................... (x,y,z) coords. of surface node labeled i  (j=1,2,3)
!                   x = p(i,1)
!                   y = p(i,2)
!                   z = p(i,3)
!   req.......................... equivalent radius
!   phi1, phi2, phi3............ angles of rotation  arround the axes
!==========================================================================================================
!==========================================================================================================
!   Modules used in this subroutine.
    USE Mod_SharedVars, ONLY: DBL, ULog, p, Pi, UDat
!==========================================================================================================
    IMPLICIT NONE
!==========================================================================================================
!   Variables IN/OUT
    INTEGER,   INTENT(IN)  :: npts, nelm
    REAL (KIND = DBL), INTENT(IN)  :: boa,  coa, req        !These variables are used in subroutine
    REAL (KIND = DBL), INTENT(IN)  :: eps                   !scale_drop_adjust.
    REAL (KIND = DBL), INTENT(IN)  :: cxp,  cyp,  czp       !
    REAL (KIND = DBL), INTENT(INOUT)  :: phi1, phi2, phi3     !
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: p

!----------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j
    REAL (KIND = DBL)  :: oot
    REAL (KIND = DBL)  :: scale
    REAL (KIND = DBL)  :: x_axis, y_axis, z_axis
    REAL (KIND = DBL)  :: cs, sn
    REAL (KIND = DBL)  :: tmpx, tmpy, tmpz
!==========================================================================================================
!   The next part coputes the expantion to specified shape and equivalent radius. The idea is introduce this
!statements in a new subroutine.
    oot  = 1.0D0/3.0D0
    scale = req/(boa*coa)**oot
    x_axis = scale
    y_axis = scale*boa
    z_axis = scale*coa
!----------------------------------------------------------------------------------------------------------
!   Scale is made her
    FORALL (i=1:npts)
        p(i,1) = x_axis*p(i,1)
        p(i,2) = y_axis*p(i,2)
        p(i,3) = z_axis*p(i,3)
    END FORALL
    WRITE (UDat,*) " ellipsoid x semi-axis = ",x_axis
    WRITE (UDat,*) " ellipsoid y semi-axis = ",y_axis
    WRITE (UDat,*) " ellipsoid z semi-axis = ",z_axis
!----------------------------------------------------------------------------------------------------------
!   This seccion is made the rotation by angles phi1,phi2,phi3 around the x,y,z axes
    IF (ABS(phi1) > eps .OR. &
```



```fortran
        &   ABS(phi2) > eps .OR. &
        &   ABS(phi3) > eps       ) THEN
        phi1 = phi1*pi                   ! scale in x axis
        phi2 = phi2*pi                   ! scale in y axis
        phi3 = phi3*pi                   ! scale ib z axis

        cs = DCOS(phi3)
        sn = DSIN(phi3)
!----------------------------------------------------------------------------------------------------------
!   Rotate about the z axis
        DO i=1,npts
            tmpx = cs*p(i,1)+sn*p(i,2)
            tmpy =-sn*p(i,1)+cs*p(i,2)
            tmpz = p(i,3)
            p(i,1) = tmpx
            p(i,2) = tmpy
            p(i,3) = tmpz
            tmpx = 0.0D0
            tmpy = 0.0D0
            tmpz = 0.0D0
        END DO
        cs = DCOS(phi1)
        sn = DSIN(phi1)
!----------------------------------------------------------------------------------------------------------
!   Rotate about the x axis
        DO i = 1, npts
            tmpx = p(i,1)
            tmpy = cs*p(i,2)+sn*p(i,3)
            tmpz =-sn*p(i,2)+cs*p(i,3)
            p(i,1) = tmpx
            p(i,2) = tmpy
            p(i,3) = tmpz
            tmpx = 0.0D0
            tmpy = 0.0D0
            tmpz = 0.0D0
        END DO
        cs = DCOS(phi2)
        sn = DSIN(phi2)
!----------------------------------------------------------------------------------------------------------
!   Rotate about the y axis
        DO i = 1, npts
            tmpx = cs*p(i,1)-sn*p(i,3)
            tmpy = p(i,2)
            tmpz = sn*p(i,1)+cs*p(i,3)
            p(i,1) = tmpx
            p(i,2) = tmpy
            p(i,3) = tmpz
            tmpx = 0.0D0
            tmpy = 0.0D0
            tmpz = 0.0D0
        END DO
!----------------------------------------------------------------------------------------------------------
!   Unscale of drop and final statement
        phi1 = phi1/pi
        phi2 = phi2/pi
        phi3 = phi3/pi
    END IF
!----------------------------------------------------------------------------------------------------------
!   Translate center to specified position
    p(:,1) = p(:,1) + cxp
    p(:,2) = p(:,2) + cyp
    p(:,3) = p(:,3) + czp
    WRITE (UDat,*) 'Translation was sucessful'
    WRITE (UDat,*) " prtcl_3d: number of nodes   : ",npts
    WRITE (UDat,*) " prtcl_3d: number of elements: ",nelm
    END SUBROUTINE drop_scale_adjust
!==========================================================================================================
    SUBROUTINE trgl_octa_read(ndiv, npts, nelm)
!==========================================================================================================
!   Triangulation of the unit sphere by subdividing a regular octahedron into six-node quadratic triangles.
!----------------------------------------------------------------------------------------------------------
!   Variables
```

```fortran
!-----------------------------------------------------------------------------------------------------------
!   nelm ........................ number of surface elements
!   npts ........................ number of nodal points
!   x(i,j), y(i,j), z(i,j) ...... Cartesian coords of point j on element i
!   p(i,j) ...................... (x,y,z) coords. of surface node labeled i  (j=1,2,3)
!                   x = p(i,1)
!                   y = p(i,2)
!                   z = p(i,3)
!   n(i,j) ...................... node number of point j on element i, where j=1,...,6
!   ne(i,j) ..................... ne(i,1) is the number of elements touching node i.
!                                 ne(i,2:ne(i,1)) are the corresponding element labels
!   nbe(i,j) .................... label of element sharing side j of element i where j = 1, 2, 3
!   ndiv ....  level of discretization of starting octahedron (0-ndiv)
!===========================================================================================================
!===========================================================================================================
!   Modules used in this subroutine.
    USE Mod_SharedVars, ONLY: DBL, ULogi,ULog, p, ne, n, nbe
!===========================================================================================================
    IMPLICIT NONE
!===========================================================================================================
!   Variables IN/OUT
!-----------------------------------------------------------------------------------------------------------
    INTEGER,   INTENT(IN)   :: ndiv
    INTEGER,   INTENT(OUT)  :: npts, nelm
!-----------------------------------------------------------------------------------------------------------
!   Variables used inside the subroutine
    INTEGER :: fstatus
    INTEGER  :: i, j, k, i1, i2, i3, i4, i5, i6
    REAL (KIND = DBL), PARAMETER :: eps=0.0000001
!===========================================================================================================
!-----------------------------------------------------------------------------------------------------------
!   Create a list of surface nodes by looping over all elements and adding nodes not alREADy found in the list.
!Fill the connectivity table n(i,j) node numbers of element points 1-6
        nelm = 8*(4 **(ndiv))
        npts= nelm + nelm + 2
        ALLOCATE(p(npts, 3), ne(npts, 7))
        ALLOCATE(n(nelm, 6), nbe(nelm, 3))
        n  = 0
        ne = 0
        nbe= 0
        p  = 0.0D0
        fstatus = 0
!-----------------------------------------------------------------------------------------------------------
!===========================================================================================================
   OPEN(UNIT = 500, FILE = 'n.dat', STATUS = 'OLD', ACTION = 'READ', IOSTAT = fstatus)
    IF (fstatus == 0) THEN
        READ(500,*) npts
        READ(500,*) nelm

        DO i = 1, nelm
            READ (500,107) n(i,1),n(i,2),n(i,3),n(i,4),n(i,5),n(i,6)
        END DO
        CLOSE(UNIT = 500)
!===========================================================================================================
    ELSE
      OPEN(UNIT = ULogi, FILE = 'Prtcl3D_Logi.log',STATUS = 'REPLACE', ACTION = 'WRITE')
            WRITE (ULogi,*)'Error, little Padawan, InputData has a mistake!'
            WRITE (ULogi, *) 'First Error index = ', fstatus
            WRITE (ULogi,*)'There was error in the n file'
            WRITE(ULogi,*)'The Force will be with you'
            CLOSE(UNIT = 10)
     END IF
!===========================================================================================================
!===========================================================================================================
   OPEN(UNIT = 500, FILE = 'nbe.dat', STATUS = 'OLD', ACTION = 'READ', IOSTAT = fstatus)
    IF (fstatus == 0) THEN
!-----------------------------------------------------------------------------------------------------------
        DO i = 1, nelm
            READ (500,108) nbe(i,1),nbe(i,2),nbe(i,3)
        END DO
        CLOSE(UNIT = 500)
!===========================================================================================================
```



```fortran
    ELSE
      OPEN(UNIT = ULogi, FILE = 'Prtcl3D_Logi.log',STATUS = 'REPLACE', ACTION = 'WRITE')
            WRITE (ULogi,*)'Error, little Padawan, InputData has a mistake!'
            WRITE (ULogi, *) 'First Error index = ', fstatus
            WRITE (ULogi,*)'There was error in the ne file'
            WRITE(ULogi,*)'The Force will be with you'
            CLOSE(UNIT = 10)
     END IF
!===========================================================================================================
!===========================================================================================================
   OPEN(UNIT = 500, FILE = 'ne.dat',STATUS = 'OLD', ACTION = 'READ', IOSTAT = fstatus)
    IF (fstatus == 0) THEN
!-----------------------------------------------------------------------------------------------------------
        DO i = 1, npts
            READ (500,107) ne(i,1),ne(i,2),ne(i,3),ne(i,4),ne(i,5),ne(i,6),ne(i,7)
        END DO
        CLOSE(UNIT = 500)
!===========================================================================================================
    ELSE
      OPEN(UNIT = ULogi, FILE = 'Prtcl3D_Logi.log',STATUS = 'REPLACE', ACTION = 'WRITE')
            WRITE (ULogi,*)'Error, little Padawan, InputData has a mistake!'
            WRITE (ULogi, *) 'First Error index = ', fstatus
            WRITE (ULogi,*)'There was error in the nbe file'
            WRITE(ULogi,*)'The Force will be with you'
            CLOSE(UNIT = 10)
     END IF
!===========================================================================================================
!===========================================================================================================
   OPEN(UNIT = 500, FILE = 'Prtcl3D_geometry.dat',STATUS = 'OLD', ACTION = 'READ', IOSTAT = fstatus)
    IF (fstatus == 0) THEN
!-----------------------------------------------------------------------------------------------------------
        !DO i = 1, npts
        !    READ (500,103) p(i,1),p(i,2),p(i,3)
        !END DO
        !CLOSE(UNIT = 500)

        DO i = 1, nelm
!   will be interpolate the normal vector and features of collocation points.
            i1 = n(i,1)
            i2 = n(i,2)
            i3 = n(i,3)
            i4 = n(i,4)
            i5 = n(i,5)
            i6 = n(i,6)
!-----------------------------------------------------------------------------------------------------------
            READ (500,103) p(i1,1), p(i1,2), p(i1,3)
            READ (500,103) p(i2,1), p(i2,2), p(i2,3)
            READ (500,103) p(i3,1), p(i3,2), p(i3,3)
            READ (500,103) p(i4,1), p(i4,2), p(i4,3)
            READ (500,103) p(i5,1), p(i5,2), p(i5,3)
            READ (500,103) p(i6,1), p(i6,2), p(i6,3)
!-----------------------------------------------------------------------------------------------------------
        END DO

!===========================================================================================================
    ELSE
      OPEN(UNIT = ULogi, FILE = 'Prtcl3D_Logi.log',STATUS = 'REPLACE', ACTION = 'WRITE')
            WRITE (ULogi,*)'Error, little Padawan, InputData has a mistake!'
            WRITE (ULogi, *) 'First Error index = ', fstatus
            WRITE (ULogi,*)'There was error in the geometry file'
            WRITE(ULogi,*)'The Force will be with you'
            CLOSE(UNIT = 10)
     END IF
!===========================================================================================================
!   DOne.
!                                                           La comedia e finita :)
!===========================================================================================================
    103 FORMAT(10(1x,ES24.16))
    107 FORMAT(8(1x,I10))
    108 FORMAT(4(1x,I10))

    END SUBROUTINE trgl_octa_read
```

```
!===============================================================================================================
    END MODULE Mod_Trgl_Octa

```

```fortran
 MODULE Mod_Prtcl_3D_Geo
!==============================================================================================================
!   Version: 0.5 created on  26 / IX / 2007
!                                                                                               C. Pozrikidis
!--------------------------------------------------------------------------------------------------------------
!   Version: 0.7 created on  -- / III /2010
!                                                                                  Marco Antonio Reyes Huesca
!--------------------------------------------------------------------------------------------------------------
!   Version: 0.9 created on 23 / 08 / 2012
!   Version: 1.0 created on 14 / 11 / 2012
!                                                                                     Alfredo Sanjuan Sanjuan
!==============================================================================================================
    CONTAINS
!==============================================================================================================
    SUBROUTINE printel(k, Index, c)
!==============================================================================================================
!    This subroutine prints drop's geometry. It has two options.
!Print successive nodes of element k in file unit 1
!   Index = 1: print the whole element
!   Index = 2: print the 4 subelements
!--------------------------------------------------------------------------------------------------------------
!   Variables
!--------------------------------------------------------------------------------------------------------------
!   k ............................ number of element
!   index ........................ type of print
!   nfour ........................ index to print only the element
!   nseven ....................... index to print the subelements
!   c ............................ firstly, it was the value of shear strain, now, it will change. ;)
!==============================================================================================================
    USE Mod_SharedVars, ONLY: DBL, ULog, UGeo, p, ne, n, nbe
!==============================================================================================================
    IMPLICIT NONE
!==============================================================================================================
    INTEGER,                         INTENT(IN) :: k, index
!--------------------------------------------------------------------------------------------------------------
    INTEGER :: nfour, nseven, i
    REAL (KIND = DBL), DIMENSION(:), INTENT(IN) :: c
!--------------------------------------------------------------------------------------------------------------
! constants
    nfour  = 4
    nseven = 1
!--------------------------------------------------------------------------------------------------------------
!   There is a CASE instrction to print the element.
!--------------------------------------------------------------------------------------------------------------
    SELECT CASE (index)
    CASE (1)
        WRITE (UGeo,100) nseven
        i = n(k,1)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,2)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,5)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,3)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,1)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
    CASE(2)
        !--- first
        WRITE (UGeo,100) nfour
        i = n(k,1)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,1)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
```



```fortran
        !--- second
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,2)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,5)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        !--- third
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,5)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        !--- fourth
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,5)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,3)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
    CASE DEFAULT
        WRITE (UGeo,100) nfour
        i = n(k,1)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,1)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,2)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,5)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,5)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,5)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,3)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        WRITE (ULog,*)
        WRITE (ULog,*) '    Geo_Printel'
        WRITE (ULog,*)
        WRITE (ULog,*) '    Chosen index is not available'
        WRITE (ULog,*) '    It was taken index= 2'
    END SELECT
100 FORMAT(1x,i4,10(1x,ES24.16))
101 FORMAT(10(1x,ES24.16))
    END SUBROUTINE printel
!==============================================================================================================
!==============================================================================================================
```

```
    SUBROUTINE abc(x1, y1, z1,  &
    &              x2, y2, z2,  &
    &              x3, y3, z3,  &
    &              x4, y4, z4,  &
    &              x5, y5, z5,  &
    &              x6, y6, z6,  &
    &              al, be, ga)
!=====================================================================================================
!   This subroutine compute the parametric representation constants alpha, beta, gamma
!=====================================================================================================
    USE Mod_SharedVars, ONLY: DBL
!=====================================================================================================
    IMPLICIT NONE
!=====================================================================================================
!   Variables
!-----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN) ::  x1, y1, z1, &     !coordinates of each point in the element
    &                                 x2, y2, z2, &
    &                                 x3, y3, z3, &
    &                                 x4, y4, z4, &
    &                                 x5, y5, z5, &
    &                                 x6, y6, z6
    REAL (KIND = DBL), INTENT(OUT) :: al, be, ga        !constants alpha, beta and gamma (weights)
!=====================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: d42, d41                       !distances of the element on the segment 1 4 2
    REAL (KIND = DBL) :: d63, d61                       !distances of the element on the segment 3 6 1
    REAL (KIND = DBL) :: d52, d53                       !distances of the element on the segment 2 5 3
!=====================================================================================================
!   Initialize
    d42 = (x4 - x2)**2 + (y4 - y2)**2 + (z4 - z2)**2
    d41 = (x4 - x1)**2 + (y4 - y1)**2 + (z4 - z1)**2
    d63 = (x6 - x3)**2 + (y6 - y3)**2 + (z6 - z3)**2
    d61 = (x6 - x1)**2 + (y6 - y1)**2 + (z6 - z1)**2
    d52 = (x5 - x2)**2 + (y5 - y2)**2 + (z5 - z2)**2
    d53 = (x5 - x3)**2 + (y5 - y3)**2 + (z5 - z3)**2
!-----------------------------------------------------------------------------------------------------
    d42 = DSQRT(d42)
    d41 = DSQRT(d41)
    d63 = DSQRT(d63)
    d61 = DSQRT(d61)
    d52 = DSQRT(d52)
    d53 = DSQRT(d53)
!-----------------------------------------------------------------------------------------------------
    al = 1.0D0/(1.0D0 + d42/d41)
    be = 1.0D0/(1.0D0 + d63/d61)
    ga = 1.0D0/(1.0D0 + d52/d53)
!-----------------------------------------------------------------------------------------------------
    END SUBROUTINE abc
!=====================================================================================================
!=====================================================================================================
    SUBROUTINE elm_geom(nelm, npts, mint,   &
    &                   xmom, ymom, zmom,   &
    &                   area, vlm,          &
    &                   cx,   cy,   cz, stride)
!=====================================================================================================
!   This subroutine is a new version of Elm_Geo Subroutine.
!Compute:
!   *The surface area of the individual elements x, y, and z moments over each element
!   *the total particle surface area and volume
!Now, (25 / Augost / 2012) this subroutine was cut.
!=====================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog,&
    &                         alphaQ,  betaQ,  gammaQ, &
    &                         arel,         crvmel,    &
    &                         vnx0,     vny0,    vnz0, &
    &                         ZZ,           WW,        &
    &                         xiq, etq, wq
!=====================================================================================================
    IMPLICIT NONE
```



```
!=====================================================================================================
!   Variables
!-----------------------------------------------------------------------------------------------------
    INTEGER,           INTENT(IN)  :: nelm                              !number of elements
    INTEGER,           INTENT(IN)  :: npts                              !number of poins on the surface
    INTEGER,           INTENT(IN)  :: mint                              !order of triangle quadrature
    INTEGER,           INTENT(IN)  :: stride              !order of triangle and Gauss-Legendre quadratures
    REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom    !coordinates of the moments of the drop
    REAL (KIND = DBL), INTENT(OUT) :: area, vlm                         !area and volume of each element
    REAL (KIND = DBL), INTENT(OUT) :: cx,   cy,   cz                    !drop's centroid coordinates
!=====================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------
    INTEGER  :: i, k                                 !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6               !indices to obtain node numbers from each element
!-----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: xi, eta                        !variables of weigth to integrate over a triangle
    REAL (KIND = DBL) :: x, y, z                        !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi            !coordinates of the tangential vector over the xi axis
    REAL (KIND = DBL) :: DxDet, DyDet, DzDet            !coordinates of the tangential vector over the eta axis
    REAL (KIND = DBL) :: vnx, vny, vnz                  !normal vector coordinates of the element
    REAL (KIND = DBL) :: hs                             !surface metric on a triangle
    REAL (KIND = DBL) :: al, be, ga, alc, bec, gac      !integration weigth coefficients
    REAL (KIND = DBL) :: cf, fil                        !integration weigth coefficients
    REAL (KIND = DBL), DIMENSION(6) ::xxi, eet          !variables of weigth over in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDx, DyDx, DzDx  !tangential vector over xi axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDe, DyDe, DzDe  !tangential vector over eta axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz       !normal vector in triangle (xi,eta)
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &           !binormal vectors around in triangle (xi,eta)
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  crvx, crvy, crvz, curv        !curvature
!=====================================================================================================
!   initialize
!-----------------------------------------------------------------------------------------------------
    area = 0.0D0
    vlm  = 0.0D0
    cx = 0.0D0
    cy = 0.0D0
    cz = 0.0D0
    arel = 0.0D0
    xmom = 0.0D0
    ymom = 0.0D0
    zmom = 0.0D0
    crvmel = 0.0D0
    fil = 0.0D0
!-----------------------------------------------------------------------------------------------------
    Outer: DO k = 1, nelm

        i1 = n(k,1)
        i2 = n(k,2)
        i3 = n(k,3)
        i4 = n(k,4)
        i5 = n(k,5)
        i6 = n(k,6)
        al  = alphaQ(k)
        be  = betaQ(k)
        ga  = gammaQ(k)
        alc = 1.0D0-al
        bec = 1.0D0-be
        gac = 1.0D0-ga
!-----------------------------------------------------------------------------------------------------
!   Compute surface area and volume of the individual elements x, y, and z moments over each element total
!particle surface area and volume
!-----------------------------------------------------------------------------------------------------
        DO i = 1, mint
            xi  = xiq(i)
            eta = etq(i)
            CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
            &             p(i2,1),   p(i2,2),  p(i2,3),    &
            &             p(i3,1),   p(i3,2),  p(i3,3),    &
            &             p(i4,1),   p(i4,2),  p(i4,3),    &
```

```fortran
          &            p(i5,1),   p(i5,2),  p(i5,3),    &
          &            p(i6,1),   p(i6,2),  p(i6,3),    &
          &            al,        be,       ga,         &
          &            xi,        eta,                  &
          &            x,         y,         z,         &
          &            DxDxi,     DyDxi,    DzDxi,      &
          &            DxDet,     DyDet,    DzDet,      &
          &            vnx,       vny,      vnz,        &
          &            hs)
          cf = hs*wq(i)
          arel(k) = arel(k) + cf
          xmom(k) = xmom(k) + cf*x
          ymom(k) = ymom(k) + cf*y
          zmom(k) = zmom(k) + cf*z
          vlm = vlm + (x*vnx+y*vny+z*vnz)*cf
      END DO
      arel(k) = 0.5D0*arel(k)
      xmom(k) = 0.5D0*xmom(k)
      ymom(k) = 0.5D0*ymom(k)
      zmom(k) = 0.5D0*zmom(k)
      area = area +arel(k)
      cx   = cx   + xmom(k)
      cy   = cy   + ymom(k)
      cz   = cz   + zmom(k)
!----------------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the nftty
!formula (4.2.10) of Pozrikidis (1997)
!----------------------------------------------------------------------------------------------------------
      xxi(1) = 0.0D0
      eet(1) = 0.0D0
      xxi(2) = 1.0D0
      eet(2) = 0.0D0
      xxi(3) = 0.0D0
      eet(3) = 1.0D0
      xxi(4) = al
      eet(4) = 0.0D0
      xxi(5) = ga
      eet(5) = gac
      xxi(6) = 0.0D0
      eet(6) = be
      xi  = 1.0D0/3.0D0
      eta = 1.0D0/3.0D0
          CALL interp_p4(p(i1,1),   p(i1,2),  p(i1,3),    &
          &            p(i2,1),   p(i2,2),  p(i2,3),    &
          &            p(i3,1),   p(i3,2),  p(i3,3),    &
          &            p(i4,1),   p(i4,2),  p(i4,3),    &
          &            p(i5,1),   p(i5,2),  p(i5,3),    &
          &            p(i6,1),   p(i6,2),  p(i6,3),    &
          &            al,        be,       ga,         &
          &            xi,        eta,                  &
          &                     curv, stride            )

!----------------------------------------------------------------------------------------------------------
! will project curvature vector onto the normal vector at the centroid only needs the colocation verctors
!In this value of curvature I made a division per 8 (1/8.0) when i expected do for 4 (1/4.0)
!----------------------------------------------------------------------------------------------------------
      crvmel(k) = curv !(0.25D0*(crvx*vnx0(k)+crvy*vny0(k)+crvz*vnz0(k))*fil)/arel(k)
   END DO Outer
!----------------------------------------------------------------------------------------------------------
! final computation of the surface-centroid and volume

!----------------------------------------------------------------------------------------------------------
      cx = cx/area
      cy = cy/area
      cz = cz/area
      vlm = vlm/6.0D0
103 FORMAT(10(1x,ES24.16))
!----------------------------------------------------------------------------------------------------------
   END SUBROUTINE elm_geom
!==========================================================================================================
!==========================================================================================================
   SUBROUTINE elm_geom4(nelm, npts, mint,   &
```



```fortran
   &                  xmom, ymom, zmom,   &
   &                  area, vlm,          &
   &                  cx,   cy,   cz)
!==========================================================================================================
!   This subroutine is a new version of Elm_Geo Subroutine.
!Compute:
!   *The surface area of the individual elements x, y, and z moments over each element
!   *the total particle surface area and volume
!Now, (25 / Augost / 2012) this subroutine was cut.
!==========================================================================================================
   USE Mod_Nodal_Interp
   USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog,&
   &                        alphaQ,  betaQ,  gammaQ, &
   &                        arel,        crvmel,    &
   &                        vnx0,     vny0,    vnz0, &
   &                        ZZ,           WW,        &
   &                        xiq, etq, wq
!==========================================================================================================
   IMPLICIT NONE
!==========================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------------
   INTEGER,           INTENT(IN)  :: nelm                              !number of elements
   INTEGER,           INTENT(IN)  :: npts                              !number of poins on the surface
   INTEGER,           INTENT(IN)  :: mint                              !order of triangle quadrature
   REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom    !coordinates of the moments of the drop
   REAL (KIND = DBL), INTENT(OUT) :: area, vlm                         !area and volume of each element
   REAL (KIND = DBL), INTENT(OUT) :: cx,   cy,   cz                    !drop's centroid coordinates
!==========================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------------
   INTEGER  :: i, k                                  !Counters
   INTEGER  :: i1, i2, i3, i4, i5, i6                !indices to obtain node numbers from each element
!----------------------------------------------------------------------------------------------------------
   REAL (KIND = DBL) :: xi, eta                          !variables of weigth to integrate over a triangle
   REAL (KIND = DBL) :: x, y, z                          !coordinates of the f(x,y,z)= F(xi,eta)
   REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi              !coordinates of the tangential vector over the xi axis
   REAL (KIND = DBL) :: DxDet, DyDet, DzDet              !coordinates of the tangential vector over the eta axis
   REAL (KIND = DBL) :: vnx, vny, vnz                    !normal vector coordinates of the element
   REAL (KIND = DBL) :: hs                               !surface metric on a triangle
   REAL (KIND = DBL) :: al, be, ga, alc, bec, gac        !integration weigth coefficients
   REAL (KIND = DBL) :: cf, fil                          !integration weigth coefficients
   REAL (KIND = DBL), DIMENSION(6) ::xxi, eet            !variables of weigth over in triangle (xi,eta)
   REAL (KIND = DBL), DIMENSION(6) ::DxDx, DyDx, DzDx    !tangential vector over xi axis in triangle (xi,eta)
   REAL (KIND = DBL), DIMENSION(6) ::DxDe, DyDe, DzDe    !tangential vector over eta axis in triangle (xi,eta)
   REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz         !normal vector in triangle (xi,eta)
   REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &             !binormal vectors around in triangle (xi,eta)
   &                     bvx2, bvy2, bvz2, &
   &                     bvx3, bvy3, bvz3
   REAL (KIND = DBL) ::  crvx, crvy, crvz                !curvature
!==========================================================================================================
!   initialize
!----------------------------------------------------------------------------------------------------------
   area = 0.0D0
   vlm  = 0.0D0
   cx = 0.0D0
   cy = 0.0D0
   cz = 0.0D0
   arel = 0.0D0
   xmom = 0.0D0
   ymom = 0.0D0
   zmom = 0.0D0
   crvmel = 0.0D0
   fil = 0.0D0
!----------------------------------------------------------------------------------------------------------
!    OPEN (9,file="curvmel.out")

   Outer: DO k = 1, nelm

       i1 = n(k,1)
       i2 = n(k,2)
       i3 = n(k,3)
```

```
        i4 = n(k,4)
        i5 = n(k,5)
        i6 = n(k,6)
        al  = alphaQ(k)
        be  = betaQ(k)
        ga  = gammaQ(k)
        alc = 1.0D0-al
        bec = 1.0D0-be
        gac = 1.0D0-ga
!------------------------------------------------------------------------------------------------------------
!   Compute surface area and volume of the individual elements x, y, and z moments over each element total
!particle surface area and volume
!------------------------------------------------------------------------------------------------------------


        DO i = 1, mint
            xi  = xiq(i)
            eta = etq(i)
            CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
            &             p(i2,1),   p(i2,2),  p(i2,3),    &
            &             p(i3,1),   p(i3,2),  p(i3,3),    &
            &             p(i4,1),   p(i4,2),  p(i4,3),    &
            &             p(i5,1),   p(i5,2),  p(i5,3),    &
            &             p(i6,1),   p(i6,2),  p(i6,3),    &
            &             al,        be,       ga,         &
            &             xi,        eta,                  &
            &             x,         y,         z,         &
            &             DxDxi,     DyDxi,    DzDxi,      &
            &             DxDet,     DyDet,    DzDet,      &
            &             vnx,       vny,      vnz,        &
            &             hs)
            cf = hs*wq(i)
            arel(k) = arel(k) + cf
            xmom(k) = xmom(k) + cf*x
            ymom(k) = ymom(k) + cf*y
            zmom(k) = zmom(k) + cf*z
            vlm = vlm + (x*vnx+y*vny+z*vnz)*cf
        END DO


        arel(k) = 0.5D0*arel(k)
        xmom(k) = 0.5D0*xmom(k)
        ymom(k) = 0.5D0*ymom(k)
        zmom(k) = 0.5D0*zmom(k)
        area = area +arel(k)
        cx   = cx   + xmom(k)
        cy   = cy   + ymom(k)
        cz   = cz   + zmom(k)
!------------------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the nftty
!formula (4.2.10) of Pozrikidis (1997)
!------------------------------------------------------------------------------------------------------------
        xxi(1) = 0.0D0
        eet(1) = 0.0D0
        xxi(2) = 1.0D0
        eet(2) = 0.0D0
        xxi(3) = 0.0D0
        eet(3) = 1.0D0
        xxi(4) = al
        eet(4) = 0.0D0
        xxi(5) = ga
        eet(5) = gac
        xxi(6) = 0.0D0
        eet(6) = be
        DO i = 1, 6
            xi  = xxi(i)
            eta = eet(i)
            CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
            &             p(i2,1),   p(i2,2),  p(i2,3),    &
            &             p(i3,1),   p(i3,2),  p(i3,3),    &
            &             p(i4,1),   p(i4,2),  p(i4,3),    &
            &             p(i5,1),   p(i5,2),  p(i5,3),    &
```



```
            &             p(i6,1),   p(i6,2),  p(i6,3),    &
            &             al,        be,       ga,         &
            &             xi,        eta,                  &
            &             x,         y,        z,          &
            &             DxDx(i),   DyDx(i),  DzDx(i),    &
            &             DxDe(i),   DyDe(i),  DzDe(i),    &
            &             vx(i),     vy(i),    vz(i),      &
            &             hs)
        END DO
!------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        crvx = 0.0D0
        crvy = 0.0D0
        crvz = 0.0D0
!------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-4-2
!------------------------------------------------------------------------------------------------------------
        bvx1 = vy(1)*DzDx(1)-vz(1)*DyDx(1)
        bvy1 = vz(1)*DxDx(1)-vx(1)*DzDx(1)
        bvz1 = vx(1)*DyDx(1)-vy(1)*DxDx(1)
!------------------------------------------------------------------------------------------------------------
        bvx2 = vy(4)*DzDx(4)-vz(4)*DyDx(4)
        bvy2 = vz(4)*DxDx(4)-vx(4)*DzDx(4)
        bvz2 = vx(4)*DyDx(4)-vy(4)*DxDx(4)
!------------------------------------------------------------------------------------------------------------
        bvx3 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
        bvy3 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
        bvz3 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
!------------------------------------------------------------------------------------------------------------
        crvx = al*bvx1 + bvx2 + alc*bvx3
        crvy = al*bvy1 + bvy2 + alc*bvy3
        crvz = al*bvz1 + bvz2 + alc*bvz3
!------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-5-3
!------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
!------------------------------------------------------------------------------------------------------------
        bvx1 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
        bvy1 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
        bvz1 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
!------------------------------------------------------------------------------------------------------------
        bvx2 = vy(5)*DzDx(5)-vz(5)*DyDx(5)
        bvy2 = vz(5)*DxDx(5)-vx(5)*DzDx(5)
        bvz2 = vx(5)*DyDx(5)-vy(5)*DxDx(5)
!------------------------------------------------------------------------------------------------------------
        bvx3 = vy(3)*DzDx(3)-vz(3)*DyDx(3)
        bvy3 = vz(3)*DxDx(3)-vx(3)*DzDx(3)
        bvz3 = vx(3)*DyDx(3)-vy(3)*DxDx(3)
!------------------------------------------------------------------------------------------------------------
        crvx = crvx - gac*bvx1 - bvx2 - ga*bvx3
        crvy = crvy - gac*bvy1 - bvy2 - ga*bvy3
        crvz = crvz - gac*bvz1 - bvz2 - ga*bvz3
!------------------------------------------------------------------------------------------------------------
        bvx1 = vy(2)*DzDe(2)-vz(2)*DyDe(2)
        bvy1 = vz(2)*DxDe(2)-vx(2)*DzDe(2)
        bvz1 = vx(2)*DyDe(2)-vy(2)*DxDe(2)
```

```fortran
!------------------------------------------------------------------------------------------------------------
        bvx2 = vy(5)*DzDe(5)-vz(5)*DyDe(5)
        bvy2 = vz(5)*DxDe(5)-vx(5)*DzDe(5)
        bvz2 = vx(5)*DyDe(5)-vy(5)*DxDe(5)
!------------------------------------------------------------------------------------------------------------
        bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
        bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
        bvz3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
!------------------------------------------------------------------------------------------------------------
        crvx = crvx + gac*bvx1 + bvx2 + ga*bvx3
        crvy = crvy + gac*bvy1 + bvy2 + ga*bvy3
        crvz = crvz + gac*bvz1 + bvz2 + ga*bvz3
!------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3-6-1
!------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
!------------------------------------------------------------------------------------------------------------
        bvx1 = vy(1)*DzDe(1)-vz(1)*DyDe(1)
        bvy1 = vz(1)*DxDe(1)-vx(1)*DzDe(1)
        bvz1 = vx(1)*DyDe(1)-vy(1)*DxDe(1)
!------------------------------------------------------------------------------------------------------------
        bvx2 = vy(6)*DzDe(6)-vz(6)*DyDe(6)
        bvy2 = vz(6)*DxDe(6)-vx(6)*DzDe(6)
        bvz2 = vx(6)*DyDe(6)-vy(6)*DxDe(6)
!------------------------------------------------------------------------------------------------------------
        bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
        bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
        bvz3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
!------------------------------------------------------------------------------------------------------------
        crvx = crvx - be*bvx1 - bvx2 - bec*bvx3
        crvy = crvy - be*bvy1 - bvy2 - bec*bvy3
        crvz = crvz - be*bvz1 - bvz2 - bec*bvz3
!------------------------------------------------------------------------------------------------------------
        fil = DSQRT(crvx**2+crvy**2+crvz**2)
!------------------------------------------------------------------------------------------------------------
        crvx = crvx/fil
        crvy = crvy/fil
        crvz = crvz/fil
!------------------------------------------------------------------------------------------------------------
! will project curvature vector onto the normal vector at the centroid only needs the colocation verctors
!In this value of curvature I made a division per 8 (1/8.0) when i expected do for 4 (1/4.0)
!------------------------------------------------------------------------------------------------------------
        crvmel(k) = (0.125D0*(crvx*vnx0(k)+crvy*vny0(k)+crvz*vnz0(k))*fil)/arel(k)
!        WRITE (9,*)   crvmel(k)

    END DO Outer

!------------------------------------------------------------------------------------------------------------
! final computation of the surface-centroid and volume

!------------------------------------------------------------------------------------------------------------
        cx = cx/area
        cy = cy/area
        cz = cz/area
        vlm = vlm/6.0D0
!    CLOSE (9)
103 FORMAT(10(1x,ES24.16))
!------------------------------------------------------------------------------------------------------------
    END SUBROUTINE elm_geom4
!============================================================================================================
!============================================================================================================
    SUBROUTINE elm_geom2(nelm, npts, mint,   &
    &                  xmom, ymom, zmom,   &
    &                  area, vlm,          &
```



```fortran
    &                  cx,   cy,   cz)
!============================================================================================================
!   This subroutine is a new version of Elm_Geo Subroutine.
!Compute:
!   *The surface area of the individual elements x, y, and z moments over each element
!   *the total particle surface area and volume
!Now, (25 / Augost / 2012) this subroutine was cut.
!============================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog, vna1, &
    &                         alphaQ,  betaQ,  gammaQ, &
    &                         arel,         crvmel,    &
    &                         vnx0,     vny0,    vnz0, &
    &                         ZZ,           WW,        &
    &                         xiq, etq, wq
!============================================================================================================
    IMPLICIT NONE
!============================================================================================================
!   Variables
!------------------------------------------------------------------------------------------------------------
    INTEGER,           INTENT(IN)  :: nelm                                 !number of elements
    INTEGER,           INTENT(IN)  :: npts                                 !number of poins on the surface
    INTEGER,           INTENT(IN)  :: mint                                 !order of triangle quadrature
    REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom    !coordinates of the moments of the drop
    REAL (KIND = DBL), INTENT(OUT) :: area, vlm                           !area and volume of each element
    REAL (KIND = DBL), INTENT(OUT) :: cx,   cy,   cz                      !drop's centroid coordinates
!============================================================================================================
!   Variables inside the subroutine
!------------------------------------------------------------------------------------------------------------
    INTEGER  :: i, k                                    !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                  !indices to obtain node numbers from each element
!------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: xi, eta                            !variables of weigth to integrate over a triangle
    REAL (KIND = DBL) :: x, y, z                            !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi                !coordinates of the tangential vector over the xi axis
    REAL (KIND = DBL) :: DxDet, DyDet, DzDet                !coordinates of the tangential vector over the eta axis
    REAL (KIND = DBL) :: vnx, vny, vnz                      !normal vector coordinates of the element
    REAL (KIND = DBL) :: hs                                 !surface metric on a triangle
    REAL (KIND = DBL) :: al, be, ga, alc, bec, gac          !integration weigth coefficients
    REAL (KIND = DBL) :: cf                                 !integration weigth coefficients
    REAL (KIND = DBL), DIMENSION(6) ::xxi, eet              !variables of weigth over in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDx, DyDx, DzDx      !tangential vector over xi axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDe, DyDe, DzDe      !tangential vector over eta axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz           !normal vector in triangle (xi,eta)
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &               !binormal vectors around in triangle (xi,eta)
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  crvx, crvy, crvz                  !curvature
!============================================================================================================
!   initialize
!------------------------------------------------------------------------------------------------------------
    area = 0.0D0
    vlm  = 0.0D0
    cx = 0.0D0
    cy = 0.0D0
    cz = 0.0D0
    arel = 0.0D0
    xmom = 0.0D0
    ymom = 0.0D0
    zmom = 0.0D0
    crvmel = 0.0D0
!------------------------------------------------------------------------------------------------------------
    OPEN (9,file="curvmel.out")

    Outer: DO k = 1, nelm

        i1 = n(k,1)
        i2 = n(k,2)
        i3 = n(k,3)
        i4 = n(k,4)
        i5 = n(k,5)
        i6 = n(k,6)
```

```
       al  = alphaQ(k)
       be  = betaQ(k)
       ga  = gammaQ(k)
       alc = 1.0D0-al
       bec = 1.0D0-be
       gac = 1.0D0-ga
!-------------------------------------------------------------------------------------------------------------
!   Compute surface area and volume of the individual elements x, y, and z moments over each element total
!particle surface area and volume
!-------------------------------------------------------------------------------------------------------------


       DO i = 1, mint
           xi  = xiq(i)
           eta = etq(i)
           CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
           &            p(i2,1),   p(i2,2),  p(i2,3),    &
           &            p(i3,1),   p(i3,2),  p(i3,3),    &
           &            p(i4,1),   p(i4,2),  p(i4,3),    &
           &            p(i5,1),   p(i5,2),  p(i5,3),    &
           &            p(i6,1),   p(i6,2),  p(i6,3),    &
           &            al,        be,       ga,         &
           &            xi,        eta,                  &
           &            x,         y,         z,         &
           &            DxDxi,     DyDxi,    DzDxi,      &
           &            DxDet,     DyDet,    DzDet,      &
           &            vnx,       vny,      vnz,        &
           &            hs)
           cf = hs*wq(i)
           arel(k) = arel(k) + cf
           xmom(k) = xmom(k) + cf*x
           ymom(k) = ymom(k) + cf*y
           zmom(k) = zmom(k) + cf*z
           vlm = vlm + (x*vnx+y*vny+z*vnz)*cf
       END DO


       arel(k) = 0.5D0*arel(k)
       xmom(k) = 0.5D0*xmom(k)
       ymom(k) = 0.5D0*ymom(k)
       zmom(k) = 0.5D0*zmom(k)
       area = area +arel(k)
       cx   = cx   + xmom(k)
       cy   = cy   + ymom(k)
       cz   = cz   + zmom(k)
!-------------------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the nftty
!formula (4.2.10) of Pozrikidis (1997)
!-------------------------------------------------------------------------------------------------------------
       xxi(1) = 0.0D0
       eet(1) = 0.0D0
       xxi(2) = 1.0D0
       eet(2) = 0.0D0
       xxi(3) = 0.0D0
       eet(3) = 1.0D0
       xxi(4) = al
       eet(4) = 0.0D0
       xxi(5) = ga
       eet(5) = gac
       xxi(6) = 0.0D0
       eet(6) = be
       DO i = 1, 6
           xi  = xxi(i)
           eta = eet(i)
           CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
           &            p(i2,1),   p(i2,2),  p(i2,3),    &
           &            p(i3,1),   p(i3,2),  p(i3,3),    &
           &            p(i4,1),   p(i4,2),  p(i4,3),    &
           &            p(i5,1),   p(i5,2),  p(i5,3),    &
           &            p(i6,1),   p(i6,2),  p(i6,3),    &
           &            al,        be,       ga,         &
           &            xi,        eta,                  &
```



```
           &            x,         y,        z,          &
           &            DxDx(i),   DyDx(i),  DzDx(i),    &
           &            DxDe(i),   DyDe(i),  DzDe(i),    &
           &            vx(i),     vy(i),    vz(i),      &
           &            hs)
       END DO
!-------------------------------------------------------------------------------------------------------------
       bvx1 = 0.0D0
       bvy1 = 0.0D0
       bvz1 = 0.0D0
       bvx2 = 0.0D0
       bvy2 = 0.0D0
       bvz2 = 0.0D0
       bvx3 = 0.0D0
       bvy3 = 0.0D0
       bvz3 = 0.0D0
       crvx = 0.0D0
       crvy = 0.0D0
       crvz = 0.0D0
!-------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-4-2
!-------------------------------------------------------------------------------------------------------------
       bvx1 = vna1(1+(6*(i-1)),2)*DzDx(1)-vna1(1+(6*(i-1)),3)*DyDx(1)
       bvy1 = vna1(1+(6*(i-1)),3)*DxDx(1)-vna1(1+(6*(i-1)),1)*DzDx(1)
       bvz1 = vna1(1+(6*(i-1)),1)*DyDx(1)-vna1(1+(6*(i-1)),2)*DxDx(1)
!-------------------------------------------------------------------------------------------------------------
           !vna1(1+(6*(i-1)),1), vna1(1+(6*(i-1)),2), vna1(1+(6*(i-1)),3), &
           !vna1(2+(6*(i-1)),1), vna1(2+(6*(i-1)),2), vna1(2+(6*(i-1)),3), &
           !vna1(3+(6*(i-1)),1), vna1(3+(6*(i-1)),2), vna1(3+(6*(i-1)),3), &
           !vna1(4+(6*(i-1)),1), vna1(4+(6*(i-1)),2), vna1(4+(6*(i-1)),3), &
           !vna1(5+(6*(i-1)),1), vna1(5+(6*(i-1)),2), vna1(5+(6*(i-1)),3), &
           !vna1(6+(6*(i-1)),1), vna1(6+(6*(i-1)),2), vna1(6+(6*(i-1)),3), &
!-------------------------------------------------------------------------------------------------------------
       bvx2 = vna1(4+(6*(i-1)),2)*DzDx(4)-vna1(4+(6*(i-1)),3)*DyDx(4)
       bvy2 = vna1(4+(6*(i-1)),3)*DxDx(4)-vna1(4+(6*(i-1)),1)*DzDx(4)
       bvz2 = vna1(4+(6*(i-1)),1)*DyDx(4)-vna1(4+(6*(i-1)),2)*DxDx(4)
!-------------------------------------------------------------------------------------------------------------
       bvx3 = vna1(2+(6*(i-1)),2)*DzDx(2)-vna1(2+(6*(i-1)),3)*DyDx(2)
       bvy3 = vna1(2+(6*(i-1)),3)*DxDx(2)-vna1(2+(6*(i-1)),1)*DzDx(2)
       bvz3 = vna1(2+(6*(i-1)),1)*DyDx(2)-vna1(2+(6*(i-1)),2)*DxDx(2)
!-------------------------------------------------------------------------------------------------------------
       crvx = al*bvx1 + bvx2 + alc*bvx3
       crvy = al*bvy1 + bvy2 + alc*bvy3
       crvz = al*bvz1 + bvz2 + alc*bvz3
!-------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-5-3
!-------------------------------------------------------------------------------------------------------------
       bvx1 = 0.0D0
       bvy1 = 0.0D0
       bvz1 = 0.0D0
       bvx2 = 0.0D0
       bvy2 = 0.0D0
       bvz2 = 0.0D0
       bvx3 = 0.0D0
       bvy3 = 0.0D0
       bvz3 = 0.0D0
!-------------------------------------------------------------------------------------------------------------
       bvx1 = vna1(2+(6*(i-1)),2)*DzDx(2)-vna1(2+(6*(i-1)),3)*DyDx(2)
       bvy1 = vna1(2+(6*(i-1)),3)*DxDx(2)-vna1(2+(6*(i-1)),1)*DzDx(2)
       bvz1 = vna1(2+(6*(i-1)),1)*DyDx(2)-vna1(2+(6*(i-1)),2)*DxDx(2)
!-------------------------------------------------------------------------------------------------------------
       bvx2 = vna1(5+(6*(i-1)),2)*DzDx(5)-vna1(5+(6*(i-1)),3)*DyDx(5)
       bvy2 = vna1(5+(6*(i-1)),3)*DxDx(5)-vna1(5+(6*(i-1)),1)*DzDx(5)
       bvz2 = vna1(5+(6*(i-1)),1)*DyDx(5)-vna1(5+(6*(i-1)),2)*DxDx(5)
!-------------------------------------------------------------------------------------------------------------
       bvx3 = vna1(3+(6*(i-1)),2)*DzDx(3)-vna1(3+(6*(i-1)),3)*DyDx(3)
       bvy3 = vna1(3+(6*(i-1)),3)*DxDx(3)-vna1(3+(6*(i-1)),1)*DzDx(3)
       bvz3 = vna1(3+(6*(i-1)),1)*DyDx(3)-vna1(3+(6*(i-1)),2)*DxDx(3)
!-------------------------------------------------------------------------------------------------------------
       crvx = crvx - gac*bvx1 - bvx2 - ga*bvx3
       crvy = crvy - gac*bvy1 - bvy2 - ga*bvy3
       crvz = crvz - gac*bvz1 - bvz2 - ga*bvz3
```

```
!----------------------------------------------------------------------------------------------------
        bvx1 = vna1(2+(6*(i-1)),2)*DzDe(2)-vna1(2+(6*(i-1)),3)*DyDe(2)
        bvy1 = vna1(2+(6*(i-1)),3)*DxDe(2)-vna1(2+(6*(i-1)),1)*DzDe(2)
        bvz1 = vna1(2+(6*(i-1)),1)*DyDe(2)-vna1(2+(6*(i-1)),2)*DxDe(2)
!----------------------------------------------------------------------------------------------------
        bvx2 = vna1(5+(6*(i-1)),2)*DzDe(5)-vna1(5+(6*(i-1)),3)*DyDe(5)
        bvy2 = vna1(5+(6*(i-1)),3)*DxDe(5)-vna1(5+(6*(i-1)),1)*DzDe(5)
        bvz2 = vna1(5+(6*(i-1)),1)*DyDe(5)-vna1(5+(6*(i-1)),2)*DxDe(5)
!----------------------------------------------------------------------------------------------------
        bvx3 = vna1(3+(6*(i-1)),2)*DzDe(3)-vna1(3+(6*(i-1)),3)*DyDe(3)
        bvy3 = vna1(3+(6*(i-1)),3)*DxDe(3)-vna1(3+(6*(i-1)),1)*DzDe(3)
        bvz3 = vna1(3+(6*(i-1)),1)*DyDe(3)-vna1(3+(6*(i-1)),2)*DxDe(3)
!----------------------------------------------------------------------------------------------------
        crvx = crvx + gac*bvx1 + bvx2 + ga*bvx3
        crvy = crvy + gac*bvy1 + bvy2 + ga*bvy3
        crvz = crvz + gac*bvz1 + bvz2 + ga*bvz3
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3-6-1
!----------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
!----------------------------------------------------------------------------------------------------
        bvx1 = vna1(1+(6*(i-1)),2)*DzDe(1)-vna1(1+(6*(i-1)),3)*DyDe(1)
        bvy1 = vna1(1+(6*(i-1)),3)*DxDe(1)-vna1(1+(6*(i-1)),1)*DzDe(1)
        bvz1 = vna1(1+(6*(i-1)),1)*DyDe(1)-vna1(1+(6*(i-1)),2)*DxDe(1)
!----------------------------------------------------------------------------------------------------
        bvx2 = vna1(6+(6*(i-1)),2)*DzDe(6)-vna1(6+(6*(i-1)),3)*DyDe(6)
        bvy2 = vna1(6+(6*(i-1)),3)*DxDe(6)-vna1(6+(6*(i-1)),1)*DzDe(6)
        bvz2 = vna1(6+(6*(i-1)),1)*DyDe(6)-vna1(6+(6*(i-1)),2)*DxDe(6)
!----------------------------------------------------------------------------------------------------
        bvx3 = vna1(3+(6*(i-1)),2)*DzDe(3)-vna1(3+(6*(i-1)),3)*DyDe(3)
        bvy3 = vna1(3+(6*(i-1)),3)*DxDe(3)-vna1(3+(6*(i-1)),1)*DzDe(3)
        bvz3 = vna1(3+(6*(i-1)),1)*DyDe(3)-vna1(3+(6*(i-1)),2)*DxDe(3)
!----------------------------------------------------------------------------------------------------
        crvx = crvx - be*bvx1 - bvx2 - bec*bvx3
        crvy = crvy - be*bvy1 - bvy2 - bec*bvy3
        crvz = crvz - be*bvz1 - bvz2 - bec*bvz3
!----------------------------------------------------------------------------------------------------
! will project curvature vector onto the normal vector at the centroid only needs the colocation verctors
!In this value of curvature I made a division per 8 (1/8.0) when i expected do for 4 (1/4.0)
!----------------------------------------------------------------------------------------------------
        crvmel(k) = 0.250D0*(crvx*vnx0(k)+crvy*vny0(k)+crvz*vnz0(k)) !/arel(k)
        WRITE (9,*)   crvmel(k)

    END DO Outer

!----------------------------------------------------------------------------------------------------
! final computation of the surface-centroid and volume

!----------------------------------------------------------------------------------------------------
        cx = cx/area
        cy = cy/area
        cz = cz/area
        vlm = vlm/6.0D0
    CLOSE (9)
103 FORMAT(10(1x,ES24.16))
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE elm_geom2
!====================================================================================================
 SUBROUTINE elm_geom3(nelm, npts, mint,   &
    &                   xmom, ymom, zmom,   &
    &                   area, vlm,          &
    &                   cx,   cy,    cz)
!====================================================================================================
!   This subroutine is a new version of Elm_Geo Subroutine.
```



```
!Compute:
!   *The surface area of the individual elements x, y, and z moments over each element
!   *the total particle surface area and volume
!Now, (25 / Augost / 2012) this subroutine was cut.
!====================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog,&
    &                         alphaQ,  betaQ,  gammaQ, &
    &                         arel,        crvmel,    &
    &                         vnx0,     vny0,    vnz0, &
    &                         ZZ,          WW,        &
    &                         xiq, etq, wq
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER,           INTENT(IN)  :: nelm                              !number of elements
    INTEGER,           INTENT(IN)  :: npts                              !number of poins on the surface
    INTEGER,           INTENT(IN)  :: mint                              !order of triangle quadrature
    REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom    !coordinates of the moments of the drop
    REAL (KIND = DBL), INTENT(OUT) :: area, vlm                         !area and volume of each element
    REAL (KIND = DBL), INTENT(OUT) :: cx,   cy,   cz                    !drop's centroid coordinates
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, k                                       !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                     !indices to obtain node numbers from each element
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: xi, eta                           !variables of weigth to integrate over a triangle
    REAL (KIND = DBL) :: x, y, z                           !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi               !coordinates of the tangential vector over the xi axis
    REAL (KIND = DBL) :: DxDet, DyDet, DzDet               !coordinates of the tangential vector over the eta axis
    REAL (KIND = DBL) :: vnx, vny, vnz                     !normal vector coordinates of the element
    REAL (KIND = DBL) :: hs, xs, es                        !surface metric on a triangle
    REAL (KIND = DBL) :: al, be, ga, alc, bec, gac         !integration weigth coefficients
    REAL (KIND = DBL) :: cf , fil                           !integration weigth coefficients
    REAL (KIND = DBL), DIMENSION(6) ::xxi, eet             !variables of weigth over in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDx, DyDx, DzDx     !tangential vector over xi axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDe, DyDe, DzDe     !tangential vector over eta axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz          !normal vector in triangle (xi,eta)
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &              !binormal vectors around in triangle (xi,eta)
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  crvx, crvy, crvz                 !curvature
!====================================================================================================
!   initialize
!----------------------------------------------------------------------------------------------------
    area = 0.0D0
    vlm  = 0.0D0
    cx = 0.0D0
    cy = 0.0D0
    cz = 0.0D0
    arel = 0.0D0
    xmom = 0.0D0
    ymom = 0.0D0
    zmom = 0.0D0
    crvmel = 0.0D0
    fil = 0.0D0
!----------------------------------------------------------------------------------------------------
    OPEN (9,file="curvmel.out")

    Outer: DO k = 1, nelm

        i1 = n(k,1)
        i2 = n(k,2)
        i3 = n(k,3)
        i4 = n(k,4)
        i5 = n(k,5)
        i6 = n(k,6)
        al  = alphaQ(k)
        be  = betaQ(k)
```

```fortran
        ga  = gammaQ(k)
        alc = 1.0D0-al
        bec = 1.0D0-be
        gac = 1.0D0-ga
!------------------------------------------------------------------------------------------------------------------
!   Compute surface area and volume of the individual elements x, y, and z moments over each element total
!particle surface area and volume
!------------------------------------------------------------------------------------------------------------------


        DO i = 1, mint
            xi  = xiq(i)
            eta = etq(i)
            CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
            &             p(i2,1),   p(i2,2),  p(i2,3),    &
            &             p(i3,1),   p(i3,2),  p(i3,3),    &
            &             p(i4,1),   p(i4,2),  p(i4,3),    &
            &             p(i5,1),   p(i5,2),  p(i5,3),    &
            &             p(i6,1),   p(i6,2),  p(i6,3),    &
            &             al,        be,       ga,         &
            &             xi,        eta,                  &
            &             x,         y,         z,         &
            &             DxDxi,     DyDxi,    DzDxi,      &
            &             DxDet,     DyDet,    DzDet,      &
            &             vnx,       vny,      vnz,        &
            &             hs)
            cf = hs*wq(i)
            arel(k) = arel(k) + cf
            xmom(k) = xmom(k) + cf*x
            ymom(k) = ymom(k) + cf*y
            zmom(k) = zmom(k) + cf*z
            vlm = vlm + (x*vnx+y*vny+z*vnz)*cf
            cf=0.0D0
        END DO


        arel(k) = 0.5D0*arel(k)
        xmom(k) = 0.5D0*xmom(k)
        ymom(k) = 0.5D0*ymom(k)
        zmom(k) = 0.5D0*zmom(k)
        area = area +arel(k)
        cx   = cx   + xmom(k)
        cy   = cy   + ymom(k)
        cz   = cz   + zmom(k)
!------------------------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the nftty
!formula (4.2.10) of Pozrikidis (1997)
!------------------------------------------------------------------------------------------------------------------
        xxi(1) = 0.0D0
        eet(1) = 0.0D0
        xxi(2) = 1.0D0
        eet(2) = 0.0D0
        xxi(3) = 0.0D0
        eet(3) = 1.0D0
        xxi(4) = al
        eet(4) = 0.0D0
        xxi(5) = ga
        eet(5) = gac
        xxi(6) = 0.0D0
        eet(6) = be
        DO i = 1, 6
            xi  = xxi(i)
            eta = eet(i)
            CALL interp_p3(p(i1,1),   p(i1,2),  p(i1,3),    &
            &             p(i2,1),   p(i2,2),  p(i2,3),    &
            &             p(i3,1),   p(i3,2),  p(i3,3),    &
            &             p(i4,1),   p(i4,2),  p(i4,3),    &
            &             p(i5,1),   p(i5,2),  p(i5,3),    &
            &             p(i6,1),   p(i6,2),  p(i6,3),    &
            &             al,        be,       ga,         &
            &             xi,        eta,                  &
            &             x,         y,        z,          &
```



```fortran
            &             DxDx(i),   DyDx(i),  DzDx(i),    &
            &             DxDe(i),   DyDe(i),  DzDe(i),    &
            &             vx(i),     vy(i),    vz(i),      &
            &             hs)
        END DO
!------------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        crvx = 0.0D0
        crvy = 0.0D0
        crvz = 0.0D0
!------------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-4-2
!------------------------------------------------------------------------------------------------------------------
        bvx1 = vy(1)*DzDx(1)-vz(1)*DyDx(1)
        bvy1 = vz(1)*DxDx(1)-vx(1)*DzDx(1)
        bvz1 = vx(1)*DyDx(1)-vy(1)*DxDx(1)
!------------------------------------------------------------------------------------------------------------------
        bvx2 = vy(4)*DzDx(4)-vz(4)*DyDx(4)
        bvy2 = vz(4)*DxDx(4)-vx(4)*DzDx(4)
        bvz2 = vx(4)*DyDx(4)-vy(4)*DxDx(4)
!------------------------------------------------------------------------------------------------------------------
        bvx3 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
        bvy3 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
        bvz3 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
!------------------------------------------------------------------------------------------------------------------
        crvx = al*bvx1 + bvx2 + alc*bvx3
        crvy = al*bvy1 + bvy2 + alc*bvy3
        crvz = al*bvz1 + bvz2 + alc*bvz3
!------------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-5-3
!------------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
!------------------------------------------------------------------------------------------------------------------
        bvx1 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
        bvy1 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
        bvz1 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
!------------------------------------------------------------------------------------------------------------------
        bvx2 = vy(5)*DzDx(5)-vz(5)*DyDx(5)
        bvy2 = vz(5)*DxDx(5)-vx(5)*DzDx(5)
        bvz2 = vx(5)*DyDx(5)-vy(5)*DxDx(5)
!------------------------------------------------------------------------------------------------------------------
        bvx3 = vy(3)*DzDx(3)-vz(3)*DyDx(3)
        bvy3 = vz(3)*DxDx(3)-vx(3)*DzDx(3)
        bvz3 = vx(3)*DyDx(3)-vy(3)*DxDx(3)
!------------------------------------------------------------------------------------------------------------------
        crvx = crvx - gac*bvx1 - bvx2 - ga*bvx3
        crvy = crvy - gac*bvy1 - bvy2 - ga*bvy3
        crvz = crvz - gac*bvz1 - bvz2 - ga*bvz3
!------------------------------------------------------------------------------------------------------------------
        bvx1 = vy(2)*DzDe(2)-vz(2)*DyDe(2)
        bvy1 = vz(2)*DxDe(2)-vx(2)*DzDe(2)
        bvz1 = vx(2)*DyDe(2)-vy(2)*DxDe(2)
!------------------------------------------------------------------------------------------------------------------
        bvx2 = vy(5)*DzDe(5)-vz(5)*DyDe(5)
        bvy2 = vz(5)*DxDe(5)-vx(5)*DzDe(5)
        bvz2 = vx(5)*DyDe(5)-vy(5)*DxDe(5)
```

```
!-------------------------------------------------------------------------------------------------------------------
        bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
        bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
        bvz3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
!-------------------------------------------------------------------------------------------------------------------
        crvx = crvx + gac*bvx1 + bvx2 + ga*bvx3
        crvy = crvy + gac*bvy1 + bvy2 + ga*bvy3
        crvz = crvz + gac*bvz1 + bvz2 + ga*bvz3
!-------------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3-6-1
!-------------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
!-------------------------------------------------------------------------------------------------------------------
        bvx1 = vy(1)*DzDe(1)-vz(1)*DyDe(1)
        bvy1 = vz(1)*DxDe(1)-vx(1)*DzDe(1)
        bvz1 = vx(1)*DyDe(1)-vy(1)*DxDe(1)
!-------------------------------------------------------------------------------------------------------------------
        bvx2 = vy(6)*DzDe(6)-vz(6)*DyDe(6)
        bvy2 = vz(6)*DxDe(6)-vx(6)*DzDe(6)
        bvz2 = vx(6)*DyDe(6)-vy(6)*DxDe(6)
!-------------------------------------------------------------------------------------------------------------------
        bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
        bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
        bvz3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
!-------------------------------------------------------------------------------------------------------------------
        crvx = crvx - be*bvx1 - bvx2 - bec*bvx3
        crvy = crvy - be*bvy1 - bvy2 - bec*bvy3
        crvz = crvz - be*bvz1 - bvz2 - bec*bvz3
!-------------------------------------------------------------------------------------------------------------------
!        fil = DSQRT(crvx**2+crvy**2+crvz**2)
!!!-------------------------------------------------------------------------------------------------------------------
-
!        crvx = crvx/fil
!        crvy = crvy/fil
!        crvz = crvz/fil
!-------------------------------------------------------------------------------------------------------------------
! will project curvature vector onto the normal vector at the centroid only needs the colocation verctors
!In this value of curvature I made a division per 8 (1/8.0) when i expected do for 4 (1/4.0)
!-------------------------------------------------------------------------------------------------------------------
        crvmel(k) = 0.250D0*(crvx*vnx0(k)+crvy*vny0(k)+crvz*vnz0(k))/arel(k)
        WRITE (9,*)   crvmel(k)

    END DO Outer

!-------------------------------------------------------------------------------------------------------------------
! final computation of the surface-centroid and volume

!-------------------------------------------------------------------------------------------------------------------
        cx = cx/area
        cy = cy/area
        cz = cz/area
        vlm = vlm/6.0D0
    CLOSE (9)
103 FORMAT(10(1x,ES24.16))
!-------------------------------------------------------------------------------------------------------------------
    END SUBROUTINE elm_geom3
!===================================================================================================================
    END MODULE Mod_Prtcl_3D_Geo
```

```fortran
 MODULE Mod_Prtcl_3D_Geo
!==========================================================================================================
!   Version: 0.5 created on  26 / IX / 2007
!                                                                                          C. Pozrikidis
!----------------------------------------------------------------------------------------------------------
!   Version: 0.7 created on  -- / III /2010
!                                                                               Marco Antonio Reyes Huesca
!----------------------------------------------------------------------------------------------------------
!   Version: 0.9 created on 23 / 08 / 2012
!   Version: 1.0 created on 14 / 11 / 2012
!                                                                                  Alfredo Sanjuan Sanjuan
!==========================================================================================================
    CONTAINS
!==========================================================================================================
    SUBROUTINE printel(k, Index, c)
!==========================================================================================================
!    This subroutine prints drop's geometry. It has two options.
!Print successive nodes of element k in file unit 1
!   Index = 1: print the whole element
!   Index = 2: print the 4 subelements
!----------------------------------------------------------------------------------------------------------
!   Variables
!----------------------------------------------------------------------------------------------------------
!   k ............................ number of element
!   index ........................ type of print
!   nfour ........................ index to print only the element
!   nseven ....................... index to print the subelements
!   c ............................ firstly, it was the value of shear strain, now, it will change. ;)
!==========================================================================================================
    USE Mod_SharedVars, ONLY: DBL, ULog, UGeo, p, ne, n, nbe
!==========================================================================================================
    IMPLICIT NONE
!==========================================================================================================
    INTEGER,                         INTENT(IN) :: k, index
!----------------------------------------------------------------------------------------------------------
    INTEGER :: nfour, nseven, i
    REAL (KIND = DBL), DIMENSION(:), INTENT(IN) :: c
!----------------------------------------------------------------------------------------------------------
! constants
    nfour  = 4
    nseven = 1
!----------------------------------------------------------------------------------------------------------
!   There is a CASE instrction to print the element.
!----------------------------------------------------------------------------------------------------------
    SELECT CASE (index)
    CASE (1)
        WRITE (UGeo,100) nseven
        i = n(k,1)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,2)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,5)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,3)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,1)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
    CASE(2)
        !--- first
        WRITE (UGeo,100) nfour
        i = n(k,1)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,1)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
```



```fortran
        !--- second
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,2)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,5)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        !--- third
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,5)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        !--- fourth
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,5)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,3)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
    CASE DEFAULT
        WRITE (UGeo,100) nfour
        i = n(k,1)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,1)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,2)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,5)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,5)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,4)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,5)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,3)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        i = n(k,6)
        WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
        WRITE (ULog,*)
        WRITE (ULog,*) '    Geo_Printel'
        WRITE (ULog,*)
        WRITE (ULog,*) '    Chosen index is not available'
        WRITE (ULog,*) '    It was taken index= 2'
    END SELECT
100 FORMAT(1x,i4,10(1x,ES24.16))
101 FORMAT(10(1x,ES24.16))
    END SUBROUTINE printel
!==========================================================================================================
!==========================================================================================================
```

```fortran
    SUBROUTINE abc(x1, y1, z1,  &
    &              x2, y2, z2,  &
    &              x3, y3, z3,  &
    &              x4, y4, z4,  &
    &              x5, y5, z5,  &
    &              x6, y6, z6,  &
    &              al, be, ga)
!=====================================================================================================
!   This subroutine compute the parametric representation constants alpha, beta, gamma
!=====================================================================================================
    USE Mod_SharedVars, ONLY: DBL
!=====================================================================================================
    IMPLICIT NONE
!=====================================================================================================
!   Variables
!-----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN) ::  x1, y1, z1, &     !coordinates of each point in the element
    &                                 x2, y2, z2, &
    &                                 x3, y3, z3, &
    &                                 x4, y4, z4, &
    &                                 x5, y5, z5, &
    &                                 x6, y6, z6
    REAL (KIND = DBL), INTENT(OUT) :: al, be, ga        !constants alpha, beta and gamma (weights)
!=====================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: d42, d41                       !distances of the element on the segment 1 4 2
    REAL (KIND = DBL) :: d63, d61                       !distances of the element on the segment 3 6 1
    REAL (KIND = DBL) :: d52, d53                       !distances of the element on the segment 2 5 3
!=====================================================================================================
!   Initialize
    d42 = (x4 - x2)**2 + (y4 - y2)**2 + (z4 - z2)**2
    d41 = (x4 - x1)**2 + (y4 - y1)**2 + (z4 - z1)**2
    d63 = (x6 - x3)**2 + (y6 - y3)**2 + (z6 - z3)**2
    d61 = (x6 - x1)**2 + (y6 - y1)**2 + (z6 - z1)**2
    d52 = (x5 - x2)**2 + (y5 - y2)**2 + (z5 - z2)**2
    d53 = (x5 - x3)**2 + (y5 - y3)**2 + (z5 - z3)**2
!-----------------------------------------------------------------------------------------------------
    d42 = DSQRT(d42)
    d41 = DSQRT(d41)
    d63 = DSQRT(d63)
    d61 = DSQRT(d61)
    d52 = DSQRT(d52)
    d53 = DSQRT(d53)
!-----------------------------------------------------------------------------------------------------
    al = 1.0D0/(1.0D0 + d42/d41)
    be = 1.0D0/(1.0D0 + d63/d61)
    ga = 1.0D0/(1.0D0 + d52/d53)
!-----------------------------------------------------------------------------------------------------
    END SUBROUTINE abc
!=====================================================================================================
!=====================================================================================================
    SUBROUTINE elm_geom(nelm, npts, mint,   &
    &                   xmom, ymom, zmom,   &
    &                   area, vlm,          &
    &                   cx,   cy,   cz, stride)
!=====================================================================================================
!   This subroutine is a new version of Elm_Geo Subroutine.
!Compute:
!   *The surface area of the individual elements x, y, and z moments over each element
!   *the total particle surface area and volume
!Now, (25 / Augost / 2012) this subroutine was cut.
!=====================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog,&
    &                         alphaQ,  betaQ,  gammaQ, &
    &                         arel,        crvmel,     &
    &                         vnx0,     vny0,    vnz0, &
    &                         ZZ,           WW,        &
    &                         xiq, etq, wq
!=====================================================================================================
    IMPLICIT NONE
```



```fortran
!=====================================================================================================
!   Variables
!-----------------------------------------------------------------------------------------------------
    INTEGER,           INTENT(IN)  :: nelm                              !number of elements
    INTEGER,           INTENT(IN)  :: npts                              !number of poins on the surface
    INTEGER,           INTENT(IN)  :: mint                              !order of triangle quadrature
    INTEGER,           INTENT(IN)  :: stride              !order of triangle and Gauss-Legendre quadratures
    REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom    !coordinates of the moments of the drop
    REAL (KIND = DBL), INTENT(OUT) :: area, vlm                         !area and volume of each element
    REAL (KIND = DBL), INTENT(OUT) :: cx,   cy,   cz                    !drop's centroid coordinates
!=====================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------
    INTEGER  :: i, k                                  !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                !indices to obtain node numbers from each element
!-----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: xi, eta                          !variables of weigth to integrate over a triangle
    REAL (KIND = DBL) :: x, y, z                          !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi              !coordinates of the tangential vector over the xi axis
    REAL (KIND = DBL) :: DxDet, DyDet, DzDet              !coordinates of the tangential vector over the eta axis
    REAL (KIND = DBL) :: vnx, vny, vnz                    !normal vector coordinates of the element
    REAL (KIND = DBL) :: hs                               !surface metric on a triangle
    REAL (KIND = DBL) :: al, be, ga, alc, bec, gac        !integration weigth coefficients
    REAL (KIND = DBL) :: cf, fil                          !integration weigth coefficients
    REAL (KIND = DBL), DIMENSION(6) ::xxi, eet            !variables of weigth over in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDx, DyDx, DzDx    !tangential vector over xi axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDe, DyDe, DzDe    !tangential vector over eta axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz         !normal vector in triangle (xi,eta)
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &             !binormal vectors around in triangle (xi,eta)
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  crvx, crvy, crvz, curv          !curvature
!=====================================================================================================
!   initialize
!-----------------------------------------------------------------------------------------------------
    area = 0.0D0
    vlm  = 0.0D0
    cx = 0.0D0
    cy = 0.0D0
    cz = 0.0D0
    arel = 0.0D0
    xmom = 0.0D0
    ymom = 0.0D0
    zmom = 0.0D0
    crvmel = 0.0D0
    fil = 0.0D0
!-----------------------------------------------------------------------------------------------------
    Outer: DO k = 1, nelm

        i1 = n(k,1)
        i2 = n(k,2)
        i3 = n(k,3)
        i4 = n(k,4)
        i5 = n(k,5)
        i6 = n(k,6)
        al  = alphaQ(k)
        be  = betaQ(k)
        ga  = gammaQ(k)
        alc = 1.0D0-al
        bec = 1.0D0-be
        gac = 1.0D0-ga
!-----------------------------------------------------------------------------------------------------
!   Compute surface area and volume of the individual elements x, y, and z moments over each element total
!particle surface area and volume
!-----------------------------------------------------------------------------------------------------
        DO i = 1, mint
            xi  = xiq(i)
            eta = etq(i)
            CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
            &             p(i2,1),   p(i2,2),  p(i2,3),    &
            &             p(i3,1),   p(i3,2),  p(i3,3),    &
            &             p(i4,1),   p(i4,2),  p(i4,3),    &
```

```fortran
            &              p(i5,1),   p(i5,2),  p(i5,3),    &
            &              p(i6,1),   p(i6,2),  p(i6,3),    &
            &              al,        be,       ga,         &
            &              xi,        eta,                  &
            &              x,         y,         z,         &
            &              DxDxi,     DyDxi,    DzDxi,      &
            &              DxDet,     DyDet,    DzDet,      &
            &              vnx,       vny,      vnz,        &
            &              hs)
            cf = hs*wq(i)
            arel(k) = arel(k) + cf
            xmom(k) = xmom(k) + cf*x
            ymom(k) = ymom(k) + cf*y
            zmom(k) = zmom(k) + cf*z
            vlm = vlm + (x*vnx+y*vny+z*vnz)*cf
        END DO
        arel(k) = 0.5D0*arel(k)
        xmom(k) = 0.5D0*xmom(k)
        ymom(k) = 0.5D0*ymom(k)
        zmom(k) = 0.5D0*zmom(k)
        area = area +arel(k)
        cx   = cx   + xmom(k)
        cy   = cy   + ymom(k)
        cz   = cz   + zmom(k)
!--------------------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the nftty
!formula (4.2.10) of Pozrikidis (1997)
!--------------------------------------------------------------------------------------------------------------
        xxi(1) = 0.0D0
        eet(1) = 0.0D0
        xxi(2) = 1.0D0
        eet(2) = 0.0D0
        xxi(3) = 0.0D0
        eet(3) = 1.0D0
        xxi(4) = al
        eet(4) = 0.0D0
        xxi(5) = ga
        eet(5) = gac
        xxi(6) = 0.0D0
        eet(6) = be
        xi  = 1.0D0/3.0D0
        eta = 1.0D0/3.0D0
            CALL interp_p4(p(i1,1),   p(i1,2),  p(i1,3),    &
            &              p(i2,1),   p(i2,2),  p(i2,3),    &
            &              p(i3,1),   p(i3,2),  p(i3,3),    &
            &              p(i4,1),   p(i4,2),  p(i4,3),    &
            &              p(i5,1),   p(i5,2),  p(i5,3),    &
            &              p(i6,1),   p(i6,2),  p(i6,3),    &
            &              al,        be,       ga,         &
            &              xi,        eta,                  &
            &                       curv, stride            )

!--------------------------------------------------------------------------------------------------------------
! will project curvature vector onto the normal vector at the centroid only needs the colocation verctors
!In this value of curvature I made a division per 8 (1/8.0) when i expected do for 4 (1/4.0)
!--------------------------------------------------------------------------------------------------------------
        crvmel(k) = curv !(0.25D0*(crvx*vnx0(k)+crvy*vny0(k)+crvz*vnz0(k))*fil)/arel(k)
    END DO Outer
!--------------------------------------------------------------------------------------------------------------
! final computation of the surface-centroid and volume

!--------------------------------------------------------------------------------------------------------------
        cx = cx/area
        cy = cy/area
        cz = cz/area
        vlm = vlm/6.0D0
103 FORMAT(10(1x,ES24.16))
!--------------------------------------------------------------------------------------------------------------
    END SUBROUTINE elm_geom
!==============================================================================================================
!==============================================================================================================
    SUBROUTINE elm_geom4(nelm, npts, mint,   &
```



```fortran
    &                   xmom, ymom, zmom,   &
    &                   area, vlm,          &
    &                   cx,   cy,   cz)
!==============================================================================================================
!   This subroutine is a new version of Elm_Geo Subroutine.
!Compute:
!   *The surface area of the individual elements x, y, and z moments over each element
!   *the total particle surface area and volume
!Now, (25 / Augost / 2012) this subroutine was cut.
!==============================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog,&
    &                         alphaQ,  betaQ,  gammaQ, &
    &                         arel,         crvmel,    &
    &                         vnx0,     vny0,    vnz0, &
    &                         ZZ,           WW,        &
    &                         xiq, etq, wq
!==============================================================================================================
    IMPLICIT NONE
!==============================================================================================================
!   Variables
!--------------------------------------------------------------------------------------------------------------
    INTEGER,            INTENT(IN)  :: nelm                              !number of elements
    INTEGER,            INTENT(IN)  :: npts                              !number of poins on the surface
    INTEGER,            INTENT(IN)  :: mint                              !order of triangle quadrature
    REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom    !coordinates of the moments of the drop
    REAL (KIND = DBL), INTENT(OUT) :: area, vlm                          !area and volume of each element
    REAL (KIND = DBL), INTENT(OUT) :: cx,   cy,   cz                     !drop's centroid coordinates
!==============================================================================================================
!   Variables inside the subroutine
!--------------------------------------------------------------------------------------------------------------
    INTEGER  :: i, k                                   !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                 !indices to obtain node numbers from each element
!--------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: xi, eta                           !variables of weigth to integrate over a triangle
    REAL (KIND = DBL) :: x, y, z                           !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi               !coordinates of the tangential vector over the xi axis
    REAL (KIND = DBL) :: DxDet, DyDet, DzDet               !coordinates of the tangential vector over the eta axis
    REAL (KIND = DBL) :: vnx, vny, vnz                     !normal vector coordinates of the element
    REAL (KIND = DBL) :: hs                                !surface metric on a triangle
    REAL (KIND = DBL) :: al, be, ga, alc, bec, gac         !integration weigth coefficients
    REAL (KIND = DBL) :: cf, fil                           !integration weigth coefficients
    REAL (KIND = DBL), DIMENSION(6) ::xxi, eet             !variables of weigth over in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDx, DyDx, DzDx     !tangential vector over xi axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDe, DyDe, DzDe     !tangential vector over eta axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz          !normal vector in triangle (xi,eta)
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &              !binormal vectors around in triangle (xi,eta)
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  crvx, crvy, crvz                 !curvature
!==============================================================================================================
!   initialize
!--------------------------------------------------------------------------------------------------------------
    area = 0.0D0
    vlm  = 0.0D0
    cx = 0.0D0
    cy = 0.0D0
    cz = 0.0D0
    arel = 0.0D0
    xmom = 0.0D0
    ymom = 0.0D0
    zmom = 0.0D0
    crvmel = 0.0D0
    fil = 0.0D0
!--------------------------------------------------------------------------------------------------------------
!    OPEN (9,file="curvmel.out")

    Outer: DO k = 1, nelm

        i1 = n(k,1)
        i2 = n(k,2)
        i3 = n(k,3)
```

```
        i4 = n(k,4)
        i5 = n(k,5)
        i6 = n(k,6)
        al  = alphaQ(k)
        be  = betaQ(k)
        ga  = gammaQ(k)
        alc = 1.0D0-al
        bec = 1.0D0-be
        gac = 1.0D0-ga
!-----------------------------------------------------------------------------------------------------
!   Compute surface area and volume of the individual elements x, y, and z moments over each element total
!particle surface area and volume
!-----------------------------------------------------------------------------------------------------


        DO i = 1, mint
            xi  = xiq(i)
            eta = etq(i)
            CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
            &             p(i2,1),   p(i2,2),  p(i2,3),    &
            &             p(i3,1),   p(i3,2),  p(i3,3),    &
            &             p(i4,1),   p(i4,2),  p(i4,3),    &
            &             p(i5,1),   p(i5,2),  p(i5,3),    &
            &             p(i6,1),   p(i6,2),  p(i6,3),    &
            &             al,        be,       ga,         &
            &             xi,        eta,                  &
            &             x,         y,         z,         &
            &             DxDxi,     DyDxi,    DzDxi,      &
            &             DxDet,     DyDet,    DzDet,      &
            &             vnx,       vny,      vnz,        &
            &             hs)
            cf = hs*wq(i)
            arel(k) = arel(k) + cf
            xmom(k) = xmom(k) + cf*x
            ymom(k) = ymom(k) + cf*y
            zmom(k) = zmom(k) + cf*z
            vlm = vlm + (x*vnx+y*vny+z*vnz)*cf
        END DO


        arel(k) = 0.5D0*arel(k)
        xmom(k) = 0.5D0*xmom(k)
        ymom(k) = 0.5D0*ymom(k)
        zmom(k) = 0.5D0*zmom(k)
        area = area +arel(k)
        cx   = cx   + xmom(k)
        cy   = cy   + ymom(k)
        cz   = cz   + zmom(k)
!-----------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the nftty
!formula (4.2.10) of Pozrikidis (1997)
!-----------------------------------------------------------------------------------------------------
        xxi(1) = 0.0D0
        eet(1) = 0.0D0
        xxi(2) = 1.0D0
        eet(2) = 0.0D0
        xxi(3) = 0.0D0
        eet(3) = 1.0D0
        xxi(4) = al
        eet(4) = 0.0D0
        xxi(5) = ga
        eet(5) = gac
        xxi(6) = 0.0D0
        eet(6) = be
        DO i = 1, 6
            xi  = xxi(i)
            eta = eet(i)
            CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
            &             p(i2,1),   p(i2,2),  p(i2,3),    &
            &             p(i3,1),   p(i3,2),  p(i3,3),    &
            &             p(i4,1),   p(i4,2),  p(i4,3),    &
            &             p(i5,1),   p(i5,2),  p(i5,3),    &
```



```
            &             p(i6,1),   p(i6,2),  p(i6,3),    &
            &             al,        be,       ga,         &
            &             xi,        eta,                  &
            &             x,         y,        z,          &
            &             DxDx(i),   DyDx(i),  DzDx(i),    &
            &             DxDe(i),   DyDe(i),  DzDe(i),    &
            &             vx(i),     vy(i),    vz(i),      &
            &             hs)
        END DO
!-----------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        crvx = 0.0D0
        crvy = 0.0D0
        crvz = 0.0D0
!-----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-4-2
!-----------------------------------------------------------------------------------------------------
        bvx1 = vy(1)*DzDx(1)-vz(1)*DyDx(1)
        bvy1 = vz(1)*DxDx(1)-vx(1)*DzDx(1)
        bvz1 = vx(1)*DyDx(1)-vy(1)*DxDx(1)
!-----------------------------------------------------------------------------------------------------
        bvx2 = vy(4)*DzDx(4)-vz(4)*DyDx(4)
        bvy2 = vz(4)*DxDx(4)-vx(4)*DzDx(4)
        bvz2 = vx(4)*DyDx(4)-vy(4)*DxDx(4)
!-----------------------------------------------------------------------------------------------------
        bvx3 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
        bvy3 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
        bvz3 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
!-----------------------------------------------------------------------------------------------------
        crvx = al*bvx1 + bvx2 + alc*bvx3
        crvy = al*bvy1 + bvy2 + alc*bvy3
        crvz = al*bvz1 + bvz2 + alc*bvz3
!-----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-5-3
!-----------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
!-----------------------------------------------------------------------------------------------------
        bvx1 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
        bvy1 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
        bvz1 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
!-----------------------------------------------------------------------------------------------------
        bvx2 = vy(5)*DzDx(5)-vz(5)*DyDx(5)
        bvy2 = vz(5)*DxDx(5)-vx(5)*DzDx(5)
        bvz2 = vx(5)*DyDx(5)-vy(5)*DxDx(5)
!-----------------------------------------------------------------------------------------------------
        bvx3 = vy(3)*DzDx(3)-vz(3)*DyDx(3)
        bvy3 = vz(3)*DxDx(3)-vx(3)*DzDx(3)
        bvz3 = vx(3)*DyDx(3)-vy(3)*DxDx(3)
!-----------------------------------------------------------------------------------------------------
        crvx = crvx - gac*bvx1 - bvx2 - ga*bvx3
        crvy = crvy - gac*bvy1 - bvy2 - ga*bvy3
        crvz = crvz - gac*bvz1 - bvz2 - ga*bvz3
!-----------------------------------------------------------------------------------------------------
        bvx1 = vy(2)*DzDe(2)-vz(2)*DyDe(2)
        bvy1 = vz(2)*DxDe(2)-vx(2)*DzDe(2)
        bvz1 = vx(2)*DyDe(2)-vy(2)*DxDe(2)
```

```
!-----------------------------------------------------------------------------------------------------------
        bvx2 = vy(5)*DzDe(5)-vz(5)*DyDe(5)
        bvy2 = vz(5)*DxDe(5)-vx(5)*DzDe(5)
        bvz2 = vx(5)*DyDe(5)-vy(5)*DxDe(5)
!-----------------------------------------------------------------------------------------------------------
        bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
        bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
        bvz3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
!-----------------------------------------------------------------------------------------------------------
        crvx = crvx + gac*bvx1 + bvx2 + ga*bvx3
        crvy = crvy + gac*bvy1 + bvy2 + ga*bvy3
        crvz = crvz + gac*bvz1 + bvz2 + ga*bvz3
!-----------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3-6-1
!-----------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
!-----------------------------------------------------------------------------------------------------------
        bvx1 = vy(1)*DzDe(1)-vz(1)*DyDe(1)
        bvy1 = vz(1)*DxDe(1)-vx(1)*DzDe(1)
        bvz1 = vx(1)*DyDe(1)-vy(1)*DxDe(1)
!-----------------------------------------------------------------------------------------------------------
        bvx2 = vy(6)*DzDe(6)-vz(6)*DyDe(6)
        bvy2 = vz(6)*DxDe(6)-vx(6)*DzDe(6)
        bvz2 = vx(6)*DyDe(6)-vy(6)*DxDe(6)
!-----------------------------------------------------------------------------------------------------------
        bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
        bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
        bvz3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
!-----------------------------------------------------------------------------------------------------------
        crvx = crvx - be*bvx1 - bvx2 - bec*bvx3
        crvy = crvy - be*bvy1 - bvy2 - bec*bvy3
        crvz = crvz - be*bvz1 - bvz2 - bec*bvz3
!-----------------------------------------------------------------------------------------------------------
        fil = DSQRT(crvx**2+crvy**2+crvz**2)
!-----------------------------------------------------------------------------------------------------------
        crvx = crvx/fil
        crvy = crvy/fil
        crvz = crvz/fil
!-----------------------------------------------------------------------------------------------------------
! will project curvature vector onto the normal vector at the centroid only needs the colocation verctors
!In this value of curvature I made a division per 8 (1/8.0) when i expected do for 4 (1/4.0)
!-----------------------------------------------------------------------------------------------------------
        crvmel(k) = (0.125D0*(crvx*vnx0(k)+crvy*vny0(k)+crvz*vnz0(k))*fil)/arel(k)
!        WRITE (9,*)   crvmel(k)

    END DO Outer

!-----------------------------------------------------------------------------------------------------------
! final computation of the surface-centroid and volume

!-----------------------------------------------------------------------------------------------------------
        cx = cx/area
        cy = cy/area
        cz = cz/area
        vlm = vlm/6.0D0
!    CLOSE (9)
103 FORMAT(10(1x,ES24.16))
!-----------------------------------------------------------------------------------------------------------
    END SUBROUTINE elm_geom4
!===========================================================================================================
!===========================================================================================================
    SUBROUTINE elm_geom2(nelm, npts, mint,   &
    &                   xmom, ymom, zmom,   &
    &                   area, vlm,          &
```



```
    &                   cx,   cy,   cz)
!===========================================================================================================
!   This subroutine is a new version of Elm_Geo Subroutine.
!Compute:
!   *The surface area of the individual elements x, y, and z moments over each element
!   *the total particle surface area and volume
!Now, (25 / Augost / 2012) this subroutine was cut.
!===========================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog, vna1, &
    &                         alphaQ,  betaQ,  gammaQ, &
    &                         arel,          crvmel,    &
    &                         vnx0,     vny0,    vnz0, &
    &                         ZZ,            WW,        &
    &                         xiq, etq, wq
!===========================================================================================================
    IMPLICIT NONE
!===========================================================================================================
!   Variables
!-----------------------------------------------------------------------------------------------------------
    INTEGER,           INTENT(IN)  :: nelm                               !number of elements
    INTEGER,           INTENT(IN)  :: npts                               !number of poins on the surface
    INTEGER,           INTENT(IN)  :: mint                               !order of triangle quadrature
    REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom    !coordinates of the moments of the drop
    REAL (KIND = DBL), INTENT(OUT) :: area, vlm                          !area and volume of each element
    REAL (KIND = DBL), INTENT(OUT) :: cx,   cy,   cz                     !drop's centroid coordinates
!===========================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------------
    INTEGER  :: i, k                                     !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                   !indices to obtain node numbers from each element
!-----------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: xi, eta                          !variables of weigth to integrate over a triangle
    REAL (KIND = DBL) :: x, y, z                          !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi              !coordinates of the tangential vector over the xi axis
    REAL (KIND = DBL) :: DxDet, DyDet, DzDet              !coordinates of the tangential vector over the eta axis
    REAL (KIND = DBL) :: vnx, vny, vnz                    !normal vector coordinates of the element
    REAL (KIND = DBL) :: hs                               !surface metric on a triangle
    REAL (KIND = DBL) :: al, be, ga, alc, bec, gac        !integration weigth coefficients
    REAL (KIND = DBL) :: cf                               !integration weigth coefficients
    REAL (KIND = DBL), DIMENSION(6) ::xxi, eet            !variables of weigth over in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDx, DyDx, DzDx    !tangential vector over xi axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDe, DyDe, DzDe    !tangential vector over eta axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz         !normal vector in triangle (xi,eta)
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &             !binormal vectors around in triangle (xi,eta)
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  crvx, crvy, crvz                !curvature
!===========================================================================================================
!   initialize
!-----------------------------------------------------------------------------------------------------------
    area = 0.0D0
    vlm  = 0.0D0
    cx = 0.0D0
    cy = 0.0D0
    cz = 0.0D0
    arel = 0.0D0
    xmom = 0.0D0
    ymom = 0.0D0
    zmom = 0.0D0
    crvmel = 0.0D0
!-----------------------------------------------------------------------------------------------------------
    OPEN (9,file="curvmel.out")

    Outer: DO k = 1, nelm

        i1 = n(k,1)
        i2 = n(k,2)
        i3 = n(k,3)
        i4 = n(k,4)
        i5 = n(k,5)
        i6 = n(k,6)
```

```
        al  = alphaQ(k)
        be  = betaQ(k)
        ga  = gammaQ(k)
        alc = 1.0D0-al
        bec = 1.0D0-be
        gac = 1.0D0-ga
!-----------------------------------------------------------------------------------------------------------------
!   Compute surface area and volume of the individual elements x, y, and z moments over each element total
!particle surface area and volume
!-----------------------------------------------------------------------------------------------------------------


        DO i = 1, mint
            xi  = xiq(i)
            eta = etq(i)
            CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
            &             p(i2,1),   p(i2,2),  p(i2,3),    &
            &             p(i3,1),   p(i3,2),  p(i3,3),    &
            &             p(i4,1),   p(i4,2),  p(i4,3),    &
            &             p(i5,1),   p(i5,2),  p(i5,3),    &
            &             p(i6,1),   p(i6,2),  p(i6,3),    &
            &             al,        be,       ga,         &
            &             xi,        eta,                  &
            &             x,         y,         z,         &
            &             DxDxi,     DyDxi,    DzDxi,      &
            &             DxDet,     DyDet,    DzDet,      &
            &             vnx,       vny,      vnz,        &
            &             hs)
            cf = hs*wq(i)
            arel(k) = arel(k) + cf
            xmom(k) = xmom(k) + cf*x
            ymom(k) = ymom(k) + cf*y
            zmom(k) = zmom(k) + cf*z
            vlm = vlm + (x*vnx+y*vny+z*vnz)*cf
        END DO


        arel(k) = 0.5D0*arel(k)
        xmom(k) = 0.5D0*xmom(k)
        ymom(k) = 0.5D0*ymom(k)
        zmom(k) = 0.5D0*zmom(k)
        area = area +arel(k)
        cx   = cx   + xmom(k)
        cy   = cy   + ymom(k)
        cz   = cz   + zmom(k)
!-----------------------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the nftty
!formula (4.2.10) of Pozrikidis (1997)
!-----------------------------------------------------------------------------------------------------------------
        xxi(1) = 0.0D0
        eet(1) = 0.0D0
        xxi(2) = 1.0D0
        eet(2) = 0.0D0
        xxi(3) = 0.0D0
        eet(3) = 1.0D0
        xxi(4) = al
        eet(4) = 0.0D0
        xxi(5) = ga
        eet(5) = gac
        xxi(6) = 0.0D0
        eet(6) = be
        DO i = 1, 6
            xi  = xxi(i)
            eta = eet(i)
            CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
            &             p(i2,1),   p(i2,2),  p(i2,3),    &
            &             p(i3,1),   p(i3,2),  p(i3,3),    &
            &             p(i4,1),   p(i4,2),  p(i4,3),    &
            &             p(i5,1),   p(i5,2),  p(i5,3),    &
            &             p(i6,1),   p(i6,2),  p(i6,3),    &
            &             al,        be,       ga,         &
            &             xi,        eta,                  &
```



```
            &             x,         y,        z,          &
            &             DxDx(i),   DyDx(i),  DzDx(i),    &
            &             DxDe(i),   DyDe(i),  DzDe(i),    &
            &             vx(i),     vy(i),    vz(i),      &
            &             hs)
        END DO
!-----------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        crvx = 0.0D0
        crvy = 0.0D0
        crvz = 0.0D0
!-----------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-4-2
!-----------------------------------------------------------------------------------------------------------------
        bvx1 = vna1(1+(6*(i-1)),2)*DzDx(1)-vna1(1+(6*(i-1)),3)*DyDx(1)
        bvy1 = vna1(1+(6*(i-1)),3)*DxDx(1)-vna1(1+(6*(i-1)),1)*DzDx(1)
        bvz1 = vna1(1+(6*(i-1)),1)*DyDx(1)-vna1(1+(6*(i-1)),2)*DxDx(1)
!-----------------------------------------------------------------------------------------------------------------
            !vna1(1+(6*(i-1)),1), vna1(1+(6*(i-1)),2), vna1(1+(6*(i-1)),3), &
            !vna1(2+(6*(i-1)),1), vna1(2+(6*(i-1)),2), vna1(2+(6*(i-1)),3), &
            !vna1(3+(6*(i-1)),1), vna1(3+(6*(i-1)),2), vna1(3+(6*(i-1)),3), &
            !vna1(4+(6*(i-1)),1), vna1(4+(6*(i-1)),2), vna1(4+(6*(i-1)),3), &
            !vna1(5+(6*(i-1)),1), vna1(5+(6*(i-1)),2), vna1(5+(6*(i-1)),3), &
            !vna1(6+(6*(i-1)),1), vna1(6+(6*(i-1)),2), vna1(6+(6*(i-1)),3), &
!-----------------------------------------------------------------------------------------------------------------
        bvx2 = vna1(4+(6*(i-1)),2)*DzDx(4)-vna1(4+(6*(i-1)),3)*DyDx(4)
        bvy2 = vna1(4+(6*(i-1)),3)*DxDx(4)-vna1(4+(6*(i-1)),1)*DzDx(4)
        bvz2 = vna1(4+(6*(i-1)),1)*DyDx(4)-vna1(4+(6*(i-1)),2)*DxDx(4)
!-----------------------------------------------------------------------------------------------------------------
        bvx3 = vna1(2+(6*(i-1)),2)*DzDx(2)-vna1(2+(6*(i-1)),3)*DyDx(2)
        bvy3 = vna1(2+(6*(i-1)),3)*DxDx(2)-vna1(2+(6*(i-1)),1)*DzDx(2)
        bvz3 = vna1(2+(6*(i-1)),1)*DyDx(2)-vna1(2+(6*(i-1)),2)*DxDx(2)
!-----------------------------------------------------------------------------------------------------------------
        crvx = al*bvx1 + bvx2 + alc*bvx3
        crvy = al*bvy1 + bvy2 + alc*bvy3
        crvz = al*bvz1 + bvz2 + alc*bvz3
!-----------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-5-3
!-----------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
!-----------------------------------------------------------------------------------------------------------------
        bvx1 = vna1(2+(6*(i-1)),2)*DzDx(2)-vna1(2+(6*(i-1)),3)*DyDx(2)
        bvy1 = vna1(2+(6*(i-1)),3)*DxDx(2)-vna1(2+(6*(i-1)),1)*DzDx(2)
        bvz1 = vna1(2+(6*(i-1)),1)*DyDx(2)-vna1(2+(6*(i-1)),2)*DxDx(2)
!-----------------------------------------------------------------------------------------------------------------
        bvx2 = vna1(5+(6*(i-1)),2)*DzDx(5)-vna1(5+(6*(i-1)),3)*DyDx(5)
        bvy2 = vna1(5+(6*(i-1)),3)*DxDx(5)-vna1(5+(6*(i-1)),1)*DzDx(5)
        bvz2 = vna1(5+(6*(i-1)),1)*DyDx(5)-vna1(5+(6*(i-1)),2)*DxDx(5)
!-----------------------------------------------------------------------------------------------------------------
        bvx3 = vna1(3+(6*(i-1)),2)*DzDx(3)-vna1(3+(6*(i-1)),3)*DyDx(3)
        bvy3 = vna1(3+(6*(i-1)),3)*DxDx(3)-vna1(3+(6*(i-1)),1)*DzDx(3)
        bvz3 = vna1(3+(6*(i-1)),1)*DyDx(3)-vna1(3+(6*(i-1)),2)*DxDx(3)
!-----------------------------------------------------------------------------------------------------------------
        crvx = crvx - gac*bvx1 - bvx2 - ga*bvx3
        crvy = crvy - gac*bvy1 - bvy2 - ga*bvy3
        crvz = crvz - gac*bvz1 - bvz2 - ga*bvz3
```

```fortran
!----------------------------------------------------------------------------------------------------
        bvx1 = vna1(2+(6*(i-1)),2)*DzDe(2)-vna1(2+(6*(i-1)),3)*DyDe(2)
        bvy1 = vna1(2+(6*(i-1)),3)*DxDe(2)-vna1(2+(6*(i-1)),1)*DzDe(2)
        bvz1 = vna1(2+(6*(i-1)),1)*DyDe(2)-vna1(2+(6*(i-1)),2)*DxDe(2)
!----------------------------------------------------------------------------------------------------
        bvx2 = vna1(5+(6*(i-1)),2)*DzDe(5)-vna1(5+(6*(i-1)),3)*DyDe(5)
        bvy2 = vna1(5+(6*(i-1)),3)*DxDe(5)-vna1(5+(6*(i-1)),1)*DzDe(5)
        bvz2 = vna1(5+(6*(i-1)),1)*DyDe(5)-vna1(5+(6*(i-1)),2)*DxDe(5)
!----------------------------------------------------------------------------------------------------
        bvx3 = vna1(3+(6*(i-1)),2)*DzDe(3)-vna1(3+(6*(i-1)),3)*DyDe(3)
        bvy3 = vna1(3+(6*(i-1)),3)*DxDe(3)-vna1(3+(6*(i-1)),1)*DzDe(3)
        bvz3 = vna1(3+(6*(i-1)),1)*DyDe(3)-vna1(3+(6*(i-1)),2)*DxDe(3)
!----------------------------------------------------------------------------------------------------
        crvx = crvx + gac*bvx1 + bvx2 + ga*bvx3
        crvy = crvy + gac*bvy1 + bvy2 + ga*bvy3
        crvz = crvz + gac*bvz1 + bvz2 + ga*bvz3
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3-6-1
!----------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
!----------------------------------------------------------------------------------------------------
        bvx1 = vna1(1+(6*(i-1)),2)*DzDe(1)-vna1(1+(6*(i-1)),3)*DyDe(1)
        bvy1 = vna1(1+(6*(i-1)),3)*DxDe(1)-vna1(1+(6*(i-1)),1)*DzDe(1)
        bvz1 = vna1(1+(6*(i-1)),1)*DyDe(1)-vna1(1+(6*(i-1)),2)*DxDe(1)
!----------------------------------------------------------------------------------------------------
        bvx2 = vna1(6+(6*(i-1)),2)*DzDe(6)-vna1(6+(6*(i-1)),3)*DyDe(6)
        bvy2 = vna1(6+(6*(i-1)),3)*DxDe(6)-vna1(6+(6*(i-1)),1)*DzDe(6)
        bvz2 = vna1(6+(6*(i-1)),1)*DyDe(6)-vna1(6+(6*(i-1)),2)*DxDe(6)
!----------------------------------------------------------------------------------------------------
        bvx3 = vna1(3+(6*(i-1)),2)*DzDe(3)-vna1(3+(6*(i-1)),3)*DyDe(3)
        bvy3 = vna1(3+(6*(i-1)),3)*DxDe(3)-vna1(3+(6*(i-1)),1)*DzDe(3)
        bvz3 = vna1(3+(6*(i-1)),1)*DyDe(3)-vna1(3+(6*(i-1)),2)*DxDe(3)
!----------------------------------------------------------------------------------------------------
        crvx = crvx - be*bvx1 - bvx2 - bec*bvx3
        crvy = crvy - be*bvy1 - bvy2 - bec*bvy3
        crvz = crvz - be*bvz1 - bvz2 - bec*bvz3
!----------------------------------------------------------------------------------------------------
! will project curvature vector onto the normal vector at the centroid only needs the colocation verctors
!In this value of curvature I made a division per 8 (1/8.0) when i expected do for 4 (1/4.0)
!----------------------------------------------------------------------------------------------------
        crvmel(k) = 0.250D0*(crvx*vnx0(k)+crvy*vny0(k)+crvz*vnz0(k)) !/arel(k)
        WRITE (9,*)   crvmel(k)

    END DO Outer

!----------------------------------------------------------------------------------------------------
! final computation of the surface-centroid and volume

!----------------------------------------------------------------------------------------------------
        cx = cx/area
        cy = cy/area
        cz = cz/area
        vlm = vlm/6.0D0
    CLOSE (9)
103 FORMAT(10(1x,ES24.16))
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE elm_geom2
!====================================================================================================
 SUBROUTINE elm_geom3(nelm, npts, mint,   &
    &                   xmom, ymom, zmom,   &
    &                   area, vlm,          &
    &                   cx,   cy,   cz)
!====================================================================================================
!   This subroutine is a new version of Elm_Geo Subroutine.
```



```fortran
!Compute:
!   *The surface area of the individual elements x, y, and z moments over each element
!   *the total particle surface area and volume
!Now, (25 / Augost / 2012) this subroutine was cut.
!====================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog,&
    &                         alphaQ,  betaQ,  gammaQ, &
    &                         arel,        crvmel,     &
    &                         vnx0,     vny0,    vnz0, &
    &                         ZZ,          WW,         &
    &                         xiq, etq, wq
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER,           INTENT(IN)  :: nelm                              !number of elements
    INTEGER,           INTENT(IN)  :: npts                              !number of poins on the surface
    INTEGER,           INTENT(IN)  :: mint                              !order of triangle quadrature
    REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom    !coordinates of the moments of the drop
    REAL (KIND = DBL), INTENT(OUT) :: area, vlm                         !area and volume of each element
    REAL (KIND = DBL), INTENT(OUT) :: cx,   cy,   cz                    !drop's centroid coordinates
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, k                                       !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                     !indices to obtain node numbers from each element
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: xi, eta                           !variables of weigth to integrate over a triangle
    REAL (KIND = DBL) :: x, y, z                           !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi               !coordinates of the tangential vector over the xi axis
    REAL (KIND = DBL) :: DxDet, DyDet, DzDet               !coordinates of the tangential vector over the eta axis
    REAL (KIND = DBL) :: vnx, vny, vnz                     !normal vector coordinates of the element
    REAL (KIND = DBL) :: hs, xs, es                        !surface metric on a triangle
    REAL (KIND = DBL) :: al, be, ga, alc, bec, gac         !integration weigth coefficients
    REAL (KIND = DBL) :: cf , fil                           !integration weigth coefficients
    REAL (KIND = DBL), DIMENSION(6) ::xxi, eet             !variables of weigth over in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDx, DyDx, DzDx     !tangential vector over xi axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDe, DyDe, DzDe     !tangential vector over eta axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz          !normal vector in triangle (xi,eta)
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &              !binormal vectors around in triangle (xi,eta)
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  crvx, crvy, crvz                 !curvature
!====================================================================================================
!   initialize
!----------------------------------------------------------------------------------------------------
    area = 0.0D0
    vlm  = 0.0D0
    cx = 0.0D0
    cy = 0.0D0
    cz = 0.0D0
    arel = 0.0D0
    xmom = 0.0D0
    ymom = 0.0D0
    zmom = 0.0D0
    crvmel = 0.0D0
    fil = 0.0D0
!----------------------------------------------------------------------------------------------------
    OPEN (9,file="curvmel.out")

    Outer: DO k = 1, nelm

        i1 = n(k,1)
        i2 = n(k,2)
        i3 = n(k,3)
        i4 = n(k,4)
        i5 = n(k,5)
        i6 = n(k,6)
        al  = alphaQ(k)
        be  = betaQ(k)
```

```
        ga  = gammaQ(k)
        alc = 1.0D0-al
        bec = 1.0D0-be
        gac = 1.0D0-ga
!---------------------------------------------------------------------------------------------------------------
!   Compute surface area and volume of the individual elements x, y, and z moments over each element total
!particle surface area and volume
!---------------------------------------------------------------------------------------------------------------


        DO i = 1, mint
            xi  = xiq(i)
            eta = etq(i)
            CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
            &             p(i2,1),   p(i2,2),  p(i2,3),    &
            &             p(i3,1),   p(i3,2),  p(i3,3),    &
            &             p(i4,1),   p(i4,2),  p(i4,3),    &
            &             p(i5,1),   p(i5,2),  p(i5,3),    &
            &             p(i6,1),   p(i6,2),  p(i6,3),    &
            &             al,        be,       ga,         &
            &             xi,        eta,                  &
            &             x,         y,         z,         &
            &             DxDxi,     DyDxi,    DzDxi,      &
            &             DxDet,     DyDet,    DzDet,      &
            &             vnx,       vny,      vnz,        &
            &             hs)
            cf = hs*wq(i)
            arel(k) = arel(k) + cf
            xmom(k) = xmom(k) + cf*x
            ymom(k) = ymom(k) + cf*y
            zmom(k) = zmom(k) + cf*z
            vlm = vlm + (x*vnx+y*vny+z*vnz)*cf
            cf=0.0D0
        END DO


        arel(k) = 0.5D0*arel(k)
        xmom(k) = 0.5D0*xmom(k)
        ymom(k) = 0.5D0*ymom(k)
        zmom(k) = 0.5D0*zmom(k)
        area = area +arel(k)
        cx   = cx   + xmom(k)
        cy   = cy   + ymom(k)
        cz   = cz   + zmom(k)
!---------------------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the nftty
!formula (4.2.10) of Pozrikidis (1997)
!---------------------------------------------------------------------------------------------------------------
        xxi(1) = 0.0D0
        eet(1) = 0.0D0
        xxi(2) = 1.0D0
        eet(2) = 0.0D0
        xxi(3) = 0.0D0
        eet(3) = 1.0D0
        xxi(4) = al
        eet(4) = 0.0D0
        xxi(5) = ga
        eet(5) = gac
        xxi(6) = 0.0D0
        eet(6) = be
        DO i = 1, 6
            xi  = xxi(i)
            eta = eet(i)
            CALL interp_p3(p(i1,1),   p(i1,2),  p(i1,3),    &
            &             p(i2,1),   p(i2,2),  p(i2,3),    &
            &             p(i3,1),   p(i3,2),  p(i3,3),    &
            &             p(i4,1),   p(i4,2),  p(i4,3),    &
            &             p(i5,1),   p(i5,2),  p(i5,3),    &
            &             p(i6,1),   p(i6,2),  p(i6,3),    &
            &             al,        be,       ga,         &
            &             xi,        eta,                  &
            &             x,         y,         z,         &
```



```
            &             DxDx(i),   DyDx(i),  DzDx(i),    &
            &             DxDe(i),   DyDe(i),  DzDe(i),    &
            &             vx(i),     vy(i),    vz(i),      &
            &             hs)
        END DO
!---------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        crvx = 0.0D0
        crvy = 0.0D0
        crvz = 0.0D0
!---------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-4-2
!---------------------------------------------------------------------------------------------------------------
        bvx1 = vy(1)*DzDx(1)-vz(1)*DyDx(1)
        bvy1 = vz(1)*DxDx(1)-vx(1)*DzDx(1)
        bvz1 = vx(1)*DyDx(1)-vy(1)*DxDx(1)
!---------------------------------------------------------------------------------------------------------------
        bvx2 = vy(4)*DzDx(4)-vz(4)*DyDx(4)
        bvy2 = vz(4)*DxDx(4)-vx(4)*DzDx(4)
        bvz2 = vx(4)*DyDx(4)-vy(4)*DxDx(4)
!---------------------------------------------------------------------------------------------------------------
        bvx3 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
        bvy3 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
        bvz3 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
!---------------------------------------------------------------------------------------------------------------
        crvx = al*bvx1 + bvx2 + alc*bvx3
        crvy = al*bvy1 + bvy2 + alc*bvy3
        crvz = al*bvz1 + bvz2 + alc*bvz3
!---------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-5-3
!---------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
!---------------------------------------------------------------------------------------------------------------
        bvx1 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
        bvy1 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
        bvz1 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
!---------------------------------------------------------------------------------------------------------------
        bvx2 = vy(5)*DzDx(5)-vz(5)*DyDx(5)
        bvy2 = vz(5)*DxDx(5)-vx(5)*DzDx(5)
        bvz2 = vx(5)*DyDx(5)-vy(5)*DxDx(5)
!---------------------------------------------------------------------------------------------------------------
        bvx3 = vy(3)*DzDx(3)-vz(3)*DyDx(3)
        bvy3 = vz(3)*DxDx(3)-vx(3)*DzDx(3)
        bvz3 = vx(3)*DyDx(3)-vy(3)*DxDx(3)
!---------------------------------------------------------------------------------------------------------------
        crvx = crvx - gac*bvx1 - bvx2 - ga*bvx3
        crvy = crvy - gac*bvy1 - bvy2 - ga*bvy3
        crvz = crvz - gac*bvz1 - bvz2 - ga*bvz3
!---------------------------------------------------------------------------------------------------------------
        bvx1 = vy(2)*DzDe(2)-vz(2)*DyDe(2)
        bvy1 = vz(2)*DxDe(2)-vx(2)*DzDe(2)
        bvz1 = vx(2)*DyDe(2)-vy(2)*DxDe(2)
!---------------------------------------------------------------------------------------------------------------
        bvx2 = vy(5)*DzDe(5)-vz(5)*DyDe(5)
        bvy2 = vz(5)*DxDe(5)-vx(5)*DzDe(5)
        bvz2 = vx(5)*DyDe(5)-vy(5)*DxDe(5)
```

```
!-----------------------------------------------------------------------------------------------------------
        bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
        bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
        bvz3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
!-----------------------------------------------------------------------------------------------------------
        crvx = crvx + gac*bvx1 + bvx2 + ga*bvx3
        crvy = crvy + gac*bvy1 + bvy2 + ga*bvy3
        crvz = crvz + gac*bvz1 + bvz2 + ga*bvz3
!-----------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3-6-1
!-----------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
!-----------------------------------------------------------------------------------------------------------
        bvx1 = vy(1)*DzDe(1)-vz(1)*DyDe(1)
        bvy1 = vz(1)*DxDe(1)-vx(1)*DzDe(1)
        bvz1 = vx(1)*DyDe(1)-vy(1)*DxDe(1)
!-----------------------------------------------------------------------------------------------------------
        bvx2 = vy(6)*DzDe(6)-vz(6)*DyDe(6)
        bvy2 = vz(6)*DxDe(6)-vx(6)*DzDe(6)
        bvz2 = vx(6)*DyDe(6)-vy(6)*DxDe(6)
!-----------------------------------------------------------------------------------------------------------
        bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
        bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
        bvz3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
!-----------------------------------------------------------------------------------------------------------
        crvx = crvx - be*bvx1 - bvx2 - bec*bvx3
        crvy = crvy - be*bvy1 - bvy2 - bec*bvy3
        crvz = crvz - be*bvz1 - bvz2 - bec*bvz3
!-----------------------------------------------------------------------------------------------------------
!        fil = DSQRT(crvx**2+crvy**2+crvz**2)
!!!-----------------------------------------------------------------------------------------------------------
    -
!        crvx = crvx/fil
!        crvy = crvy/fil
!        crvz = crvz/fil
!-----------------------------------------------------------------------------------------------------------
! will project curvature vector onto the normal vector at the centroid only needs the colocation verctors
!In this value of curvature I made a division per 8 (1/8.0) when i expected do for 4 (1/4.0)
!-----------------------------------------------------------------------------------------------------------
        crvmel(k) = 0.250D0*(crvx*vnx0(k)+crvy*vny0(k)+crvz*vnz0(k))/arel(k)
        WRITE (9,*)   crvmel(k)

    END DO Outer

!-----------------------------------------------------------------------------------------------------------
! final computation of the surface-centroid and volume

!-----------------------------------------------------------------------------------------------------------
        cx = cx/area
        cy = cy/area
        cz = cz/area
        vlm = vlm/6.0D0
    CLOSE (9)
103 FORMAT(10(1x,ES24.16))
!-----------------------------------------------------------------------------------------------------------
    END SUBROUTINE elm_geom3
!===========================================================================================================
    END MODULE Mod_Prtcl_3D_Geo
```

```fortran
MODULE Mod_Nodal_Interp
!================================================================================================================
!Version 0.9            10 / Augost / 2012
!Version 1.0            14 / 11 / 2012
!                                                                   Alfredo Sanjuan Sanjuan, in others words, me :)
!================================================================================================================
CONTAINS
!================================================================================================================
    SUBROUTINE interp_p(   x1,    y1,    z1,  &
    &                      x2,    y2,    z2,  &
    &                      x3,    y3,    z3,  &
    &                      x4,    y4,    z4,  &
    &                      x5,    y5,    z5,  &
    &                      x6,    y6,    z6,  &
    &                      al,    be,    ga,  &
    &                      xi,   eta,         &
    &                       x,     y,     z,  &
    &                   DxDxi, DyDxi, DzDxi,  &
    &                   DxDet, DyDet, DzDet,  &
    &                     vnx,   vny,   vnz,  &
    &                      hs                 )
!================================================================================================================
!   This subroutine interpolate over an element to compute geometrical variables including the following:

!   1) Position vector
!   2) Tangential vectors in the xi and eta directions
!   3) Unit normal vector
!   4) Area and volume of each element
!================================================================================================================
    USE Mod_SharedVars, ONLY: DBL
!================================================================================================================
    IMPLICIT NONE
!================================================================================================================
!   Variables
    REAL (KIND = DBL), INTENT(IN)  :: x1, y1, z1, &             !coordinates of first node of the element
    &                                 x2, y2, z2, &             !coordinates of second node of the element
    &                                 x3, y3, z3, &             !coordinates of third node of the element
    &                                 x4, y4, z4, &             !coordinates of fourth node of the element
    &                                 x5, y5, z5, &             !coordinates of fifth node of the element
    &                                 x6, y6, z6                !coordinates of sixth node of the element
    REAL (KIND = DBL), INTENT(IN)  :: al, be, ga                !constants alpha, beta and gamma
    REAL (KIND = DBL), INTENT(IN)  :: xi, eta                   !variables to integrate over a triangle
    REAL (KIND = DBL), INTENT(OUT) :: x, y, z                   !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL), INTENT(OUT) :: DxDxi, DyDxi, DzDxi, &    !Derivates of the tangential vectors over the
    &                                 DxDet, DyDet, DzDet       !element
    REAL (KIND = DBL), INTENT(OUT) :: vnx, vny, vnz             !normal vector of the element
    REAL (KIND = DBL), INTENT(OUT) :: hs                        !surface metric
!================================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: alc,   bec,   gac, xs, es, hhs         !complements of alpha, beta and gamma
    REAL (KIND = DBL) :: alalc, bebec, gagac                    !other constants of alpha, beta and gamma
    REAL (KIND = DBL) :: ph1,   ph2,   ph3,  ph4,  ph5,  ph6    !components phi(xi,eta)
    REAL (KIND = DBL) :: dph1,  dph2,  dph3, dph4, dph5, dph6   !derivate of phi(xi,eta) wrt xi
    REAL (KIND = DBL) :: pph1,  pph2,  pph3, pph4, pph5, pph6   !derivate of phi(xi,eta) wrt eta
!----------------------------------------------------------------------------------------------------------------
!   Prepare the constants of this subroutine.
!----------------------------------------------------------------------------------------------------------------
    alc = 1.0D0-al
    bec = 1.0D0-be
    gac = 1.0D0-ga
    alalc = al*alc
    bebec = be*bec
    gagac = ga*gac
!----------------------------------------------------------------------------------------------------------------
!   In this part, the subroutine evaluates basis functions, it obtains the tangential and normal vectors.
!This is based on Numerical Computation in Science and Engineering, C.Pozrikidis, 1998, pp.305-312
!----------------------------------------------------------------------------------------------------------------
    ph2 = xi *(xi - al + eta*(al - ga)/gac)/alc
    ph3 = eta*(eta - be + xi*(be + ga - 1.0D0)/ga)/bec
    ph4 = xi *(1.0D0 - xi - eta)/alalc
    ph5 = xi*eta/gagac
```



```fortran
    ph6 = eta*(1.0D0 - xi - eta)/bebec
    ph1 = 1.0D0-ph2-ph3-ph4-ph5-ph6
!----------------------------------------------------------------------------------------------------------------
!   Interpolate the position vector (x, y, z)
!----------------------------------------------------------------------------------------------------------------
    x = x1*ph1 + x2*ph2 + x3*ph3 + x4*ph4 + x5*ph5 + x6*ph6
    y = y1*ph1 + y2*ph2 + y3*ph3 + y4*ph4 + y5*ph5 + y6*ph6
    z = z1*ph1 + z2*ph2 + z3*ph3 + z4*ph4 + z5*ph5 + z6*ph6
!----------------------------------------------------------------------------------------------------------------
!   In this part suroutine evaluates xi derivatives of basis functions
!----------------------------------------------------------------------------------------------------------------
    dph2 =  (2.0D0*xi - al + eta*(al - ga)/gac)/alc
    dph3 =  eta*(be + ga - 1.0D0)/(ga*bec)
    dph4 =  (1.0D0 - 2.0D0*xi - eta)/alalc
    dph5 =  eta/gagac
    dph6 = -eta/bebec
    dph1 = -dph2 - dph3 - dph4 - dph5 - dph6
!----------------------------------------------------------------------------------------------------------------
!   Compute dx/dxi from xi derivatives of phi
!----------------------------------------------------------------------------------------------------------------
    DxDxi = x1*dph1 + x2*dph2 + x3*dph3 + x4*dph4 + x5*dph5 + x6*dph6
    DyDxi = y1*dph1 + y2*dph2 + y3*dph3 + y4*dph4 + y5*dph5 + y6*dph6
    DzDxi = z1*dph1 + z2*dph2 + z3*dph3 + z4*dph4 + z5*dph5 + z6*dph6
    xs = DSQRT(DxDxi*DxDxi + DyDxi*DyDxi + DzDxi*DzDxi)
!----------------------------------------------------------------------------------------------------------------
!   Normalization of normal vector
!----------------------------------------------------------------------------------------------------------------
    DxDxi = DxDxi/xs
    DyDxi = DyDxi/xs
    DzDxi = DzDxi/xs
!---------------------------------------------------------------------------------------------------------------!
    ----------------------------------------------------------------------------------------------------------------
    --
!   In this part suroutine evaluates eta derivatives of basis functions
!----------------------------------------------------------------------------------------------------------------
    pph2 =  xi*(al - ga)/(alc*gac)
    pph3 =  (2.0D0*eta - be + xi*(be + ga - 1.0D0)/ga)/bec
    pph4 =  -xi/alalc
    pph5 =   xi/gagac
    pph6 =  (1.0D0 - xi - 2.0D0*eta)/bebec
    pph1 = - pph2 - pph3 - pph4 - pph5 - pph6
!----------------------------------------------------------------------------------------------------------------
!   Compute dx/deta from eta derivatives of phi
!----------------------------------------------------------------------------------------------------------------
    DxDet = x1*pph1 + x2*pph2 + x3*pph3 + x4*pph4 + x5*pph5 + x6*pph6
    DyDet = y1*pph1 + y2*pph2 + y3*pph3 + y4*pph4 + y5*pph5 + y6*pph6
    DzDet = z1*pph1 + z2*pph2 + z3*pph3 + z4*pph4 + z5*pph5 + z6*pph6
    es = DSQRT(DxDet*DxDet + DyDet*DyDet + DzDet*DzDet)
!----------------------------------------------------------------------------------------------------------------
!   Normalization of normal vector
!----------------------------------------------------------------------------------------------------------------
    DxDet = DxDet/es
    DyDet = DyDet/es
    DzDet = DzDet/es
!----------------------------------------------------------------------------------------------------------------
!----------------------------------------------------------------------------------------------------------------
!   Normal vector vn = (DxDxi)^(DxDeta)
!   Surface metric   hs = norm(vn)
!----------------------------------------------------------------------------------------------------------------
    vnx = DyDxi * DzDet - DyDet * DzDxi
    vny = DzDxi * DxDet - DzDet * DxDxi
    vnz = DxDxi * DyDet - DxDet * DyDxi
    hhs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
!----------------------------------------------------------------------------------------------------------------
!   Normalization of normal vector
!----------------------------------------------------------------------------------------------------------------
    vnx = vnx/hhs
    vny = vny/hhs
    vnz = vnz/hhs
    hs= es*xs*hhs
!----------------------------------------------------------------------------------------------------------------
    END SUBROUTINE interp_p
```

```fortran
!====================================================================================================
!   Subroutine INTERP_E is a new method to obtain the normal vectors on every node on dorp's surface. The
!name has the end _E which is an abreviation of element. Firstly, this decision aids to differenciate this
!method of subroutine INTERP_P. Second idea was to continue with this notation or manner to name subroutines to
!have similar enviroment names. Finaly INTERP_E is very importan becouse this subroutine is fundamental to
!advanced the dinamic of dorp deformation in the time. Therefore, this subroutine is in this new MODULE
!====================================================================================================
    SUBROUTINE interp_e( npts,    ne,   nbe,&
                       & vnx0,  vny0,  vnz0,&
                       & vna)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
    USE Mod_SharedVars, ONLY: DBL !, p , x0, y0, z0
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: npts                                             !number of nodes
    INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: ne                  !table of conectivities per node
    INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: nbe                 !table of conectivities per element
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: vnx0, vny0, vnz0  !arrays of the components of
                                                                            !normal vectors over collocation points
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(OUT) :: vna  !array whit the values of unit normal
                                                                            !vector of every node over the surface
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                  !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: norma
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: dist1,dist2, absdist
!====================================================================================================
!   Initialize
    ALLOCATE(vna(npts,3), norma(npts), dist1(npts),dist2(npts),absdist(npts))
    vna = 0.0D0
    norma = 0.0D0
    dist1 = 0.0D0
    dist2 = 0.0D0
    absdist=0.0D0

!----------------------------------------------------------------------------------------------------
    WHERE(ne(:,1) == 4)
        vna(:,1)=(vnx0(ne(:,2)) + vnx0(ne(:,3)) + vnx0(ne(:,4)) + vnx0(ne(:,5)))/4.0D0
        vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(ne(:,4)) + vny0(ne(:,5)))/4.0D0
        vna(:,3)=(vnz0(ne(:,2)) + vnz0(ne(:,3)) + vnz0(ne(:,4)) + vnz0(ne(:,5)))/4.0D0
    END WHERE
!----------------------------------------------------------------------------------------------------
!----------------------------------------------------------------------------------------------------
    !WHERE(ne(:,1)== 4)
    !    vna(:,1)=4.0D0
    !    vna(:,2)=4.0D0
    !    vna(:,3)=4.0D0
    !END WHERE
!----------------------------------------------------------------------------------------------------
    WHERE(ne(:,1) == 6)
        vna(:,1)=(vnx0(ne(:,2)) + vnx0(ne(:,3)) + vnx0(ne(:,4)) + vnx0(ne(:,5)) &
              & + vnx0(ne(:,6)) + vnx0(ne(:,7)))/6.0D0
        vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(ne(:,4)) + vny0(ne(:,5)) &
              & + vny0(ne(:,6)) + vny0(ne(:,7)))/6.0D0
        vna(:,3)=(vnz0(ne(:,2)) + vnz0(ne(:,3)) + vnz0(ne(:,4)) + vnz0(ne(:,5)) &
              & + vnz0(ne(:,6)) + vnz0(ne(:,7)))/6.0D0
    END WHERE
!----------------------------------------------------------------------------------------------------
!----------------------------------------------------------------------------------------------------
    !WHERE(ne(:,1) == 6)
    !    vna(:,1)=6.0D0
    !    vna(:,2)=6.0D0
    !    vna(:,3)=6.0D0
    !END WHERE
!----------------------------------------------------------------------------------------------------
    WHERE (ne(:,1) == 2)
```



```fortran

        !vna(:,1)=(vnx0(ne(:,2)) + vnx0(ne(:,3))+ vnx0(nbe(ne(:,2),1)) + vnx0(nbe(ne(:,2),2)) + vnx0(nbe(ne(:,2),3)) &
        !& + vnx0(nbe(ne(:,3),1)) + vnx0(nbe(ne(:,3),2)) + vnx0(nbe(ne(:,3),3)))/8.0D0
        !vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3))+ vny0(nbe(ne(:,2),1)) + vny0(nbe(ne(:,2),2)) + vny0(nbe(ne(:,2),3)) &
        !& + vny0(nbe(ne(:,3),1)) + vny0(nbe(ne(:,3),2)) + vny0(nbe(ne(:,3),3)))/8.0D0
        !vna(:,3)=(vnz0(ne(:,2)) + vnz0(ne(:,3))+ vnz0(nbe(ne(:,2),1)) + vnz0(nbe(ne(:,2),2)) + vnz0(nbe(ne(:,2),3)) &
        !& + vnz0(nbe(ne(:,3),1)) + vnz0(nbe(ne(:,3),2)) + vnz0(nbe(ne(:,3),3)))/8.0D0
!----------------------------------------------------------------------------------------------------
        !vna(:,1)=(2.0D0*vnx0(ne(:,2)) + 2.0D0*vnx0(ne(:,3))+ vnx0(nbe(ne(:,2),1)) + vnx0(nbe(ne(:,2),2)) + vnx0(nbe(ne(:,2),3)) &
        !& + vnx0(nbe(ne(:,3),1)) + vnx0(nbe(ne(:,3),2)) + vnx0(nbe(ne(:,3),3)))/10.0D0
        !vna(:,2)=(2.0D0*vny0(ne(:,2)) + 2.0D0*vny0(ne(:,3))+ vny0(nbe(ne(:,2),1)) + vny0(nbe(ne(:,2),2)) + vny0(nbe(ne(:,2),3)) &
        !& + vny0(nbe(ne(:,3),1)) + vny0(nbe(ne(:,3),2)) + vny0(nbe(ne(:,3),3)))/10.0D0
        !vna(:,3)=(2.0D0*vnz0(ne(:,2)) + 2.0D0*vnz0(ne(:,3))+ vnz0(nbe(ne(:,2),1)) + vnz0(nbe(ne(:,2),2)) + vnz0(nbe(ne(:,2),3)) &
        !& + vnz0(nbe(ne(:,3),1)) + vnz0(nbe(ne(:,3),2)) + vnz0(nbe(ne(:,3),3)))/10.0D0
!----------------------------------------------------------------------------------------------------
        !vna(:,1)=(3.0D0*vnx0(nbe(ne(:,2),1)) + 3.0D0*vnx0(nbe(ne(:,2),2)) + 3.0D0*vnx0(nbe(ne(:,2),3)) + 3.0D0*vnx0(nbe(ne(:,3),1)) &
        !& + 3.0D0*vnx0(nbe(ne(:,3),2)) + 3.0D0*vnx0(nbe(ne(:,3),3))+ vnx0(ne(:,2)) + vnx0(ne(:,3)))/20.0D0
        !vna(:,2)=(3.0D0*vny0(nbe(ne(:,2),1)) + 3.0D0*vny0(nbe(ne(:,2),2)) + 3.0D0*vny0(nbe(ne(:,2),3)) + 3.0D0*vny0(nbe(ne(:,3),1)) &
        !& + 3.0D0*vny0(nbe(ne(:,3),2)) + 3.0D0*vny0(nbe(ne(:,3),3)) + vny0(ne(:,2)) + vny0(ne(:,3)))/20.0D0
        !vna(:,3)=(3.0D0*vnz0(nbe(ne(:,2),1)) + 3.0D0*vnz0(nbe(ne(:,2),2)) + 3.0D0*vnz0(nbe(ne(:,2),3)) + 3.0D0*vnz0(nbe(ne(:,3),1)) &
        !& + 3.0D0*vnz0(nbe(ne(:,3),2)) + 3.0D0*vnz0(nbe(ne(:,3),3)) + vnz0(ne(:,2)) + vnz0(ne(:,3)))/20.0D0
!----------------------------------------------------------------------------------------------------
        vna(:,1)= (vnx0(ne(:,2)) + vnx0(ne(:,3)))/2.0D0
        vna(:,2)= (vny0(ne(:,2)) + vny0(ne(:,3)))/2.0D0
        vna(:,3)= (vnz0(ne(:,2)) + vnz0(ne(:,3)))/2.0D0

!----------------------------------------------------------------------------------------------------
        !vna(:,1)= 2.0D0
        !vna(:,2)= 2.0D0
        !vna(:,3)= 2.0D0
        !
!!        dist1(:)=DSQRT((p(:,1)-x0(ne(:,2)))**2+(p(:,2)-y0(ne(:,2)))**2+(p(:,3)-z0(ne(:,2)))**2)
!        dist2(:)=DSQRT((p(:,1)-x0(ne(:,3)))**2+(p(:,2)-y0(ne(:,3)))**2+(p(:,3)-z0(ne(:,3)))**2)
!!----------------------------------------------------------------------------------------------------
!        absdist(:)= DABS((1.0D0/dist1(:)) + (1.0D0/dist2(:)))
!!----------------------------------------------------------------------------------------------------
!        vna(:,1)= ((vnx0(ne(:,2))*dist1(:))+(vnx0(ne(:,3))*dist2(:)))/absdist(:)
!        vna(:,2)= ((vny0(ne(:,2))*dist1(:))+(vny0(ne(:,3))*dist2(:)))/absdist(:)
!        vna(:,3)= ((vnz0(ne(:,2))*dist1(:))+(vnz0(ne(:,3))*dist2(:)))/absdist(:)
!----------------------------------------------------------------------------------------------------
    END WHERE
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE interp_e
!====================================================================================================
    SUBROUTINE interp_evn( npts,    ne,   nbe,&
                         & vnx0,  vny0,  vnz0,&
                         & vna)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
    USE Mod_SharedVars, ONLY: DBL, p, x0, y0, z0
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: npts                                             !number of nodes
    INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: ne                  !table of conectivities per node
    INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: nbe                 !table of conectivities per element
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: vnx0, vny0, vnz0  !arrays of the components of
                                                                            !normal vectors over collocation points
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(OUT) :: vna  !array whit the values of unit normal
                                                                            !vector of every node over the surface
```

```fortran
!=====================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                        !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: norma
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: dist1,dist2, absdist
!=====================================================================================================
!   Initialize
    ALLOCATE(vna(npts,3), norma(npts), dist1(npts),dist2(npts),absdist(npts))
    vna = 0.0D0
    norma = 0.0D0
    dist1 = 0.0D0
    dist2 = 0.0D0
    absdist=0.0D0
!-----------------------------------------------------------------------------------------------------
    WHERE(ne(:,1)==4)
        vna(:,1)=(vnx0(ne(:,2)) + vnx0(ne(:,3)) + vnx0(ne(:,4)) + vnx0(ne(:,5)))/4.0D0
        vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(ne(:,4)) + vny0(ne(:,5)))/4.0D0
        vna(:,3)=(vnz0(ne(:,2)) + vnz0(ne(:,3)) + vnz0(ne(:,4)) + vnz0(ne(:,5)))/4.0D0
!-----------------------------------------------------------------------------------------------------
        !vna(:,1)=(vnx0(ne(:,2)) + vnx0(ne(:,3)) + vnx0(ne(:,4)) + vnx0(ne(:,5))&
        !& + vnx0(nbe(ne(:,2),1)) + vnx0(nbe(ne(:,2),2)) + vnx0(nbe(ne(:,2),3)) &
        !& + vnx0(nbe(ne(:,3),1)) + vnx0(nbe(ne(:,3),2)) + vnx0(nbe(ne(:,3),3)) &
        !& + vnx0(nbe(ne(:,4),1)) + vnx0(nbe(ne(:,4),2)) + vnx0(nbe(ne(:,4),3)) &
        !& + vnx0(nbe(ne(:,5),1)) + vnx0(nbe(ne(:,5),2)) + vnx0(nbe(ne(:,5),3)))/16.0D0
        !vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(ne(:,4)) +vny0(ne(:,5))&
        !& + vny0(nbe(ne(:,2),1)) + vny0(nbe(ne(:,2),2)) + vny0(nbe(ne(:,2),3)) &
        !& + vny0(nbe(ne(:,3),1)) + vny0(nbe(ne(:,3),2)) + vny0(nbe(ne(:,3),3)) &
        !& + vny0(nbe(ne(:,4),1)) + vny0(nbe(ne(:,4),2)) + vny0(nbe(ne(:,4),3)) &
        !& + vny0(nbe(ne(:,5),1)) + vny0(nbe(ne(:,5),2)) + vny0(nbe(ne(:,5),3)))/16.0D0
        !vna(:,3)=(vnz0(ne(:,2)) + vnz0(ne(:,3)) + vnz0(ne(:,4)) + vnz0(ne(:,5))&
        !& + vnz0(nbe(ne(:,2),1)) + vnz0(nbe(ne(:,2),2)) + vnz0(nbe(ne(:,2),3)) &
        !& + vnz0(nbe(ne(:,3),1)) + vnz0(nbe(ne(:,3),2)) + vnz0(nbe(ne(:,3),3)) &
        !& + vnz0(nbe(ne(:,4),1)) + vnz0(nbe(ne(:,4),2)) + vnz0(nbe(ne(:,4),3)) &
        !& + vnz0(nbe(ne(:,5),1)) + vnz0(nbe(ne(:,5),2)) + vnz0(nbe(ne(:,5),3)))/16.0D0
!-----------------------------------------------------------------------------------------------------
        !vna(:,1)=(vnx0(ne(:,2)) + vnx0(ne(:,3)) + vnx0(ne(:,4)) + vnx0(ne(:,5))&
        !& + 3.0D0*(vnx0(nbe(ne(:,2),1)) + vnx0(nbe(ne(:,2),2)) + vnx0(nbe(ne(:,2),3))) &
        !& + 3.0D0*(vnx0(nbe(ne(:,3),1)) + vnx0(nbe(ne(:,3),2)) + vnx0(nbe(ne(:,3),3))) &
        !& + 3.0D0*(vnx0(nbe(ne(:,4),1)) + vnx0(nbe(ne(:,4),2)) + vnx0(nbe(ne(:,4),3))) &
        !& + 3.0D0*(vnx0(nbe(ne(:,5),1)) + vnx0(nbe(ne(:,5),2)) + vnx0(nbe(ne(:,5),3))))/40.0D0
        !vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(ne(:,4)) +vny0(ne(:,5))&
        !& + 3.0D0*(vny0(nbe(ne(:,2),1)) + vny0(nbe(ne(:,2),2)) + vny0(nbe(ne(:,2),3))) &
        !& + 3.0D0*(vny0(nbe(ne(:,3),1)) + vny0(nbe(ne(:,3),2)) + vny0(nbe(ne(:,3),3))) &
        !& + 3.0D0*(vny0(nbe(ne(:,4),1)) + vny0(nbe(ne(:,4),2)) + vny0(nbe(ne(:,4),3))) &
        !& + 3.0D0*(vny0(nbe(ne(:,5),1)) + vny0(nbe(ne(:,5),2)) + vny0(nbe(ne(:,5),3))))/40.0D0
        !vna(:,3)=(vnz0(ne(:,2)) + vnz0(ne(:,3)) + vnz0(ne(:,4)) + vnz0(ne(:,5))&
        !& + 3.0D0*(vnz0(nbe(ne(:,2),1)) + vnz0(nbe(ne(:,2),2)) + vnz0(nbe(ne(:,2),3))) &
        !& + 3.0D0*(vnz0(nbe(ne(:,3),1)) + vnz0(nbe(ne(:,3),2)) + vnz0(nbe(ne(:,3),3))) &
        !& + 3.0D0*(vnz0(nbe(ne(:,4),1)) + vnz0(nbe(ne(:,4),2)) + vnz0(nbe(ne(:,4),3))) &
        !& + 3.0D0*(vnz0(nbe(ne(:,5),1)) + vnz0(nbe(ne(:,5),2)) + vnz0(nbe(ne(:,5),3))))/40.0D0
    END WHERE
!-----------------------------------------------------------------------------------------------------
    WHERE(ne(:,1) == 6)
        vna(:,1)=(vnx0(ne(:,2)) + vnx0(ne(:,3)) + vnx0(ne(:,4)) + vnx0(ne(:,5)) &
              & + vnx0(ne(:,6)) + vnx0(ne(:,7)))/6.0D0
        vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(ne(:,4)) + vny0(ne(:,5)) &
              & + vny0(ne(:,6)) + vny0(ne(:,7)))/6.0D0
        vna(:,3)=(vnz0(ne(:,2)) + vnz0(ne(:,3)) + vnz0(ne(:,4)) + vnz0(ne(:,5)) &
              & + vnz0(ne(:,6)) + vnz0(ne(:,7)))/6.0D0
!-----------------------------------------------------------------------------------------------------
        !vna(:,1)=(vnx0(ne(:,2)) + vnx0(ne(:,3)) + vnx0(ne(:,4)) + vnx0(ne(:,5)) &
        !& + vnx0(ne(:,6)) + vnx0(ne(:,7))                                       &
        !& + vnx0(nbe(ne(:,2),1)) + vnx0(nbe(ne(:,2),2)) + vnx0(nbe(ne(:,2),3))  &
        !& + vnx0(nbe(ne(:,3),1)) + vnx0(nbe(ne(:,3),2)) + vnx0(nbe(ne(:,3),3))  &
        !& + vnx0(nbe(ne(:,4),1)) + vnx0(nbe(ne(:,4),2)) + vnx0(nbe(ne(:,4),3))  &
        !& + vnx0(nbe(ne(:,5),1)) + vnx0(nbe(ne(:,5),2)) + vnx0(nbe(ne(:,5),3))  &
        !& + vnx0(nbe(ne(:,6),1)) + vnx0(nbe(ne(:,6),2)) + vnx0(nbe(ne(:,6),3))  &
        !& + vnx0(nbe(ne(:,7),1)) + vnx0(nbe(ne(:,7),2)) + vnx0(nbe(ne(:,7),3)))/24.0D0
        !
        !vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(ne(:,4)) + vny0(ne(:,5)) &
        !& + vny0(ne(:,6)) + vny0(ne(:,7))                                       &
```



```fortran
        !& + vny0(nbe(ne(:,2),1)) + vny0(nbe(ne(:,2),2)) + vny0(nbe(ne(:,2),3))  &
        !& + vny0(nbe(ne(:,3),1)) + vny0(nbe(ne(:,3),2)) + vny0(nbe(ne(:,3),3))  &
        !& + vny0(nbe(ne(:,4),1)) + vny0(nbe(ne(:,4),2)) + vny0(nbe(ne(:,4),3))  &
        !& + vny0(nbe(ne(:,5),1)) + vny0(nbe(ne(:,5),2)) + vny0(nbe(ne(:,5),3))  &
        !& + vny0(nbe(ne(:,6),1)) + vny0(nbe(ne(:,6),2)) + vny0(nbe(ne(:,6),3))  &
        !& + vny0(nbe(ne(:,7),1)) + vny0(nbe(ne(:,7),2)) + vny0(nbe(ne(:,7),3)))/24.0D0
        !
        !vna(:,3)=(vnz0(ne(:,2)) + vnz0(ne(:,3)) + vnz0(ne(:,4)) + vnz0(ne(:,5)) &
        !& + vnz0(ne(:,6)) + vnz0(ne(:,7))                                       &
        !& + vnz0(nbe(ne(:,2),1)) + vnz0(nbe(ne(:,2),2)) + vnz0(nbe(ne(:,2),3))  &
        !& + vnz0(nbe(ne(:,3),1)) + vnz0(nbe(ne(:,3),2)) + vnz0(nbe(ne(:,3),3))  &
        !& + vnz0(nbe(ne(:,4),1)) + vnz0(nbe(ne(:,4),2)) + vnz0(nbe(ne(:,4),3))  &
        !& + vnz0(nbe(ne(:,5),1)) + vnz0(nbe(ne(:,5),2)) + vnz0(nbe(ne(:,5),3))  &
        !& + vnz0(nbe(ne(:,6),1)) + vnz0(nbe(ne(:,6),2)) + vnz0(nbe(ne(:,6),3))  &
        !& + vnz0(nbe(ne(:,7),1)) + vnz0(nbe(ne(:,7),2)) + vnz0(nbe(ne(:,7),3)))/24.0D0
    END WHERE
!-----------------------------------------------------------------------------------------------------
    WHERE (ne(:,1)==2)
        !vna(:,1)=(vnx0(ne(:,2)) + vnx0(ne(:,3))+ vnx0(nbe(ne(:,2),1)) + vnx0(nbe(ne(:,2),2)) + vnx0(nbe(ne(:,2),
    3)) &
        !& + vnx0(nbe(ne(:,3),1)) + vnx0(nbe(ne(:,3),2)) + vnx0(nbe(ne(:,3),3)))/8.0D0
        !vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3))+ vny0(nbe(ne(:,2),1)) + vny0(nbe(ne(:,2),2)) + vny0(nbe(ne(:,2),
    3)) &
        !& + vny0(nbe(ne(:,3),1)) + vny0(nbe(ne(:,3),2)) + vny0(nbe(ne(:,3),3)))/8.0D0
        !vna(:,3)=(vnz0(ne(:,2)) + vnz0(ne(:,3))+ vnz0(nbe(ne(:,2),1)) + vnz0(nbe(ne(:,2),2)) + vnz0(nbe(ne(:,2),
    3)) &
        !& + vnz0(nbe(ne(:,3),1)) + vnz0(nbe(ne(:,3),2)) + vnz0(nbe(ne(:,3),3)))/8.0D0
!-----------------------------------------------------------------------------------------------------
        !vna(:,1)=(2.0D0*vnx0(ne(:,2)) + 2.0D0*vnx0(ne(:,3))+ vnx0(nbe(ne(:,2),1)) + vnx0(nbe(ne(:,2),2)) + vnx0
    (nbe(ne(:,2),3)) &
        !& + vnx0(nbe(ne(:,3),1)) + vnx0(nbe(ne(:,3),2)) + vnx0(nbe(ne(:,3),3)))/10.0D0
        !vna(:,2)=(2.0D0*vny0(ne(:,2)) + 2.0D0*vny0(ne(:,3))+ vny0(nbe(ne(:,2),1)) + vny0(nbe(ne(:,2),2)) + vny0
    (nbe(ne(:,2),3)) &
        !& + vny0(nbe(ne(:,3),1)) + vny0(nbe(ne(:,3),2)) + vny0(nbe(ne(:,3),3)))/10.0D0
        !vna(:,3)=(2.0D0*vnz0(ne(:,2)) + 2.0D0*vnz0(ne(:,3))+ vnz0(nbe(ne(:,2),1)) + vnz0(nbe(ne(:,2),2)) + vnz0
    (nbe(ne(:,2),3)) &
        !& + vnz0(nbe(ne(:,3),1)) + vnz0(nbe(ne(:,3),2)) + vnz0(nbe(ne(:,3),3)))/10.0D0
!-----------------------------------------------------------------------------------------------------
        !vna(:,1)=(3.0D0*vnx0(nbe(ne(:,2),1)) + 3.0D0*vnx0(nbe(ne(:,2),2)) + 3.0D0*vnx0(nbe(ne(:,2),3)) + 3.0D0*
    vnx0(nbe(ne(:,3),1)) &
        !& + 3.0D0*vnx0(nbe(ne(:,3),2)) + 3.0D0*vnx0(nbe(ne(:,3),3))+ vnx0(ne(:,2)) + vnx0(ne(:,3)))/20.0D0
        !vna(:,2)=(3.0D0*vny0(nbe(ne(:,2),1)) + 3.0D0*vny0(nbe(ne(:,2),2)) + 3.0D0*vny0(nbe(ne(:,2),3)) + 3.0D0*
    vny0(nbe(ne(:,3),1)) &
        !& + 3.0D0*vny0(nbe(ne(:,3),2)) + 3.0D0*vny0(nbe(ne(:,3),3)) + vny0(ne(:,2)) + vny0(ne(:,3)))/20.0D0
        !vna(:,3)=(3.0D0*vnz0(nbe(ne(:,2),1)) + 3.0D0*vnz0(nbe(ne(:,2),2)) + 3.0D0*vnz0(nbe(ne(:,2),3)) + 3.0D0*
    vnz0(nbe(ne(:,3),1)) &
        !& + 3.0D0*vnz0(nbe(ne(:,3),2)) + 3.0D0*vnz0(nbe(ne(:,3),3)) + vnz0(ne(:,2)) + vnz0(ne(:,3)))/20.0D0
!-----------------------------------------------------------------------------------------------------
        vna(:,1)= (vnx0(ne(:,2)) + vnx0(ne(:,3)))/2.0D0
        vna(:,2)= (vny0(ne(:,2)) + vny0(ne(:,3)))/2.0D0
        vna(:,3)= (vnz0(ne(:,2)) + vnz0(ne(:,3)))/2.0D0

!        dist1(:)=DSQRT((p(:,1)-x0(ne(:,2)))**2+(p(:,2)-y0(ne(:,2)))**2+(p(:,3)-z0(ne(:,2)))**2)
!        dist2(:)=DSQRT((p(:,1)-x0(ne(:,3)))**2+(p(:,2)-y0(ne(:,3)))**2+(p(:,3)-z0(ne(:,3)))**2)
!!----------------------------------------------------------------------------------------------------
!        absdist(:)= DABS((1.0D0/dist1(:)) + (1.0D0/dist2(:)))
!!----------------------------------------------------------------------------------------------------
!        vna(:,1)= ((vnx0(ne(:,2))*dist1(:))+(vnx0(ne(:,3))*dist2(:)))/absdist(:)
!        vna(:,2)= ((vny0(ne(:,2))*dist1(:))+(vny0(ne(:,3))*dist2(:)))/absdist(:)
!        vna(:,3)= ((vnz0(ne(:,2))*dist1(:))+(vnz0(ne(:,3))*dist2(:)))/absdist(:)
!

    END WHERE

!-----------------------------------------------------------------------------------------------------
    !FORALL (i=1:npts)
    !    vna(i,:)= vna(i,:)*DBLE(REAL(npts))
    !END FORALL
!-----------------------------------------------------------------------------------------------------
    FORALL (i=1:npts)
        norma(i)= DSQRT(SUM(vna(i,:)**2))
    END FORALL
```

```fortran
!----------------------------------------------------------------------------------------------------
    FORALL (i=1:npts)
        vna(i,:)= vna(i,:)/norma(i)
    END FORALL
!----------------------------------------------------------------------------------------------------
   ! WHERE(norma(:)>1.0D0)
   !     vna(:,1)= 25.0D0 !vna(:,1)/norma(:)
   !     vna(:,2)= 35.0D0 !vna(:,2)/norma(:)
   !     vna(:,3)= 45.0D0 !vna(:,3)/norma(:)
   !END WHERE
   !
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE interp_evn
!====================================================================================================
!====================================================================================================
    SUBROUTINE interp_ven( npts,  ne,   nbe,&
                         & Vel1x, Vel1y, Vel1z, Vel)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
    USE Mod_SharedVars, ONLY: DBL, p
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: npts                                         !number of nodes
    INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: ne              !table of conectivities per node
    INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: nbe             !table of conectivities per element


    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: Vel1x, Vel1y, Vel1z !arrays of the components of
                                                                        !normal vectors over collocation points
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(OUT) :: Vel    !array whit the values of unit normal
                                                                        !vector of every node over the surface
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                !Counters
    ALLOCATE(Vel(npts,3))
    Vel = 0.0D0
!====================================================================================================
    WHERE(ne(:,1)==4)
        Vel(:,1)=(Vel1x(ne(:,2)) + Vel1x(ne(:,3)) + Vel1x(ne(:,4)) + Vel1x(ne(:,5)))/4.0D0
        Vel(:,2)=(Vel1y(ne(:,2)) + Vel1y(ne(:,3)) + Vel1y(ne(:,4)) + Vel1y(ne(:,5)))/4.0D0
        Vel(:,3)=(Vel1z(ne(:,2)) + Vel1z(ne(:,3)) + Vel1z(ne(:,4)) + Vel1z(ne(:,5)))/4.0D0
    END WHERE
!----------------------------------------------------------------------------------------------------
    WHERE(ne(:,1) == 6)
        Vel(:,1)=(Vel1x(ne(:,2)) + Vel1x(ne(:,3)) + Vel1x(ne(:,4)) + Vel1x(ne(:,5)) &
              & + Vel1x(ne(:,6)) + Vel1x(ne(:,7)))/6.0D0
        Vel(:,2)=(Vel1y(ne(:,2)) + Vel1y(ne(:,3)) + Vel1y(ne(:,4)) + Vel1y(ne(:,5)) &
              & + Vel1y(ne(:,6)) + Vel1y(ne(:,7)))/6.0D0
        Vel(:,3)=(Vel1z(ne(:,2)) + Vel1z(ne(:,3)) + Vel1z(ne(:,4)) + Vel1z(ne(:,5)) &
              & + Vel1z(ne(:,6)) + Vel1z(ne(:,7)))/6.0D0
!----------------------------------------------------------------------------------------------------
    END WHERE
!----------------------------------------------------------------------------------------------------
    WHERE (ne(:,1)==2)
!----------------------------------------------------------------------------------------------------
        !Vel(:,1)=(Vel1x(ne(:,2)) + Vel1x(ne(:,3))+ Vel1x(nbe(ne(:,2),1)) + Vel1x(nbe(ne(:,2),2)) + Vel1x(nbe(ne( ↙
    :,2),3)) &
        !& +Vel1x(nbe(ne(:,3),1)) + Vel1x(nbe(ne(:,3),2)) + Vel1x(nbe(ne(:,3),3)))/8.0D0
        !Vel(:,2)=(Vel1y(ne(:,2)) + Vel1y(ne(:,3))+ Vel1y(nbe(ne(:,2),1)) + Vel1y(nbe(ne(:,2),2)) + Vel1y(nbe(ne( ↙
    :,2),3)) &
        !& +Vel1y(nbe(ne(:,3),1)) + Vel1y(nbe(ne(:,3),2)) + Vel1y(nbe(ne(:,3),3)))/8.0D0
        !Vel(:,3)=(Vel1z(ne(:,2)) + Vel1z(ne(:,3))+ Vel1z(nbe(ne(:,2),1)) + Vel1z(nbe(ne(:,2),2)) + Vel1z(nbe(ne( ↙
    :,2),3)) &
        !& + Vel1z(nbe(ne(:,3),1)) + Vel1z(nbe(ne(:,3),2)) + Vel1z(nbe(ne(:,3),3)))/8.0D0
!----------------------------------------------------------------------------------------------------
        Vel(:,1)= (Vel1x(ne(:,2)) + Vel1x(ne(:,3)))/2.0D0
        Vel(:,2)= (Vel1y(ne(:,2)) + Vel1y(ne(:,3)))/2.0D0
```



```fortran
        Vel(:,3)= (Vel1z(ne(:,2)) + Vel1z(ne(:,3)))/2.0D0
    END WHERE
!!---------------------------------------------------------------------------------------------------
!    !FORALL (i=1:npts)
!    !    vna(i,:)= vna(i,:)*DBLE(REAL(npts))
!    !END FORALL
!!---------------------------------------------------------------------------------------------------
!    FORALL (i=1:npts)
!        norma(i)= DSQRT(SUM(vna(i,:)**2))
!    END FORALL
!!---------------------------------------------------------------------------------------------------
!    FORALL (i=1:npts)
!        vna(i,:)= vna(i,:)/norma(i)
!    END FORALL
!!---------------------------------------------------------------------------------------------------
!   ! WHERE(norma(:)>1.0D0)
!   !     vna(:,1)= 25.0D0 !vna(:,1)/norma(:)
!   !     vna(:,2)= 35.0D0 !vna(:,2)/norma(:)
!   !     vna(:,3)= 45.0D0 !vna(:,3)/norma(:)
!   !END WHERE
!   !
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE interp_ven
!====================================================================================================
    SUBROUTINE interp_curv( nelm,  ne,   nbe,&
                          & crvmel)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
    USE Mod_SharedVars, ONLY: DBL, p
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: nelm                                         !number of elements
    INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: ne              !table of conectivities per node
    INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: nbe             !table of conectivities per element
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: crvmel    !array whit the values of kappa
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: cuvi
    ALLOCATE(cuvi(nelm))                            !Dummy array
    cuvi = 0.0D0
!====================================================================================================
    WHERE(ne(:,1)==4)
        cuvi(:)=(crvmel(ne(:,2)) + crvmel(ne(:,3)) +crvmel(ne(:,4)) + crvmel(ne(:,5)))/4.0D0
    END WHERE
!----------------------------------------------------------------------------------------------------
    WHERE(ne(:,1) == 6)
        cuvi(:)=(crvmel(ne(:,2)) + crvmel(ne(:,3)) + crvmel(ne(:,4)) + crvmel(ne(:,5)) &
              & + crvmel(ne(:,6)) + crvmel(ne(:,7)))/6.0D0
!----------------------------------------------------------------------------------------------------
    END WHERE
!----------------------------------------------------------------------------------------------------
    WHERE (ne(:,1)==2)
!----------------------------------------------------------------------------------------------------
        cuvi(:)= (crvmel(ne(:,2)) + crvmel(ne(:,3)))/2.0D0
!----------------------------------------------------------------------------------------------------
    END WHERE
!----------------------------------------------------------------------------------------------------
    crvmel = cuvi
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE interp_curv
!====================================================================================================
!====================================================================================================
    SUBROUTINE do_t_produc(Mdim, nelm, gamma,&
                         &  GM, RM, crvmel,   &
                         &  vnx0, vny0, vnz0  )
!====================================================================================================
```

```fortran
!   This subroutine generate th array fo G(x,X0) dot delta f
!-----------------------------------------------------------------------------------------------------------
!   Variables
!-----------------------------------------------------------------------------------------------------------
!   vnx0, vny0, vnz0 .......... arrays of the components of normal vectors over collocation points
!   vna ....................... array whit the values of unit normal vector of every node over the surface
!   ne ........................ table of conectivities per node
!   nbe ....................... table of conectivities per element
!   npts ...................... number of nodes
!===========================================================================================================
    USE Mod_SharedVars, ONLY: DBL
!===========================================================================================================
    IMPLICIT NONE
!===========================================================================================================
!   Variables
!-----------------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: Mdim, nelm                                     !
    REAL (KIND = DBL), INTENT(IN)  :: gamma                               !seurface tension constant
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN)   :: vnx0, vny0, vnz0  !arrays of the components of
                                                                        !normal vectors over collocation points
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN)   :: crvmel  !
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: GM      !The matrix with the information of
 !Stokeslets on the surface
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(OUT)  :: RM      !The matrix with the information of
!===========================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------------
    INTEGER :: i,j
!-----------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: MM
!    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: MM
!===========================================================================================================
!   Initialize
    ALLOCATE(MM(Mdim), RM(Mdim))
!    ALLOCATE(MM(Mdim,3), RM(Mdim))
    MM = 0.0D0
    RM = 0.0D0
!!-----------------------------------------------------------------------------------------------------------
!    FORALL (i=1:nelm)
!        MM(i,1)= SUM(GM(i,1:nelm)*vnx0(i))
!        MM(i,2)= SUM(GM(i,1+nelm:nelm+nelm)*vny0(i))
!        MM(i,3)= SUM(GM(i,1+nelm+nelm:Mdim)*vnz0(i))
!    END FORALL
!    FORALL (i=1:nelm)
!        MM(i+nelm,1)=SUM(GM(i+nelm,1:nelm)*vnx0(i))
!        MM(i+nelm,2)=SUM(GM(i+nelm,1+nelm:nelm+nelm)*vny0(i))
!        MM(i+nelm,3)=SUM(GM(i+nelm,1+nelm+nelm:Mdim)*vnz0(i))
!    END FORALL
!    FORALL (i=1:nelm)
!        MM(i+nelm+nelm,1)=SUM(GM(i+nelm+nelm,1:nelm)*vnx0(i))
!        MM(i+nelm+nelm,2)=SUM(GM(i+nelm+nelm,1+nelm:nelm+nelm)*vny0(i))
!        MM(i+nelm+nelm,3)=SUM(GM(i+nelm+nelm,1+nelm+nelm:Mdim)*vnz0(i))
!    END FORALL
!!-----------------------------------------------------------------------------------------------------------
!    FORALL (i=1:nelm)
!        RM(i)= SUM(MM(i,:))
!        RM(i+nelm)=SUM(MM(i+nelm,:))
!        RM(i+nelm+nelm)=SUM(MM(i+nelm+nelm,:))
!    END FORALL
!!-----------------------------------------------------------------------------------------------------------
!    FORALL (i=1:nelm)
!        RM(i)= RM(i) *crvmel(i)*gamma
!        RM(i+nelm)=RM(i+nelm) *crvmel(i)*gamma
!        RM(i+nelm+nelm)=RM(i+nelm+nelm)*crvmel(i)*gamma
!    END FORALL
!!-----------------------------------------------------------------------------------------------------------
!-----------------------------------------------------------------------------------------------------------
!    FORALL (i=1:nelm)
!        MM(i)= vnx0(i) *crvmel(i)*gamma
!        MM(i+nelm)=vny0(i) *crvmel(i)*gamma
!        MM(i+nelm+nelm)=vnz0(i) *crvmel(i)*gamma
!    END FORALL
```



```fortran
!!-----------------------------------------------------------------------------------------------------------
!    RM=MATMUL(GM,MM)
!!-----------------------------------------------------------------------------------------------------------
    FORALL (i=1:nelm)
        RM(i)= GM(i) !*crvmel(i) !*gamma
        RM(i+nelm)= GM(i+nelm) !*crvmel(i) !*gamma
        RM(i+nelm+nelm)= GM(i+nelm+nelm) !*crvmel(i) !*gamma
    END FORALL

    END SUBROUTINE do_t_produc
!===========================================================================================================
    SUBROUTINE dissociate(nelm,  Vel1,       &
                        & Vel1x, Vel1y, Vel1z)
!===========================================================================================================
    USE Mod_SharedVars, ONLY: DBL
!===========================================================================================================
    IMPLICIT NONE
!===========================================================================================================
!   Variables
!-----------------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) ::nelm                                              !number of elements
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN)    :: Vel1 !array whit the values of unit normal
                                                                            !vector of every node over the surface
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: Vel1x, Vel1y, Vel1z !arrays of the components
                                                                            ! of normal vectors over collocation points
!===========================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------------
    INTEGER :: i                                                      !Counter
!-----------------------------------------------------------------------------------------------------------
!===========================================================================================================
    Vel1x = 0.0D0
    Vel1y = 0.0D0
    Vel1z = 0.0D0
!-----------------------------------------------------------------------------------------------------------


    Vel1x= Vel1(1:nelm)
    Vel1y= Vel1(nelm+1:nelm+nelm)
    Vel1z= Vel1(nelm+nelm+1:)

!-----------------------------------------------------------------------------------------------------------
    END SUBROUTINE dissociate
!===========================================================================================================
    SUBROUTINE interp_e2( npts,vna1,vna)
!===========================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!===========================================================================================================
    USE Mod_SharedVars, ONLY: DBL !, p , x0, y0, z0
!===========================================================================================================
    IMPLICIT NONE
!===========================================================================================================
!   Variables
!-----------------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: npts                                                 !number of nodes
!    INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: ne              !table of conectivities per node
!    INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: nbe             !table of conectivities per element
!    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: vnx0, vny0, vnz0  !arrays of the components of
                                                                                !normal vectors over collocation points
    REAL (KIND = DBL), DIMENSION(:,:), INTENT(IN) :: vna1  !array whit the values of unit normal
                                                                                !vector of every node (local)
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(OUT) :: vna  !array whit the values of unit normal
                                                                                !vector of every node over the surface
!===========================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                   !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: norma
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: dist1,dist2,dist3, absdist
!===========================================================================================================
!   Initialize
    ALLOCATE(vna(npts,3), norma(npts), dist1(npts),dist2(npts),dist3(npts),absdist(npts))
    vna = 0.0D0
```

```
    norma = 0.0D0
    dist1 = 0.0D0
    dist2 = 0.0D0
    dist3 = 0.0D0
!    absdist=0.0D0
!-----------------------------------------------------------------------------------------------------
    FORALL (i=1:npts)
        dist1(i)= SUM(vna1(:,1), INT(vna1(:,4))==i)
        dist2(i)= SUM(vna1(:,2), INT(vna1(:,4))==i)
        dist3(i)= SUM(vna1(:,3), INT(vna1(:,4))==i)
    END FORALL
!-----------------------------------------------------------------------------------------------------
!-----------------------------------------------------------------------------------------------------
    FORALL (i=1:npts)
        norma(i)= DSQRT(dist1(i)**2+dist2(i)**2+dist3(i)**2)
        vna(i,1)= dist1(i)/norma(i)
        vna(i,2)= dist2(i)/norma(i)
        vna(i,3)= dist3(i)/norma(i)
       ! vna(i,:)= (/dist1,dist2,dist3/)/norma(i)
    END FORALL
!-----------------------------------------------------------------------------------------------------
    !vna(:,1)= dist1
    !vna(:,2)= dist2
    !vna(:,3)= dist2
 !    vna(i,:)= (/dist1,dist2,dist3/)/norma(i)
!-----------------------------------------------------------------------------------------------------
    END SUBROUTINE interp_e2

!=====================================================================================================
    SUBROUTINE interp_p2(  x1,    y1,    z1,  &
    &                      x2,    y2,    z2,  &
    &                      x3,    y3,    z3,  &
    &                      x4,    y4,    z4,  &
    &                      x5,    y5,    z5,  &
    &                      x6,    y6,    z6,  &
    &                       x,     y,     z,  &
    &                     nx1,   ny1,   nz1,  &
    &                     nx2,   ny2,   nz2,  &
    &                     nx3,   ny3,   nz3,  &
    &                     nx4,   ny4,   nz4,  &
    &                     nx5,   ny5,   nz5,  &
    &                     nx6,   ny6,   nz6,  &
    &                     vnx,   vny,   vnz,  &
    &                      hs                 )
!=====================================================================================================
!   This subroutine interpolate over an element to compute geometrical variables including the following:

!   1) Position vector
!   2) Tangential vectors in the xi and eta directions
!   3) Unit normal vector
!   4) Area and volume of each element
!=====================================================================================================
    USE Mod_SharedVars, ONLY: DBL, vna1
!=====================================================================================================
    IMPLICIT NONE
!=====================================================================================================
!   Variables
    REAL (KIND = DBL), INTENT(IN)  :: x1, y1, z1, &            !coordinates of first node of the element
    &                                 x2, y2, z2, &            !coordinates of second node of the element
    &                                 x3, y3, z3, &            !coordinates of third node of the element
    &                                 x4, y4, z4, &            !coordinates of fourth node of the element
    &                                 x5, y5, z5, &            !coordinates of fifth node of the element
    &                                 x6, y6, z6               !coordinates of sixth node of the element
    REAL (KIND = DBL), INTENT(OUT) :: x, y, z                  !coordinates of the collocation point
    REAL (KIND = DBL), INTENT(OUT) :: nx1, ny1, nz1, &         !normal vector coordinates of first node of the
    element
    &                                 nx2, ny2, nz2, &         !normal vector coordinates of second node of the
    element
    &                                 nx3, ny3, nz3, &         !normal vector coordinates of third node of the
    element
    &                                 nx4, ny4, nz4, &         !normal vector coordinates of fourth node of the
    element
```



```
    &                                 nx5, ny5, nz5, &         !normal vector coordinates of fifth node of the
    element
    &                                 nx6, ny6, nz6            !normal vector coordinates of sixth node of the
    element
    REAL (KIND = DBL), INTENT(OUT) :: vnx, vny, vnz            !normal vector of the element
    REAL (KIND = DBL), INTENT(OUT) :: hs                       !surface metric
!=====================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: w12, w13, w21, w31, w23, w32          !components weiths
    REAL (KIND = DBL) :: b210x, b210y, b210z, b120x, b120y, b120z, & !components of bezier surface
                       & b021x, b021y, b021z, b012x, b012y, b012z, &
                       & b102x, b102y, b102z, b201x, b201y, b201z
    REAL (KIND = DBL) :: ex, ey, ez                                  !components of bezier surface
    REAL (KIND = DBL) :: n1, n2, n3, n4, n5, n6                      !modules of normal vectors
    REAL (KIND = DBL) :: v1x, v1y, v1z                               !components of bezier surface
    REAL (KIND = DBL) :: v12, v23, v31                               !components weiths
    REAL (KIND = DBL) :: h110x, h110y, h110z, h011x, h011y, h011z, & !components of bezier surface
                       & h101x, h101y, h101z
!-----------------------------------------------------------------------------------------------------
!   Prepare the normal vector of vertices.
!-----------------------------------------------------------------------------------------------------
    nx1= -y4*z1 + y6*z1 + y1*z4 - y6*z4 - y1*z6 + y4*z6
    ny1= x4*z1 - x6*z1 - x1*z4 + x6*z4 + x1*z6 - x4*z6
    nz1= -x4*y1 + x6*y1 + x1*y4 - x6*y4 - x1*y6 + x4*y6
    nx2= -y5*z2 + y4*z2 + y2*z5 - y4*z5 - y2*z4 + y5*z4
    ny2= x5*z2 - x4*z2 - x2*z5 + x4*z5 + x2*z4 - x5*z4
    nz2= -x5*y2 + x4*y2 + x2*y5 - x4*y5 - x2*y4 + x5*y4
    nx3= -y6*z3 + y5*z3 + y3*z6 - y5*z6 - y3*z5 + y6*z5
    ny3= x6*z3 - x5*z3 - x3*z6 + x5*z6 + x3*z5 - x6*z5
    nz3= -x6*y3 + x5*y3 + x3*y6 - x5*y6 - x3*y5 + x6*y5
!-----------------------------------------------------------------------------------------------------
    n1= DSQRT(nx1**2+ny1**2+nz1**2)
    n2= DSQRT(nx2**2+ny2**2+nz2**2)
    n3= DSQRT(nx3**2+ny3**2+nz3**2)
!-----------------------------------------------------------------------------------------------------
    nx1= nx1/n1
    ny1= ny1/n1
    nz1= nz1/n1
    nx2= nx2/n2
    ny2= ny2/n2
    nz2= nz2/n2
    nx3= nx3/n3
    ny3= ny3/n3
    nz3= nz3/n3
!-----------------------------------------------------------------------------------------------------
    w12=((x2-x1)*nx1+(y2-y1)*ny1+(z2-z1)*nz1)
    w13=((x3-x1)*nx1+(y3-y1)*ny1+(z3-z1)*nz1)
    w21=((x1-x2)*nx2+(y1-y2)*ny2+(z1-z2)*nz2)
    w31=((x1-x3)*nx3+(y1-y3)*ny3+(z1-z3)*nz3)
    w23=((x3-x2)*nx2+(y3-y2)*ny2+(z3-z2)*nz2)
    w32=((x2-x3)*nx3+(y2-y3)*ny3+(z2-z3)*nz3)
!-----------------------------------------------------------------------------------------------------
    b210x=(2*x1+x2-w12*nx1)/3.0D0
    b210y=(2*y1+y2-w12*ny1)/3.0D0
    b210z=(2*z1+z2-w12*nz1)/3.0D0
!-----------------------------------------------------------------------------------------------------
    b120x=(2*x2+x1-w21*nx2)/3.0D0
    b120y=(2*y2+y1-w21*ny2)/3.0D0
    b120z=(2*z2+z1-w21*nz2)/3.0D0
!-----------------------------------------------------------------------------------------------------
    b021x=(2*x2+x3-w23*nx2)/3.0D0
    b021y=(2*y2+y3-w23*ny2)/3.0D0
    b021z=(2*z2+z3-w23*nz2)/3.0D0
!-----------------------------------------------------------------------------------------------------
    b012x=(2*x3+x2-w32*nx3)/3.0D0
    b012y=(2*y3+y2-w32*ny3)/3.0D0
    b012z=(2*z3+z2-w32*nz3)/3.0D0
!-----------------------------------------------------------------------------------------------------
    b102x=(2*x3+x1-w31*nx3)/3.0D0
    b102y=(2*y3+y1-w31*ny3)/3.0D0
    b102z=(2*z3+z1-w31*nz3)/3.0D0
```

```
!-----------------------------------------------------------------------------------------------------------
    b201x=(2*x1+x3-w13*nx1)/3.0D0
    b201y=(2*y1+y3-w13*ny1)/3.0D0
    b201z=(2*z1+z3-w13*nz1)/3.0D0
!-----------------------------------------------------------------------------------------------------------
    ex=(b210x+b120x+b021x+b012x+b102x+b201x)/6.0D0
    ey=(b210y+b120y+b021y+b012y+b102y+b201y)/6.0D0
    ez=(b210z+b120z+b021z+b012z+b102z+b201z)/6.0D0
!-----------------------------------------------------------------------------------------------------------
    v1x=(x1+x2+x3)/3.0D0
    v1y=(y1+y2+y3)/3.0D0
    v1z=(z1+z2+z3)/3.0D0
!-----------------------------------------------------------------------------------------------------------
    x= ex + (ex-v1x)/2.0D0
    y= ey + (ey-v1y)/2.0D0
    z= ez + (ez-v1z)/2.0D0
!-----------------------------------------------------------------------------------------------------------
    v12=2.0D0*(((x2-x1)*(nx1+nx2)+(y2-y1)*(ny1+ny2)+(z2-z1)*(nz1+nz2))/((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1)+(z2-z1)*
    (z2-z1)))
    v23=2.0D0*(((x3-x2)*(nx2+nx3)+(y3-y2)*(ny2+ny3)+(z3-z2)*(nz2+nz3))/((x3-x2)*(x3-x2)+(y3-y2)*(y3-y2)+(z3-z2)*
    (z3-z2)))
    v31=2.0D0*(((x1-x3)*(nx3+nx1)+(y1-y3)*(ny1+ny3)+(z1-z3)*(nz1+nz3))/((x1-x3)*(x1-x3)+(y1-y3)*(y1-y3)+(z1-z3)*
    (z1-z3)))
!-----------------------------------------------------------------------------------------------------------
    h110x = nx1 + nx2 - v12*(x2 - x1)
    h110y = ny1 + ny2 - v12*(y2 - y1)
    h110z = nz1 + nz2 - v12*(z2 - z1)
    h011x = nx2 + nx3 - v23*(x3 - x2)
    h011y = ny2 + ny3 - v23*(y3 - y2)
    h011z = nz2 + nz3 - v23*(z3 - z2)
    h101x = nx3 + nx1 - v31*(x1 - x3)
    h101y = ny3 + ny1 - v31*(y1 - y3)
    h101z = nz3 + nz1 - v31*(z1 - z3)
!-----------------------------------------------------------------------------------------------------------
    n4 = DSQRT(h110x**2 + h110y**2 + h110z**2)
    n5 = DSQRT(h011x**2 + h011y**2 + h011z**2)
    n6 = DSQRT(h101x**2 + h101y**2 + h101z**2)
!-----------------------------------------------------------------------------------------------------------
    nx4 = h110x/n4
    ny4 = h110y/n4
    nz4 = h110z/n4
    nx5 = h011x/n5
    ny5 = h011y/n5
    nz5 = h011z/n5
    nx6 = h101x/n6
    ny6 = h101y/n6
    nz6 = h101z/n6
!-----------------------------------------------------------------------------------------------------------
    vnx =  nx1 + nx2 + nx3 + nx4 + nx5 + nx6
    vny =  ny1 + ny2 + ny3 + ny4 + ny5 + ny6
    vnz =  nz1 + nz2 + nz3 + nz4 + nz5 + nz6
    hs = DSQRT(vnx**2 + vny**2 + vnz**2)
!-----------------------------------------------------------------------------------------------------------
    vnx = vnx/hs
    vny = vny/hs
    vnz = vnz/hs
!-----------------------------------------------------------------------------------------------------------
    END SUBROUTINE interp_p2
!===========================================================================================================
!===========================================================================================================
    SUBROUTINE interp_p3(   x1,    y1,    z1,  &
    &                      x2,    y2,    z2,  &
    &                      x3,    y3,    z3,  &
    &                      x4,    y4,    z4,  &
    &                      x5,    y5,    z5,  &
    &                      x6,    y6,    z6,  &
    &                      al,    be,    ga,  &
    &                      xi,   eta,         &
    &                       x,     y,     z,  &
    &                   DxDxi, DyDxi, DzDxi,  &
    &                   DxDet, DyDet, DzDet,  &
    &                     vnx,   vny,   vnz,  &
```



```
    &                      hs)
!===========================================================================================================
!   This subroutine interpolate over an element to compute geometrical variables including the following:

!   1) Position vector
!   2) Tangential vectors in the xi and eta directions
!   3) Unit normal vector
!   4) Area and volume of each element
!===========================================================================================================
    USE Mod_SharedVars, ONLY: DBL
!===========================================================================================================
    IMPLICIT NONE
!===========================================================================================================
!   Variables
    REAL (KIND = DBL), INTENT(IN)  :: x1, y1, z1, &             !coordinates of first node of the element
    &                                 x2, y2, z2, &             !coordinates of second node of the element
    &                                 x3, y3, z3, &             !coordinates of third node of the element
    &                                 x4, y4, z4, &             !coordinates of fourth node of the element
    &                                 x5, y5, z5, &             !coordinates of fifth node of the element
    &                                 x6, y6, z6                !coordinates of sixth node of the element
    REAL (KIND = DBL), INTENT(IN)  :: al, be, ga                !constants alpha, beta and gamma
    REAL (KIND = DBL), INTENT(IN)  :: xi, eta                   !variables to integrate over a triangle
    REAL (KIND = DBL), INTENT(OUT) :: x, y, z                   !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL), INTENT(OUT) :: DxDxi, DyDxi, DzDxi, &    !Derivates of the tangential vectors over the
    &                                 DxDet, DyDet, DzDet       !element
    REAL (KIND = DBL), INTENT(OUT) :: vnx, vny, vnz             !normal vector of the element
    REAL (KIND = DBL), INTENT(OUT) :: hs !, es, xs                !surface metric
!===========================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: alc,   bec,   gac ,   xs, es, hhs                     !complements of alpha, beta and
    gamma
    REAL (KIND = DBL) :: alalc, bebec, gagac                    !other constants of alpha, beta and gamma
    REAL (KIND = DBL) :: ph1,   ph2,   ph3,  ph4,  ph5,  ph6    !components phi(xi,eta)
    REAL (KIND = DBL) :: dph1,  dph2,  dph3, dph4, dph5, dph6   !derivate of phi(xi,eta) wrt xi
    REAL (KIND = DBL) :: pph1,  pph2,  pph3, pph4, pph5, pph6   !derivate of phi(xi,eta) wrt eta
!-----------------------------------------------------------------------------------------------------------
!   Prepare the constants of this subroutine.
!-----------------------------------------------------------------------------------------------------------
    alc = 1.0D0-al
    bec = 1.0D0-be
    gac = 1.0D0-ga
    alalc = al*alc
    bebec = be*bec
    gagac = ga*gac
!-----------------------------------------------------------------------------------------------------------
!   In this part, the subroutine evaluates basis functions, it obtains the tangential and normal vectors.
!This is based on Numerical Computation in Science and Engineering, C.Pozrikidis, 1998, pp.305-312
!-----------------------------------------------------------------------------------------------------------
    ph2 = xi *(xi -al+eta*(al-ga)/gac)/alc
    ph3 = eta*(eta-be+xi *(be+ga-1.0D0)/ga)/bec
    ph4 = xi *(1.0D0-xi-eta)/alalc
    ph5 = xi*eta          /gagac
    ph6 = eta*(1.0D0-xi-eta)/bebec
    ph1 = 1.0D0-ph2-ph3-ph4-ph5-ph6
!-----------------------------------------------------------------------------------------------------------
!   Interpolate the position vector (x, y, z)
!-----------------------------------------------------------------------------------------------------------
    x = x1*ph1 + x2*ph2 + x3*ph3 + x4*ph4 + x5*ph5 + x6*ph6
    y = y1*ph1 + y2*ph2 + y3*ph3 + y4*ph4 + y5*ph5 + y6*ph6
    z = z1*ph1 + z2*ph2 + z3*ph3 + z4*ph4 + z5*ph5 + z6*ph6
!-----------------------------------------------------------------------------------------------------------
!   In this part suroutine evaluates xi derivatives of basis functions
!-----------------------------------------------------------------------------------------------------------
    dph2 =  (2.0D0*xi-al +eta*(al-ga)/gac)/alc
    dph3 =  eta*(be+ga-1.0D0)/(ga*bec)
    dph4 =  (1.0D0-2.0D0*xi-eta)/alalc
    dph5 =  eta/gagac
    dph6 = -eta/bebec
    dph1 = -dph2-dph3-dph4-dph5-dph6
!-----------------------------------------------------------------------------------------------------------
!   Compute dx/dxi from xi derivatives of phi
```

```fortran
!----------------------------------------------------------------------------------------------------
    DxDxi = x1*dph1 + x2*dph2 + x3*dph3 + x4*dph4 + x5*dph5 + x6*dph6
    DyDxi = y1*dph1 + y2*dph2 + y3*dph3 + y4*dph4 + y5*dph5 + y6*dph6
    DzDxi = z1*dph1 + z2*dph2 + z3*dph3 + z4*dph4 + z5*dph5 + z6*dph6
    xs = DSQRT(DxDxi*DxDxi + DyDxi*DyDxi + DzDxi*DzDxi)
!----------------------------------------------------------------------------------------------------
!   Normalization of normal vector
!----------------------------------------------------------------------------------------------------
    DxDxi = DxDxi
    DyDxi = DyDxi
    DzDxi = DzDxi
!----------------------------------------------------------------------------------------------------!
    ----------------------------------------------------------------------------------------------------
    --
!   In this part suroutine evaluates eta derivatives of basis functions
!----------------------------------------------------------------------------------------------------
    pph2 =  xi*(al-ga)/(alc*gac)
    pph3 =  (2.0D0*eta-be+xi*(be+ga-1.0D0)/ga)/bec
    pph4 =  -xi/alalc
    pph5 =   xi/gagac
    pph6 =  (1.0D0-xi-2.0D0*eta)/bebec
    pph1 = -pph2-pph3-pph4-pph5-pph6
!----------------------------------------------------------------------------------------------------
!   Compute dx/deta from eta derivatives of phi
!----------------------------------------------------------------------------------------------------
    DxDet = x1*pph1 + x2*pph2 + x3*pph3 + x4*pph4 + x5*pph5 + x6*pph6
    DyDet = y1*pph1 + y2*pph2 + y3*pph3 + y4*pph4 + y5*pph5 + y6*pph6
    DzDet = z1*pph1 + z2*pph2 + z3*pph3 + z4*pph4 + z5*pph5 + z6*pph6
    es = DSQRT(DxDet*DxDet + DyDet*DyDet + DzDet*DzDet)
!----------------------------------------------------------------------------------------------------
!   Normalization of normal vector
!----------------------------------------------------------------------------------------------------
    DxDet = DxDet
    DyDet = DyDet
    DzDet = DzDet
!----------------------------------------------------------------------------------------------------
!----------------------------------------------------------------------------------------------------
!   Normal vector vn = (DxDxi)^(DxDeta)
!   Surface metric   hs = norm(vn)
!----------------------------------------------------------------------------------------------------
    vnx = DyDxi * DzDet - DyDet * DzDxi
    vny = DzDxi * DxDet - DzDet * DxDxi
    vnz = DxDxi * DyDet - DxDet * DyDxi
    hs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
!----------------------------------------------------------------------------------------------------
!   Normalization of normal vector
!----------------------------------------------------------------------------------------------------
    vnx = vnx/hs
    vny = vny/hs
    vnz = vnz/hs
    !hs= es*xs
!!----------------------------------------------------------------------------------------------------
!     vnx = (DyDxi/xs) * (DzDet/es) - (DyDet/es) * (DzDxi/xs)
!     vny = (DzDxi/xs) * (DxDet/es) - (DzDet/es) * (DxDxi/xs)
!     vnz = (DxDxi/xs) * (DyDet/es) - (DxDet/es) * (DyDxi/xs)
!     hhs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
!!----------------------------------------------------------------------------------------------------
!!    Normalization of normal vector
!!----------------------------------------------------------------------------------------------------
!     vnx = vnx/hhs
!     vny = vny/hhs
!     vnz = vnz/hhs
    !hs= es*xs
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE interp_p3
!====================================================================================================
    SUBROUTINE interp_p4(   x1,    y1,    z1,  &
    &                      x2,    y2,    z2,  &
    &                      x3,    y3,    z3,  &
    &                      x4,    y4,    z4,  &
    &                      x5,    y5,    z5,  &
    &                      x6,    y6,    z6,  &
```



```fortran
    &                      al,    be,    ga,  &
    &                      xi,   eta,         &
    &                      curv, stride       )
!====================================================================================================
!   This subroutine interpolate over an element to compute geometrical variables including the following:

!   1) Position vector
!   2) Tangential vectors in the xi and eta directions
!   3) Unit normal vector
!   4) Area and volume of each element
!====================================================================================================
    USE Mod_SharedVars, ONLY: DBL
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
    INTEGER,           INTENT(IN)  :: stride                     !stride
    REAL (KIND = DBL), INTENT(IN)  :: x1, y1, z1, &              !coordinates of first node of the element
    &                                 x2, y2, z2, &              !coordinates of second node of the element
    &                                 x3, y3, z3, &              !coordinates of third node of the element
    &                                 x4, y4, z4, &              !coordinates of fourth node of the element
    &                                 x5, y5, z5, &              !coordinates of fifth node of the element
    &                                 x6, y6, z6                 !coordinates of sixth node of the element
    REAL (KIND = DBL), INTENT(IN)  :: al, be, ga                 !constants alpha, beta and gamma
    REAL (KIND = DBL), INTENT(IN)  :: xi, eta                    !variables to integrate over a triangle
    REAL (KIND = DBL), INTENT(OUT) :: curv                       !mean curvature
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: x, y, z                   !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL) :: vnx, vny, vnz             !normal vector of the element
    REAL (KIND = DBL) :: hs                        !surface metric
    REAL (KIND = DBL) :: alc,   bec,   gac                      !complements of alpha, beta and gamma
    REAL (KIND = DBL) :: xs, es, hhs                            !abs value of derivates wrt etha and xi
    REAL (KIND = DBL) :: alalc, bebec, gagac                    !other constants of alpha, beta and gamma
    REAL (KIND = DBL) :: ph1,   ph2,   ph3,  ph4,  ph5,  ph6    !components phi(xi,eta)
    REAL (KIND = DBL) :: dph1,  dph2,  dph3, dph4, dph5, dph6   !derivate of phi(xi,eta) wrt xi
    REAL (KIND = DBL) :: pph1,  pph2,  pph3, pph4, pph5, pph6   !derivate of phi(xi,eta) wrt eta
    REAL (KIND = DBL) :: DxDxi,   DyDxi,   DzDxi                !Derivates wrt xi
    REAL (KIND = DBL) :: DxDet,   DyDet,   DzDet                !Derivates wrt eta
    REAL (KIND = DBL) :: DxDxi2,  DyDxi2,  DzDxi2               !Derivates wrt xi and xi
    REAL (KIND = DBL) :: DxDet2,  DyDet2,  DzDet2               !Derivates wrt eta and eta
    REAL (KIND = DBL) :: DxDxiet, DyDxiet, DzDxiet              !Derivates wrt xi and eta
    REAL (KIND = DBL) :: pph21,  pph22,  pph23, pph24, pph25, pph26   !second derivate of phi(xi,eta) wrt eta
    REAL (KIND = DBL) :: dph21,  dph22,  dph23, dph24, dph25, dph26   !second derivate of phi(xi,eta) wrt xi
    REAL (KIND = DBL) :: pdph1,  pdph2,  pdph3, pdph4, pdph5, pdph6   !second derivate of phi(xi,eta) wrt etaxi
    REAL (KIND = DBL) :: EK1, FK1, GK1                          !constants of first Fundamental form
    REAL (KIND = DBL) :: EK2, FK2, GK2                          !constants of first Fundamental form
    REAL (KIND = DBL) :: EK3, FK3, GK3                          !constants of first Fundamental form
    REAL (KIND = DBL) :: LK1, MK1, NK1                          !constants of second Fundamental form
    REAL (KIND = DBL) :: LK2, MK2, NK2                          !constants of second Fundamental form
    REAL (KIND = DBL) :: LK3, MK3, NK3                          !constants of second Fundamental form
    REAL (KIND = DBL) :: EK, FK, GK                             !constants of first Fundamental form
    REAL (KIND = DBL) :: LK, MK, NK                             !constants of second Fundamental form
    REAL (KIND = DBL) :: strider                             !constants of real of stride
!----------------------------------------------------------------------------------------------------
!   Prepare the constants of this subroutine.
!----------------------------------------------------------------------------------------------------
    alc = 1.0D0-al
    bec = 1.0D0-be
    gac = 1.0D0-ga
    alalc = al*alc
    bebec = be*bec
    gagac = ga*gac
!----------------------------------------------------------------------------------------------------
!   In this part, the subroutine evaluates basis functions, it obtains the tangential and normal vectors.
!This is based on Numerical Computation in Science and Engineering, C.Pozrikidis, 1998, pp.305-312
!----------------------------------------------------------------------------------------------------
    ph2 = xi *(xi - al + eta*(al - ga)/gac)/alc
    ph3 = eta*(eta - be + xi*(be + ga - 1.0D0)/ga)/bec
    ph4 = xi *(1.0D0 - xi - eta)/alalc
    ph5 = xi*eta/gagac
```

```
    ph6 = eta*(1.0D0 - xi - eta)/bebec
    ph1 = 1.0D0-ph2-ph3-ph4-ph5-ph6
!-----------------------------------------------------------------------------------------------------
!   Interpolate the position vector (x, y, z)
!-----------------------------------------------------------------------------------------------------
    x = x1*ph1 + x2*ph2 + x3*ph3 + x4*ph4 + x5*ph5 + x6*ph6
    y = y1*ph1 + y2*ph2 + y3*ph3 + y4*ph4 + y5*ph5 + y6*ph6
    z = z1*ph1 + z2*ph2 + z3*ph3 + z4*ph4 + z5*ph5 + z6*ph6
!-----------------------------------------------------------------------------------------------------
!   In this part suroutine evaluates xi derivatives of basis functions
!-----------------------------------------------------------------------------------------------------
    dph2 =  (2.0D0*xi - al + eta*(al - ga)/gac)/alc
    dph3 =  eta*(be + ga - 1.0D0)/(ga*bec)
    dph4 =  (1.0D0 - 2.0D0*xi - eta)/alalc
    dph5 =  eta/gagac
    dph6 = -eta/bebec
    dph1 = -dph2 - dph3 - dph4 - dph5 - dph6
!-----------------------------------------------------------------------------------------------------
!   Compute dx/dxi from xi derivatives of phi
!-----------------------------------------------------------------------------------------------------
    DxDxi = x1*dph1 + x2*dph2 + x3*dph3 + x4*dph4 + x5*dph5 + x6*dph6
    DyDxi = y1*dph1 + y2*dph2 + y3*dph3 + y4*dph4 + y5*dph5 + y6*dph6
    DzDxi = z1*dph1 + z2*dph2 + z3*dph3 + z4*dph4 + z5*dph5 + z6*dph6
    xs = DSQRT(DxDxi*DxDxi + DyDxi*DyDxi + DzDxi*DzDxi)
!-----------------------------------------------------------------------------------------------------
!   In this part suroutine evaluates eta derivatives of basis functions
!-----------------------------------------------------------------------------------------------------
    pph2 =  xi*(al - ga)/(alc*gac)
    pph3 =  (2.0D0*eta - be + xi*(be + ga - 1.0D0)/ga)/bec
    pph4 =  -xi/alalc
    pph5 =   xi/gagac
    pph6 =  (1.0D0 - xi - 2.0D0*eta)/bebec
    pph1 = - pph2 - pph3 - pph4 - pph5 - pph6
!-----------------------------------------------------------------------------------------------------
!   Compute dx/deta from eta derivatives of phi
!-----------------------------------------------------------------------------------------------------
    DxDet = x1*pph1 + x2*pph2 + x3*pph3 + x4*pph4 + x5*pph5 + x6*pph6
    DyDet = y1*pph1 + y2*pph2 + y3*pph3 + y4*pph4 + y5*pph5 + y6*pph6
    DzDet = z1*pph1 + z2*pph2 + z3*pph3 + z4*pph4 + z5*pph5 + z6*pph6
    es = DSQRT(DxDet*DxDet + DyDet*DyDet + DzDet*DzDet)
!-----------------------------------------------------------------------------------------------------
!   Normal vector vn = (DxDxi)^(DxDeta)
!   Surface metric   hs = norm(vn)
!-----------------------------------------------------------------------------------------------------
    vnx = DyDxi * DzDet - DyDet * DzDxi
    vny = DzDxi * DxDet - DzDet * DxDxi
    vnz = DxDxi * DyDet - DxDet * DyDxi
    hhs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
!-----------------------------------------------------------------------------------------------------
!   Normalization of normal vector
!-----------------------------------------------------------------------------------------------------
    vnx = vnx/hhs
    vny = vny/hhs
    vnz = vnz/hhs
    hs= hhs
!-----------------------------------------------------------------------------------------------------
!   In this part, the subroutine computes the mean curvature using the analisys of Stoker Chapter IV.
!First the vectors E, F and G are computed to estimate the First Fundamental Form
!-----------------------------------------------------------------------------------------------------
    EK1 = DxDxi**2
    EK2 = DyDxi**2
    EK3 = DzDxi**2
!-----------------------------------------------------------------------------------------------------
    FK1 = DxDxi*DxDet
    FK2 = DyDxi*DyDet
    FK3 = DzDxi*DzDet
!-----------------------------------------------------------------------------------------------------
    GK1 = DxDet**2
    GK2 = DyDet**2
    GK3 = DzDet**2
!-----------------------------------------------------------------------------------------------------
!   Evaluation of E, F, and G
```



```
!-----------------------------------------------------------------------------------------------------
    EK  = EK1 + EK2 + EK3
    FK  = FK1 + FK2 + FK3
    GK  = GK1 + GK2 + GK3
!-----------------------------------------------------------------------------------------------------
!   Second derivatives of the element.
!In this part suroutine evaluates the second derivatives wrt xi
!-----------------------------------------------------------------------------------------------------
    dph22 =  (2.0D0)/alc
    dph23 =  0.0D0
    dph24 =  -2.0D0/alalc
    dph25 =  0.0D0
    dph26 =  0.0D0
    dph21 = -dph22 - dph23 - dph24 - dph25 - dph26
!-----------------------------------------------------------------------------------------------------
!   Compute dx/dxi**2 of phi
!-----------------------------------------------------------------------------------------------------
    DxDxi2 = x1*dph21 + x2*dph22 + x3*dph23 + x4*dph24 + x5*dph25 + x6*dph26

    DyDxi2 = y1*dph21 + y2*dph22 + y3*dph23 + y4*dph24 + y5*dph25 + y6*dph26

    DzDxi2 = z1*dph21 + z2*dph22 + z3*dph23 + z4*dph24 + z5*dph25 + z6*dph26

!-----------------------------------------------------------------------------------------------------
!In this part suroutine evaluates the second derivatives wrt xi then wrt eta
!-----------------------------------------------------------------------------------------------------
    pdph2 =  ((al - ga)/gac*alc)
    pdph3 =  (be + ga - 1.0D0)/(ga*bec)
    pdph4 =  (-1.0D0)/alalc
    pdph5 =  1.0D0/gagac
    pdph6 = -1.0D0/bebec
    pdph1 = -pdph2 - pdph3 - pdph4 - pdph5 - pdph6
!-----------------------------------------------------------------------------------------------------
!   Compute dx/dxidet of phi
!-----------------------------------------------------------------------------------------------------
    DxDxiet = x1*pdph1 + x2*pdph2 + x3*pdph3 + x4*pdph4 + x5*pdph5 + x6*pdph6

    DyDxiet = y1*pdph1 + y2*pdph2 + y3*pdph3 + y4*pdph4 + y5*pdph5 + y6*pdph6

    DzDxiet = z1*pdph1 + z2*pdph2 + z3*pdph3 + z4*pdph4 + z5*pdph5 + z6*pdph6

!-----------------------------------------------------------------------------------------------------
!   In this part suroutine evaluates eta second derivatives of phi
!-----------------------------------------------------------------------------------------------------
    pph22 =  0.0D0
    pph23 =  (2.0D0)/bec
    pph24 =  0.0D0
    pph25 =  0.0D0
    pph26 =  (-2.0D0)/bebec
    pph21 = - pph22 - pph23 - pph24 - pph25 - pph26
!-----------------------------------------------------------------------------------------------------
!   Compute dx/det**2 from eta derivatives of phi
!-----------------------------------------------------------------------------------------------------
    DxDet2 = x1*pph21 + x2*pph22 + x3*pph23 + x4*pph24 + x5*pph25 + x6*pph26

    DyDet2 = y1*pph21 + y2*pph22 + y3*pph23 + y4*pph24 + y5*pph25 + y6*pph26

    DzDet2 = z1*pph21 + z2*pph22 + z3*pph23 + z4*pph24 + z5*pph25 + z6*pph26

!-----------------------------------------------------------------------------------------------------
!   In this part, the subroutine computes the mean curvature using the analisys of Stoker Chapter IV.
!First the vectors L, M and N are computed to estimate the Second Fundamental Form
!-----------------------------------------------------------------------------------------------------
    LK1 = DxDxi2*vnx / DSQRT(EK*GK - FK**2)
    LK2 = DyDxi2*vny / DSQRT(EK*GK - FK**2)
    LK3 = DzDxi2*vnz / DSQRT(EK*GK - FK**2)
!-----------------------------------------------------------------------------------------------------
    MK1 = DxDxiet*vnx / DSQRT(EK*GK - FK**2)
    MK2 = DyDxiet*vny / DSQRT(EK*GK - FK**2)
    MK3 = DzDxiet*vnz / DSQRT(EK*GK - FK**2)
!-----------------------------------------------------------------------------------------------------
    NK1 = DxDet2*vnx / DSQRT(EK*GK - FK**2)
```

```fortran
    NK2 = DyDet2*vny / DSQRT(EK*GK - FK**2)
    NK3 = DzDet2*vnz / DSQRT(EK*GK - FK**2)
!-----------------------------------------------------------------------------------------------------------
!   Evaluation of L, M and N
!-----------------------------------------------------------------------------------------------------------
    LK  = LK1 + LK2 + LK3
    MK  = MK1 + MK2 + MK3
    NK  = NK1 + NK2 + NK3
!-----------------------------------------------------------------------------------------------------------
!   Mean Curvature is obtained
!-----------------------------------------------------------------------------------------------------------
    curv = (0.5D0*((EK*NK + GK*LK - 2.0D0*FK*MK) / (EK*GK - FK**2))) !+(0.001*((LK*NK - MK*MK) / (EK*GK - FK**
    2)))
!    curv = DSIGN(0.5D0*((EK*NK + GK*LK - 2.0D0*FK*MK) / (EK*GK - FK**2)),((LK*NK - MK*MK) / (EK*GK - FK**2)))
    strider = REAL(stride)
    curv = ((-1.0D0))*curv* hs
!-----------------------------------------------------------------------------------------------------------
    END SUBROUTINE interp_p4
!===========================================================================================================
!   Subroutine INTERP_E is a new method to obtain the normal vectors on every node on dorp's surface. The
!name has the end _E which is an abreviation of element. Firstly, this decision aids to differenciate this
!method of subroutine INTERP_P. Second idea was to continue with this notation or manner to name subroutines to
!have similar enviroment names. Finaly INTERP_E is very importan becouse this subroutine is fundamental to
!advanced the dinamic of dorp deformation in the time. Therefore, this subroutine is in this new MODULE
!===========================================================================================================
END MODULE Mod_Nodal_Interp
```

```fortran

    MODULE Mod_Gauss_Coefs
!====================================================================================================
!   Version: 0.5 created on  26 / IX / 2007
!                                                                                        C. Pozrikidis
!----------------------------------------------------------------------------------------------------
!   Version: 0.7 created on  -- / III /2010
!                                                                              Marco Antonio Reyes Huesca
!----------------------------------------------------------------------------------------------------
!   Version: 0.9 created on 21 / 08 / 2012
!   Version: 1.0 created on 14 / 11 / 2012
!                                                                                  Alfredo Sanjuan Sanjuan
!====================================================================================================
    CONTAINS
!====================================================================================================
    SUBROUTINE Gauss_Legendre(NGL)
!====================================================================================================
!   This Subroutine obtains abscissas and weights for the Gauss-LegENDre quadrature with NGL points.
!The value NGL or NGL as this code is the number of Gauss-Legendre base points for singular integrals.
! SYMBOLS:
!   NGL = 1,2,3,4,5,6,8,12 and 20
!Default value is 20
!====================================================================================================
!====================================================================================================
    USE Mod_SharedVars, ONLY: DBL, UDat, ULog, ZZ, WW
!====================================================================================================
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!====================================================================================================
!   Variables
    INTEGER,   INTENT(INOUT)   :: NGL               !order of Gauss-Legendre quadrature
!----------------------------------------------------------------------------------------------------
!   The next step is use a Case statement to obtain the coeficientes.
    SELECT CASE(NGL)
!----------------------------------------------------------------------------------------------------
    CASE (1)
        ALLOCATE(ZZ(NGL), WW(NGL))
        ZZ = 0.0D0
        WW = 0.0D0

        ZZ(1) = 0.0D0
        WW(1) = 2.0D0
!----------------------------------------------------------------------------------------------------
    CASE (2)
        ALLOCATE(ZZ(NGL), WW(NGL))
        ZZ = 0.0D0
        WW = 0.0D0
!----------------------------------------------------------------------------------------------------
        ZZ(1) = -0.57735026918962576450D0
        ZZ(2) = -ZZ(1)

        WW(1) = 1.0D0
        WW(2) = 1.0D0
!----------------------------------------------------------------------------------------------------
    CASE (3)
        ALLOCATE(ZZ(NGL), WW(NGL))
        ZZ = 0.0D0
        WW = 0.0D0
!----------------------------------------------------------------------------------------------------
        ZZ(1) = -0.77459666924148337703D0
        ZZ(2) =  0.0D0
        ZZ(3) = -ZZ(1)
!----------------------------------------------------------------------------------------------------
        WW(1) = 0.55555555555555555556D0
        WW(2) = 0.88888888888888888889D0
        WW(3) = 0.55555555555555555556D0
!----------------------------------------------------------------------------------------------------
    CASE (4)
        ALLOCATE(ZZ(NGL), WW(NGL))
        ZZ = 0.0D0
        WW = 0.0D0
```



```fortran
!----------------------------------------------------------------------------------------------------
        ZZ(1) = -0.86113631159405257522D0
        ZZ(2) = -0.33998104358485626480D0
        ZZ(3) = -ZZ(2)
        ZZ(4) = -ZZ(1)
!----------------------------------------------------------------------------------------------------
        WW(1) = 0.34785484513745385737D0
        WW(2) = 0.65214515486254614262D0
        WW(3) = WW(2)
        WW(4) = WW(1)
!----------------------------------------------------------------------------------------------------
    CASE (5)
        ALLOCATE(ZZ(NGL), WW(NGL))
        ZZ = 0.0D0
        WW = 0.0D0
!----------------------------------------------------------------------------------------------------
        ZZ(1) = -0.90617984593866399279D0
        ZZ(2) = -0.53846931010568309103D0
        ZZ(3) =  0.0D0
        ZZ(4) = -ZZ(2)
        ZZ(5) = -ZZ(1)
!----------------------------------------------------------------------------------------------------
        WW(1) = 0.23692688505618908751D0
        WW(2) = 0.47862867049936646804D0
        WW(3) = 0.56888888888888888889D0
        WW(4) = WW(2)
        WW(5) = WW(1)
!----------------------------------------------------------------------------------------------------
    CASE (6)
        ALLOCATE(ZZ(NGL), WW(NGL))
        ZZ = 0.0D0
        WW = 0.0D0
!----------------------------------------------------------------------------------------------------
        ZZ(1) = -0.932469514203152D0
        ZZ(2) = -0.661209386466265D0
        ZZ(3) = -0.238619186083197D0
        ZZ(4) = -ZZ(3)
        ZZ(5) = -ZZ(2)
        ZZ(6) = -ZZ(1)
!----------------------------------------------------------------------------------------------------
        WW(1) = 0.171324492379170D0
        WW(2) = 0.360761573048139D0
        WW(3) = 0.467913934572691D0
        WW(4) = WW(3)
        WW(5) = WW(2)
        WW(6) = WW(1)
!----------------------------------------------------------------------------------------------------
    CASE (8)
        ALLOCATE(ZZ(NGL), WW(NGL))
        ZZ = 0.0D0
        WW = 0.0D0
!----------------------------------------------------------------------------------------------------
        ZZ(1) = -0.96028985649753623168D0
        ZZ(2) = -0.79666647741362673959D0
        ZZ(3) = -0.52553240991632898581D0
        ZZ(4) = -0.18343464249564980493D0
        ZZ(5) = -ZZ(4)
        ZZ(6) = -ZZ(3)
        ZZ(7) = -ZZ(2)
        ZZ(8) = -ZZ(1)
!----------------------------------------------------------------------------------------------------
        WW(1) = 0.10122853629037625915D0
        WW(2) = 0.22238103445337447054D0
        WW(3) = 0.31370664587788728733D0
        WW(4) = 0.36268378337836198296D0
        WW(5) = WW(4)
        WW(6) = WW(3)
        WW(7) = WW(2)
        WW(8) = WW(1)
!----------------------------------------------------------------------------------------------------
    CASE (12)
        ALLOCATE(ZZ(NGL), WW(NGL))
```

```
        ZZ = 0.0D0
        WW = 0.0D0
!-----------------------------------------------------------------------------------------------------------
        ZZ(1) = -0.981560634246719D0
        ZZ(2) = -0.904117256370475D0
        ZZ(3) = -0.769902674194305D0
        ZZ(4) = -0.587317954286617D0
        ZZ(5) = -0.367831498998180D0
        ZZ(6) = -0.125233408511469D0
        ZZ(7) = -ZZ(6)
        ZZ(8) = -ZZ(5)
        ZZ(9) = -ZZ(4)
        ZZ(10)= -ZZ(3)
        ZZ(11)= -ZZ(2)
        ZZ(12)= -ZZ(1)
!-----------------------------------------------------------------------------------------------------------
        WW(1) = 0.047175336386512D0
        WW(2) = 0.106939325995318D0
        WW(3) = 0.160078328543346D0
        WW(4) = 0.203167426723066D0
        WW(5) = 0.233492536538355D0
        WW(6) = 0.249147045813403D0
        WW(7) = WW(6)
        WW(8) = WW(5)
        WW(9) = WW(4)
        WW(10)= WW(3)
        WW(11)= WW(2)
        WW(12)= WW(1)
!-----------------------------------------------------------------------------------------------------------
    CASE (20)
        ALLOCATE(ZZ(NGL), WW(NGL))
        ZZ = 0.0D0
        WW = 0.0D0
!-----------------------------------------------------------------------------------------------------------
        ZZ(1) = -0.993128599185094924786D0
        ZZ(2) = -0.963971927277913791268D0
        ZZ(3) = -0.912234428251325905868D0
        ZZ(4) = -0.839116971822218823395D0
        ZZ(5) = -0.746331906460150792614D0
        ZZ(6) = -0.636053680726515025453D0
        ZZ(7) = -0.510867001950827098004D0
        ZZ(8) = -0.373706088715419560673D0
        ZZ(9) = -0.227785851141645078080D0
        ZZ(10)= -0.076526521133497333755D0
        ZZ(11) = -ZZ(10)
        ZZ(12) = -ZZ(9)
        ZZ(13) = -ZZ(8)
        ZZ(14) = -ZZ(7)
        ZZ(15) = -ZZ(6)
        ZZ(16) = -ZZ(5)
        ZZ(17) = -ZZ(4)
        ZZ(18) = -ZZ(3)
        ZZ(19) = -ZZ(2)
        ZZ(20) = -ZZ(1)
!-----------------------------------------------------------------------------------------------------------
        WW(1) = 0.017614007139152118312D0
        WW(2) = 0.040601429800386941331D0
        WW(3) = 0.062672048334109063570D0
        WW(4) = 0.083276741576704748725D0
        WW(5) = 0.101930119817240435037D0
        WW(6) = 0.118194531961518417312D0
        WW(7) = 0.131688638449176626898D0
        WW(8) = 0.142096109318382051329D0
        WW(9) = 0.149172986472603746788D0
        WW(10)= 0.152753387130725850698D0
        WW(11) = WW(10)
        WW(12) = WW(9)
        WW(13) = WW(8)
        WW(14) = WW(7)
        WW(15) = WW(6)
        WW(16) = WW(5)
        WW(17) = WW(4)
```



```
        WW(18) = WW(3)
        WW(19) = WW(2)
        WW(20) = WW(1)
!-----------------------------------------------------------------------------------------------------------
    CASE DEFAULT
        NGL = 20
        ALLOCATE(ZZ(NGL), WW(NGL))
        ZZ = 0.0D0
        WW = 0.0D0
!-----------------------------------------------------------------------------------------------------------
        ZZ(1) = -0.993128599185094924786D0
        ZZ(2) = -0.963971927277913791268D0
        ZZ(3) = -0.912234428251325905868D0
        ZZ(4) = -0.839116971822218823395D0
        ZZ(5) = -0.746331906460150792614D0
        ZZ(6) = -0.636053680726515025453D0
        ZZ(7) = -0.510867001950827098004D0
        ZZ(8) = -0.373706088715419560673D0
        ZZ(9) = -0.227785851141645078080D0
        ZZ(10)= -0.076526521133497333755D0
        ZZ(11) = -ZZ(10)
        ZZ(12) = -ZZ(9)
        ZZ(13) = -ZZ(8)
        ZZ(14) = -ZZ(7)
        ZZ(15) = -ZZ(6)
        ZZ(16) = -ZZ(5)
        ZZ(17) = -ZZ(4)
        ZZ(18) = -ZZ(3)
        ZZ(19) = -ZZ(2)
        ZZ(20) = -ZZ(1)
        WW(1) = 0.017614007139152118312D0
        WW(2) = 0.040601429800386941331D0
        WW(3) = 0.062672048334109063570D0
        WW(4) = 0.083276741576704748725D0
        WW(5) = 0.101930119817240435037D0
        WW(6) = 0.118194531961518417312D0
        WW(7) = 0.131688638449176626898D0
        WW(8) = 0.142096109318382051329D0
        WW(9) = 0.149172986472603746788D0
        WW(10)= 0.152753387130725850698D0
        WW(11) = WW(10)
        WW(12) = WW(9)
        WW(13) = WW(8)
        WW(14) = WW(7)
        WW(15) = WW(6)
        WW(16) = WW(5)
        WW(17) = WW(4)
        WW(18) = WW(3)
        WW(19) = WW(2)
        WW(20) = WW(1)
!-----------------------------------------------------------------------------------------------------------
        WRITE (ULog,*)
        WRITE (ULog,*) ' Gauss_LegENDre:'
        WRITE (ULog,*)
        WRITE (ULog,*) '   Chosen number of Gaussian points'
        WRITE (ULog,*) '   is not available; It was taken NGL=20'
    END SELECT
    END SUBROUTINE Gauss_Legendre
!===========================================================================================================
!===========================================================================================================
    SUBROUTINE Gauss_Trgl(mint)
!===========================================================================================================
!   This Subroutine obtains abscissas and weights for Gaussian integration over a triangle.
! Integration is performed with respect to the triangle barycentric coordinates
! SYMBOLS:
!   mint: order of the quadrature choose from 1,3,4,5,7,9,12,13
!Default value is 13
!===========================================================================================================
    USE Mod_SharedVars, ONLY: DBL, ULog, xiq, etq, wq
!===========================================================================================================
    IMPLICIT NONE
!===========================================================================================================
```

```fortran
!   Variables
!----------------------------------------------------------------------------------------------------------------
    INTEGER,   INTENT(INOUT)   :: mint          ! order of the triangle quadrature
!================================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: al, be, ga, de         !constants with the weith values
    REAL (KIND = DBL) :: rh, qa, ru
    REAL (KIND = DBL) :: o1, o2, o3, o4
!================================================================================================================
!   The next step is use a Case statement to obtain the coeficientes.
    SELECT CASE (mint)
    CASE(1)
        ALLOCATE(xiq(mint), etq(mint), wq(mint))
        xiq = 0.0D0
        etq = 0.0D0
        wq  = 0.0D0
!----------------------------------------------------------------------------------------------------------------
        xiq(1) = 1.0D0/3.0D0
        etq(1) = 1.0D0/3.0D0
        wq(1) = 1.0D0
!----------------------------------------------------------------------------------------------------------------
    CASE(3)
        ALLOCATE(xiq(mint), etq(mint), wq(mint))
        xiq = 0.0D0
        etq = 0.0D0
        wq  = 0.0D0
!----------------------------------------------------------------------------------------------------------------
        xiq(1) = 1.0D0/6.0D0
        xiq(2) = 2.0D0/3.0D0
        xiq(3) = 1.0D0/6.0D0
        etq(1) = 1.0D0/6.0D0
        etq(2) = 1.0D0/6.0D0
        etq(3) = 2.0D0/3.0D0
        wq(1) = 1.0D0/3.0D0
        wq(2) = wq(1)
        wq(3) = wq(1)
!----------------------------------------------------------------------------------------------------------------
    CASE(4)
        ALLOCATE(xiq(mint), etq(mint), wq(mint))
        xiq = 0.0D0
        etq = 0.0D0
        wq  = 0.0D0
!----------------------------------------------------------------------------------------------------------------
        xiq(1) =   1.0D0/3.0D0
        xiq(2) =  1.0D0/5.0D0
        xiq(3) =  3.0D0/5.0D0
        xiq(4) =  1.0D0/5.0D0
        etq(1) =   1.0D0/3.0D0
        etq(2) =  1.0D0/5.0D0
        etq(3) =  1.0D0/5.0D0
        etq(4) =  3.0D0/5.0D0
        wq(1) = -27.0D0/48.0D0
        wq(2) = 25.0D0/48.0D0
        wq(3) = 25.0D0/48.0D0
        wq(4) = 25.0D0/48.0D0
!----------------------------------------------------------------------------------------------------------------
    CASE(6)
        ALLOCATE(xiq(mint), etq(mint), wq(mint))
        xiq = 0.0D0
        etq = 0.0D0
        wq  = 0.0D0
!----------------------------------------------------------------------------------------------------------------
        al = 0.816847572980459D0
        be = 0.445948490915965D0
        ga = 0.108103018168070D0
        de = 0.091576213509771D0
        o1 = 0.109951743655322D0
        o2 = 0.223381589678011D0
!----------------------------------------------------------------------------------------------------------------
        xiq(1) = de
        xiq(2) = al
```



```fortran
        xiq(3) = de
        xiq(4) = be
        xiq(5) = ga
        xiq(6) = be
        etq(1) = de
        etq(2) = de
        etq(3) = al
        etq(4) = be
        etq(5) = be
        etq(6) = ga
        wq(1) = o1
        wq(2) = o1
        wq(3) = o1
        wq(4) = o2
        wq(5) = o2
        wq(6) = o2
!----------------------------------------------------------------------------------------------------------------
    CASE(7)
        ALLOCATE(xiq(mint), etq(mint), wq(mint))
        xiq = 0.0D0
        etq = 0.0D0
        wq  = 0.0D0
!----------------------------------------------------------------------------------------------------------------
        al = 0.797426985353087D0
        be = 0.470142064105115D0
        ga = 0.059715871789770D0
        de = 0.101286507323456D0
        o1 = 0.125939180544827D0
        o2 = 0.132394152788506D0
!----------------------------------------------------------------------------------------------------------------
        xiq(1) = de
        xiq(2) = al
        xiq(3) = de
        xiq(4) = be
        xiq(5) = ga
        xiq(6) = be
        xiq(7) = 1.0D0/3.0D0
        etq(1) = de
        etq(2) = de
        etq(3) = al
        etq(4) = be
        etq(5) = be
        etq(6) = ga
        etq(7) = 1.0D0/3.0D0
        wq(1) = o1
        wq(2) = o1
        wq(3) = o1
        wq(4) = o2
        wq(5) = o2
        wq(6) = o2
        wq(7) = 0.225D0
!----------------------------------------------------------------------------------------------------------------
    CASE(9)
        ALLOCATE(xiq(mint), etq(mint), wq(mint))
        xiq = 0.0D0
        etq = 0.0D0
        wq  = 0.0D0
!----------------------------------------------------------------------------------------------------------------
        al = 0.124949503233232D0
        qa = 0.165409927389841D0
        rh = 0.797112651860071D0
        de = 0.437525248383384D0
        ru = 0.037477420750088D0
        o1 = 0.205950504760887D0
        o2 = 0.063691414286223D0
!----------------------------------------------------------------------------------------------------------------
        xiq(1) = de
        xiq(2) = al
        xiq(3) = de
        xiq(4) = qa
        xiq(5) = ru
        xiq(6) = rh
```

```
        xiq(7) = qa
        xiq(8) = ru
        xiq(9) = rh
        etq(1) = de
        etq(2) = de
        etq(3) = al
        etq(4) = ru
        etq(5) = qa
        etq(6) = qa
        etq(7) = rh
        etq(8) = rh
        etq(9) = ru
        wq(1) = o1
        wq(2) = o1
        wq(3) = o1
        wq(4) = o2
        wq(5) = o2
        wq(6) = o2
        wq(7) = o2
        wq(8) = o2
        wq(9) = o2
!-----------------------------------------------------------------------------------------------------------------
    CASE(12)
        ALLOCATE(xiq(mint), etq(mint), wq(mint))
        xiq = 0.0D0
        etq = 0.0D0
        wq  = 0.0D0
!-----------------------------------------------------------------------------------------------------------------
        al = 0.873821971016996D0
        be = 0.249286745170910D0
        ga = 0.501426509658179D0
        de = 0.063089014491502D0
        rh = 0.636502499121399D0
        qa = 0.310352451033785D0
        ru = 0.053145049844816D0
        o1 = 0.050844906370207D0
        o2 = 0.116786275726379D0
        o3 = 0.082851075618374D0
!-----------------------------------------------------------------------------------------------------------------
        xiq(1)  = de
        xiq(2)  = al
        xiq(3)  = de
        xiq(4)  = be
        xiq(5)  = ga
        xiq(6)  = be
        xiq(7)  = qa
        xiq(8)  = ru
        xiq(9)  = rh
        xiq(10) = qa
        xiq(11) = ru
        xiq(12) = rh
!-----------------------------------------------------------------------------------------------------------------
        etq(1)  = de
        etq(2)  = de
        etq(3)  = al
        etq(4)  = be
        etq(5)  = be
        etq(6)  = ga
        etq(7)  = ru
        etq(8)  = qa
        etq(9)  = qa
        etq(10) = rh
        etq(11) = rh
        etq(12) = ru
!-----------------------------------------------------------------------------------------------------------------
        wq(1)  = o1
        wq(2)  = o1
        wq(3)  = o1
        wq(4)  = o2
        wq(5)  = o2
        wq(6)  = o2
        wq(7)  = o3
```



```
        wq(8)  = o3
        wq(9)  = o3
        wq(10) = o3
        wq(11) = o3
        wq(12) = o3
!-----------------------------------------------------------------------------------------------------------------
    CASE(13)
        ALLOCATE(xiq(mint), etq(mint), wq(mint))
        xiq = 0.0D0
        etq = 0.0D0
        wq  = 0.0D0
!-----------------------------------------------------------------------------------------------------------------
        al = 0.479308067841923D0
        be = 0.065130102902216D0
        ga = 0.869739794195568D0
        de = 0.260345966079038D0
        rh = 0.638444188569809D0
        qa = 0.312865496004875D0
        ru = 0.048690315425316D0
        o1 = 0.175615257433204D0
        o2 = 0.053347235608839D0
        o3 = 0.077113760890257D0
        o4 =-0.149570044467670D0
!-----------------------------------------------------------------------------------------------------------------
        xiq(1)  = de
        xiq(2)  = al
        xiq(3)  = de
        xiq(4)  = be
        xiq(5)  = ga
        xiq(6)  = be
        xiq(7)  = qa
        xiq(8)  = ru
        xiq(9)  = rh
        xiq(10) = qa
        xiq(11) = ru
        xiq(12) = rh
        xiq(13) = 1.0D0/3.0D0
!-----------------------------------------------------------------------------------------------------------------
        etq(1)  = de
        etq(2)  = de
        etq(3)  = al
        etq(4)  = be
        etq(5)  = be
        etq(6)  = ga
        etq(7)  = ru
        etq(8)  = qa
        etq(9)  = qa
        etq(10) = rh
        etq(11) = rh
        etq(12) = ru
        etq(13) = 1.0D0/3.0D0
!-----------------------------------------------------------------------------------------------------------------
        wq(1)  = o1
        wq(2)  = o1
        wq(3)  = o1
        wq(4)  = o2
        wq(5)  = o2
        wq(6)  = o2
        wq(7)  = o3
        wq(8)  = o3
        wq(9)  = o3
        wq(10) = o3
        wq(11) = o3
        wq(12) = o3
        wq(13) = o4
!-----------------------------------------------------------------------------------------------------------------
    CASE DEFAULT
        mint = 13
        ALLOCATE(xiq(mint), etq(mint), wq(mint))
        xiq = 0.0D0
        etq = 0.0D0
        wq  = 0.0D0
```

```
!---------------------------------------------------------------------------------------------------------------
        al = 0.479308067841923D0
        be = 0.065130102902216D0
        ga = 0.869739794195568D0
        de = 0.260345966079038D0
        rh = 0.638444188569809D0
        qa = 0.312865496004875D0
        ru = 0.048690315425316D0
        o1 = 0.175615257433204D0
        o2 = 0.053347235608839D0
        o3 = 0.077113760890257D0
        o4 =-0.149570044467670D0
        xiq(1)  = de
        xiq(2)  = al
        xiq(3)  = de
        xiq(4)  = be
        xiq(5)  = ga
        xiq(6)  = be
        xiq(7)  = qa
        xiq(8)  = ru
        xiq(9)  = rh
        xiq(10) = qa
        xiq(11) = ru
        xiq(12) = rh
        xiq(13) = 1.0D0/3.0D0
!---------------------------------------------------------------------------------------------------------------
        etq(1)  = de
        etq(2)  = de
        etq(3)  = al
        etq(4)  = be
        etq(5)  = be
        etq(6)  = ga
        etq(7)  = ru
        etq(8)  = qa
        etq(9)  = qa
        etq(10) = rh
        etq(11) = rh
        etq(12) = ru
        etq(13) = 1.0D0/3.0D0
!---------------------------------------------------------------------------------------------------------------
        wq(1)  = o1
        wq(2)  = o1
        wq(3)  = o1
        wq(4)  = o2
        wq(5)  = o2
        wq(6)  = o2
        wq(7)  = o3
        wq(8)  = o3
        wq(9)  = o3
        wq(10) = o3
        wq(11) = o3
        wq(12) = o3
        wq(13) = o4
!---------------------------------------------------------------------------------------------------------------
        WRITE (ULog,*)
        WRITE (ULog,*) ' Gauss_Trgl:'
        WRITE (ULog,*)
        WRITE (ULog,*) '   Number of Gauss triangle quadrature'
        WRITE (ULog,*) '   is not available; It was taken mint  =13'
    END SELECT
   END SUBROUTINE Gauss_Trgl
!===============================================================================================================
END MODULE Mod_Gauss_Coefs
```

```
MODULE Mod_Builder_Matrix_Arrays
!=====================================================================================================
!Version 1.0            07 / November / 2012
!                                                                  Alfredo Sanjuan Sanjuan, in others words, me :)
!=====================================================================================================
!   The first action is tho generate the matrix form of the single and doble integral potential. To obtain this
!the program call the necesary subroutines. Then the code buid the matrix sistem, and finally the sistem is
!solved with the module Mod_Solution_Ax_b.
!-----------------------------------------------------------------------------------------------------
!   three rows at a time,
! corresponding to the x, y, z
! components of the integral equation
!-----------------------------------------------------------------------------------------------------

CONTAINS
!=====================================================================================================
    SUBROUTINE Builder_GM(GM, cdg, nelm, gamma, mint, NGL)
!=====================================================================================================
!   This subroutine builds the matrix array from the Stokelets GE(x,x0).
!=====================================================================================================
    USE Mod_SharedVars
    USE Mod_Prtcl3D_SLP
    USE Mod_sgf_3d_fs
    USE Mod_Prtcl3D_DLP
    USE Mod_sgf_3d_sfs
    USE mkl95_precision
    USE omp_lib
!=====================================================================================================
    IMPLICIT NONE
!=====================================================================================================
!   Variables
!-----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: GM     !The matrix with the information of
  !Stokeslets on the surface
    REAL (KIND = DBL), INTENT(IN)  :: cdg                   !coefficient of SLP
    INTEGER,           INTENT(IN)  :: nelm                  !number of elements
    REAL (KIND = DBL), INTENT(IN)  :: gamma
    INTEGER,           INTENT(IN)  :: mint, NGL             !order of triangle and Gauss-Legendre quadratures
!=====================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------
    INTEGER ::   i, j                                       !Counters
    INTEGER ::   inelm, inelm2
!-----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL)  :: GEx, GEy, GEz                  !Integrated ij component over the element
!-----------------------------------------------------------------------------------------------------
!The loop over collocations points serves to build the matrix form sistem
    GM =0.0D0
!!!$omp parallel do
!!        Upper: DO i=1,nelm-1
!!            inelm = i + nelm
!!            inelm2= i + nelm + nelm
!!!----------------------------------------------------------------------------------------------------
!!            DO j = i+1, nelm
!!!----------------------------------------------------------------------------------------------------
!!!   The values of single layer potential are obtained in normal kind.
!!                CALL Intgr_Trgl(x0(i),  y0(i),  z0(i),  &
!!                            &   x0(j),  y0(j),  z0(j),  &
!!                            &   j,                      &
!!                            &   GEx,   GEy,   GEz,   &
!!                            &   mint                    )
!!!----------------------------------------------------------------------------------------------------
!!!----------------------------------------------------------------------------------------------------
!!            !CALL intr_lin_sing4(x0(i),   y0(i),   z0(i),    &
!!            !                &  j,                &
!!            !                &  GEx, GEy, GEz)
!!!----------------------------------------------------------------------------------------------------
!!                GM(i)            =  (GM(i)     + GEx)
!!                GM(inelm)        =  (GM(inelm) + GEy)
!!                GM(inelm2)       =  (GM(inelm2)+ GEz)
!!!----------------------------------------------------------------------------------------------------
!!            END DO
```



```
!!!----------------------------------------------------------------------------------------------------
!!        END DO Upper
!!!$omp end parallel
!!!----------------------------------------------------------------------------------------------------
!!!$omp parallel do
!!        Lower: DO i = 2, nelm
!!            inelm = i + nelm
!!            inelm2= i + nelm + nelm
!!            DO j = 1, i-1
!!!----------------------------------------------------------------------------------------------------
!!!   The values of single layer potential are obtained in normal kind.
!!                CALL Intgr_Trgl(x0(i),  y0(i),  z0(i),  &
!!                            &   x0(j),  y0(j),  z0(j),  &
!!                            &   j,                      &
!!                            &   GEx,   GEy,   GEz,   &
!!                            &   mint                    )
!!!----------------------------------------------------------------------------------------------------
!!!----------------------------------------------------------------------------------------------------
!!            !CALL intr_lin_sing4(x0(i),   y0(i),   z0(i),    &
!!            !                &  j,                &
!!            !                &  GEx, GEy, GEz)
!!!----------------------------------------------------------------------------------------------------
!!!   The code will made the algebraic assembly of the matrix
!!                GM(i)            =  (GM(i)+GEx)
!!                GM(inelm)        =  (GM(inelm)+GEy)
!!                GM(inelm2)       =  (GM(inelm2)+GEz)
!!!----------------------------------------------------------------------------------------------------
!!            END DO
!!        END DO Lower
!!!$omp end parallel
!!!----------------------------------------------------------------------------------------------------
!!!   The values of single layer potential are obtained with an interpolation.
!!!$omp parallel do
!!        DO i=1,nelm
!!            inelm = i + nelm
!!            inelm2= i + nelm + nelm
!!!----------------------------------------------------------------------------------------------------
!!            CALL Intgr_Trgl_Sing(x0(i),   y0(i),   z0(i),   &
!!                            &  i,                          &
!!                            &  GEx, GEy, GEz,              &
!!                            &  mint                         )
!!!!---------------------------------------------------------------------------------------------------
  -
!!            !CALL intr_lin_sing4(x0(i),   y0(i),   z0(i),    &
!!            !                &  i,                &
!!            !                &  GEx, GEy, GEz)
!!!----------------------------------------------------------------------------------------------------
!!!   The condition is over ande then the code will made the algebraic assembly of the matrix
!!                GM(i)            =  (GM(i)+GEx)
!!                GM(inelm)        =  (GM(inelm)+GEy)
!!                GM(inelm2)       =  (GM(inelm2)+GEz)
!!!----------------------------------------------------------------------------------------------------
!!        END DO
!!!$omp end parallel
!!!----------------------------------------------------------------------------------------------------
!!!   Add the constant cdg to GM

!$omp parallel
!$omp do private(i, inelm, inelm2, j, GEx, GEY, GEz)
        DO i=1, nelm
            inelm = i + nelm
            inelm2= i + nelm + nelm
!----------------------------------------------------------------------------------------------------
            DO j = 1, nelm
                IF (i==j) THEN
!----------------------------------------------------------------------------------------------------
                   CALL Intgr_Trgl_Sing(x0(i),   y0(i),   z0(i),   &
                            &  i,                          &
                            &  GEx, GEy, GEz,              &
                            &  mint                         )
!----------------------------------------------------------------------------------------------------
            !CALL intr_lin_sing4(x0(i),   y0(i),   z0(i),    &
```

```
            !                &  i,               &
            !                &  GEx, GEy, GEz)
!----------------------------------------------------------------------------------------------------------------
                ELSE
!----------------------------------------------------------------------------------------------------------------
                    CALL Intgr_Trgl(x0(i),  y0(i),  z0(i),  &
                            &   x0(j),  y0(j),  z0(j),  &
                            &   j,                      &
                            &   GEx,   GEy,   GEz,   &
                            &   mint                    )
!----------------------------------------------------------------------------------------------------------------
            !CALL intr_lin_sing4(x0(i),   y0(i),   z0(i),    &
            !                &  j,               &
            !                &  GEx, GEy, GEz)
!----------------------------------------------------------------------------------------------------------------
                END IF
!----------------------------------------------------------------------------------------------------------------
                GM(i)            =  (GM(i)     + GEx)
                GM(inelm)        =  (GM(inelm) + GEy)
                GM(inelm2)       =  (GM(inelm2)+ GEz)
!----------------------------------------------------------------------------------------------------------------
            END DO
!----------------------------------------------------------------------------------------------------------------
        END DO
!$omp end do
!$omp end parallel
        GM = cdg*gamma*GM
!================================================================================================================
    END SUBROUTINE Builder_GM
!================================================================================================================
!================================================================================================================
    SUBROUTINE Builder_TM(TM, cdt, lmu, nelm, mint, NGL)
!================================================================================================================
!   This subroutine builds the matrix array from the Stresslets TE(x,x0).
!----------------------------------------------------------------------------------------------------------------
!   Variables
!----------------------------------------------------------------------------------------------------------------
!   TM ............................. The matrix with the information of Stresslets on the surface
!   cdt ............................ coefficient
!   lmu ............................ coefficient lamdamu
!   nelm ........................... number of elements
!   mint ........................... order of Gauss' quadrature
!   NGL ............................ order of triangle quadrature
!   x0, y0, z0 ..................... coordinates of collocation point of the element
!   TExx, TEyx, TEzx ............... Integrated ij component over the element
!   TEyx, TEyy, TEyz
!   TEzx, TEzy, TEzz
!================================================================================================================
    USE Mod_SharedVars
    USE Mod_Prtcl3D_DLP
    USE Mod_sgf_3d_sfs
    USE mkl95_precision
    USE omp_lib
!================================================================================================================
    IMPLICIT NONE
!================================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: TM     !The matrix with the information of
 !Stresslets on the surface
    REAL (KIND = DBL), INTENT(IN)  :: cdt, lmu              !coefficients of DLP
    INTEGER,           INTENT(IN)  :: nelm                  !number of elements
    INTEGER,           INTENT(IN)  :: mint, NGL             !order of triangle and Gauss-Legendre quadratures
!================================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------------------
    INTEGER ::   i, j                                       !Counters
    INTEGER ::   inelm, inelm2
    INTEGER ::   jnelm, jnelm2
!----------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL)  :: TExx, TExy, TExz, &               !Integrated ij component over the element
    &                     TEyx, TEyy, TEyz, &
```



```
    &                     TEzx, TEzy, TEzz
!----------------------------------------------------------------------------------------------------------------
    TM= 0.0D0
!----------------------------------------------------------------------------------------------------------------
!The loop over collocations points serves to build the matrix form sistem.
!================================================================================================================
    IF(lmu==1.0D0)THEN
!================================================================================================================
!$omp parallel
!$omp do private(i, inelm, inelm2)
        DO i= 1, nelm
            inelm = i + nelm
            inelm2= i + nelm + nelm
!----------------------------------------------------------------------------------------------------------------
!   The values of doble layer potential are obtained in normal kind.
            TM(i,i)             = - ((1.0D0+lmu)/2.0D0)
            TM(inelm,inelm)     = - ((1.0D0+lmu)/2.0D0)
            TM(inelm2,inelm2)   = - ((1.0D0+lmu)/2.0D0)
        END DO
!$omp end do
!$omp end parallel
!================================================================================================================
    ELSE
!================================================================================================================
!$omp parallel
!$omp do private(i, inelm, inelm2, j, jnelm, jnelm2, TExx,TExy,TExz, TEyx,TEyy,TEyz, TEzx,TEzy,TEzz)
        DO i= 1, nelm
            inelm = i + nelm
            inelm2= i + nelm + nelm
            DO j = 1, nelm
                jnelm  = j + nelm
                jnelm2 = j + nelm + nelm
!----------------------------------------------------------------------------------------------------------------
!   The values of doble layer potential are obtained in normal kind.
        IF(i==j)THEN
!----------------------------------------------------------------------------------------------------------------
            CALL Intgr_Trgl_Sing_s(x0(j),y0(j),z0(j),  &
            &      j,                                  &
            &      TExx,TExy,TExz,                     &
            &      TEyx,TEyy,TEyz,                     &
            &      TEzx,TEzy,TEzz,                     &
            &      mint)
!----------------------------------------------------------------------------------------------------------------
            !CALL intr_lin_sing_s4s(x0(i),y0(i),z0(i),    &
            !                &  x0(i),y0(i),z0(i),  &
            !                &  i,                  &
            !                &  TExx, TExy, TExz,   &
            !                &  TEyx, TEyy, TEyz,   &
            !                &  TEzx, TEzy, TEzz    )
!----------------------------------------------------------------------------------------------------------------
            TM(j,j)             = (cdt*TExx) - ((1.0D0 + lmu)/2.0D0)
            TM(j,jnelm)         = cdt*TExy
            TM(j,jnelm2)        = cdt*TExz
            TM(jnelm,j)         = cdt*TEyx
            TM(jnelm,jnelm)     = (cdt*TEyy) - ((1.0D0 + lmu)/2.0D0)
            TM(jnelm,jnelm2)    = cdt*TEyz
            TM(jnelm2,j)        = cdt*TEzx
            TM(jnelm2,jnelm)    = cdt*TEzy
            TM(jnelm2,jnelm2)   = (cdt*TEzz) - ((1.0D0 + lmu)/2.0D0)
!----------------------------------------------------------------------------------------------------------------
            !TM(j,j)             = ((0.05D0*cdt*TExx) - ((1.0D0+lmu)/2.0D0))
            !TM(j,jnelm)         = 0.05D0*cdt*TExy
            !TM(j,jnelm2)        = 0.05D0*cdt*TExz
            !TM(jnelm,j)         = 0.05D0*cdt*TEyx
            !TM(jnelm,jnelm)     = ((0.05D0*cdt*TEyy) - ((1.0D0+lmu)/2.0D0))
            !TM(jnelm,jnelm2)    = 0.05D0*cdt*TEyz
            !TM(jnelm2,j)        = 0.05D0*cdt*TEzx
            !TM(jnelm2,jnelm)    = 0.05D0*cdt*TEzy
            !TM(jnelm2,jnelm2)   = ((0.05D0*cdt*TEzz) - ((1.0D0+lmu)/2.0D0))
!----------------------------------------------------------------------------------------------------------------
        ELSE
!----------------------------------------------------------------------------------------------------------------
```

```
            CALL Intgr_Trgl_s(x0(i),  y0(i),  z0(i),  &
                        &     x0(j),  y0(j),  z0(j),  &
                        &     j,                      &
                        &     TExx,   TExy,   TExz,   &
                        &     TEyx,   TEyy,   TEyz,   &
                        &     TEzx,   TEzy,   TEzz,   &
                        &     mint)
!-------------------------------------------------------------------------------------------------------------------
            !CALL intr_lin_sing_s8(x0(j),y0(j),z0(j),    &
            !                &  x0(i),y0(i),z0(i),  &
            !                &  i,                  &
            !                &  TExx, TExy, TExz,   &
            !                &  TEyx, TEyy, TEyz,   &
            !                &  TEzx, TEzy, TEzz    )
!-------------------------------------------------------------------------------------------------------------------
            TM(i,j)             = cdt*TExx
            TM(i,jnelm)         = cdt*TExy
            TM(i,jnelm2)        = cdt*TExz
            TM(inelm,j)         = cdt*TEyx
            TM(inelm,jnelm)     = cdt*TEyy
            TM(inelm,jnelm2)    = cdt*TEyz
            TM(inelm2,j)        = cdt*TEzx
            TM(inelm2,jnelm)    = cdt*TEzy
            TM(inelm2,jnelm2)   = cdt*TEzz
!-------------------------------------------------------------------------------------------------------------------
            !TM(i,j)             = cdt*TExx*1.5D0
            !TM(i,jnelm)         = cdt*TExy*1.5D0
            !TM(i,jnelm2)        = cdt*TExz*1.5D0
            !TM(inelm,j)         = cdt*TEyx*1.5D0
            !TM(inelm,jnelm)     = cdt*TEyy*1.5D0
            !TM(inelm,jnelm2)    = cdt*TEyz*1.5D0
            !TM(inelm2,j)        = cdt*TEzx*1.5D0
            !TM(inelm2,jnelm)    = cdt*TEzy*1.5D0
            !TM(inelm2,jnelm2)   = cdt*TEzz*1.5D0
!-------------------------------------------------------------------------------------------------------------------
        END IF
            END DO
        END DO
!$omp end do
!$omp end parallel
!===================================================================================================================
    END IF
!===================================================================================================================
    END SUBROUTINE Builder_TM
!===================================================================================================================
END MODULE Mod_Builder_Matrix_Arrays
```

```
MODULE Mod_Prtcl3D_SLP
!==============================================================================================================
!   Version: 0.5 created on  26 / IX / 2007
!                                                                                                C. Pozrikidis
!--------------------------------------------------------------------------------------------------------------
!   Version: 0.7 created on  -- / III /2010
!                                                                                      Marco Antonio Reyes Huesca
!--------------------------------------------------------------------------------------------------------------
!   Version: 0.8 created on 22 / 03 / 2012
!   Version: 0.9 created on 03 / 09 / 2012
!   Version: 1.0 created on 12 / 11 / 2012
!                                                                                         Alfredo Sanjuan Sanjuan
!==============================================================================================================
    CONTAINS
!==============================================================================================================
    SUBROUTINE Intgr_Trgl(x0,   y0,   z0,   &
                        & x,    y,    z, &
!                        & x1,    y1,    z1, &
                        & k,          &
                        & GEx, GEy, GEz, &
                        & mint              )
!==============================================================================================================
!   This subroutine integrates the Green's function over a non-singular triangle numbered k
!==============================================================================================================
    USE Mod_SharedVars, ONLY: DBL, ULog, eps, Pi,      &
    &                         p, ne, n, nbe,arel,      &
    &                         alphaQ, betaQ, gammaQ,   &
    &                         xiq, etq, wq,            &
    &                         Ns,Np
    USE Mod_sgf_3d_fs
    USE Mod_Nodal_Interp
!==============================================================================================================
    IMPLICIT NONE
!==============================================================================================================
!   Variables
!--------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN)    :: x0,  y0,  z0        !coordinates of collocation point of analisis
!    REAL (KIND = DBL), INTENT(IN)    :: x1, y1, z1             !coordinates of collocation point of the element
    REAL (KIND = DBL), INTENT(IN)    :: x, y, z             !coordinates of collocation point of the element
    INTEGER,           INTENT(IN)  :: k                   !element index
    REAL (KIND = DBL), INTENT(OUT) :: GEx, GEy, GEz   !Integrated ij component over the element
    INTEGER,           INTENT(IN)  :: mint                !order of triangle quadrature
!==============================================================================================================
!   Variables inside the subroutine
!--------------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6              !indices to obtain node numbers from each element
!--------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: xi, eta                    !variables to integrate over a triangle
!    REAL (KIND = DBL) :: x, y, z                    !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi, &     !Derivates of the tangential vectors over the element
    &                    DxDet, DyDet, DzDet
    REAL (KIND = DBL) :: vnx, vny, vnz              !normal vector coordinates of the element
    REAL (KIND = DBL) :: hs                         !surface metric on a triangle
    REAL (KIND = DBL) :: fc                         !integration weigth coefficient
    REAL (KIND = DBL) :: Gxx, Gxy, Gxz,       &     !Free-space Green´s function of Stokeslet. integrated ij
    &                    Gyx, Gyy, Gyz,       &     !component over the element
    &                    Gzx, Gzy, Gzz
!==============================================================================================================
!   Initialize
    Gxx = 0.0D0
    Gxy = 0.0D0
    Gxz = 0.0D0
    Gyx = 0.0D0
    Gyy = 0.0D0
    Gyz = 0.0D0
    Gzx = 0.0D0
    Gzy = 0.0D0
    Gzz = 0.0D0
!--------------------------------------------------------------------------------------------------------------
!   prepare the array
!--------------------------------------------------------------------------------------------------------------
```



```
    GEx = 0.0D0
    GEy = 0.0D0
    GEz = 0.0D0
!--------------------------------------------------------------------------------------------------------------
!   vertices of the kth triangle
!--------------------------------------------------------------------------------------------------------------
    i1 = n(k,1)
    i2 = n(k,2)
    i3 = n(k,3)
    i4 = n(k,4)
    i5 = n(k,5)
    i6 = n(k,6)
!==============================================================================================================
!    DO i = 1, mint
!        xi  = xiq(i)
!        eta = etq(i)
!        CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
!        &             p(i2,1),   p(i2,2),  p(i2,3),    &
!        &             p(i3,1),   p(i3,2),  p(i3,3),    &
!        &             p(i4,1),   p(i4,2),  p(i4,3),    &
!        &             p(i5,1),   p(i5,2),  p(i5,3),    &
!        &             p(i6,1),   p(i6,2),  p(i6,3),    &
!        &             alphaQ(k), betaQ(k), gammaQ(k),  &
!        &             xi,        eta,                  &
!        &             x,         y,         z,         &
!        &             DxDxi,     DyDxi,    DzDxi,      &
!        &             DxDet,     DyDet,    DzDet,      &
!        &             vnx,       vny,      vnz,        &
!        &             hs)
!!-------------------------------------------------------------------------------------------------------------
!!   Call subroutine sgf_fs
!!=============================================================================================================
!        CALL sgf_3d_fs(x,    y,    z,   &
!                     & x0,   y0,   z0,  &
!                     & Gxx,  Gxy,  Gxz, &
!                     & Gyx,  Gyy,  Gyz, &
!                     & Gzx,  Gzy,  Gzz  )
!!-------------------------------------------------------------------------------------------------------------
!!   Computes the integral
!!-------------------------------------------------------------------------------------------------------------
!        fc = wq(i)*hs*0.5D0
!!-------------------------------------------------------------------------------------------------------------
!        GEx = GEx + (Gxx*vnx+Gyx*vny+Gzx*vnz)*fc
!        GEy = GEy + (Gxy*vnx+Gyy*vny+Gzy*vnz)*fc
!        GEz = GEz + (Gxz*vnx+Gyz*vny+Gzz*vnz)*fc
!!-------------------------------------------------------------------------------------------------------------
!    END DO
!==============================================================================================================
    CALL intr_lin_sing2( x0,     y0,     z0,     &
                     &    x,      y,      z,      &
                     &  p(i1,1),p(i1,2),p(i1,3),  &
                     &  p(i4,1),p(i4,2),p(i4,3),  &
                     &  k,  mint,                 &
                     &  GEx,  GEy,  GEz)
!--------------------------------------------------------------------------------------------------------------
    CALL intr_lin_sing2( x0,     y0,     z0,     &
                     &    x,      y,      z,      &
                     &  p(i4,1),p(i4,2),p(i4,3),&
                     &  p(i2,1),p(i2,2),p(i2,3),&
                     &  k,  mint,                  &
                     &  GEx,  GEy,  GEz)
!--------------------------------------------------------------------------------------------------------------
    CALL intr_lin_sing2( x0,     y0,     z0,     &
                     &    x,      y,      z,      &
                     &  p(i2,1),p(i2,2),p(i2,3),&
                     &  p(i5,1),p(i5,2),p(i5,3),&
                     &  k,  mint,                  &
                      &  GEx,  GEy,  GEz)
!--------------------------------------------------------------------------------------------------------------
    CALL intr_lin_sing2( x0,     y0,     z0,     &
                     &    x,      y,      z,      &
                     &  p(i5,1),p(i5,2),p(i5,3),&
```

```fortran
                     &  p(i3,1),p(i3,2),p(i3,3),&
                     &  k,  mint,                 &
                     &  GEx,  GEy,  GEz)
!----------------------------------------------------------------------------------------------------
    CALL intr_lin_sing2( x0,     y0,     z0,     &
                     &    x,     y,      z,      &
                     &  p(i3,1),p(i3,2),p(i3,3),&
                     &  p(i6,1),p(i6,2),p(i6,3),&
                     &  k,  mint,                 &
                     &  GEx,  GEy,  GEz)
!----------------------------------------------------------------------------------------------------
    CALL intr_lin_sing2( x0,     y0,     z0,     &
                     &    x,     y,      z,      &
                     &  p(i6,1),p(i6,2),p(i6,3),&
                     &  p(i1,1),p(i1,2),p(i1,3),&
                     &  k,  mint,                &
                     &  GEx,  GEy,  GEz)
!----------------------------------------------------------------------------------------------------
! DOne
!----------------------------------------------------------------------------------------------------
END SUBROUTINE Intgr_Trgl
!====================================================================================================
!====================================================================================================
    SUBROUTINE Intgr_Trgl_Sing(x0,   y0,   z0,    &
                            &  k,                 &
                            &  GEx, GEy, GEz,  &
                            &  mint                )
!====================================================================================================
!   Integrate the Green's function over the kth singular quadratic triangle.
! This is done by breaking up the SINgular triangle into six flat triangles, and THEN integrating individually
!over the flat triangles in local polar coordinates. The singularity
!====================================================================================================
    USE Mod_SharedVars, ONLY: DBL, ULog, eps,    &
    &                         Pi, p, ne, n, nbe, &
    &                         arel, crvmel,       &
    &                         xiq, etq, wq,      &
    &                         Ns, Np
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
    REAL (KIND = DBL), INTENT(IN)  :: x0,   y0,   z0        !coordinates of collocationpoint of the element
    INTEGER,           INTENT(IN)  :: k                     !element index
    REAL (KIND = DBL), INTENT(OUT) :: GEx, GEy, GEz   !Integrated ij component over the element
    INTEGER,           INTENT(IN)  :: mint                   !order of triangle quadrature
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                        !   i, j ........................... Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                      !indices to obtain node numbers from each element
!    INTEGER  :: mint                                        !order of triangle quadrature
!====================================================================================================
!   Obtain the information of the nodes in each element.
!----------------------------------------------------------------------------------------------------
    i1 = n(k,1)
    i2 = n(k,2)
    i3 = n(k,3)
    i4 = n(k,4)
    i5 = n(k,5)
    i6 = n(k,6)
!----------------------------------------------------------------------------------------------------
!   Initialize the array
    GEx = 0.0D0
    GEy = 0.0D0
    GEz = 0.0D0
!====================================================================================================
!   Integrate over three flat triangles
!----------------------------------------------------------------------------------------------------
!    CALL intr_lin_sing( x0,     y0,     z0,     &
!                     &  p(i1,1),p(i1,2),p(i1,3),&
!                     &  p(i2,1),p(i2,2),p(i2,3),&
!                      &  GEx,  GEy,  GEz,  &
!                      !&  GEyx,  GEyy,  GEyz,  &
```



```fortran
!                      !&  GEzx,  GEzy,  GEzz,  &
!                      &  NGL                  )
!!---------------------------------------------------------------------------------------------------
!    CALL intr_lin_sing( x0,     y0,     z0,     &
!                     &  p(i2,1),p(i2,2),p(i2,3),&
!                     &  p(i3,1),p(i3,2),p(i3,3),&
!                      &  GEx,  GEy,  GEz,  &
!                      !&  GEyx,  GEyy,  GEyz,  &
!                      !&  GEzx,  GEzy,  GEzz,  &
!                      &  NGL                  )
!!---------------------------------------------------------------------------------------------------
!    CALL intr_lin_sing( x0,     y0,     z0,     &
!                     &  p(i3,1),p(i3,2),p(i3,3),&
!                     &  p(i1,1),p(i1,2),p(i1,3),&
!                      &  GEx,  GEy,  GEz,  &
!                      !&  GEyx,  GEyy,  GEyz,  &
!                      !&  GEzx,  GEzy,  GEzz,  &
!                      &  NGL                  )
!===================================================================================================
!   Integrate over six flat triangles
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing( x0,     y0,     z0,     &
                     &  p(i1,1),p(i1,2),p(i1,3),&
                     &  p(i4,1),p(i4,2),p(i4,3),&
                      &  k,                    &
                      &  GEx,  GEy,  GEz,  &
                      !&  GEyx,  GEyy,  GEyz,  &
                      !&  GEzx,  GEzy,  GEzz,  &
                      &  mint                  )
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing( x0,     y0,     z0,     &
                     &  p(i4,1),p(i4,2),p(i4,3),&
                     &  p(i2,1),p(i2,2),p(i2,3),&
                      &  k,                    &
                      &  GEx,  GEy,  GEz,  &
                      !&  GEyx,  GEyy,  GEyz,  &
                      !&  GEzx,  GEzy,  GEzz,  &
                      &  mint                  )
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing( x0,     y0,     z0,     &
                     &  p(i2,1),p(i2,2),p(i2,3),&
                     &  p(i5,1),p(i5,2),p(i5,3),&
                      &  k,                    &
                      &  GEx,  GEy,  GEz,  &
                      !&  GEyx,  GEyy,  GEyz,  &
                      !&  GEzx,  GEzy,  GEzz,  &
                      &  mint                  )
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing( x0,     y0,     z0,     &
                     &  p(i5,1),p(i5,2),p(i5,3),&
                     &  p(i3,1),p(i3,2),p(i3,3),&
                      &  k,                    &
                      &  GEx,  GEy,  GEz,  &
                      !&  GEyx,  GEyy,  GEyz,  &
                      !&  GEzx,  GEzy,  GEzz,  &
                      &  mint                  )
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing( x0,     y0,     z0,     &
                     &  p(i3,1),p(i3,2),p(i3,3),&
                     &  p(i6,1),p(i6,2),p(i6,3),&
                      &  k,                    &
                      &  GEx,  GEy,  GEz,  &
                      !&  GEyx,  GEyy,  GEyz,  &
                      !&  GEzx,  GEzy,  GEzz,  &
                      &  mint                  )
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing( x0,     y0,     z0,     &
                     &  p(i6,1),p(i6,2),p(i6,3),&
                     &  p(i1,1),p(i1,2),p(i1,3),&
                      &  k,                    &
                      &  GEx,  GEy,  GEz,  &
                      !&  GEyx,  GEyy,  GEyz,  &
```

```
                      !&  GEzx,  GEzy,  GEzz,  &
                      &  mint                  )
!----------------------------------------------------------------------------------------------------
    !GEx = GEx*crvmel(k)
    !GEy = GEy*crvmel(k)
    !GEz = GEz*crvmel(k)
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE Intgr_Trgl_Sing
!====================================================================================================
!====================================================================================================
SUBROUTINE intr_lin_sing(x1,    y1,    z1,    &
                      &  x2,    y2,    z2,    &
                      &  x3,    y3,    z3,    &
                      &  k,                   &
                      &  GEx,  GEy,  GEz,  &
                      &  mint                  )
!====================================================================================================
!   Integrates the Green's function over a flat triangle in local polar coordinates with origin at singular
!point: (x1,y1,z1). The subrotine is based from Pozriquidis 2002 pp. 119-120 in the technique 5.2.8 - 5.2.11
!====================================================================================================
   USE Mod_SharedVars, ONLY: DBL, ULog, eps, Pi,   &
   &                         ZZ, WW,               &
   &                         Ns, Np, crvmel, xiq, etq, wq
!====================================================================================================
   USE Mod_sgf_3d_fs
!====================================================================================================
   IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN)    :: x1, y1, z1          !coordinates of flat triangle vertice
    REAL (KIND = DBL), INTENT(IN)    :: x2, y2, z2
    REAL (KIND = DBL), INTENT(IN)    :: x3, y3, z3
    INTEGER,           INTENT(IN)    :: k                   !element index
    REAL (KIND = DBL), INTENT(INOUT) :: GEx, GEy, GEz !Integrated ij component over the element
    INTEGER,           INTENT(IN)    :: mint                 !order of Gauss-Legendre quadrature
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                            !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6          !indices to obtain the node number in the vertices of each element
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: pi4, piq           !constants to integrate over a circle element in Phi and r
    REAL (KIND = DBL) :: xi, et, zt         !constants of weigth to integrate over a flat triangle
    REAL (KIND = DBL) :: dx, dy             !surface metrics
    REAL (KIND = DBL) :: vnx, vny, vnz      !triangle normal vector
    REAL (KIND = DBL) :: area, hs           !triangle area and surface metric on a flat triangle
    REAL (KIND = DBL) :: ph, cph, sph       !variable Phi, Cos(phi) and Sin(Phi)
    REAL (KIND = DBL) :: r, rmax, rmaxh     !variable r and values of variable r to integrate
    REAL (KIND = DBL) :: x, y, z            !coordinates of f(xi, et, zt)
    REAL (KIND = DBL) :: cf, cf1, cf2       !integrate weigths
    REAL (KIND = DBL) :: asm, bsm           !triangle area computed by numerical integration and its derivate
    REAL (KIND = DBL) :: B, C, ABC          !integrate arc coefficients
    REAL (KIND = DBL) :: b1, b2, c1         !dummy integrate arc constants
    REAL (KIND = DBL), DIMENSION(3) ::xxi, eet, zzt          !variables of weigth over in triangle (xi,eta)
    REAL (KIND = DBL) :: Gxx, Gxy, Gxz, &    !Free-space Green´s function of Stokeslet. integrated ij component
    &                    Gyx, Gyy, Gyz, &   !over the element
    &                    Gzx, Gzy, Gzz
    REAL (KIND = DBL) :: Rxx, Rxy, Rxz, &   !Stokeslets Dummy coefficients
    &                    Ryx, Ryy, Ryz, &
    &                    Rzx, Rzy, Rzz
    REAL (KIND = DBL) :: Mxx, Mxy, Mxz, &   !Stokeslets Dummy coefficients
    &                    Myx, Myy, Myz, &
    &                    Mzx, Mzy, Mzz
!====================================================================================================
!   constants
!----------------------------------------------------------------------------------------------------
    piq = 0.25D0*pi
    pi4 = 4.0D0*pi
!----------------------------------------------------------------------------------------------------
!   Initialize
    xxi(1) = 0.5D0
```



```
    eet(1) = 0.5D0
    zzt(1) = 0.0D0
    xxi(2) = 0.5D0
    eet(2) = 0.0D0
    zzt(2) = 0.5D0
    xxi(3) = 0.0D0
    eet(3) = 0.5D0
    zzt(3) = 0.5D0
!----------------------------------------------------------------------------------------------------
    !xxi(1) = 1.0D0 / 6.0D0
    !eet(1) = 1.0D0 / 6.0D0
    !zzt(1) = 2.0D0 / 3.0D0
    !xxi(2) = 2.0D0 / 3.0D0
    !eet(2) = 1.0D0 / 6.0D0
    !zzt(2) = 1.0D0 / 6.0D0
    !xxi(3) = 1.0D0 / 6.0D0
    !eet(3) = 2.0D0 / 3.0D0
    !zzt(3) = 1.0D0 / 6.0D0
!----------------------------------------------------------------------------------------------------
    Gxx = 0.0D0
    Gxy = 0.0D0
    Gxz = 0.0D0
    Gyx = 0.0D0
    Gyy = 0.0D0
    Gyz = 0.0D0
    Gzx = 0.0D0
    Gzy = 0.0D0
    Gzz = 0.0D0
!----------------------------------------------------------------------------------------------------
    Rxx = 0.0D0
    Rxy = 0.0D0
    Rxz = 0.0D0
    Ryx = 0.0D0
    Ryy = 0.0D0
    Ryz = 0.0D0
    Rzx = 0.0D0
    Rzy = 0.0D0
    Rzz = 0.0D0
!----------------------------------------------------------------------------------------------------
    Mxx = 0.0D0
    Mxy = 0.0D0
    Mxz = 0.0D0
    Myx = 0.0D0
    Myy = 0.0D0
    Myz = 0.0D0
    Mzx = 0.0D0
    Mzy = 0.0D0
    Mzz = 0.0D0
!----------------------------------------------------------------------------------------------------
!   compute surface metric: hs
!----------------------------------------------------------------------------------------------------
    dx = DSQRT( ((x2-x1)**2)+((y2-y1)**2)+((z2-z1)**2) )
    dy = DSQRT( ((x3-x1)**2)+((y3-y1)**2)+((z3-z1)**2) )
!----------------------------------------------------------------------------------------------------
    vnx = ((y2-y1)*(z3-z1))-((z2-z1)*(y3-y1))
    vny = ((z2-z1)*(x3-x1))-((x2-x1)*(z3-z1))
    vnz = ((x2-x1)*(y3-y1))-((y2-y1)*(x3-x1))
!----------------------------------------------------------------------------------------------------
    hs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
    vnx = vnx/hs
    vny = vny/hs
    vnz = vnz/hs
!----------------------------------------------------------------------------------------------------
!   Surface metric on a flat triangle
    area = hs/2.0D0
!----------------------------------------------------------------------------------------------------
!   Initialize
!----------------------------------------------------------------------------------------------------
!   Triangle area
    asm = 0.0D0
!****************************************************************************************************
!!---------------------------------------------------------------------------------------------------
```

```
!!   Apply the forulas from Pozrikidis (2002, p. 119)
!    b1=0.0D0
!    b2=0.0D0
!    c1=0.0D0
!    ABC=0.0D0
!    B=0.0D0
!    C=0.0D0
!    b1=((x3-x1)*(x2-x1))+((y3-y1)*(y2-y1))+((z3-z1)*(z2-z1))
!    b2=DSQRT((x2-x1)**2+(y2-y1)**2+(z2-z1)**2)
!    c1=DSQRT((x3-x1)**2+(y3-y1)**2+(z3-z1)**2)
!    B= b1/b2**2
!    C= c1**2/b2**2
!!----------------------------------------------------------------------------------------------------------------
!!   Apply the Double quadrature
!!----------------------------------------------------------------------------------------------------------------
!!   Integration wrt phi
!!----------------------------------------------------------------------------------------------------------------
!    Cycle1: DO i = 1, NGL
!        ph    = piq*(1.0D0 + zz(i))
!        cph   = DCOS(ph)
!        sph   = DSIN(ph)
!        rmax  = 1.0D0/(cph+sph)
!        rmaxh = 0.5D0*rmax
!        ABC= (cph**2 + (B*DSIN(2*ph)) + (C*(sph**2)))
!!----------------------------------------------------------------------------------------------------------------
!!   Derivative of asm
!        bsm = 0.0D0
!!----------------------------------------------------------------------------------------------------------------
!!   Integration wrt r
!        Cycle2: DO j=1,NGL
!            r  = rmaxh*(1.0D0+zz(j))
!            xi = r*cph
!            et = r*sph
!            zt = 1.0D0-xi-et
!!----------------------------------------------------------------------------------------------------------------
!            x = x1*zt + x2*xi + x3*et
!            y = y1*zt + y2*xi + y3*et
!            z = z1*zt + z2*xi + z3*et
!!----------------------------------------------------------------------------------------------------------------
!            CALL sgf_3d_fsing(x,   y,   z,   &
!                        &  x1,  y1,  z1,  &
!                        &  Mxx, Mxy, Mxz, &
!                        &  Myx, Myy, Myz, &
!                        &  Mzx, Mzy, Mzz, &
!                        &  ABC)
!            !WRITE(*,*)'Mxx'
!            !WRITE(*,*) Mxx
!!----------------------------------------------------------------------------------------------------------------
!            cf1 = ww(j)*r
!            !WRITE(*,*)'cf1'
!            !WRITE(*,*) cf1
!!----------------------------------------------------------------------------------------------------------------
!            bsm = bsm + cf1
!!----------------------------------------------------------------------------------------------------------------
!            Rxx = Rxx + cf1*Mxx
!            Rxy = Rxy + cf1*Mxy
!            Rxz = Rxz + cf1*Mxz
!            Ryx = Ryx + cf1*Myx
!            Ryy = Ryy + cf1*Myy
!            Ryz = Ryz + cf1*Myz
!            Rzx = Rzx + cf1*Mzx
!            Rzy = Rzy + cf1*Mzy
!            Rzz = Rzz + cf1*Mzz
!!----------------------------------------------------------------------------------------------------------------
!        END DO Cycle2
!!----------------------------------------------------------------------------------------------------------------
!        cf2 = ww(i)*rmaxh
!
!            !WRITE(*,*)'cf2'
!            !WRITE(*,*) cf2
!!----------------------------------------------------------------------------------------------------------------
!        asm = asm + bsm*cf2
```



```
!!----------------------------------------------------------------------------------------------------------------
!        Gxx = Gxx + (cf2*Rxx/DSQRT(ABC))
!        Gxy = Gxy + (cf2*Rxy/DSQRT(ABC))
!        Gxz = Gxz + (cf2*Rxz/DSQRT(ABC))
!        Gyx = Gyx + (cf2*Ryx/DSQRT(ABC))
!        Gyy = Gyy + (cf2*Ryy/DSQRT(ABC))
!        Gyz = Gyz + (cf2*Ryz/DSQRT(ABC))
!        Gzx = Gzx + (cf2*Rzx/DSQRT(ABC))
!        Gzy = Gzy + (cf2*Rzy/DSQRT(ABC))
!        Gzz = Gzz + (cf2*Rzz/DSQRT(ABC))
!!----------------------------------------------------------------------------------------------------------------
!!   initialize again to the next triangle
!        Rxx = 0.0D0
!        Rxy = 0.0D0
!        Rxz = 0.0D0
!        Ryx = 0.0D0
!        Ryy = 0.0D0
!        Ryz = 0.0D0
!        Rzx = 0.0D0
!        Rzy = 0.0D0
!        Rzz = 0.0D0
!!----------------------------------------------------------------------------------------------------------------
!    END DO Cycle1
!!----------------------------------------------------------------------------------------------------------------
!!   complete the quadrature
!!----------------------------------------------------------------------------------------------------------------
!    cf = hs/piq
!!----------------------------------------------------------------------------------------------------------------
!    asm  = asm*cf
!    GEx = GEx + (cf*(Gxx*vnx+Gyx*vny+Gzx*vnz))/b2
!    GEy = GEy + (cf*(Gxy*vnx+Gyy*vny+Gzy*vnz))/b2
!    GEz = GEz + (cf*(Gxz*vnx+Gyz*vny+Gzz*vnz))/b2
!!****************************************************************************************************************
!****************************************************************************************************************
!            xi = 1.0D0/3.0D0 !r*cph
!            et = 1.0D0/3.0D0 !r*sph
!            zt = 1.0D0/3.0D0 !1.0D0-xi-et
!!----------------------------------------------------------------------------------------------------------------
!            x = x1*zt + x2*xi + x3*et
!            y = y1*zt + y2*xi + y3*et
!            z = z1*zt + z2*xi + z3*et
!!----------------------------------------------------------------------------------------------------------------
!            CALL sgf_3d_fs(x,   y,   z,   &
!                        &  x1,  y1,  z1,  &
!                        &  Mxx, Mxy, Mxz, &
!                        &  Myx, Myy, Myz, &
!                        &  Mzx, Mzy, Mzz  )
!!----------------------------------------------------------------------------------------------------------------
!            cf1 = 1.0D0 ! ww(j)*r
!!----------------------------------------------------------------------------------------------------------------
!            bsm = bsm + cf1
!!----------------------------------------------------------------------------------------------------------------
!            Rxx = Rxx + cf1*Mxx
!            Rxy = Rxy + cf1*Mxy
!            Rxz = Rxz + cf1*Mxz
!            Ryx = Ryx + cf1*Myx
!            Ryy = Ryy + cf1*Myy
!            Ryz = Ryz + cf1*Myz
!            Rzx = Rzx + cf1*Mzx
!            Rzy = Rzy + cf1*Mzy
!            Rzz = Rzz + cf1*Mzz
!!----------------------------------------------------------------------------------------------------------------
!        cf2 = 1.0D0 !ww(i)*rmaxh
!!----------------------------------------------------------------------------------------------------------------
!        asm = asm + bsm*cf2
!!----------------------------------------------------------------------------------------------------------------
!        Gxx = Gxx + (cf2*Rxx) !(cf2*Rxx/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!        Gxy = Gxy + (cf2*Rxy) !(cf2*Rxy/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!        Gxz = Gxz + (cf2*Rxz) !(cf2*Rxz/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!        Gyx = Gyx + (cf2*Ryx) !(cf2*Ryx/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!        Gyy = Gyy + (cf2*Ryy) !(cf2*Ryy/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!        Gyz = Gyz + (cf2*Ryz) !(cf2*Ryz/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
```

```fortran
!        Gzx = Gzx + (cf2*Rzx) !(cf2*Rzx/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!        Gzy = Gzy + (cf2*Rzy) !(cf2*Rzy/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!        Gzz = Gzz + (cf2*Rzz) !(cf2*Rzz/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!!-----------------------------------------------------------------------------------------------------------
!!   initialize again to the next triangle
!        Rxx = 0.0D0
!        Rxy = 0.0D0
!        Rxz = 0.0D0
!        Ryx = 0.0D0
!        Ryy = 0.0D0
!        Ryz = 0.0D0
!        Rzx = 0.0D0
!        Rzy = 0.0D0
!        Rzz = 0.0D0
!**********************************************************************************************************
!**********************************************************************************************************
    DO i = 1, mint
        xi  = xiq(i)
        et  = etq(i)
        zt  = 1.0D0- et-xi
        cf1 = wq(i)
!----------------------------------------------------------------------------------------------------------
    !DO i = 1, 3
    !    xi  = xxi(i)
    !    et  = eet(i)
    !    zt  = zzt(i)
    !    cf1 = 1.0D0/3.0D0 ! wq(i)
!----------------------------------------------------------------------------------------------------------
            x = x1*zt + x2*xi + x3*et
            y = y1*zt + y2*xi + y3*et
            z = z1*zt + z2*xi + z3*et
!----------------------------------------------------------------------------------------------------------
            CALL sgf_3d_fs(x,   y,   z,   &
                        &  x1,  y1,  z1,  &
                        &  Rxx, Rxy, Rxz, &
                        &  Ryx, Ryy, Ryz, &
                        &  Rzx, Rzy, Rzz  )
!----------------------------------------------------------------------------------------------------------
!            cf1 = 1.0D0/3.0D0
!----------------------------------------------------------------------------------------------------------
            bsm=0.0D0
            bsm =cf1
!----------------------------------------------------------------------------------------------------------
            Rxx = Rxx*cf1
            Rxy = Rxy*cf1
            Rxz = Rxz*cf1
            Ryx = Ryx*cf1
            Ryy = Ryy*cf1
            Ryz = Ryz*cf1
            Rzx = Rzx*cf1
            Rzy = Rzy*cf1
            Rzz = Rzz*cf1
!----------------------------------------------------------------------------------------------------------
        asm =asm + bsm
!----------------------------------------------------------------------------------------------------------
        Gxx = Gxx + Rxx
        Gxy = Gxy + Rxy
        Gxz = Gxz + Rxz
        Gyx = Gyx + Ryx
        Gyy = Gyy + Ryy
        Gyz = Gyz + Ryz
        Gzx = Gzx + Rzx
        Gzy = Gzy + Rzy
        Gzz = Gzz + Rzz
!----------------------------------------------------------------------------------------------------------
    END DO
!**********************************************************************************************************
!   complete the quadrature
!----------------------------------------------------------------------------------------------------------
    cf = area
!----------------------------------------------------------------------------------------------------------
    asm  = asm*cf
```



```fortran
!**********************************************************************************************************
    GEx = GEx + cf*(Gxx*vnx+Gyx*vny+Gzx*vnz) *crvmel(k)
    GEy = GEy + cf*(Gxy*vnx+Gyy*vny+Gzy*vnz) *crvmel(k)
    GEz = GEz + cf*(Gxz*vnx+Gyz*vny+Gzz*vnz) *crvmel(k)
!**********************************************************************************************************
!----------------------------------------------------------------------------------------------------------
!  IF all went well, asm should be equal to "area"
!  WRITE (Ulog,100) i,area,asm
!----------------------------------------------------------------------------------------------------------
!  100 FORMAT (1x,i3,2(f10.5))
!----------------------------------------------------------------------------------------------------------
END SUBROUTINE intr_lin_sing
!==========================================================================================================
SUBROUTINE intr_lin_sing2(x0,    y0,    z0,   &
                     &  x1,    y1,    z1,    &
                     &  x2,    y2,    z2,    &
                     &  x3,    y3,    z3,    &
                     &  k,     mint,         &
                     &  GEx,  GEy,  GEz)
!==========================================================================================================
!   Integrates the Green's function over a flat triangle in local polar coordinates with origin at singular
!point: (x1,y1,z1). The subrotine is based from Pozriquidis 2002 pp. 119-120 in the technique 5.2.8 - 5.2.11
!==========================================================================================================
   USE Mod_SharedVars, ONLY: DBL, ULog, eps, Pi,   &
   &                         Ns, Np,crvmel, xiq, etq, wq
!==========================================================================================================
   USE Mod_sgf_3d_fs
!==========================================================================================================
   IMPLICIT NONE
!==========================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN)    :: x0, y0, z0
    REAL (KIND = DBL), INTENT(IN)    :: x1, y1, z1          !coordinates of flat triangle vertice
    REAL (KIND = DBL), INTENT(IN)    :: x2, y2, z2
    REAL (KIND = DBL), INTENT(IN)    :: x3, y3, z3
    INTEGER,           INTENT(IN)    :: k                   !element index
    INTEGER,           INTENT(IN)    :: mint                !element index
    REAL (KIND = DBL), INTENT(INOUT) :: GEx, GEy, GEz       !Integrated ij component over the element
    !&                                  GEyx, GEyy, GEyz, &
    !&                                  GEzx, GEzy, GEzz
    !REAL (KIND = DBL), INTENT(IN)  :: a,   b,   c           !coordinates of collocationpoint of the element
!==========================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                          !Counters
    REAL (KIND = DBL) :: xi, et, zt           !constants of weigth to integrate over a flat triangle
    REAL (KIND = DBL) :: dx, dy               !surface metrics
    REAL (KIND = DBL) :: vnx, vny, vnz        !triangle normal vector
    REAL (KIND = DBL) :: area, hs             !triangle area and surface metric on a flat triangle
    REAL (KIND = DBL) :: x, y, z              !coordinates of f(xi, et, zt)
    REAL (KIND = DBL) :: cf, cf1, cf2         !integrate weigths
    REAL (KIND = DBL) :: asm, bsm             !triangle area computed by numerical integration and its derivate
    REAL (KIND = DBL), DIMENSION(3) ::xxi, eet, zzt          !variables of weigth over in triangle (xi,eta)
    REAL (KIND = DBL) :: Gxx, Gxy, Gxz, &   !Free-space Green´s function of Stokeslet. integrated ij component
    &                    Gyx, Gyy, Gyz, &   !over the element
    &                    Gzx, Gzy, Gzz
    REAL (KIND = DBL) :: Rxx, Rxy, Rxz, &   !Stokeslets Dummy coefficients
    &                    Ryx, Ryy, Ryz, &
    &                    Rzx, Rzy, Rzz
!==========================================================================================================
!   Initialize
    xxi(1) = 0.5D0
    eet(1) = 0.5D0
    zzt(1) = 0.0D0
    xxi(2) = 0.5D0
    eet(2) = 0.0D0
    zzt(2) = 0.5D0
    xxi(3) = 0.0D0
    eet(3) = 0.5D0
    zzt(3) = 0.5D0
!----------------------------------------------------------------------------------------------------------
```

```
    Gxx = 0.0D0
    Gxy = 0.0D0
    Gxz = 0.0D0
    Gyx = 0.0D0
    Gyy = 0.0D0
    Gyz = 0.0D0
    Gzx = 0.0D0
    Gzy = 0.0D0
    Gzz = 0.0D0
!------------------------------------------------------------------------------------------------------
    Rxx = 0.0D0
    Rxy = 0.0D0
    Rxz = 0.0D0
    Ryx = 0.0D0
    Ryy = 0.0D0
    Ryz = 0.0D0
    Rzx = 0.0D0
    Rzy = 0.0D0
    Rzz = 0.0D0

!------------------------------------------------------------------------------------------------------
!   compute surface metric: hs
!------------------------------------------------------------------------------------------------------
    dx = DSQRT( (x2-x1)**2+(y2-y1)**2+(z2-z1)**2 )
    dy = DSQRT( (x3-x1)**2+(y3-y1)**2+(z3-z1)**2 )
!------------------------------------------------------------------------------------------------------
    vnx = (y2-y1)*(z3-z1)-(z2-z1)*(y3-y1)
    vny = (z2-z1)*(x3-x1)-(x2-x1)*(z3-z1)
    vnz = (x2-x1)*(y3-y1)-(y2-y1)*(x3-x1)
!------------------------------------------------------------------------------------------------------
    hs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
    vnx = vnx/hs
    vny = vny/hs
    vnz = vnz/hs
!------------------------------------------------------------------------------------------------------
!   Surface metric on a flat triangle
    area = hs/2.0D0
!------------------------------------------------------------------------------------------------------
!   Initialize
!------------------------------------------------------------------------------------------------------
!   Triangle area
    asm = 0.0D0
!------------------------------------------------------------------------------------------------------
    DO i = 1, mint
       xi  = xiq(i)
       et  = etq(i)
       zt  = 1.0D0 - xi-et
       cf1 = wq(i)
!------------------------------------------------------------------------------------------------------
           x = x1*zt + x2*xi + x3*et
           y = y1*zt + y2*xi + y3*et
           z = z1*zt + z2*xi + z3*et
!------------------------------------------------------------------------------------------------------
           CALL sgf_3d_fs(x,   y,   z,   &
                      &  x0,  y0,  z0,  &
                      &  Rxx, Rxy, Rxz, &
                      &  Ryx, Ryy, Ryz, &
                      &  Rzx, Rzy, Rzz  )
!------------------------------------------------------------------------------------------------------
!           cf1 = 1.0D0/3.0D0
!------------------------------------------------------------------------------------------------------
           bsm=0.0D0
           bsm =cf1
!------------------------------------------------------------------------------------------------------
           Rxx = Rxx*cf1
           Rxy = Rxy*cf1
           Rxz = Rxz*cf1
           Ryx = Ryx*cf1
           Ryy = Ryy*cf1
           Ryz = Ryz*cf1
           Rzx = Rzx*cf1
           Rzy = Rzy*cf1
```



```
           Rzz = Rzz*cf1
!------------------------------------------------------------------------------------------------------
       asm =asm + bsm
!------------------------------------------------------------------------------------------------------
       Gxx = Gxx + Rxx
       Gxy = Gxy + Rxy
       Gxz = Gxz + Rxz
       Gyx = Gyx + Ryx
       Gyy = Gyy + Ryy
       Gyz = Gyz + Ryz
       Gzx = Gzx + Rzx
       Gzy = Gzy + Rzy
       Gzz = Gzz + Rzz
!------------------------------------------------------------------------------------------------------
    END DO
!******************************************************************************************************
!   complete the quadrature
!------------------------------------------------------------------------------------------------------
    cf = area
!------------------------------------------------------------------------------------------------------
    asm  = asm*cf
!!******************************************************************************************************
    GEx = GEx + cf*(Gxx*vnx+Gyx*vny+Gzx*vnz) *crvmel(k)
    GEy = GEy + cf*(Gxy*vnx+Gyy*vny+Gzy*vnz) *crvmel(k)
    GEz = GEz + cf*(Gxz*vnx+Gyz*vny+Gzz*vnz) *crvmel(k)
!------------------------------------------------------------------------------------------------------
!  IF all went well, asm should be equal to "area"
!  WRITE (Ulog,100) i,area,asm
!------------------------------------------------------------------------------------------------------
!  100 FORMAT (1x,i3,2(f10.5))
!------------------------------------------------------------------------------------------------------
    END SUBROUTINE intr_lin_sing2
!======================================================================================================
    SUBROUTINE intr_lin_sing3(x0,   y0,   z0,    &
                            &  k,                &
                            &  GEx, GEy, GEz    )
!======================================================================================================
!   This subroutine is a new version stokeslet Subroutine.
!Compute:
!   *The value of the Stokeslet over each singular element
!Now, (March/ 09 / 2015) this subroutine was made.
!======================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog,&
    &                         alphaQ,  betaQ,  gammaQ, &
    &                         vnx0,     vny0,    vnz0, &
    &                         ZZ,           WW,        &
    &                         xiq, etq, wq, crvmel
!======================================================================================================
    IMPLICIT NONE
!======================================================================================================
!   Variables
!------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN) :: x0,   y0,   z0                !singulatity coordinates
    INTEGER,           INTENT(IN)  :: k                            !number of element
    REAL (KIND = DBL), INTENT(OUT) :: GEx, GEy, GEz                !value of stokeslet in the singular element
!======================================================================================================
!   Variables inside the subroutine
!------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                   !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                 !indices to obtain node numbers from each element
!------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: xi, eta                           !variables of weigth to integrate over a triangle
    REAL (KIND = DBL) :: x, y, z                           !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi               !coordinates of the tangential vector over the xi axis
    REAL (KIND = DBL) :: DxDet, DyDet, DzDet               !coordinates of the tangential vector over the eta axis
    REAL (KIND = DBL) :: vnx, vny, vnz                     !normal vector coordinates of the element
    REAL (KIND = DBL) :: hs                                !surface metric on a triangle
    REAL (KIND = DBL) :: al, be, ga, alc, bec, gac         !integration weigth coefficients
    REAL (KIND = DBL) :: cf, fill                          !integration weigth coefficients
    REAL (KIND = DBL), DIMENSION(6) ::xxi, eet             !variables of weigth over in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDx, DyDx, DzDx     !tangential vector over xi axis in triangle (xi,eta)
```

```
    REAL (KIND = DBL), DIMENSION(6) ::DxDe, DyDe, DzDe  !tangential vector over eta axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz       !normal vector in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) :: hss              !weigth
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &           !binormal vectors around in triangle (xi,eta)
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  yvx, yvy, yvz                 !vector (y-x0)
!====================================================================================================
!   Initialize
!----------------------------------------------------------------------------------------------------
    GEx = 0.0D0
    GEy = 0.0D0
    GEz = 0.0D0

    yvx = 0.0D0
    yvy = 0.0D0
    yvz = 0.0D0
!----------------------------------------------------------------------------------------------------
!   vertices of the kth triangle
!----------------------------------------------------------------------------------------------------
    i1 = n(k,1)
    i2 = n(k,2)
    i3 = n(k,3)
    i4 = n(k,4)
    i5 = n(k,5)
    i6 = n(k,6)
!----------------------------------------------------------------------------------------------------

        al  = alphaQ(k)
        be  = betaQ(k)
        ga  = gammaQ(k)
        alc = 1.0D0-al
        bec = 1.0D0-be
        gac = 1.0D0-ga
        cf  = 0.0D0
!----------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the nftty
!formula (4.2.10) of Pozrikidis (1997)
!----------------------------------------------------------------------------------------------------
        xxi(1) = 0.0D0
        eet(1) = 0.0D0
        xxi(2) = 1.0D0
        eet(2) = 0.0D0
        xxi(3) = 0.0D0
        eet(3) = 1.0D0
        xxi(4) = al
        eet(4) = 0.0D0
        xxi(5) = ga
        eet(5) = gac
        xxi(6) = 0.0D0
        eet(6) = be
        DO i = 1, 6
            xi  = xxi(i)
            eta = eet(i)
            CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
            &             p(i2,1),   p(i2,2),  p(i2,3),    &
            &             p(i3,1),   p(i3,2),  p(i3,3),    &
            &             p(i4,1),   p(i4,2),  p(i4,3),    &
            &             p(i5,1),   p(i5,2),  p(i5,3),    &
            &             p(i6,1),   p(i6,2),  p(i6,3),    &
            &             al,        be,       ga,         &
            &             xi,        eta,                  &
            &             x,         y,        z,          &
            &             DxDx(i),   DyDx(i),  DzDx(i),    &
            &             DxDe(i),   DyDe(i),  DzDe(i),    &
            &             vx(i),     vy(i),    vz(i),      &
            &             hss(i) )
        END DO
!----------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
```



```
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx  = 0.0D0
        yvy  = 0.0D0
        yvz  = 0.0D0
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-4-2
!----------------------------------------------------------------------------------------------------
        yvx = p(i1,1) - x0
        yvy = p(i1,2) - y0
        yvz = p(i1,3) - z0
        fill = (DSQRT(yvx**2 + yvy**2 +yvz**2))/hss(1)
!----------------------------------------------------------------------------------------------------
        bvx1 = (DyDx(1)*yvz - DzDx(1)*yvy)/fill
        bvy1 = (DzDx(1)*yvx - DxDx(1)*yvz)/fill
        bvz1 = (DxDx(1)*yvy - DyDx(1)*yvx)/fill
!----------------------------------------------------------------------------------------------------
        yvx = p(i4,1) - x0
        yvy = p(i4,2) - y0
        yvz = p(i4,3) - z0
        fill = (DSQRT(yvx**2 + yvy**2 +yvz**2))/hss(4)
!----------------------------------------------------------------------------------------------------
        bvx2 = (DyDx(4)*yvz - DzDx(4)*yvy)/fill
        bvy2 = (DzDx(4)*yvx - DxDx(4)*yvz)/fill
        bvz2 = (DxDx(4)*yvy - DyDx(4)*yvx)/fill
!----------------------------------------------------------------------------------------------------
        yvx = p(i2,1) - x0
        yvy = p(i2,2) - y0
        yvz = p(i2,3) - z0
        fill = (DSQRT(yvx**2 + yvy**2 +yvz**2))/hss(2)
!----------------------------------------------------------------------------------------------------
        bvx3 = (DyDx(2)*yvz - DzDx(2)*yvy)/fill
        bvy3 = (DzDx(2)*yvx - DxDx(2)*yvz)/fill
        bvz3 = (DxDx(2)*yvy - DyDx(2)*yvx)/fill
!----------------------------------------------------------------------------------------------------
        GEx  = al*bvx1 + bvx2 + alc*bvx3
        GEy  = al*bvy1 + bvy2 + alc*bvy3
        GEz  = al*bvz1 + bvz2 + alc*bvz3
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-5-3
!----------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
!----------------------------------------------------------------------------------------------------
        yvx = p(i2,1) - x0
        yvy = p(i2,2) - y0
        yvz = p(i2,3) - z0
        fill = (DSQRT(yvx**2 + yvy**2 +yvz**2))/hss(2)
!----------------------------------------------------------------------------------------------------
        bvx1 = (DyDx(2)*yvz - DzDx(2)*yvy)/fill
        bvy1 = (DzDx(2)*yvx - DxDx(2)*yvz)/fill
        bvz1 = (DxDx(2)*yvy - DyDx(2)*yvx)/fill
!----------------------------------------------------------------------------------------------------
        yvx = p(i5,1) - x0
        yvy = p(i5,2) - y0
        yvz = p(i5,3) - z0
        fill = (DSQRT(yvx**2 + yvy**2 +yvz**2))/hss(5)
!----------------------------------------------------------------------------------------------------
        bvx2 = (DyDx(5)*yvz - DzDx(5)*yvy)/fill
        bvy2 = (DzDx(5)*yvx - DxDx(5)*yvz)/fill
        bvz2 = (DxDx(5)*yvy - DyDx(5)*yvx)/fill
```

```
!--------------------------------------------------------------------------------------------------------
        yvx = p(i3,1) - x0
        yvy = p(i3,2) - y0
        yvz = p(i3,3) - z0
        fill = (DSQRT(yvx**2 + yvy**2 +yvz**2))/hss(3)
!--------------------------------------------------------------------------------------------------------
        bvx3 = (DyDx(3)*yvz - DzDx(3)*yvy)/fill
        bvy3 = (DzDx(3)*yvx - DxDx(3)*yvz)/fill
        bvz3 = (DxDx(3)*yvy - DyDx(3)*yvx)/fill
!--------------------------------------------------------------------------------------------------------
        GEx  = GEx - (gac*bvx1 + bvx2 + ga*bvx3)
        GEy  = GEy - (gac*bvy1 + bvy2 + ga*bvy3)
        GEz  = GEz - (gac*bvz1 + bvz2 + ga*bvz3)
!--------------------------------------------------------------------------------------------------------
        yvx = p(i2,1) - x0
        yvy = p(i2,2) - y0
        yvz = p(i2,3) - z0
        fill = (DSQRT(yvx**2 + yvy**2 +yvz**2))/hss(2)
!--------------------------------------------------------------------------------------------------------
        bvx1 = (DyDe(2)*yvz - DzDe(2)*yvy)/fill
        bvy1 = (DzDe(2)*yvx - DxDe(2)*yvz)/fill
        bvz1 = (DxDe(2)*yvy - DyDe(2)*yvx)/fill
!--------------------------------------------------------------------------------------------------------
        yvx = p(i5,1) - x0
        yvy = p(i5,2) - y0
        yvz = p(i5,3) - z0
        fill = (DSQRT(yvx**2 + yvy**2 +yvz**2))/hss(5)
!--------------------------------------------------------------------------------------------------------
        bvx2 = (DyDe(5)*yvz - DzDe(5)*yvy)/fill
        bvy2 = (DzDe(5)*yvx - DxDe(5)*yvz)/fill
        bvz2 = (DxDe(5)*yvy - DyDe(5)*yvx)/fill
!--------------------------------------------------------------------------------------------------------
        yvx = p(i3,1) - x0
        yvy = p(i3,2) - y0
        yvz = p(i3,3) - z0
        fill = (DSQRT(yvx**2 + yvy**2 +yvz**2))/hss(3)
!--------------------------------------------------------------------------------------------------------
        bvx3 = (DyDe(3)*yvz - DzDe(3)*yvy)/fill
        bvy3 = (DzDe(3)*yvx - DxDe(3)*yvz)/fill
        bvz3 = (DxDe(3)*yvy - DyDe(3)*yvx)/fill
!--------------------------------------------------------------------------------------------------------
        GEx  = GEx + (gac*bvx1 + bvx2 + ga*bvx3)
        GEy  = GEy + (gac*bvy1 + bvy2 + ga*bvy3)
        GEz  = GEz + (gac*bvz1 + bvz2 + ga*bvz3)
!--------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3-6-1
!--------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
!--------------------------------------------------------------------------------------------------------
        yvx = p(i3,1) - x0
        yvy = p(i3,2) - y0
        yvz = p(i3,3) - z0
        fill = (DSQRT(yvx**2 + yvy**2 +yvz**2))/hss(3)
!--------------------------------------------------------------------------------------------------------
        bvx1 = (DyDe(3)*yvz - DzDe(3)*yvy)/fill
        bvy1 = (DzDe(3)*yvx - DxDe(3)*yvz)/fill
        bvz1 = (DxDe(3)*yvy - DyDe(3)*yvx)/fill
!--------------------------------------------------------------------------------------------------------
        yvx = p(i6,1) - x0
        yvy = p(i6,2) - y0
        yvz = p(i6,3) - z0
        fill = (DSQRT(yvx**2 + yvy**2 +yvz**2))/hss(6)
!--------------------------------------------------------------------------------------------------------
        bvx2 = (DyDe(6)*yvz - DzDe(6)*yvy)/fill
```



```
        bvy2 = (DzDe(6)*yvx - DxDe(6)*yvz)/fill
        bvz2 = (DxDe(6)*yvy - DyDe(6)*yvx)/fill
!--------------------------------------------------------------------------------------------------------
        yvx = p(i1,1) - x0
        yvy = p(i1,2) - y0
        yvz = p(i1,3) - z0
        fill = (DSQRT(yvx**2 + yvy**2 +yvz**2))/hss(1)
!--------------------------------------------------------------------------------------------------------
        bvx3 = (DyDe(1)*yvz - DzDe(1)*yvy)/fill
        bvy3 = (DzDe(1)*yvx - DxDe(1)*yvz)/fill
        bvz3 = (DxDe(1)*yvy - DyDe(1)*yvx)/fill
!--------------------------------------------------------------------------------------------------------
        GEx  = GEx - be*bvx1 - bvx2 - bec*bvx3
        GEy  = GEy - be*bvy1 - bvy2 - bec*bvy3
        GEz  = GEz - be*bvz1 - bvz2 - bec*bvz3
!--------------------------------------------------------------------------------------------------------
    GEx = GEx*crvmel(k)
    GEy = GEy*crvmel(k)
    GEz = GEz*crvmel(k)
    END SUBROUTINE intr_lin_sing3
!========================================================================================================
    SUBROUTINE intr_lin_sing4(x0,   y0,   z0,    &
                           &  k,                &
                           &  GEx, GEy, GEz    )
!========================================================================================================
!   This subroutine is a new version stokeslet Subroutine.
!Compute:
!   *The value of the Stokeslet over each singular element
!Now, (March/ 09 / 2015) this subroutine was made.
!========================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog,&
    &                          crvmel
!========================================================================================================
    IMPLICIT NONE
!========================================================================================================
!   Variables
!--------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN) :: x0,   y0,   z0                !singularity coordinates
    INTEGER,           INTENT(IN)  :: k                            !number of element
    REAL (KIND = DBL), INTENT(OUT) :: GEx, GEy, GEz                !value of stokeslet in the singular element
!========================================================================================================
!   Variables inside the subroutine
!--------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                  !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                !indices to obtain node numbers from each element
!--------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: cf, fill                         !integration weigth coefficients
    REAL (KIND = DBL) :: DxDx, DyDx, DzDx  !tangential vector over xi axis in triangle (xi,eta)
    REAL (KIND = DBL) :: hss              !weigth
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &          !binormal vectors around in triangle (xi,eta)
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  yvx, yvy, yvz                !vector (y-x0)
!========================================================================================================
!   Initialize
!--------------------------------------------------------------------------------------------------------
    GEx = 0.0D0
    GEy = 0.0D0
    GEz = 0.0D0

    yvx = 0.0D0
    yvy = 0.0D0
    yvz = 0.0D0
!--------------------------------------------------------------------------------------------------------
!   vertices of the kth triangle
!--------------------------------------------------------------------------------------------------------
    i1 = n(k,1)
    i2 = n(k,2)
    i3 = n(k,3)
    i4 = n(k,4)
    i5 = n(k,5)
```

```fortran
    i6 = n(k,6)
!------------------------------------------------------------------------------------------------------------------
        cf  = 0.0D0
!------------------------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
! rule formula
!------------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx  = 0.0D0
        yvy  = 0.0D0
        yvz  = 0.0D0
!------------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-4
!------------------------------------------------------------------------------------------------------------------
        hss  = DSQRT((p(i4,1) - p(i1,1))**2 +(p(i4,2) - p(i1,2))**2 + (p(i4,3) - p(i1,3))**2)
        DxDx = (p(i4,1) - p(i1,1))/hss
        DyDx = (p(i4,2) - p(i1,2))/hss
        DzDx = (p(i4,3) - p(i1,3))/hss
!------------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i1,1) - x0)**2 + (p(i1,2) - y0)**2 +(p(i1,3) - z0)**2)
        yvx = p(i1,1) - x0
        yvy = p(i1,2) - y0
        yvz = p(i1,3) - z0
!------------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz - DzDx*yvy)/fill
        bvy1 = (DzDx*yvx - DxDx*yvz)/fill
        bvz1 = (DxDx*yvy - DyDx*yvx)/fill
!------------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i4,1) - x0)**2 + (p(i4,2) - y0)**2 +(p(i4,3) - z0)**2)
        yvx = p(i4,1) - x0
        yvy = p(i4,2) - y0
        yvz = p(i4,3) - z0
!------------------------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz - DzDx*yvy)/fill
        bvy2 = (DzDx*yvx - DxDx*yvz)/fill
        bvz2 = (DxDx*yvy - DyDx*yvx)/fill
!------------------------------------------------------------------------------------------------------------------
        GEx  = 0.5D0*hss*(bvx1 + bvx2)
        GEy  = 0.5D0*hss*(bvy1 + bvy2)
        GEz  = 0.5D0*hss*(bvz1 + bvz2)
!------------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 4-2
!------------------------------------------------------------------------------------------------------------------
        hss  = DSQRT((p(i2,1) - p(i4,1))**2 +(p(i2,2) - p(i4,2))**2 + (p(i2,3) - p(i4,3))**2)
        DxDx = (p(i2,1) - p(i4,1))/hss
        DyDx = (p(i2,2) - p(i4,2))/hss
        DzDx = (p(i2,3) - p(i4,3))/hss
!------------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i4,1) - x0)**2 + (p(i4,2) - y0)**2 +(p(i4,3) - z0)**2)
        yvx = p(i4,1) - x0
        yvy = p(i4,2) - y0
        yvz = p(i4,3) - z0
!------------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz - DzDx*yvy)/fill
        bvy1 = (DzDx*yvx - DxDx*yvz)/fill
        bvz1 = (DxDx*yvy - DyDx*yvx)/fill
!------------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i2,1) - x0)**2 + (p(i2,2) - y0)**2 +(p(i2,3) - z0)**2)
        yvx = p(i2,1) - x0
        yvy = p(i2,2) - y0
        yvz = p(i2,3) - z0
!------------------------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz - DzDx*yvy)/fill
        bvy2 = (DzDx*yvx - DxDx*yvz)/fill
```



```fortran
        bvz2 = (DxDx*yvy - DyDx*yvx)/fill
!------------------------------------------------------------------------------------------------------------------
        GEx  = GEx + 0.5D0*hss*(bvx1 + bvx2)
        GEy  = GEy + 0.5D0*hss*(bvy1 + bvy2)
        GEz  = GEz + 0.5D0*hss*(bvz1 + bvz2)
!------------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-5
!------------------------------------------------------------------------------------------------------------------
        hss  = DSQRT((p(i5,1) - p(i2,1))**2 +(p(i5,2) - p(i2,2))**2 + (p(i5,3) - p(i2,3))**2)
        DxDx = (p(i5,1) - p(i2,1))/hss
        DyDx = (p(i5,2) - p(i2,2))/hss
        DzDx = (p(i5,3) - p(i2,3))/hss
!------------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i2,1) - x0)**2 + (p(i2,2) - y0)**2 +(p(i2,3) - z0)**2)
        yvx = p(i2,1) - x0
        yvy = p(i2,2) - y0
        yvz = p(i2,3) - z0
!------------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz - DzDx*yvy)/fill
        bvy1 = (DzDx*yvx - DxDx*yvz)/fill
        bvz1 = (DxDx*yvy - DyDx*yvx)/fill
!------------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i5,1) - x0)**2 + (p(i5,2) - y0)**2 +(p(i5,3) - z0)**2)
        yvx = p(i5,1) - x0
        yvy = p(i5,2) - y0
        yvz = p(i5,3) - z0
!------------------------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz - DzDx*yvy)/fill
        bvy2 = (DzDx*yvx - DxDx*yvz)/fill
        bvz2 = (DxDx*yvy - DyDx*yvx)/fill
!------------------------------------------------------------------------------------------------------------------
        GEx  = GEx + 0.5D0*hss*(bvx1 + bvx2)
        GEy  = GEy + 0.5D0*hss*(bvy1 + bvy2)
        GEz  = GEz + 0.5D0*hss*(bvz1 + bvz2)
!------------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 5-3
!------------------------------------------------------------------------------------------------------------------
        hss  = DSQRT((p(i3,1) - p(i5,1))**2 +(p(i3,2) - p(i5,2))**2 + (p(i3,3) - p(i5,3))**2)
        DxDx = (p(i3,1) - p(i5,1))/hss
        DyDx = (p(i3,2) - p(i5,2))/hss
        DzDx = (p(i3,3) - p(i5,3))/hss
!------------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i5,1) - x0)**2 + (p(i5,2) - y0)**2 +(p(i5,3) - z0)**2)
        yvx = p(i5,1) - x0
        yvy = p(i5,2) - y0
        yvz = p(i5,3) - z0
!------------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz - DzDx*yvy)/fill
        bvy1 = (DzDx*yvx - DxDx*yvz)/fill
        bvz1 = (DxDx*yvy - DyDx*yvx)/fill
!------------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i3,1) - x0)**2 + (p(i3,2) - y0)**2 +(p(i3,3) - z0)**2)
        yvx = p(i3,1) - x0
        yvy = p(i3,2) - y0
        yvz = p(i3,3) - z0
!------------------------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz - DzDx*yvy)/fill
        bvy2 = (DzDx*yvx - DxDx*yvz)/fill
        bvz2 = (DxDx*yvy - DyDx*yvx)/fill
!------------------------------------------------------------------------------------------------------------------
        GEx  = GEx + 0.5D0*hss*(bvx1 + bvx2)
        GEy  = GEy + 0.5D0*hss*(bvy1 + bvy2)
        GEz  = GEz + 0.5D0*hss*(bvz1 + bvz2)
!------------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3-6
!------------------------------------------------------------------------------------------------------------------
        hss  = DSQRT((p(i6,1) - p(i3,1))**2 +(p(i6,2) - p(i3,2))**2 + (p(i6,3) - p(i3,3))**2)
        DxDx = (p(i6,1) - p(i3,1))/hss
        DyDx = (p(i6,2) - p(i3,2))/hss
        DzDx = (p(i6,3) - p(i3,3))/hss
!------------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i3,1) - x0)**2 + (p(i3,2) - y0)**2 +(p(i3,3) - z0)**2)
```

```
        yvx = p(i3,1) - x0
        yvy = p(i3,2) - y0
        yvz = p(i3,3) - z0
!---------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz - DzDx*yvy)/fill
        bvy1 = (DzDx*yvx - DxDx*yvz)/fill
        bvz1 = (DxDx*yvy - DyDx*yvx)/fill
!---------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i6,1) - x0)**2 + (p(i6,2) - y0)**2 +(p(i6,3) - z0)**2)
        yvx = p(i6,1) - x0
        yvy = p(i6,2) - y0
        yvz = p(i6,3) - z0
!---------------------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz - DzDx*yvy)/fill
        bvy2 = (DzDx*yvx - DxDx*yvz)/fill
        bvz2 = (DxDx*yvy - DyDx*yvx)/fill
!---------------------------------------------------------------------------------------------------------------
        GEx  = GEx + 0.5D0*hss*(bvx1 + bvx2)
        GEy  = GEy + 0.5D0*hss*(bvy1 + bvy2)
        GEz  = GEz + 0.5D0*hss*(bvz1 + bvz2)
!---------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 6-1
!---------------------------------------------------------------------------------------------------------------
        hss  = DSQRT((p(i1,1) - p(i6,1))**2 +(p(i1,2) - p(i6,2))**2 + (p(i1,3) - p(i6,3))**2)
        DxDx = (p(i1,1) - p(i6,1))/hss
        DyDx = (p(i1,2) - p(i6,2))/hss
        DzDx = (p(i1,3) - p(i6,3))/hss
!---------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i6,1) - x0)**2 + (p(i6,2) - y0)**2 +(p(i6,3) - z0)**2)
        yvx = p(i6,1) - x0
        yvy = p(i6,2) - y0
        yvz = p(i6,3) - z0
!---------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz - DzDx*yvy)/fill
        bvy1 = (DzDx*yvx - DxDx*yvz)/fill
        bvz1 = (DxDx*yvy - DyDx*yvx)/fill
!---------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i1,1) - x0)**2 + (p(i1,2) - y0)**2 +(p(i1,3) - z0)**2)
        yvx = p(i1,1) - x0
        yvy = p(i1,2) - y0
        yvz = p(i1,3) - z0
!---------------------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz - DzDx*yvy)/fill
        bvy2 = (DzDx*yvx - DxDx*yvz)/fill
        bvz2 = (DxDx*yvy - DyDx*yvx)/fill
!---------------------------------------------------------------------------------------------------------------
        GEx  = GEx + 0.5D0*hss*(bvx1 + bvx2)
        GEy  = GEy + 0.5D0*hss*(bvy1 + bvy2)
        GEz  = GEz + 0.5D0*hss*(bvz1 + bvz2)
!---------------------------------------------------------------------------------------------------------------
!===============================================================================================================
! computation of curvature line integral along segment 1-6
!---------------------------------------------------------------------------------------------------------------
        hss  = DSQRT((p(i6,1) - p(i1,1))**2 +(p(i6,2) - p(i1,2))**2 + (p(i6,3) - p(i1,3))**2)
        DxDx = (p(i6,1) - p(i1,1))/hss
        DyDx = (p(i6,2) - p(i1,2))/hss
        DzDx = (p(i6,3) - p(i1,3))/hss
!---------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i1,1) - x0)**2 + (p(i1,2) - y0)**2 +(p(i1,3) - z0)**2)
        yvx = p(i1,1) - x0
        yvy = p(i1,2) - y0
        yvz = p(i1,3) - z0
!---------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz - DzDx*yvy)/fill
        bvy1 = (DzDx*yvx - DxDx*yvz)/fill
        bvz1 = (DxDx*yvy - DyDx*yvx)/fill
!---------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i6,1) - x0)**2 + (p(i6,2) - y0)**2 +(p(i6,3) - z0)**2)
        yvx = p(i6,1) - x0
        yvy = p(i6,2) - y0
        yvz = p(i6,3) - z0
!---------------------------------------------------------------------------------------------------------------
```



```
        bvx2 = (DyDx*yvz - DzDx*yvy)/fill
        bvy2 = (DzDx*yvx - DxDx*yvz)/fill
        bvz2 = (DxDx*yvy - DyDx*yvx)/fill
!---------------------------------------------------------------------------------------------------------------
        GEx  = GEx + 0.5D0*hss*(bvx1 + bvx2)
        GEy  = GEy + 0.5D0*hss*(bvy1 + bvy2)
        GEz  = GEz + 0.5D0*hss*(bvz1 + bvz2)
!---------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 6-3
!---------------------------------------------------------------------------------------------------------------
        hss  = DSQRT((p(i3,1) - p(i6,1))**2 +(p(i3,2) - p(i6,2))**2 + (p(i3,3) - p(i6,3))**2)
        DxDx = (p(i3,1) - p(i6,1))/hss
        DyDx = (p(i3,2) - p(i6,2))/hss
        DzDx = (p(i3,3) - p(i6,3))/hss
!---------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i6,1) - x0)**2 + (p(i6,2) - y0)**2 +(p(i6,3) - z0)**2)
        yvx = p(i6,1) - x0
        yvy = p(i6,2) - y0
        yvz = p(i6,3) - z0
!---------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz - DzDx*yvy)/fill
        bvy1 = (DzDx*yvx - DxDx*yvz)/fill
        bvz1 = (DxDx*yvy - DyDx*yvx)/fill
!---------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i3,1) - x0)**2 + (p(i3,2) - y0)**2 +(p(i3,3) - z0)**2)
        yvx = p(i3,1) - x0
        yvy = p(i3,2) - y0
        yvz = p(i3,3) - z0
!---------------------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz - DzDx*yvy)/fill
        bvy2 = (DzDx*yvx - DxDx*yvz)/fill
        bvz2 = (DxDx*yvy - DyDx*yvx)/fill
!---------------------------------------------------------------------------------------------------------------
        GEx  = GEx + 0.5D0*hss*(bvx1 + bvx2)
        GEy  = GEy + 0.5D0*hss*(bvy1 + bvy2)
        GEz  = GEz + 0.5D0*hss*(bvz1 + bvz2)
!---------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3-5
!---------------------------------------------------------------------------------------------------------------
        hss  = DSQRT((p(i5,1) - p(i3,1))**2 +(p(i5,2) - p(i3,2))**2 + (p(i5,3) - p(i3,3))**2)
        DxDx = (p(i5,1) - p(i3,1))/hss
        DyDx = (p(i5,2) - p(i3,2))/hss
        DzDx = (p(i5,3) - p(i3,3))/hss
!---------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i3,1) - x0)**2 + (p(i3,2) - y0)**2 +(p(i3,3) - z0)**2)
        yvx = p(i3,1) - x0
        yvy = p(i3,2) - y0
        yvz = p(i3,3) - z0
!---------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz - DzDx*yvy)/fill
        bvy1 = (DzDx*yvx - DxDx*yvz)/fill
        bvz1 = (DxDx*yvy - DyDx*yvx)/fill
!---------------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i5,1) - x0)**2 + (p(i5,2) - y0)**2 +(p(i5,3) - z0)**2)
        yvx = p(i5,1) - x0
        yvy = p(i5,2) - y0
        yvz = p(i5,3) - z0
!---------------------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz - DzDx*yvy)/fill
        bvy2 = (DzDx*yvx - DxDx*yvz)/fill
        bvz2 = (DxDx*yvy - DyDx*yvx)/fill
!---------------------------------------------------------------------------------------------------------------
        GEx  = GEx + 0.5D0*hss*(bvx1 + bvx2)
        GEy  = GEy + 0.5D0*hss*(bvy1 + bvy2)
        GEz  = GEz + 0.5D0*hss*(bvz1 + bvz2)
!---------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 5-2
!---------------------------------------------------------------------------------------------------------------
        hss  = DSQRT((p(i2,1) - p(i5,1))**2 +(p(i2,2) - p(i5,2))**2 + (p(i2,3) - p(i5,3))**2)
        DxDx = (p(i2,1) - p(i5,1))/hss
        DyDx = (p(i2,2) - p(i5,2))/hss
        DzDx = (p(i2,3) - p(i5,3))/hss
```

```
!-----------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i5,1) - x0)**2 + (p(i5,2) - y0)**2 +(p(i5,3) - z0)**2)
        yvx = p(i5,1) - x0
        yvy = p(i5,2) - y0
        yvz = p(i5,3) - z0
!-----------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz - DzDx*yvy)/fill
        bvy1 = (DzDx*yvx - DxDx*yvz)/fill
        bvz1 = (DxDx*yvy - DyDx*yvx)/fill
!-----------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i2,1) - x0)**2 + (p(i2,2) - y0)**2 +(p(i2,3) - z0)**2)
        yvx = p(i2,1) - x0
        yvy = p(i2,2) - y0
        yvz = p(i2,3) - z0
!-----------------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz - DzDx*yvy)/fill
        bvy2 = (DzDx*yvx - DxDx*yvz)/fill
        bvz2 = (DxDx*yvy - DyDx*yvx)/fill
!-----------------------------------------------------------------------------------------------------------
        GEx  = GEx + 0.5D0*hss*(bvx1 + bvx2)
        GEy  = GEy + 0.5D0*hss*(bvy1 + bvy2)
        GEz  = GEz + 0.5D0*hss*(bvz1 + bvz2)
!-----------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-4
!-----------------------------------------------------------------------------------------------------------
        hss  = DSQRT((p(i4,1) - p(i2,1))**2 +(p(i4,2) - p(i2,2))**2 + (p(i4,3) - p(i2,3))**2)
        DxDx = (p(i4,1) - p(i2,1))/hss
        DyDx = (p(i4,2) - p(i2,2))/hss
        DzDx = (p(i4,3) - p(i2,3))/hss
!-----------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i2,1) - x0)**2 + (p(i2,2) - y0)**2 +(p(i2,3) - z0)**2)
        yvx = p(i2,1) - x0
        yvy = p(i2,2) - y0
        yvz = p(i2,3) - z0
!-----------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz - DzDx*yvy)/fill
        bvy1 = (DzDx*yvx - DxDx*yvz)/fill
        bvz1 = (DxDx*yvy - DyDx*yvx)/fill
!-----------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i4,1) - x0)**2 + (p(i4,2) - y0)**2 +(p(i4,3) - z0)**2)
        yvx = p(i4,1) - x0
        yvy = p(i4,2) - y0
        yvz = p(i4,3) - z0
!-----------------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz - DzDx*yvy)/fill
        bvy2 = (DzDx*yvx - DxDx*yvz)/fill
        bvz2 = (DxDx*yvy - DyDx*yvx)/fill
!-----------------------------------------------------------------------------------------------------------
        GEx  = GEx + 0.5D0*hss*(bvx1 + bvx2)
        GEy  = GEy + 0.5D0*hss*(bvy1 + bvy2)
        GEz  = GEz + 0.5D0*hss*(bvz1 + bvz2)
!-----------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 4-1
!-----------------------------------------------------------------------------------------------------------
        hss  = DSQRT((p(i1,1) - p(i4,1))**2 +(p(i1,2) - p(i4,2))**2 + (p(i1,3) - p(i4,3))**2)
        DxDx = (p(i1,1) - p(i4,1))/hss
        DyDx = (p(i1,2) - p(i4,2))/hss
        DzDx = (p(i1,3) - p(i4,3))/hss
!-----------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i4,1) - x0)**2 + (p(i4,2) - y0)**2 +(p(i4,3) - z0)**2)
        yvx = p(i4,1) - x0
        yvy = p(i4,2) - y0
        yvz = p(i4,3) - z0
!-----------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz - DzDx*yvy)/fill
        bvy1 = (DzDx*yvx - DxDx*yvz)/fill
        bvz1 = (DxDx*yvy - DyDx*yvx)/fill
!-----------------------------------------------------------------------------------------------------------
        fill = DSQRT((p(i1,1) - x0)**2 + (p(i1,2) - y0)**2 +(p(i1,3) - z0)**2)
        yvx = p(i1,1) - x0
        yvy = p(i1,2) - y0
        yvz = p(i1,3) - z0
```



```
!-----------------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz - DzDx*yvy)/fill
        bvy2 = (DzDx*yvx - DxDx*yvz)/fill
        bvz2 = (DxDx*yvy - DyDx*yvx)/fill
!-----------------------------------------------------------------------------------------------------------
        GEx  = GEx + 0.5D0*hss*(bvx1 + bvx2)
        GEy  = GEy + 0.5D0*hss*(bvy1 + bvy2)
        GEz  = GEz + 0.5D0*hss*(bvz1 + bvz2)
!-----------------------------------------------------------------------------------------------------------
    GEx = 0.5D0*GEx
    GEy = 0.5D0*GEy
    GEz = 0.5D0*GEz
!===========================================================================================================
    GEx = GEx*crvmel(k)
    GEy = GEy*crvmel(k)
    GEz = GEz*crvmel(k)
!===========================================================================================================
    END SUBROUTINE intr_lin_sing4
!===========================================================================================================
!===========================================================================================================
   SUBROUTINE intr_lin_sing5(x0,   y0,   z0,    &
                           &  k,                &
                           &  GEx, GEy, GEz    )
!===========================================================================================================
!   This subroutine is a new version stokeslet Subroutine.
!Compute:
!   *The value of the Stokeslet over each singular element
!Now, (March/ 09 / 2015) this subroutine was made.
!===========================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog,&
    &                         crvmel
!===========================================================================================================
    IMPLICIT NONE
!===========================================================================================================
!   Variables
!-----------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN) :: x0,   y0,   z0                !singulatity coordinates
    INTEGER,           INTENT(IN)  :: k                            !number of element
    REAL (KIND = DBL), INTENT(OUT) :: GEx, GEy, GEz                !value of stokeslet in the singular element
!===========================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                   !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                 !indices to obtain node numbers from each element
!-----------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: cf, fill                          !integration weigth coefficients
    REAL (KIND = DBL) :: DxDx, DyDx, DzDx  !tangential vector over xi axis in triangle (xi,eta)
    REAL (KIND = DBL) :: hss                !weigth
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &          !binormal vectors around in triangle (xi,eta)
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3, &
    &                     bvx4, bvy4, bvz4, &
    &                     bvx5, bvy5, bvz5, &
    &                     bvx6, bvy6, bvz6
    REAL (KIND = DBL) ::  yvx1, yvy1, yvz1                  !vector y1
    REAL (KIND = DBL) ::  yvx2, yvy2, yvz2                  !vector y2
    REAL (KIND = DBL) ::  yvx3, yvy3, yvz3                  !vector y3
    REAL (KIND = DBL) ::  yvx4, yvy4, yvz4                  !vector y4
    REAL (KIND = DBL) ::  yvx5, yvy5, yvz5                  !vector y5
    REAL (KIND = DBL) ::  yvx6, yvy6, yvz6                  !vector y6
    REAL (KIND = DBL) ::  zvx1, zvy1, zvz1                  !vector (z1-x0)
    REAL (KIND = DBL) ::  zvx2, zvy2, zvz2                  !vector (z2-x0)
    REAL (KIND = DBL) ::  zvx3, zvy3, zvz3                  !vector (z3-x0)
    REAL (KIND = DBL) ::  zvx4, zvy4, zvz4                  !vector (z4-x0)
    REAL (KIND = DBL) ::  zvx5, zvy5, zvz5                  !vector (z5-x0)
    REAL (KIND = DBL) ::  zvx6, zvy6, zvz6                  !vector (z6-x0)
    REAL (KIND = DBL) ::  px2, py2, pz2                 !preasure vector
    REAL (KIND = DBL) ::  a1, a2, a3                    !vector a
    REAL (KIND = DBL) ::  be1, be2, be3, be4, be5, be6  !betha angle between zi-x0 and tangent vector
    REAL (KIND = DBL) ::  ga1, ga2, ga3, ga4, ga5, ga6  !gamma angle between z1-x0 and tangent vector
    REAL (KIND = DBL) ::  e1, e2, e3, e4, e5, e6        !constant
```

```fortran
    REAL (KIND = DBL) ::  g1, g2, g3, g4, g5, g6        !constant
!===================================================================================================
!   Initialize
!---------------------------------------------------------------------------------------------------
    GEx = 0.0D0
    GEy = 0.0D0
    GEz = 0.0D0

!---------------------------------------------------------------------------------------------------
!   vertices of the kth triangle
!---------------------------------------------------------------------------------------------------
    i1 = n(k,1)
    i2 = n(k,2)
    i3 = n(k,3)
    i4 = n(k,4)
    i5 = n(k,5)
    i6 = n(k,6)
!---------------------------------------------------------------------------------------------------
        cf  = 0.0D0
!---------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
! rule formula
!---------------------------------------------------------------------------------------------------
    DxDx = p(i4,1) - p(i1,1)
    DyDx = p(i4,2) - p(i1,2)
    DzDx = p(i4,3) - p(i1,3)
    hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
    DxDx = DxDx/hss
    DyDx = DyDx/hss
    DzDx = DzDx/hss
!---------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
! rule formula
!---------------------------------------------------------------------------------------------------
!   Firstly, to compute all vectors and angles involved.
!---------------------------------------------------------------------------------------------------
    yvx1 = p(i1,1)-x0
    yvy1 = p(i1,2)-y0
    yvz1 = p(i1,3)-z0
    yvx2 = p(i2,1)-x0
    yvy2 = p(i2,2)-y0
    yvz2 = p(i2,3)-z0
    yvx3 = p(i3,1)-x0
    yvy3 = p(i3,2)-y0
    yvz3 = p(i3,3)-z0
    yvx4 = p(i4,1)-x0
    yvy4 = p(i4,2)-y0
    yvz4 = p(i4,3)-z0
    yvx5 = p(i5,1)-x0
    yvy5 = p(i5,2)-y0
    yvz5 = p(i5,3)-z0
    yvx6 = p(i6,1)-x0
    yvy6 = p(i6,2)-y0
    yvz6 = p(i6,3)-z0
!---------------------------------------------------------------------------------------------------
    zvx1 = yvx1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
    zvy1 = yvy1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
    zvz1 = yvz1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
    zvx2 = yvx2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
    zvy2 = yvy2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
    zvz2 = yvz2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
    zvx3 = yvx3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
    zvy3 = yvy3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
    zvz3 = yvz3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
    zvx4 = yvx4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
    zvy4 = yvy4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
    zvz4 = yvz4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
    zvx5 = yvx5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
    zvy5 = yvy5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
    zvz5 = yvz5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
    zvx6 = yvx6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
    zvy6 = yvy6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
```



```fortran
    zvz6 = yvz6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
!---------------------------------------------------------------------------------------------------
!   Angles
!---------------------------------------------------------------------------------------------------
    be1 = DACOS( ( ( yvx4-yvx1 )*zvx1 + ( yvy4-yvy1 )*zvy1 + ( yvz4-yvz1 )*zvz1 ) &
                & /DSQRT( ( yvx4-yvx1 )**2 + ( yvy4-yvy1 )**2 + ( yvz4-yvz1 )**2 ) )
    ga1 = DACOS( ( ( yvx4-yvx1 )*zvx4 + ( yvy4-yvy1 )*zvy4 + ( yvz4-yvz1 )*zvz4 ) &
                & /DSQRT( ( yvx4-yvx1 )**2 + ( yvy4-yvy1 )**2 + ( yvz4-yvz1 )**2 ) )
    be2 = DACOS( ( ( yvx2-yvx4 )*zvx4 + ( yvy2-yvy4 )*zvy4 + ( yvz2-yvz4 )*zvz4 ) &
                & /DSQRT( ( yvx2-yvx4 )**2 + ( yvy2-yvy4 )**2 + ( yvz2-yvz4 )**2 ) )
    ga2 = DACOS( ( ( yvx2-yvx4 )*zvx2 + ( yvy2-yvy4 )*zvy2 + ( yvz2-yvz4 )*zvz2 ) &
                & /DSQRT( ( yvx2-yvx4 )**2 + ( yvy2-yvy4 )**2 + ( yvz2-yvz4 )**2 ) )
    be3 = DACOS( ( ( yvx5-yvx2 )*zvx2 + ( yvy5-yvy2 )*zvy2 + ( yvz5-yvz2 )*zvz2 ) &
                & /DSQRT( ( yvx5-yvx2 )**2 + ( yvy5-yvy2 )**2 + ( yvz5-yvz2 )**2 ) )
    ga3 = DACOS( ( ( yvx5-yvx2 )*zvx5 + ( yvy5-yvy2 )*zvy5 + ( yvz5-yvz2 )*zvz5 ) &
                & /DSQRT( ( yvx5-yvx2 )**2 + ( yvy5-yvy2 )**2 + ( yvz5-yvz2 )**2 ) )
    be4 = DACOS( ( ( yvx3-yvx5 )*zvx5 + ( yvy3-yvy5 )*zvy5 + ( yvz3-yvz5 )*zvz5 ) &
                & /DSQRT( ( yvx3-yvx5 )**2 + ( yvy3-yvy5 )**2 + ( yvz3-yvz5 )**2 ) )
    ga4 = DACOS( ( ( yvx3-yvx5 )*zvx3 + ( yvy3-yvy5 )*zvy3 + ( yvz3-yvz5 )*zvz3 ) &
                & /DSQRT( ( yvx3-yvx5 )**2 + ( yvy3-yvy5 )**2 + ( yvz3-yvz5 )**2 ) )
    be5 = DACOS( ( ( yvx6-yvx3 )*zvx3 + ( yvy6-yvy3 )*zvy3 + ( yvz6-yvz3 )*zvz3 ) &
                & /DSQRT( ( yvx6-yvx3 )**2 + ( yvy6-yvy3 )**2 + ( yvz6-yvz3 )**2 ) )
    ga5 = DACOS( ( ( yvx6-yvx3 )*zvx6 + ( yvy6-yvy3 )*zvy6 + ( yvz6-yvz3 )*zvz6 ) &
                & /DSQRT( ( yvx6-yvx3 )**2 + ( yvy6-yvy3 )**2 + ( yvz6-yvz3 )**2 ) )
    be6 = DACOS( ( ( yvx1-yvx6 )*zvx6 + ( yvy1-yvy6 )*zvy6 + ( yvz1-yvz6 )*zvz6 ) &
                & /DSQRT( ( yvx1-yvx6 )**2 + ( yvy1-yvy6 )**2 + ( yvz1-yvz6 )**2 ) )
    ga6 = DACOS( ( ( yvx1-yvx6 )*zvx1 + ( yvy1-yvy6 )*zvy1 + ( yvz1-yvz6 )*zvz1 ) &
                & /DSQRT( ( yvx1-yvx6 )**2 + ( yvy1-yvy6 )**2 + ( yvz1-yvz6 )**2 ) )
!---------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1
!---------------------------------------------------------------------------------------------------
        bvx1 = (yvy1*yvz4 - yvz1*yvy4)
        bvy1 = (yvz1*yvx4 - yvx1*yvz4)
        bvz1 = (yvx1*yvy4 - yvy1*yvx4)
!---------------------------------------------------------------------------------------------------
        bvx1 = bvx1/DSQRT((yvy1*yvz4 - yvz1*yvy4)**2 + (yvz1*yvx4 - yvx1*yvz4)**2 + (yvx1*yvy4 - yvy1*yvx4)**2)
        bvy1 = bvy1/DSQRT((yvy1*yvz4 - yvz1*yvy4)**2 + (yvz1*yvx4 - yvx1*yvz4)**2 + (yvx1*yvy4 - yvy1*yvx4)**2)
        bvz1 = bvz1/DSQRT((yvy1*yvz4 - yvz1*yvy4)**2 + (yvz1*yvx4 - yvx1*yvz4)**2 + (yvx1*yvy4 - yvy1*yvx4)**2)
!---------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 4
!---------------------------------------------------------------------------------------------------
        bvx2 = (yvy4*yvz2 - yvz4*yvy2)
        bvy2 = (yvz4*yvx2 - yvx4*yvz2)
        bvz2 = (yvx4*yvy2 - yvy4*yvx2)
!---------------------------------------------------------------------------------------------------
        bvx2 = bvx2/DSQRT((yvy4*yvz2 - yvz4*yvy2)**2 +(yvz4*yvx2 - yvx4*yvz2)**2 +(yvx4*yvy2 - yvy4*yvx2)**2 )
        bvy2 = bvy2/DSQRT((yvy4*yvz2 - yvz4*yvy2)**2 +(yvz4*yvx2 - yvx4*yvz2)**2 +(yvx4*yvy2 - yvy4*yvx2)**2 )
        bvz2 = bvz2/DSQRT((yvy4*yvz2 - yvz4*yvy2)**2 +(yvz4*yvx2 - yvx4*yvz2)**2 +(yvx4*yvy2 - yvy4*yvx2)**2 )
!---------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2
!---------------------------------------------------------------------------------------------------
        bvx3 = (yvy2*yvz5 - yvz2*yvy5)
        bvy3 = (yvz2*yvx5 - yvx2*yvz5)
        bvz3 = (yvx2*yvy5 - yvy2*yvx5)
!---------------------------------------------------------------------------------------------------
        bvx3 = bvx3/DSQRT((yvy2*yvz5 - yvz2*yvy5)**2 +(yvz2*yvx5 - yvx2*yvz5)**2 +(yvx2*yvy5 - yvy2*yvx5)**2 )
        bvy3 = bvy3/DSQRT((yvy2*yvz5 - yvz2*yvy5)**2 +(yvz2*yvx5 - yvx2*yvz5)**2 +(yvx2*yvy5 - yvy2*yvx5)**2 )
        bvz3 = bvz3/DSQRT((yvy2*yvz5 - yvz2*yvy5)**2 +(yvz2*yvx5 - yvx2*yvz5)**2 +(yvx2*yvy5 - yvy2*yvx5)**2 )
!---------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 5
!---------------------------------------------------------------------------------------------------
        bvx4 = (yvy5*yvz3 - yvz5*yvy3)
        bvy4 = (yvz5*yvx3 - yvx5*yvz3)
        bvz4 = (yvx5*yvy3 - yvy5*yvx3)
!---------------------------------------------------------------------------------------------------
        bvx4 = bvx4/DSQRT((yvy5*yvz3 - yvz5*yvy3)**2 +(yvz5*yvx3 - yvx5*yvz3)**2 +(yvx5*yvy3 - yvy5*yvx3)**2 )
        bvy4 = bvy4/DSQRT((yvy5*yvz3 - yvz5*yvy3)**2 +(yvz5*yvx3 - yvx5*yvz3)**2 +(yvx5*yvy3 - yvy5*yvx3)**2 )
        bvz4 = bvz4/DSQRT((yvy5*yvz3 - yvz5*yvy3)**2 +(yvz5*yvx3 - yvx5*yvz3)**2 +(yvx5*yvy3 - yvy5*yvx3)**2 )
!---------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3
!---------------------------------------------------------------------------------------------------
        bvx5 = (yvy3*yvz6 - yvz3*yvy6)
```

```
        bvy5 = (yvz3*yvx6 - yvx3*yvz6)
        bvz5 = (yvx3*yvy6 - yvy3*yvx6)
!-----------------------------------------------------------------------------------------------------
        bvx5 = bvx5/DSQRT((yvy3*yvz6 - yvz3*yvy6)**2 +(yvz3*yvx6 - yvx3*yvz6)**2 +(yvx3*yvy6 - yvy3*yvx6)**2 )
        bvy5 = bvy5/DSQRT((yvy3*yvz6 - yvz3*yvy6)**2 +(yvz3*yvx6 - yvx3*yvz6)**2 +(yvx3*yvy6 - yvy3*yvx6)**2 )
        bvz5 = bvz5/DSQRT((yvy3*yvz6 - yvz3*yvy6)**2 +(yvz3*yvx6 - yvx3*yvz6)**2 +(yvx3*yvy6 - yvy3*yvx6)**2 )
!-----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 6
!-----------------------------------------------------------------------------------------------------
        bvx6 = (yvy6*yvz1 - yvz6*yvy1)
        bvy6 = (yvz6*yvx1 - yvx6*yvz1)
        bvz6 = (yvx6*yvy1 - yvy6*yvx1)
!-----------------------------------------------------------------------------------------------------
        bvx6 = bvx6/DSQRT((yvy6*yvz1 - yvz6*yvy1)**2 +(yvz6*yvx1 - yvx6*yvz1)**2 +(yvx6*yvy1 - yvy6*yvx1)**2 )
        bvy6 = bvy6/DSQRT((yvy6*yvz1 - yvz6*yvy1)**2 +(yvz6*yvx1 - yvx6*yvz1)**2 +(yvx6*yvy1 - yvy6*yvx1)**2 )
        bvz6 = bvz6/DSQRT((yvy6*yvz1 - yvz6*yvy1)**2 +(yvz6*yvx1 - yvx6*yvz1)**2 +(yvx6*yvy1 - yvy6*yvx1)**2 )
!-----------------------------------------------------------------------------------------------------
!  Estimation along point 1
!-----------------------------------------------------------------------------------------------------
        GEx  = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)*DSIN(be1) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be1))*(1.0D0 + DCOS(ga1)))/(DSIN(be1)*DSIN(ga1)) )**2 ) )*bvx1
        GEy  = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)*DSIN(be1) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be1))*(1.0D0 + DCOS(ga1)))/(DSIN(be1)*DSIN(ga1)) )**2 ) )*bvy1
        GEz  = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)*DSIN(be1) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be1))*(1.0D0 + DCOS(ga1)))/(DSIN(be1)*DSIN(ga1)) )**2 ) )*bvz1
!-----------------------------------------------------------------------------------------------------
!  Estimation along point 4
!-----------------------------------------------------------------------------------------------------
        GEx  = GEx + DSQRT(yvx4**2 + yvy4**2 + yvz4**2)*DSIN(be2) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be2))*(1.0D0 + DCOS(ga2)))/(DSIN(be2)*DSIN(ga2)) )**2 ) )*bvx2
        GEy  = GEy + DSQRT(yvx4**2 + yvy4**2 + yvz4**2)*DSIN(be2) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be2))*(1.0D0 + DCOS(ga2)))/(DSIN(be2)*DSIN(ga2)) )**2 ) )*bvy2
        GEz  = GEz + DSQRT(yvx4**2 + yvy4**2 + yvz4**2)*DSIN(be2) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be2))*(1.0D0 + DCOS(ga2)))/(DSIN(be2)*DSIN(ga2)) )**2 ) )*bvz2
!-----------------------------------------------------------------------------------------------------
!  Estimation along point 2
!-----------------------------------------------------------------------------------------------------
        GEx  = GEx + DSQRT(yvx2**3 + yvy2**2 + yvz2**2)*DSIN(be3) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be3))*(1.0D0 + DCOS(ga3)))/(DSIN(be3)*DSIN(ga3)) )**2 ) )*bvx3
        GEy  = GEy + DSQRT(yvx2**3 + yvy2**2 + yvz2**2)*DSIN(be3) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be3))*(1.0D0 + DCOS(ga3)))/(DSIN(be3)*DSIN(ga3)) )**2 ) )*bvy3
        GEz  = GEz + DSQRT(yvx2**3 + yvy2**2 + yvz2**2)*DSIN(be3) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be3))*(1.0D0 + DCOS(ga3)))/(DSIN(be3)*DSIN(ga3)) )**2 ) )*bvz3
!-----------------------------------------------------------------------------------------------------
!  Estimation along point 5
!-----------------------------------------------------------------------------------------------------
        GEx  = GEx + DSQRT(yvx5**2 + yvy5**2 + yvz5**2)*DSIN(be4) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be4))*(1.0D0 + DCOS(ga4)))/(DSIN(be4)*DSIN(ga4)) )**2 ) )*bvx4
        GEy  = GEy + DSQRT(yvx5**2 + yvy5**2 + yvz5**2)*DSIN(be4) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be4))*(1.0D0 + DCOS(ga4)))/(DSIN(be4)*DSIN(ga4)) )**2 ) )*bvy4
        GEz  = GEz + DSQRT(yvx5**2 + yvy5**2 + yvz5**2)*DSIN(be4) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be4))*(1.0D0 + DCOS(ga4)))/(DSIN(be4)*DSIN(ga4)) )**2 ) )*bvz4
!-----------------------------------------------------------------------------------------------------
!  Estimation along point 3
!-----------------------------------------------------------------------------------------------------
        GEx  = GEx + DSQRT(yvx3**2 + yvy3**2 + yvz3**2)*DSIN(be5) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be5))*(1.0D0 + DCOS(ga5)))/(DSIN(be5)*DSIN(ga5)) )**2 ) )*bvx5
        GEy  = GEy + DSQRT(yvx3**2 + yvy3**2 + yvz3**2)*DSIN(be5) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be5))*(1.0D0 + DCOS(ga5)))/(DSIN(be5)*DSIN(ga5)) )**2 ) )*bvy5
        GEz  = GEz + DSQRT(yvx3**2 + yvy3**2 + yvz3**2)*DSIN(be5) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be5))*(1.0D0 + DCOS(ga5)))/(DSIN(be5)*DSIN(ga5)) )**2 ) )*bvz5
!-----------------------------------------------------------------------------------------------------
!  Estimation along point 6
!-----------------------------------------------------------------------------------------------------
        GEx  = GEx + DSQRT(yvx6**2 + yvy6**2 + yvz6**2)*DSIN(be6) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be6))*(1.0D0 + DCOS(ga6)))/(DSIN(be6)*DSIN(ga6)) )**2 ) )*bvx6
        GEy  = GEy + DSQRT(yvx6**2 + yvy6**2 + yvz6**2)*DSIN(be6) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be6))*(1.0D0 + DCOS(ga6)))/(DSIN(be6)*DSIN(ga6)) )**2 ) )*bvy6
        GEz  = GEz + DSQRT(yvx6**2 + yvy6**2 + yvz6**2)*DSIN(be6) *&
            &    DLOG( DSQRT( ( ((1.0D0 + DCOS(be6))*(1.0D0 + DCOS(ga6)))/(DSIN(be6)*DSIN(ga6)) )**2 ) )*bvz6
!-----------------------------------------------------------------------------------------------------
    GEx = GEx*crvmel(k)
```



```
    GEy = GEy*crvmel(k)
    GEz = GEz*crvmel(k)
!=====================================================================================================
    END SUBROUTINE intr_lin_sing5
!=====================================================================================================
!=====================================================================================================
 END MODULE Mod_Prtcl3D_SLP
```

```
MODULE Mod_sgf_3d_fs
!===========================================================================================================
!   Version: 0.5 created on  26 / IX / 2007
!                                                                                              C. Pozrikidis
!-----------------------------------------------------------------------------------------------------------
!   Version: 0.7 created on  -- / III /2010
!                                                                                  Marco Antonio Reyes Huesca
!-----------------------------------------------------------------------------------------------------------
!   Version: 0.8 created on 22 / 03 / 2012
!   Version: 0.9 created on 03 / 09 / 2012
!   Version: 1.0 created on 13 / 11 / 2012
!                                                                                     Alfredo Sanjuan Sanjuan
!===========================================================================================================
    CONTAINS
!===========================================================================================================
    SUBROUTINE sgf_3d_fs(x,    y,    z,   &
                       & x0,   y0,   z0,  &
                       & Gxx,  Gxy,  Gxz, &
                       & Gyx,  Gyy,  Gyz, &
                       & Gzx,  Gzy,  Gzz  )

!===========================================================================================================
!   Free-space Green's function: Stokeslet Pozrikidis (1992, p. 23)
!   This new version of sfg_3d_fs only calculates the Green's function: Stokeslet. The stress asociated is
!calculated in other module.
!===========================================================================================================
    USE Mod_SharedVars, ONLY: DBL, ULog
!===========================================================================================================
    IMPLICIT NONE
!===========================================================================================================
!   Variables
!-----------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN)    :: x,   y,   z      !coordinates of collocation point of the element
    REAL (KIND = DBL), INTENT(IN)    :: x0,  y0,  z0     !coordinates of the singularity
    REAL (KIND = DBL), INTENT(INOUT) :: Gxx, Gxy, Gxz, & !Free-space Green´s function of Stokeslet. integrated
    &                                   Gyx, Gyy, Gyz, & !ij component over the element
    &                                   Gzx, Gzy, Gzz
!===========================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: dx,  dy,  dz       !differences between the (x,y,z) and (x0,y0,z0) coordinates
    REAL (KIND = DBL) :: dxx, dxy, dxz,  &  !square distance di*dj=dij
    &                    dyy, dyz, dzz
    REAL (KIND = DBL) :: r, r3              !distance r between the (x,y,z) point and (x0,y0,z0) and r**3
    REAL (KIND = DBL) :: ri, ri3            !inverse of distance r  and inverse of r**3
!===========================================================================================================
!   Initialize
    Gxx = 0.0D0
    Gxy = 0.0D0
    Gxz = 0.0D0
    Gyx = 0.0D0
    Gyy = 0.0D0
    Gyz = 0.0D0
    Gzx = 0.0D0
    Gzy = 0.0D0
    Gzz = 0.0D0
    dx  = 0.0D0
    dy  = 0.0D0
    dz  = 0.0D0
!-----------------------------------------------------------------------------------------------------------
    dxx = 0.0D0
    dxy = 0.0D0
    dxz = 0.0D0
    dyy = 0.0D0
    dyz = 0.0D0
    dzz = 0.0D0
!-----------------------------------------------------------------------------------------------------------
    r   = 0.0D0
    r3  = 0.0D0
    ri  = 0.0D0
    ri3 = 0.0D0
!-----------------------------------------------------------------------------------------------------------
```



```
!Start the magic
    dx  = (x-x0 )
    dy  = (y-y0 )
    dz  = (z-z0 )
!-----------------------------------------------------------------------------------------------------------
    dxx = dx*dx
    dxy = dx*dy
    dxz = dx*dz
    dyy = dy*dy
    dyz = dy*dz
    dzz = dz*dz
!-----------------------------------------------------------------------------------------------------------
    r   = DSQRT(dxx + dyy + dzz)
    r3  = r**3
    ri  = 1.0D0/r
    ri3 = 1.0D0/r3
!-----------------------------------------------------------------------------------------------------------
    Gxx = ri + dxx*ri3
    Gxy =      dxy*ri3
    Gxz =      dxz*ri3
    Gyy = ri + dyy*ri3
    Gyz =      dyz*ri3
    Gzz = ri + dzz*ri3
!-----------------------------------------------------------------------------------------------------------
    Gyx = Gxy
    Gzx = Gxz
    Gzy = Gyz
!-----------------------------------------------------------------------------------------------------------
! Done
!-----------------------------------------------------------------------------------------------------------
END SUBROUTINE sgf_3d_fs
!===========================================================================================================
!===========================================================================================================
    SUBROUTINE sgf_3d_fsing(x,    y,    z,   &
                          & x0,   y0,   z0,  &
                          & Gxx,  Gxy,  Gxz, &
                          & Gyx,  Gyy,  Gyz, &
                          & Gzx,  Gzy,  Gzz, &
                          & ABS)

!===========================================================================================================
!   Free-space Green's function: Stokeslet Pozrikidis (1992, p. 23)
!   This new version of sfg_3d_fs only calculates the Green's function: Stokeslet. The stress asociated is
!calculated in other module.
!===========================================================================================================
    USE Mod_SharedVars, ONLY: DBL, ULog
!===========================================================================================================
    IMPLICIT NONE
!===========================================================================================================
!   Variables
!-----------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN)    :: x,   y,   z      !coordinates of collocation point of the element
    REAL (KIND = DBL), INTENT(IN)    :: x0,  y0,  z0     !coordinates of the singularity
    REAL (KIND = DBL), INTENT(INOUT) :: Gxx, Gxy, Gxz, & !Free-space Green´s function of Stokeslet. integrated
    &                                   Gyx, Gyy, Gyz, & !ij component over the element
    &                                   Gzx, Gzy, Gzz
    REAL (KIND = DBL), INTENT(IN) :: ABS             !var on posrikidis

!===========================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: dx,  dy,  dz       !differences between the (x,y,z) and (x0,y0,z0) coordinates
    REAL (KIND = DBL) :: dxx, dxy, dxz,  &  !square distance di*dj=dij
    &                    dyy, dyz, dzz
    REAL (KIND = DBL) :: r, r3              !distance r between the (x,y,z) point and (x0,y0,z0) and r**3
    REAL (KIND = DBL) :: ri, ri3            !inverse of distance r  and inverse of r**3
!===========================================================================================================
!   Initialize
    Gxx = 0.0D0
    Gxy = 0.0D0
    Gxz = 0.0D0
    Gyx = 0.0D0
```

```fortran
    Gyy = 0.0D0
    Gyz = 0.0D0
    Gzx = 0.0D0
    Gzy = 0.0D0
    Gzz = 0.0D0
    dx  = 0.0D0
    dy  = 0.0D0
    dz  = 0.0D0
!-------------------------------------------------------------------------------------------------------------
    dxx = 0.0D0
    dxy = 0.0D0
    dxz = 0.0D0
    dyy = 0.0D0
    dyz = 0.0D0
    dzz = 0.0D0
!-------------------------------------------------------------------------------------------------------------
    r   = 0.0D0
    r3  = 0.0D0
    ri  = 0.0D0
    ri3 = 0.0D0
!-------------------------------------------------------------------------------------------------------------
!Start the magic
    dx  = (x-x0 )
    dy  = (y-y0 )
    dz  = (z-z0 )
!-------------------------------------------------------------------------------------------------------------
    dxx = dx*dx
    dxy = dx*dy
    dxz = dx*dz
    dyy = dy*dy
    dyz = dy*dz
    dzz = dz*dz
!-------------------------------------------------------------------------------------------------------------
    r   = DSQRT(dxx + dyy + dzz)
    r3  = r**3
    ri  = 1.0D0/r
    ri3 = 1.0D0/r3
!-------------------------------------------------------------------------------------------------------------
    Gxx = 1.0D0  + dxx/ABS**2
    Gxy = dxy/ABS**2
    Gxz = dxz/ABS**2
    Gyy = 1.0D0  + dyy/ABS**2
    Gyz = dyz/ABS**2
    Gzz = 1.0D0  + dzz/ABS**2
!-------------------------------------------------------------------------------------------------------------
    Gyx = Gxy
    Gzx = Gxz
    Gzy = Gyz
!-------------------------------------------------------------------------------------------------------------
! Done
!-------------------------------------------------------------------------------------------------------------
END SUBROUTINE sgf_3d_fsing
!=============================================================================================================
END MODULE Mod_sgf_3d_fs
```

```
MODULE Mod_Prtcl3D_DLP
!====================================================================================================
!   Version: 0.5 created on  26 / IX / 2007
!                                                                                       C. Pozrikidis
!----------------------------------------------------------------------------------------------------
!   Version: 0.7 created on  -- / III /2010
!                                                                            Marco Antonio Reyes Huesca
!----------------------------------------------------------------------------------------------------
!   Version: 0.9 created on 22 / 03 / 2012
!   Version: 1.0 created on 13 / 10 / 2012
!                                                                              Alfredo Sanjuan Sanjuan
!====================================================================================================
    CONTAINS
!====================================================================================================
    SUBROUTINE Intgr_Trgl_s(x0,   y0,   z0,   &
                          & x1,    y1,    z1,    &
                          & k,                &
                          & TExx, TExy, TExz, &
                          & TEyx, TEyy, TEyz, &
                          & TEzx, TEzy, TEzz, &
                          & mint)
!====================================================================================================
!   This subroutine integrates the Green's function over a non-singular triangle numbered k
!====================================================================================================
    USE Mod_SharedVars, ONLY: DBL, ULog, eps, Pi,      &
    &                         p, ne, n, nbe,           &
    &                         alphaQ, betaQ, gammaQ,   &
    &                         xiq, etq, wq,            &
    &                         Ns,Np
!====================================================================================================
    USE Mod_sgf_3d_sfs
    USE Mod_Nodal_Interp
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
    REAL (KIND = DBL), INTENT(IN)  :: x0, y0, z0           !coordinates of collocation point analysis
    REAL (KIND = DBL), INTENT(IN)  :: x1, y1, z1           !coordinates of element collocation point
    INTEGER,           INTENT(IN)  :: k                    !element index
    REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TExz, &  !Integrated ij component over the element
    &                                 TEyx, TEyy, TEyz, &
    &                                 TEzx, TEzy, TEzz
    INTEGER,           INTENT(IN)  :: mint                 !order of triangle quadrature
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6              !indices to obtain node numbers from each element
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: xi, eta                    !constants of weigth to integrate over a triangle
    REAL (KIND = DBL) :: x, y, z                    !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi, &     !Derivates of the tangential vectors over the element
    &                    DxDet, DyDet, DzDet
    REAL (KIND = DBL) :: vnx, vny, vnz              !normal vector coordinates of the element
    REAL (KIND = DBL) :: hs                         !surface metric on a triangle
    REAL (KIND = DBL) :: fc                         !integration weigth coefficient
    REAL (KIND = DBL) :: Txxx, Txxy, Txxz, &        !Free-space Green´s function of Stresslet. integrated ijk
    &                    Txyx, Txyy, Txyz, &        !component over the element
    &                    Txzx, Txzy, Txzz, &
    &                    Tyxx, Tyxy, Tyxz, &
    &                    Tyyx, Tyyy, Tyyz, &
    &                    Tyzx, Tyzy, Tyzz, &
    &                    Tzxx, Tzxy, Tzxz, &
    &                    Tzyx, Tzyy, Tzyz, &
    &                    Tzzx, Tzzy, Tzzz
!----------------------------------------------------------------------------------------------------
!   Initialize
    TExx = 0.0D0
    TExy = 0.0D0
    TExz = 0.0D0
!----------------------------------------------------------------------------------------------------
    TEyx = 0.0D0
```



```
    TEyy = 0.0D0
    TEyz = 0.0D0
!----------------------------------------------------------------------------------------------------
    TEzx = 0.0D0
    TEzy = 0.0D0
    TEzz = 0.0D0
!----------------------------------------------------------------------------------------------------
!   vertices of the kth triangle
    i1 = n(k,1)
    i2 = n(k,2)
    i3 = n(k,3)
    i4 = n(k,4)
    i5 = n(k,5)
    i6 = n(k,6)
!----------------------------------------------------------------------------------------------------
!    DO i = 1, mint
!        xi  = xiq(i)
!        eta = etq(i)
!!---------------------------------------------------------------------------------------------------
!        CALL interp_p(p(i1,1),   p(i1,2),  p(i1,3),    &
!        &             p(i2,1),   p(i2,2),  p(i2,3),    &
!        &             p(i3,1),   p(i3,2),  p(i3,3),    &
!        &             p(i4,1),   p(i4,2),  p(i4,3),    &
!        &             p(i5,1),   p(i5,2),  p(i5,3),    &
!        &             p(i6,1),   p(i6,2),  p(i6,3),    &
!        &             alphaQ(k), betaQ(k), gammaQ(k),  &
!        &             xi,        eta,                  &
!        &             x,         y,         z,         &
!        &             DxDxi,     DyDxi,    DzDxi,      &
!        &             DxDet,     DyDet,    DzDet,      &
!        &             vnx,       vny,      vnz,        &
!        &             hs)
!!---------------------------------------------------------------------------------------------------
!        CALL sgf_3d_sfs( x,   y,   z,     &
!                      &  x0,  y0,  z0,    &
!                      &  Txxx,Txxy,Txxz,  &
!                      &  Txyx,Txyy,Txyz,  &
!                      &  Txzx,Txzy,Txzz,  &
!                      &  Tyxx,Tyxy,Tyxz,  &
!                      &  Tyyx,Tyyy,Tyyz,  &
!                      &  Tyzx,Tyzy,Tyzz,  &
!                      &  Tzxx,Tzxy,Tzxz,  &
!                      &  Tzyx,Tzyy,Tzyz,  &
!                      &  Tzzx,Tzzy,Tzzz   )
!!---------------------------------------------------------------------------------------------------
!        fc = 0.5D0*hs*wq(i)
!!---------------------------------------------------------------------------------------------------
!        TExx = TExx + (Txxx*vnx*fc + Txxy*vny*fc + Txxz*vnz*fc)
!        TExy = TExy + (Txyx*vnx*fc + Txyy*vny*fc + Txyz*vnz*fc)
!        TExz = TExz + (Txzx*vnx*fc + Txzy*vny*fc + Txzz*vnz*fc)
!        TEyx = TEyx + (Tyxx*vnx*fc + Tyxy*vny*fc + Tyxz*vnz*fc)
!        TEyy = TEyy + (Tyyx*vnx*fc + Tyyy*vny*fc + Tyyz*vnz*fc)
!        TEyz = TEyz + (Tyzx*vnx*fc + Tyzy*vny*fc + Tyzz*vnz*fc)
!        TEzx = TEzx + (Tzxx*vnx*fc + Tzxy*vny*fc + Tzxz*vnz*fc)
!        TEzy = TEzy + (Tzyx*vnx*fc + Tzyy*vny*fc + Tzyz*vnz*fc)
!        TEzz = TEzz + (Tzzx*vnx*fc + Tzzy*vny*fc + Tzzz*vnz*fc)
!    END DO
!***************************************************************************************************
! Integrate over six flat triangles
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing_s2( x0,     y0,     z0,     &
                        &  x1,     y1,     z1,     &
                        &  p(i1,1),p(i1,2),p(i1,3),&
                        &  p(i4,1),p(i4,2),p(i4,3),&
                        &  mint,                   &
                        &  TExx,   TExy,   TExz,   &
                        &  TEyx,   TEyy,   TEyz,   &
                        &  TEzx,   TEzy,   TEzz    )
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing_s2( x0,     y0,     z0,     &
                        &  x1,     y1,     z1,     &
                        &  p(i4,1),p(i4,2),p(i4,3),&
```

```
                        &  p(i2,1),p(i2,2),p(i2,3),&
                        &  mint,                   &
                        &  TExx,   TExy,   TExz,   &
                        &  TEyx,   TEyy,   TEyz,   &
                        &  TEzx,   TEzy,   TEzz    )
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing_s2( x0,     y0,     z0,     &
                        &  x1,     y1,     z1,     &
                        &  p(i2,1),p(i2,2),p(i2,3),&
                        &  p(i5,1),p(i5,2),p(i5,3),&
                        &  mint,                   &
                        &  TExx,   TExy,   TExz,   &
                        &  TEyx,   TEyy,   TEyz,   &
                        &  TEzx,   TEzy,   TEzz    )
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing_s2( x0,     y0,     z0,     &
                        &  x1,     y1,     z1,     &
                        &  p(i5,1),p(i5,2),p(i5,3),&
                        &  p(i3,1),p(i3,2),p(i3,3),&
                        &  mint,                   &
                        &  TExx,   TExy,   TExz,   &
                        &  TEyx,   TEyy,   TEyz,   &
                        &  TEzx,   TEzy,   TEzz    )
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing_s2( x0,     y0,     z0,     &
                        &  x1,     y1,     z1,     &
                        &  p(i3,1),p(i3,2),p(i3,3),&
                        &  p(i6,1),p(i6,2),p(i6,3),&
                        &  mint,                   &
                        &  TExx,   TExy,   TExz,   &
                        &  TEyx,   TEyy,   TEyz,   &
                        &  TEzx,   TEzy,   TEzz    )
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing_s2( x0,     y0,     z0,     &
                        &  x1,     y1,     z1,     &
                        &  p(i6,1),p(i6,2),p(i6,3),&
                        &  p(i1,1),p(i1,2),p(i1,3),&
                        &  mint,                   &
                        &  TExx,   TExy,   TExz,   &
                        &  TEyx,   TEyy,   TEyz,   &
                        &  TEzx,   TEzy,   TEzz    )
!---------------------------------------------------------------------------------------------------
! DOne
!---------------------------------------------------------------------------------------------------
    END SUBROUTINE Intgr_Trgl_s
!===================================================================================================
!===================================================================================================
    SUBROUTINE Intgr_Trgl_Sing_s(x0,   y0,   z0,   &
                               & k,                &
                               & TExx, TExy, TExz, &
                               & TEyx, TEyy, TEyz, &
                               & TEzx, TEzy, TEzz, &
                               & mint               )
!===================================================================================================
!   Integrate the Green's function over the kth singular quadratic triangle. This is done by breaking up the
!singular triangle into six flat triangles, and then integrating individually over the flat triangles in local
!polar coordinates.
!===================================================================================================
    USE Mod_SharedVars, ONLY: DBL, ULog, eps, Pi,   &
    &                         p, ne, n, nbe,        &
    &                         xiq, etq, wq,         &
    &                         Ns,Np
!===================================================================================================
    IMPLICIT NONE
!===================================================================================================
    REAL (KIND = DBL), INTENT(IN)  :: x0, y0, z0           !coordinates of collocationpoint of the element
    INTEGER,           INTENT(IN)  :: k                    !element index
    REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TExz,  & !Integrated ij component over the element
    &                                 TEyx, TEyy, TEyz,  &
    &                                 TEzx, TEzy, TEzz
    INTEGER,           INTENT(IN)  :: mint                  !order of Gauss-Legendre quadrature
!===================================================================================================
```



```
!===================================================================================================
!   Variables inside the subroutine
!---------------------------------------------------------------------------------------------------
    INTEGER  :: i, j
    INTEGER  :: i1, i2, i3, i4, i5, i6
!===================================================================================================
!   launching
    i1 = n(k,1)
    i2 = n(k,2)
    i3 = n(k,3)
    i4 = n(k,4)
    i5 = n(k,5)
    i6 = n(k,6)
!---------------------------------------------------------------------------------------------------
    TExx = 0.0D0
    TExy = 0.0D0
    TExz = 0.0D0
!---------------------------------------------------------------------------------------------------
    TEyx = 0.0D0
    TEyy = 0.0D0
    TEyz = 0.0D0
!---------------------------------------------------------------------------------------------------
    TEzx = 0.0D0
    TEzy = 0.0D0
    TEzz = 0.0D0
!===================================================================================================
!   Integrate over three flat triangles
!---------------------------------------------------------------------------------------------------
!     CALL intr_lin_sing_s( x0,     y0,     z0,     &
!                        &  p(i1,1),p(i1,2),p(i1,3),&
!                        &  p(i2,1),p(i2,2),p(i2,3),&
!                        &  TExx,   TExy,   TExz,   &
!                        &  TEyx,   TEyy,   TEyz,   &
!                        &  TEzx,   TEzy,   TEzz,   &
!                        &  cdt,    NGL             )
!---------------------------------------------------------------------------------------------------
!     CALL intr_lin_sing_s( x0,     y0,     z0,     &
!                        &  p(i2,1),p(i2,2),p(i2,3),&
!                        &  p(i3,1),p(i3,2),p(i3,3),&
!                        &  TExx,   TExy,   TExz,   &
!                        &  TEyx,   TEyy,   TEyz,   &
!                        &  TEzx,   TEzy,   TEzz,   &
!                        &  cdt,    NGL             )
!---------------------------------------------------------------------------------------------------
!     CALL intr_lin_sing_s( x0,     y0,     z0,     &
!                        &  p(i3,1),p(i3,2),p(i3,3),&
!                        &  p(i1,1),p(i1,2),p(i1,3),&
!                        &  TExx,   TExy,   TExz,   &
!                        &  TEyx,   TEyy,   TEyz,   &
!                        &  TEzx,   TEzy,   TEzz,   &
!                        &  cdt,    NGL             )
!===================================================================================================
! Integrate over six flat triangles
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing_s( x0,     y0,     z0,     &
                       &  p(i1,1),p(i1,2),p(i1,3),&
                       &  p(i4,1),p(i4,2),p(i4,3),&
                       &  TExx,   TExy,   TExz,   &
                       &  TEyx,   TEyy,   TEyz,   &
                       &  TEzx,   TEzy,   TEzz,   &
                       &  mint                     )
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing_s( x0,     y0,     z0,     &
                       &  p(i4,1),p(i4,2),p(i4,3),&
                       &  p(i2,1),p(i2,2),p(i2,3),&
                       &  TExx,   TExy,   TExz,   &
                       &  TEyx,   TEyy,   TEyz,   &
                       &  TEzx,   TEzy,   TEzz,   &
                       &  mint                     )
!---------------------------------------------------------------------------------------------------
    CALL intr_lin_sing_s( x0,     y0,     z0,     &
                       &  p(i2,1),p(i2,2),p(i2,3),&
```

```fortran
                        &  p(i5,1),p(i5,2),p(i5,3),&
                        &  TExx,   TExy,   TExz,   &
                        &  TEyx,   TEyy,   TEyz,   &
                        &  TEzx,   TEzy,   TEzz,   &
                        &  mint                      )
!----------------------------------------------------------------------------------------------------
    CALL intr_lin_sing_s( x0,     y0,     z0,     &
                        &  p(i5,1),p(i5,2),p(i5,3),&
                        &  p(i3,1),p(i3,2),p(i3,3),&
                        &  TExx,   TExy,   TExz,   &
                        &  TEyx,   TEyy,   TEyz,   &
                        &  TEzx,   TEzy,   TEzz,   &
                        &  mint                      )
!----------------------------------------------------------------------------------------------------
    CALL intr_lin_sing_s( x0,     y0,     z0,     &
                        &  p(i3,1),p(i3,2),p(i3,3),&
                        &  p(i6,1),p(i6,2),p(i6,3),&
                        &  TExx,   TExy,   TExz,   &
                        &  TEyx,   TEyy,   TEyz,   &
                        &  TEzx,   TEzy,   TEzz,   &
                        &  mint                      )
!----------------------------------------------------------------------------------------------------
    CALL intr_lin_sing_s( x0,     y0,     z0,     &
                        &  p(i6,1),p(i6,2),p(i6,3),&
                        &  p(i1,1),p(i1,2),p(i1,3),&
                        &  TExx,   TExy,   TExz,   &
                        &  TEyx,   TEyy,   TEyz,   &
                        &  TEzx,   TEzy,   TEzz,   &
                        &  mint                      )
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE Intgr_Trgl_Sing_s
!====================================================================================================
!====================================================================================================
    SUBROUTINE intr_lin_sing_s(x1,   y1,   z1,   &
                            & x2,   y2,   z2,   &
                            & x3,   y3,   z3,   &
                            & TExx, TExy, TExz, &
                            & TEyx, TEyy, TEyz, &
                            & TEzx, TEzy, TEzz, &
                            & mint             )
!====================================================================================================
!   Integrates the Green's function over a flat triangle in local polar coordinates with origin at the singular
!point: (x1,y1,z1). The subrotine is based from Pozriquidis 2002 pp. 119-120 in the technique 5.2.8 - 5.2.11
!====================================================================================================
    USE Mod_SharedVars, ONLY: DBL, ULog, eps, Pi,   &
    &                         ZZ, WW,               &
    &                         Ns, Np, xiq, etq, wq
    USE Mod_sgf_3d_sfs
!====================================================================================================
   IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN)    :: x1, y1, z1          !coordinates of flat triangle vertice
    REAL (KIND = DBL), INTENT(IN)    :: x2, y2, z2
    REAL (KIND = DBL), INTENT(IN)    :: x3, y3, z3
    REAL (KIND = DBL), INTENT(INOUT) :: TExx, TExy, TExz, & !Integrated ij component over the element
    &                                   TEyx, TEyy, TEyz, &
    &                                   TEzx, TEzy, TEzz
    INTEGER,           INTENT(IN)    :: mint                 !order of Gauss-Legendre quadrature
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                            !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6       !indices to obtain the node number in the vertices of each element
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: pi4, piq              !constants to integrate over a circle element in Phi and r
    REAL (KIND = DBL) :: xi, et, zt            !constants of weigth to integrate over a flat triangle
    REAL (KIND = DBL) :: dx, dy                !surface metrics
    REAL (KIND = DBL) :: vnx, vny, vnz         !triangle normal vector
    REAL (KIND = DBL) :: area, hs              !triangle area and surface metric on a flat triangle
    REAL (KIND = DBL) :: ph, cph, sph          !variable Phi, Cos(phi) and Sin(Phi)
```



```fortran
    REAL (KIND = DBL) :: r, rmax, rmaxh         !variable r and values of variable r to integrate
    REAL (KIND = DBL) :: x, y, z                !coordinates of f(xi, et, zt)
    REAL (KIND = DBL) :: cf, cf1, cf2           !integrate weigths
    REAL (KIND = DBL) :: asm, bsm               !triangle area and its derivative
    REAL (KIND = DBL) :: B, C                   !integrate arc coefficients
    REAL (KIND = DBL) :: b1, b2, c1             !dummy integrate arc constants
    REAL (KIND = DBL), DIMENSION(3) ::xxi, eet, zzt          !variables of weigth over in triangle (xi,eta)
    REAL (KIND = DBL) :: Txxx, Txxy, Txxz, &    !Free-space Green´s function of Stresslet. integrated ijk
    &                    Txyx, Txyy, Txyz, &    !component over the element
    &                    Txzx, Txzy, Txzz, &
    &                    Tyxx, Tyxy, Tyxz, &
    &                    Tyyx, Tyyy, Tyyz, &
    &                    Tyzx, Tyzy, Tyzz, &
    &                    Tzxx, Tzxy, Tzxz, &
    &                    Tzyx, Tzyy, Tzyz, &
    &                    Tzzx, Tzzy, Tzzz
    REAL (KIND = DBL) :: Txx, Txy, Txz, &       !Stresslets Dummy coefficients
    &                    Tyx, Tyy, Tyz, &
    &                    Tzx, Tzy, Tzz
    REAL (KIND = DBL) :: Rxx, Rxy, Rxz, &       !Stresslets Dummy coefficients
    &                    Ryx, Ryy, Ryz, &
    &                    Rzx, Rzy, Rzz

!====================================================================================================
!   constants
      piq = 0.25D0*pi
      pi4 = 4.0D0*pi
!----------------------------------------------------------------------------------------------------
!   Initialize
    xxi(1) = 0.5D0
    eet(1) = 0.5D0
    zzt(1) = 0.0D0
    xxi(2) = 0.5D0
    eet(2) = 0.0D0
    zzt(2) = 0.5D0
    xxi(3) = 0.0D0
    eet(3) = 0.5D0
    zzt(3) = 0.5D0
!----------------------------------------------------------------------------------------------------
    Txx = 0.0D0
    Txy = 0.0D0
    Txz = 0.0D0
    Tyx = 0.0D0
    Tyy = 0.0D0
    Tyz = 0.0D0
    Tzx = 0.0D0
    Tzy = 0.0D0
    Tzz = 0.0D0
!----------------------------------------------------------------------------------------------------
    Rxx = 0.0D0
    Rxy = 0.0D0
    Rxz = 0.0D0
    Ryx = 0.0D0
    Ryy = 0.0D0
    Ryz = 0.0D0
    Rzx = 0.0D0
    Rzy = 0.0D0
    Rzz = 0.0D0
!----------------------------------------------------------------------------------------------------
!   compute surface metric: hs
!----------------------------------------------------------------------------------------------------
    dx = DSQRT( (x2-x1)**2+(y2-y1)**2+(z2-z1)**2 )
    dy = DSQRT( (x3-x1)**2+(y3-y1)**2+(z3-z1)**2 )
!----------------------------------------------------------------------------------------------------
    vnx = ((y2-y1)*(z3-z1))-((z2-z1)*(y3-y1))
    vny = ((z2-z1)*(x3-x1))-((x2-x1)*(z3-z1))
    vnz = ((x2-x1)*(y3-y1))-((y2-y1)*(x3-x1))
!----------------------------------------------------------------------------------------------------
    hs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
    vnx = vnx/hs
    vny = vny/hs
    vnz = vnz/hs
```

```
!----------------------------------------------------------------------------------------------------
!   Surface metric on a flat triangle
    area = hs/2.0D0
!----------------------------------------------------------------------------------------------------
!   Initialize
!----------------------------------------------------------------------------------------------------
!   Triangle area
    asm = 0.0D0
!----------------------------------------------------------------------------------------------------
!!   Apply the forulas from Pozrikidis (2002, p. 119)
!    b1=((x3-x1)*(x2-x1))+((y3-y1)*(y2-y1))+((z3-z1)*(z2-z1))
!    b2=DSQRT((x2-x1)**2+(y2-y1)**2+(z2-z1)**2)
!    c1=DSQRT((x3-x1)**2+(y3-y1)**2+(z3-z1)**2)
!    B= b1/b2**2
!    C= c1**2/b2**2
!!----------------------------------------------------------------------------------------------------
!!   Apply the Double quadrature
!!----------------------------------------------------------------------------------------------------
!!   Integration wrt phi
!    Cycle1: DO i = 1, NGL
!        ph    = piq*(1.0D0+zz(i))
!        cph   = DCOS(ph)
!        sph   = DSIN(ph)
!        rmax  = 1.0D0/(cph+sph)
!        rmaxh = 0.5D0*rmax
!!----------------------------------------------------------------------------------------------------
!!   Derivative of asm
!        bsm = 0.0D0
!!----------------------------------------------------------------------------------------------------
!!   Integration wrt r
!        Cycle2: DO j=1,NGL
!            r  = rmaxh*(1.0D0+zz(j))
!            xi = r*cph
!            et = r*sph
!            zt = 1.0D0-xi-et
!!----------------------------------------------------------------------------------------------------
!            x = x1*zt + x2*xi + x3*et
!            y = y1*zt + y2*xi + y3*et
!            z = z1*zt + z2*xi + z3*et
!!----------------------------------------------------------------------------------------------------
!            CALL sgf_3d_sfs( x,   y,   z,     &
!                          &  x1,  y1,  z1,    &
!                          &  Txxx,Txxy,Txxz,  &
!                          &  Txyx,Txyy,Txyz,  &
!                          &  Txzx,Txzy,Txzz,  &
!                          &  Tyxx,Tyxy,Tyxz,  &
!                          &  Tyyx,Tyyy,Tyyz,  &
!                          &  Tyzx,Tyzy,Tyzz,  &
!                          &  Tzxx,Tzxy,Tzxz,  &
!                          &  Tzyx,Tzyy,Tzyz,  &
!                          &  Tzzx,Tzzy,Tzzz   )
!!----------------------------------------------------------------------------------------------------
!            cf1 = ww(j)*r
!!----------------------------------------------------------------------------------------------------
!            bsm = bsm + cf1
!!----------------------------------------------------------------------------------------------------
!            Rxx = Rxx + (Txxx*vnx + Txxy*vny + Txxz*vnz)*cf1
!            Rxy = Rxy + (Txyx*vnx + Txyy*vny + Txyz*vnz)*cf1
!            Rxz = Rxz + (Txzx*vnx + Txzy*vny + Txzz*vnz)*cf1
!            Ryx = Ryx + (Tyxx*vnx + Tyxy*vny + Tyxz*vnz)*cf1
!            Ryy = Ryy + (Tyyx*vnx + Tyyy*vny + Tyyz*vnz)*cf1
!            Ryz = Ryz + (Tyzx*vnx + Tyzy*vny + Tyzz*vnz)*cf1
!            Rzx = Rzx + (Tzxx*vnx + Tzxy*vny + Tzxz*vnz)*cf1
!            Rzy = Rzy + (Tzyx*vnx + Tzyy*vny + Tzyz*vnz)*cf1
!            Rzz = Rzz + (Tzzx*vnx + Tzzy*vny + Tzzz*vnz)*cf1
!        END DO Cycle2
!!----------------------------------------------------------------------------------------------------
!        cf2 = ww(i)*rmaxh
!!----------------------------------------------------------------------------------------------------
!        asm = asm + bsm*cf2
!!----------------------------------------------------------------------------------------------------
!        Txx = Txx + (cf2*Rxx/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
```



```
!        Txy = Txy + (cf2*Rxy/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!        Txz = Txz + (cf2*Rxz/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!        Tyx = Tyx + (cf2*Ryx/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!        Tyy = Tyy + (cf2*Ryy/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!        Tyz = Tyz + (cf2*Ryz/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!        Tzx = Tzx + (cf2*Rzx/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!        Tzy = Tzy + (cf2*Rzy/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!        Tzz = Tzz + (cf2*Rzz/DSQRT(cph**2+(B*DSIN(2*ph))+(C*sph**2)))
!!----------------------------------------------------------------------------------------------------
!!   initialize again to the next triangle
!        Rxx = 0.0D0
!        Rxy = 0.0D0
!        Rxz = 0.0D0
!        Ryx = 0.0D0
!        Ryy = 0.0D0
!        Ryz = 0.0D0
!        Rzx = 0.0D0
!        Rzy = 0.0D0
!        Rzz = 0.0D0
!    END DO Cycle1
!!----------------------------------------------------------------------------------------------------
!****************************************************************************************************
    DO i = 1, mint
        xi  = xiq(i)
        et  = etq(i)
        zt  = 1.0D0 - xi - et
!----------------------------------------------------------------------------------------------------
            x = x1*zt + x2*xi + x3*et
            y = y1*zt + y2*xi + y3*et
            z = z1*zt + z2*xi + z3*et
!----------------------------------------------------------------------------------------------------
            CALL sgf_3d_sfs( x,   y,   z,     &
                          &  x1,  y1,  z1,    &
                          &  Txxx,Txxy,Txxz,  &
                          &  Txyx,Txyy,Txyz,  &
                          &  Txzx,Txzy,Txzz,  &
                          &  Tyxx,Tyxy,Tyxz,  &
                          &  Tyyx,Tyyy,Tyyz,  &
                          &  Tyzx,Tyzy,Tyzz,  &
                          &  Tzxx,Tzxy,Tzxz,  &
                          &  Tzyx,Tzyy,Tzyz,  &
                          &  Tzzx,Tzzy,Tzzz   )
!----------------------------------------------------------------------------------------------------
            cf1 = wq(i)
!----------------------------------------------------------------------------------------------------
            bsm=0.0D0
            bsm =cf1
!----------------------------------------------------------------------------------------------------
            Rxx = (Txxx*vnx + Txxy*vny + Txxz*vnz)*cf1
            Rxy = (Txyx*vnx + Txyy*vny + Txyz*vnz)*cf1
            Rxz = (Txzx*vnx + Txzy*vny + Txzz*vnz)*cf1
            Ryx = (Tyxx*vnx + Tyxy*vny + Tyxz*vnz)*cf1
            Ryy = (Tyyx*vnx + Tyyy*vny + Tyyz*vnz)*cf1
            Ryz = (Tyzx*vnx + Tyzy*vny + Tyzz*vnz)*cf1
            Rzx = (Tzxx*vnx + Tzxy*vny + Tzxz*vnz)*cf1
            Rzy = (Tzyx*vnx + Tzyy*vny + Tzyz*vnz)*cf1
            Rzz = (Tzzx*vnx + Tzzy*vny + Tzzz*vnz)*cf1
!----------------------------------------------------------------------------------------------------
        asm =asm + bsm
!----------------------------------------------------------------------------------------------------
        Txx = Txx + Rxx
        Txy = Txy + Rxy
        Txz = Txz + Rxz
        Tyx = Tyx + Ryx
        Tyy = Tyy + Ryy
        Tyz = Tyz + Ryz
        Tzx = Tzx + Rzx
        Tzy = Tzy + Rzy
        Tzz = Tzz + Rzz

    END DO
!****************************************************************************************************
```

```fortran
!   complete the quadrature
!--------------------------------------------------------------------------------------------------------------
      cf = area
!--------------------------------------------------------------------------------------------------------------
      asm  = asm*cf
!--------------------------------------------------------------------------------------------------------------
      TExx = TExx + cf*Txx
      TExy = TExy + cf*Txy
      TExz = TExz + cf*Txz
      TEyx = TEyx + cf*Tyx
      TEyy = TEyy + cf*Tyy
      TEyz = TEyz + cf*Tyz
      TEzx = TEzx + cf*Tzx
      TEzy = TEzy + cf*Tzy
      TEzz = TEzz + cf*Tzz
!--------------------------------------------------------------------------------------------------------------
!  IF all went well, asm should be equal to "area"
!  WRITE (Ulog,100) i,area,asm
!--------------------------------------------------------------------------------------------------------------
!  100 FORMAT (1x,i3,2(f10.5))
!--------------------------------------------------------------------------------------------------------------
END SUBROUTINE intr_lin_sing_s
!==============================================================================================================
    SUBROUTINE intr_lin_sing_s2(x0,   y0,   z0,   &
                             &  x1,   y1,   z1,   &
                             &  x2,   y2,   z2,   &
                             &  x3,   y3,   z3,   &
                             &  mint, &
                             &  TExx, TExy, TExz, &
                             &  TEyx, TEyy, TEyz, &
                             &  TEzx, TEzy, TEzz  )
!==============================================================================================================
!   Integrates the Green's function over a flat triangle in local polar coordinates with origin at the singular
!point: (x1,y1,z1). The subrotine is based from Pozriquidis 2002 pp. 119-120 in the technique 5.2.8 - 5.2.11
!==============================================================================================================
    USE Mod_SharedVars, ONLY: DBL, ULog, eps, Pi,   &
    &                         ZZ, WW,               &
    &                         Ns, Np, xiq, etq, wq
    USE Mod_sgf_3d_sfs
!==============================================================================================================
   IMPLICIT NONE
!==============================================================================================================
!   Variables
!--------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN)    :: x0, y0, z0          !coordinates of collocation point
    REAL (KIND = DBL), INTENT(IN)    :: x1, y1, z1          !coordinates of flat triangle vertice
    REAL (KIND = DBL), INTENT(IN)    :: x2, y2, z2
    REAL (KIND = DBL), INTENT(IN)    :: x3, y3, z3
    INTEGER,           INTENT(IN)    :: mint                !element index
    REAL (KIND = DBL), INTENT(INOUT) :: TExx, TExy, TExz, & !Integrated ij component over the element
    &                                   TEyx, TEyy, TEyz, &
    &                                   TEzx, TEzy, TEzz
!==============================================================================================================
!   Variables inside the subroutine
!--------------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                            !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6          !indices to obtain the node number in the vertices of each element
!--------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: pi4, piq               !constants to integrate over a circle element in Phi and r
    REAL (KIND = DBL) :: xi, et, zt             !constants of weigth to integrate over a flat triangle
    REAL (KIND = DBL) :: dx, dy                 !surface metrics
    REAL (KIND = DBL) :: vnx, vny, vnz          !triangle normal vector
    REAL (KIND = DBL) :: area, hs               !triangle area and surface metric on a flat triangle
    REAL (KIND = DBL) :: x, y, z                !coordinates of f(xi, et, zt)
    REAL (KIND = DBL) :: cf, cf1, cf2           !integrate weigths
    REAL (KIND = DBL) :: asm, bsm               !triangle area and its derivative
    REAL (KIND = DBL), DIMENSION(3) ::xxi, eet, zzt          !variables of weigth over in triangle (xi,eta)
    REAL (KIND = DBL) :: Txxx, Txxy, Txxz, &    !Free-space Green´s function of Stresslet. integrated ijk
    &                    Txyx, Txyy, Txyz, &    !component over the element
    &                    Txzx, Txzy, Txzz, &
    &                    Tyxx, Tyxy, Tyxz, &
    &                    Tyyx, Tyyy, Tyyz, &
```



```fortran
    &                    Tyzx, Tyzy, Tyzz, &
    &                    Tzxx, Tzxy, Tzxz, &
    &                    Tzyx, Tzyy, Tzyz, &
    &                    Tzzx, Tzzy, Tzzz
    REAL (KIND = DBL) :: Txx, Txy, Txz, &       !Stresslets Dummy coefficients
    &                    Tyx, Tyy, Tyz, &
    &                    Tzx, Tzy, Tzz
    REAL (KIND = DBL) :: Rxx, Rxy, Rxz, &       !Stresslets Dummy coefficients
    &                    Ryx, Ryy, Ryz, &
    &                    Rzx, Rzy, Rzz

!==============================================================================================================
!   Initialize
    xxi(1) = 0.5D0
    eet(1) = 0.5D0
    zzt(1) = 0.0D0
    xxi(2) = 0.5D0
    eet(2) = 0.0D0
    zzt(2) = 0.5D0
    xxi(3) = 0.0D0
    eet(3) = 0.5D0
    zzt(3) = 0.5D0
!--------------------------------------------------------------------------------------------------------------
    !xxi(1) = 1.0D0 / 6.0D0
    !eet(1) = 1.0D0 / 6.0D0
    !zzt(1) = 2.0D0 / 3.0D0
    !xxi(2) = 2.0D0 / 3.0D0
    !eet(2) = 1.0D0 / 6.0D0
    !zzt(2) = 1.0D0 / 6.0D0
    !xxi(3) = 1.0D0 / 6.0D0
    !eet(3) = 2.0D0 / 3.0D0
    !zzt(3) = 1.0D0 / 6.0D0
!--------------------------------------------------------------------------------------------------------------
    Txx = 0.0D0
    Txy = 0.0D0
    Txz = 0.0D0
    Tyx = 0.0D0
    Tyy = 0.0D0
    Tyz = 0.0D0
    Tzx = 0.0D0
    Tzy = 0.0D0
    Tzz = 0.0D0
!--------------------------------------------------------------------------------------------------------------
    Rxx = 0.0D0
    Rxy = 0.0D0
    Rxz = 0.0D0
    Ryx = 0.0D0
    Ryy = 0.0D0
    Ryz = 0.0D0
    Rzx = 0.0D0
    Rzy = 0.0D0
    Rzz = 0.0D0
!--------------------------------------------------------------------------------------------------------------
!   compute surface metric: hs
!--------------------------------------------------------------------------------------------------------------
    dx = DSQRT( (x2-x1)**2+(y2-y1)**2+(z2-z1)**2 )
    dy = DSQRT( (x3-x1)**2+(y3-y1)**2+(z3-z1)**2 )
!--------------------------------------------------------------------------------------------------------------
    vnx = ((y2-y1)*(z3-z1))-((z2-z1)*(y3-y1))
    vny = ((z2-z1)*(x3-x1))-((x2-x1)*(z3-z1))
    vnz = ((x2-x1)*(y3-y1))-((y2-y1)*(x3-x1))
!--------------------------------------------------------------------------------------------------------------
    hs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
    vnx = vnx/hs
    vny = vny/hs
    vnz = vnz/hs
!--------------------------------------------------------------------------------------------------------------
!   Surface metric on a flat triangle
    area = hs/2.0D0
!--------------------------------------------------------------------------------------------------------------
!   Initialize
!--------------------------------------------------------------------------------------------------------------
```

```fortran
!   Triangle area
    asm = 0.0D0
!----------------------------------------------------------------------------------------------------
    DO i = 1, mint
        xi  = xiq(i)
        et  = etq(i)
        zt  = 1.0D0- et-xi
        cf1 = wq(i)
!----------------------------------------------------------------------------------------------------
    !DO i = 1, 3
    !    xi  = xxi(i)
    !    et  = eet(i)
    !    zt  = zzt(i)
    !    cf1 = 1.0D0/3.0D0
!----------------------------------------------------------------------------------------------------
            x = x1*zt + x2*xi + x3*et
            y = y1*zt + y2*xi + y3*et
            z = z1*zt + z2*xi + z3*et
!----------------------------------------------------------------------------------------------------
            CALL sgf_3d_sfs( x,   y,   z,     &
                          &  x0,  y0,  z0,    &
                          &  Txxx,Txxy,Txxz,  &
                          &  Txyx,Txyy,Txyz,  &
                          &  Txzx,Txzy,Txzz,  &
                          &  Tyxx,Tyxy,Tyxz,  &
                          &  Tyyx,Tyyy,Tyyz,  &
                          &  Tyzx,Tyzy,Tyzz,  &
                          &  Tzxx,Tzxy,Tzxz,  &
                          &  Tzyx,Tzyy,Tzyz,  &
                          &  Tzzx,Tzzy,Tzzz   )
!----------------------------------------------------------------------------------------------------
            bsm=0.0D0
            bsm =cf1
!----------------------------------------------------------------------------------------------------
            Rxx = (Txxx*vnx + Txxy*vny + Txxz*vnz)*cf1
            Rxy = (Txyx*vnx + Txyy*vny + Txyz*vnz)*cf1
            Rxz = (Txzx*vnx + Txzy*vny + Txzz*vnz)*cf1
            Ryx = (Tyxx*vnx + Tyxy*vny + Tyxz*vnz)*cf1
            Ryy = (Tyyx*vnx + Tyyy*vny + Tyyz*vnz)*cf1
            Ryz = (Tyzx*vnx + Tyzy*vny + Tyzz*vnz)*cf1
            Rzx = (Tzxx*vnx + Tzxy*vny + Tzxz*vnz)*cf1
            Rzy = (Tzyx*vnx + Tzyy*vny + Tzyz*vnz)*cf1
            Rzz = (Tzzx*vnx + Tzzy*vny + Tzzz*vnz)*cf1
!----------------------------------------------------------------------------------------------------
        asm =asm + bsm
!----------------------------------------------------------------------------------------------------
        Txx = Txx + Rxx
        Txy = Txy + Rxy
        Txz = Txz + Rxz
        Tyx = Tyx + Ryx
        Tyy = Tyy + Ryy
        Tyz = Tyz + Ryz
        Tzx = Tzx + Rzx
        Tzy = Tzy + Rzy
        Tzz = Tzz + Rzz

    END DO
!----------------------------------------------------------------------------------------------------
!   complete the quadrature
!----------------------------------------------------------------------------------------------------
      cf = area
!----------------------------------------------------------------------------------------------------
      asm  = asm*cf
!----------------------------------------------------------------------------------------------------
      TExx = TExx + cf*Txx
      TExy = TExy + cf*Txy
      TExz = TExz + cf*Txz
      TEyx = TEyx + cf*Tyx
      TEyy = TEyy + cf*Tyy
      TEyz = TEyz + cf*Tyz
      TEzx = TEzx + cf*Tzx
      TEzy = TEzy + cf*Tzy
```



```fortran
      TEzz = TEzz + cf*Tzz
!----------------------------------------------------------------------------------------------------
!  IF all went well, asm should be equal to "area"
!  WRITE (Ulog,100) i,area,asm
!----------------------------------------------------------------------------------------------------
!  100 FORMAT (1x,i3,2(f10.5))
!----------------------------------------------------------------------------------------------------
END SUBROUTINE intr_lin_sing_s2
!====================================================================================================
    SUBROUTINE intr_lin_sing_s4(x0e,   y0e,   z0e,    &
                            &  x00,   y00,   z00,  &
                            &  k,                  &
                            &  TExx, TExy, TExz,   &
                            &  TEyx, TEyy, TEyz,   &
                            &  TEzx, TEzy, TEzz    )
!====================================================================================================
!   This subroutine is a new version stokeslet Subroutine.
!Compute:
!   *The value of the Stokeslet over each singular element
!Now, (March/ 09 / 2015) this subroutine was made.
!====================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog, pi, eps, vnx0, vny0, vnz0, x0, y0, z0
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN) :: x00,   y00,   z00             !singulatity coordinates
    REAL (KIND = DBL), INTENT(IN) :: x0e,   y0e,   z0e             !singulatity coordinates element
    INTEGER,           INTENT(IN)  :: k                         !number of element
    REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TExz          !value of stresslet in the singular element
    REAL (KIND = DBL), INTENT(OUT) :: TEyx, TEyy, TEyz          !value of stresslet in the singular element
    REAL (KIND = DBL), INTENT(OUT) :: TEzx, TEzy, TEzz          !value of stresslet in the singular element
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                  !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                !indices to obtain node numbers from each element
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: cf, fil1, fil2, fil3           !integration weigth coefficients
    REAL (KIND = DBL) :: DxDx, DyDx, DzDx               !tangential vector around the triangle (xi,eta)
    REAL (KIND = DBL) :: hss                            !weigth
    REAL (KIND = DBL) :: cpi                            !Selection to add the solid angle
    REAL (KIND = DBL) :: modx0                          !weigth
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &           !vectoreial product
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  QExx, QExy, QExz, &           !Iq tensor
    &                     QEyx, QEyy, QEyz, &
    &                     QEzx, QEzy, QEzz
    REAL (KIND = DBL) ::  PEx, PEy, PEz, PE             !Ip tensor
    REAL (KIND = DBL) ::  yvx1, yvy1, yvz1                 !vector (y1-x0)
    REAL (KIND = DBL) ::  yvx2, yvy2, yvz2                 !vector (y2-x0)
    REAL (KIND = DBL) ::  yvx3, yvy3, yvz3                 !vector (y2-x0)
    REAL (KIND = DBL) ::  px2, py2, pz2                 !vector (y2-x0)
    REAL (KIND = DBL) ::  a1, a2, a3                    !vector a
    REAL (KIND = DBL) ::  ay1, ay2, ay3                 !vector dot product a*y1, a*y2
    REAL (KIND = DBL) ::  by1, by2, by3                 !product yi*(yi+a.yi)
!====================================================================================================
!   Initialize
!----------------------------------------------------------------------------------------------------
    TExx = 0.0D0
    TExy = 0.0D0
    TExz = 0.0D0
    TEyx = 0.0D0
    TEyy = 0.0D0
    TEyz = 0.0D0
    TEzx = 0.0D0
    TEzy = 0.0D0
    TEzz = 0.0D0
```

```fortran
    QExx = 0.0D0
    QExy = 0.0D0
    QExz = 0.0D0
    QEyx = 0.0D0
    QEyy = 0.0D0
    QEyz = 0.0D0
    QEzx = 0.0D0
    QEzy = 0.0D0
    QEzz = 0.0D0

    PEx = 0.0D0
    PEy = 0.0D0
    PEz = 0.0D0
    PE  = 0.0D0

    a1   = x0e-x00
    a2   = y0e-y00
    a3   = z0e-z00
    modx0= DSQRT(a1**2+a2**2+a3**2)

    IF (modx0 >= 1.0D0 + eps) THEN
        cpi = 0.0D0
    ELSE IF (modx0 <= 1.0D0 - eps) THEN
        cpi = -8.0D0*pi
    ELSE
        cpi = -4.0D0*pi
    end if
    a1   = -x0e
    a2   = -y0e
    a3   = -z0e
    modx0= DSQRT(a1**2+a2**2+a3**2)
    a1   = a1/modx0
    a2   = a2/modx0
    a3   = a3/modx0

    !OPEN (9,file="TE.out")
!---------------------------------------------------------------------------------------------------------------
!   vertices of the kth triangle
!---------------------------------------------------------------------------------------------------------------
    i1 = n(k,1)
    i2 = n(k,2)
    i3 = n(k,3)
    i4 = n(k,4)
    i5 = n(k,5)
    i6 = n(k,6)
!---------------------------------------------------------------------------------------------------------------
        cf  = 0.0D0
!---------------------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
! rule formula
!---------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!---------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-4
!---------------------------------------------------------------------------------------------------------------
```



```fortran
        DxDx = p(i4,1) - p(i1,1)
        DyDx = p(i4,2) - p(i1,2)
        DzDx = p(i4,3) - p(i1,3)
!---------------------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!---------------------------------------------------------------------------------------------------------------
        yvx1 = p(i1,1) - x00
        yvy1 = p(i1,2) - y00
        yvz1 = p(i1,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!---------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!---------------------------------------------------------------------------------------------------------------
        yvx3 = p(i4,1) - x00
        yvy3 = p(i4,2) - y00
        yvz3 = p(i4,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!---------------------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!---------------------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - (bvx3*(yvx3/fil3**3)) )/2.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - (bvy3*(yvx3/fil3**3)) )/2.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - (bvz3*(yvx3/fil3**3)) )/2.0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - (bvx3*(yvy3/fil3**3)) )/2.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - (bvy3*(yvy3/fil3**3)) )/2.0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - (bvx3*(yvz3/fil3**3)) )/2.0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - (bvy3*(yvz3/fil3**3)) )/2.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.0D0
!---------------------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!---------------------------------------------------------------------------------------------------------------
        by1  = fil1*(fil1+ay1)
        by3  = fil3*(fil3+ay3)
!---------------------------------------------------------------------------------------------------------------
        PEx  = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
!---------------------------------------------------------------------------------------------------------------
        PE   = PE + hss*( PEx + PEy + PEz )/2.0D0
!---------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 4-2
!---------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!---------------------------------------------------------------------------------------------------------------
        DxDx = p(i2,1) - p(i4,1)
        DyDx = p(i2,2) - p(i4,2)
```

```
        DzDx = p(i2,3) - p(i4,3)
!---------------------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!---------------------------------------------------------------------------------------------------------------
        yvx1 = p(i4,1) - x00
        yvy1 = p(i4,2) - y00
        yvz1 = p(i4,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!---------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!---------------------------------------------------------------------------------------------------------------
        yvx3 = p(i2,1) - x00
        yvy3 = p(i2,2) - y00
        yvz3 = p(i2,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!---------------------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!---------------------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - (bvx3*(yvx3/fil3**3)) )/2.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - (bvy3*(yvx3/fil3**3)) )/2.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - (bvz3*(yvx3/fil3**3)) )/2.0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - (bvx3*(yvy3/fil3**3)) )/2.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - (bvy3*(yvy3/fil3**3)) )/2.0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - (bvx3*(yvz3/fil3**3)) )/2.0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - (bvy3*(yvz3/fil3**3)) )/2.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.0D0
!---------------------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!---------------------------------------------------------------------------------------------------------------
        by1  = fil1*(fil1+ay1)
        by3  = fil3*(fil3+ay3)
!---------------------------------------------------------------------------------------------------------------
        PEx  = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
!---------------------------------------------------------------------------------------------------------------
        PE   = PE + hss*( PEx + PEy + PEz )/2.0D0
!---------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-5
!---------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!---------------------------------------------------------------------------------------------------------------
        DxDx = p(i5,1) - p(i2,1)
        DyDx = p(i5,2) - p(i2,2)
        DzDx = p(i5,3) - p(i2,3)
!---------------------------------------------------------------------------------------------------------------
```



```
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!---------------------------------------------------------------------------------------------------------------
        yvx1 = p(i2,1) - x00
        yvy1 = p(i2,2) - y00
        yvz1 = p(i2,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!---------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!---------------------------------------------------------------------------------------------------------------
        yvx3 = p(i5,1) - x00
        yvy3 = p(i5,2) - y00
        yvz3 = p(i5,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!---------------------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!---------------------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - (bvx3*(yvx3/fil3**3)) )/2.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - (bvy3*(yvx3/fil3**3)) )/2.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - (bvz3*(yvx3/fil3**3)) )/2.0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - (bvx3*(yvy3/fil3**3)) )/2.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - (bvy3*(yvy3/fil3**3)) )/2.0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - (bvx3*(yvz3/fil3**3)) )/2.0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - (bvy3*(yvz3/fil3**3)) )/2.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.0D0
!---------------------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!---------------------------------------------------------------------------------------------------------------
        by1  = fil1*(fil1+ay1)
        by3  = fil3*(fil3+ay3)
!---------------------------------------------------------------------------------------------------------------
        PEx  = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
!---------------------------------------------------------------------------------------------------------------
        PE   = PE + hss*( PEx + PEy + PEz )/2.0D0
!---------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 5-3
!---------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!---------------------------------------------------------------------------------------------------------------
        DxDx = p(i3,1) - p(i5,1)
        DyDx = p(i3,2) - p(i5,2)
        DzDx = p(i3,3) - p(i5,3)
!---------------------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
```

```fortran
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!-----------------------------------------------------------------------------------------------------------
        yvx1 = p(i5,1) - x00
        yvy1 = p(i5,2) - y00
        yvz1 = p(i5,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!-----------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!-----------------------------------------------------------------------------------------------------------
        yvx3 = p(i3,1) - x00
        yvy3 = p(i3,2) - y00
        yvz3 = p(i3,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!-----------------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!-----------------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - (bvx3*(yvx3/fil3**3)) )/2.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - (bvy3*(yvx3/fil3**3)) )/2.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - (bvz3*(yvx3/fil3**3)) )/2.0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - (bvx3*(yvy3/fil3**3)) )/2.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - (bvy3*(yvy3/fil3**3)) )/2.0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - (bvx3*(yvz3/fil3**3)) )/2.0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - (bvy3*(yvz3/fil3**3)) )/2.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.0D0
!-----------------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!-----------------------------------------------------------------------------------------------------------
        by1  = fil1*(fil1+ay1)
        by3  = fil3*(fil3+ay3)
!-----------------------------------------------------------------------------------------------------------
        PEx  = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
!-----------------------------------------------------------------------------------------------------------
        PE   = PE + hss*( PEx + PEy + PEz )/2.0D0
!-----------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3-6
!-----------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!-----------------------------------------------------------------------------------------------------------
        DxDx = p(i6,1) - p(i3,1)
        DyDx = p(i6,2) - p(i3,2)
        DzDx = p(i6,3) - p(i3,3)
!-----------------------------------------------------------------------------------------------------------
        hss   = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
```



```fortran
!-----------------------------------------------------------------------------------------------------------
        yvx1 = p(i3,1) - x00
        yvy1 = p(i3,2) - y00
        yvz1 = p(i3,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!-----------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!-----------------------------------------------------------------------------------------------------------
        yvx3 = p(i6,1) - x00
        yvy3 = p(i6,2) - y00
        yvz3 = p(i6,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!-----------------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!-----------------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - (bvx3*(yvx3/fil3**3)) )/2.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - (bvy3*(yvx3/fil3**3)) )/2.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - (bvz3*(yvx3/fil3**3)) )/2.0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - (bvx3*(yvy3/fil3**3)) )/2.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - (bvy3*(yvy3/fil3**3)) )/2.0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - (bvx3*(yvz3/fil3**3)) )/2.0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - (bvy3*(yvz3/fil3**3)) )/2.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.0D0
!-----------------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!-----------------------------------------------------------------------------------------------------------
        by1  = fil1*(fil1+ay1)
        by3  = fil3*(fil3+ay3)
!-----------------------------------------------------------------------------------------------------------
        PEx  = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
!-----------------------------------------------------------------------------------------------------------
        PE   = PE + hss*( PEx + PEy + PEz )/2.0D0
!-----------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 6-1
!-----------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!-----------------------------------------------------------------------------------------------------------
        DxDx = p(i1,1) - p(i6,1)
        DyDx = p(i1,2) - p(i6,2)
        DzDx = p(i1,3) - p(i6,3)
!-----------------------------------------------------------------------------------------------------------
        hss   = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!-----------------------------------------------------------------------------------------------------------
        yvx1 = p(i6,1) - x00
```

```fortran
        yvy1 = p(i6,2) - y00
        yvz1 = p(i6,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!-----------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!-----------------------------------------------------------------------------------------------------------
        yvx3 = p(i1,1) - x00
        yvy3 = p(i1,2) - y00
        yvz3 = p(i1,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!-----------------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!-----------------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - (bvx3*(yvx3/fil3**3)) )/2.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - (bvy3*(yvx3/fil3**3)) )/2.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - (bvz3*(yvx3/fil3**3)) )/2.0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - (bvx3*(yvy3/fil3**3)) )/2.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - (bvy3*(yvy3/fil3**3)) )/2.0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - (bvx3*(yvz3/fil3**3)) )/2.0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - (bvy3*(yvz3/fil3**3)) )/2.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.0D0
!-----------------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!-----------------------------------------------------------------------------------------------------------
        by1  = fil1*(fil1+ay1)
        by3  = fil3*(fil3+ay3)
!-----------------------------------------------------------------------------------------------------------
        PEx  = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
!-----------------------------------------------------------------------------------------------------------
        PE   = PE + hss*( PEx + PEy + PEz )/2.0D0
!        Close(9)
!-----------------------------------------------------------------------------------------------------------
    TExx = 2.0D0*( QExx + cpi -PE )
    TExy = 2.0D0*( QExy )
    TExz = 2.0D0*( QExz )
    TEyx = 2.0D0*( QEyx )
    TEyy = 2.0D0*( QEyy + cpi -PE )
    TEyz = 2.0D0*( QEyz )
    TEzx = 2.0D0*( QEzx )
    TEzy = 2.0D0*( QEzy )
    TEzz = 2.0D0*( QEzz + cpi -PE )
!-----------------------------------------------------------------------------------------------------------
END SUBROUTINE intr_lin_sing_s4
!===========================================================================================================!
    ===========================================================================================================
    ==
    SUBROUTINE intr_lin_sing_s4s(x0e,   y0e,   z0e,    &
                            &  x00,   y00,   z00,  &
                            &  k,                  &
                            &  TExx, TExy, TExz,   &
                            &  TEyx, TEyy, TEyz,   &
                            &  TEzx, TEzy, TEzz    )
!===========================================================================================================
!   This subroutine is a new version stokeslet Subroutine.
!Compute:
!   *The value of the Stokeslet over each singular element
!Now, (March/ 09 / 2015) this subroutine was made.
!===========================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog, pi, eps, vnx0, vny0, vnz0, x0, y0, z0
!===========================================================================================================
    IMPLICIT NONE
!===========================================================================================================
!   Variables
```



```fortran
!-----------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN) :: x00,   y00,   z00            !singulatity coordinates
    REAL (KIND = DBL), INTENT(IN) :: x0e,   y0e,   z0e            !singulatity coordinates element
    INTEGER,           INTENT(IN)  :: k                         !number of element
    REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TExz          !value of stresslet in the singular element
    REAL (KIND = DBL), INTENT(OUT) :: TEyx, TEyy, TEyz          !value of stresslet in the singular element
    REAL (KIND = DBL), INTENT(OUT) :: TEzx, TEzy, TEzz          !value of stresslet in the singular element
!===========================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                    !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                  !indices to obtain node numbers from each element
!-----------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: cf, fil1, fil2, fil3           !integration weigth coefficients
    REAL (KIND = DBL) :: DxDx, DyDx, DzDx               !tangential vector around the triangle (xi,eta)
    REAL (KIND = DBL) :: hss                            !weigth
    REAL (KIND = DBL) :: cpi                            !Selection to add the solid angle
    REAL (KIND = DBL) :: modx0                          !weigth
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &           !vectoreial product
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  QExx, QExy, QExz, &           !Iq tensor
    &                     QEyx, QEyy, QEyz, &
    &                     QEzx, QEzy, QEzz
    REAL (KIND = DBL) ::  PEx, PEy, PEz, PE             !Ip tensor
    REAL (KIND = DBL) ::  yvx1, yvy1, yvz1                 !vector (y1-x0)
    REAL (KIND = DBL) ::  yvx2, yvy2, yvz2                 !vector (y2-x0)
    REAL (KIND = DBL) ::  yvx3, yvy3, yvz3                 !vector (y2-x0)
    REAL (KIND = DBL) ::  px2, py2, pz2                 !vector (y2-x0)
    REAL (KIND = DBL) ::  a1, a2, a3                    !vector a
    REAL (KIND = DBL) ::  ay1, ay2, ay3                 !vector dot product a*y1, a*y2
    REAL (KIND = DBL) ::  by1, by2, by3                 !product yi*(yi+a.yi)
!===========================================================================================================
!   Initialize
!-----------------------------------------------------------------------------------------------------------
    TExx = 0.0D0
    TExy = 0.0D0
    TExz = 0.0D0
    TEyx = 0.0D0
    TEyy = 0.0D0
    TEyz = 0.0D0
    TEzx = 0.0D0
    TEzy = 0.0D0
    TEzz = 0.0D0

    QExx = 0.0D0
    QExy = 0.0D0
    QExz = 0.0D0
    QEyx = 0.0D0
    QEyy = 0.0D0
    QEyz = 0.0D0
    QEzx = 0.0D0
    QEzy = 0.0D0
    QEzz = 0.0D0

    PEx = 0.0D0
    PEy = 0.0D0
    PEz = 0.0D0
    PE  = 0.0D0

    a1   = -x0(k)
    a2   = -y0(k)
    a3   = -z0(k)
    modx0= DSQRT(a1**2+a2**2+a3**2)

        cpi = 4.0D0*pi

    !IF((ABS(a1) <= eps) .AND. &
    !&  (ABS(a2) <= eps) .AND. &
    !&  (ABS(a3) <= eps)) THEN
    !    a1   = x0(k)
    !    a2   = y0(k)
```

```
    !    a3   = z0(k)
    !ELSE
    !a1   = x0e-x00
    !a2   = y0e-y00
    !a3   = z0e-z00
    !END IF
    !
    a1   = a1/modx0
    a2   = a2/modx0
    a3   = a3/modx0

    !OPEN (9,file="TE.out")
!------------------------------------------------------------------------------------------------------------
!   vertices of the kth triangle
!------------------------------------------------------------------------------------------------------------
    i1 = n(k,1)
    i2 = n(k,2)
    i3 = n(k,3)
    i4 = n(k,4)
    i5 = n(k,5)
    i6 = n(k,6)
!------------------------------------------------------------------------------------------------------------
        cf  = 0.0D0
!------------------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
! rule formula
!------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-4
!------------------------------------------------------------------------------------------------------------
        DxDx = p(i4,1) - p(i1,1)
        DyDx = p(i4,2) - p(i1,2)
        DzDx = p(i4,3) - p(i1,3)
!------------------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!------------------------------------------------------------------------------------------------------------
        yvx1 = p(i1,1) - x00
        yvy1 = p(i1,2) - y00
        yvz1 = p(i1,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!------------------------------------------------------------------------------------------------------------
        yvx3 = p(i4,1) - x00
        yvy3 = p(i4,2) - y00
        yvz3 = p(i4,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!------------------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
```



```
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!------------------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - (bvx3*(yvx3/fil3**3)) )/2.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - (bvy3*(yvx3/fil3**3)) )/2.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - (bvz3*(yvx3/fil3**3)) )/2.0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - (bvx3*(yvy3/fil3**3)) )/2.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - (bvy3*(yvy3/fil3**3)) )/2.0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - (bvx3*(yvz3/fil3**3)) )/2.0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - (bvy3*(yvz3/fil3**3)) )/2.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.0D0
!------------------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!------------------------------------------------------------------------------------------------------------
        by1  = fil1*(fil1+ay1)
        by3  = fil3*(fil3+ay3)
!------------------------------------------------------------------------------------------------------------
        PEx  = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
!------------------------------------------------------------------------------------------------------------
        PE   = PE + hss*( PEx + PEy + PEz )/2.0D0
!------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 4-2
!------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!------------------------------------------------------------------------------------------------------------
        DxDx = p(i2,1) - p(i4,1)
        DyDx = p(i2,2) - p(i4,2)
        DzDx = p(i2,3) - p(i4,3)
!------------------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!------------------------------------------------------------------------------------------------------------
        yvx1 = p(i4,1) - x00
        yvy1 = p(i4,2) - y00
        yvz1 = p(i4,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!------------------------------------------------------------------------------------------------------------
        yvx3 = p(i2,1) - x00
        yvy3 = p(i2,2) - y00
        yvz3 = p(i2,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!------------------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
```

```
!-----------------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - (bvx3*(yvx3/fil3**3)) )/2.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - (bvy3*(yvx3/fil3**3)) )/2.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - (bvz3*(yvx3/fil3**3)) )/2.0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - (bvx3*(yvy3/fil3**3)) )/2.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - (bvy3*(yvy3/fil3**3)) )/2.0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - (bvx3*(yvz3/fil3**3)) )/2.0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - (bvy3*(yvz3/fil3**3)) )/2.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.0D0
!-----------------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!-----------------------------------------------------------------------------------------------------------
        by1  = fil1*(fil1+ay1)
        by3  = fil3*(fil3+ay3)
!-----------------------------------------------------------------------------------------------------------
        PEx  = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
!-----------------------------------------------------------------------------------------------------------
        PE   = PE + hss*( PEx + PEy + PEz )/2.0D0
!-----------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-5
!-----------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!-----------------------------------------------------------------------------------------------------------
        DxDx = p(i5,1) - p(i2,1)
        DyDx = p(i5,2) - p(i2,2)
        DzDx = p(i5,3) - p(i2,3)
!-----------------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!-----------------------------------------------------------------------------------------------------------
        yvx1 = p(i2,1) - x00
        yvy1 = p(i2,2) - y00
        yvz1 = p(i2,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!-----------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!-----------------------------------------------------------------------------------------------------------
        yvx3 = p(i5,1) - x00
        yvy3 = p(i5,2) - y00
        yvz3 = p(i5,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!-----------------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!-----------------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - (bvx3*(yvx3/fil3**3)) )/2.0D0
```



```
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - (bvy3*(yvx3/fil3**3)) )/2.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - (bvz3*(yvx3/fil3**3)) )/2.0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - (bvx3*(yvy3/fil3**3)) )/2.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - (bvy3*(yvy3/fil3**3)) )/2.0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - (bvx3*(yvz3/fil3**3)) )/2.0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - (bvy3*(yvz3/fil3**3)) )/2.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.0D0
!-----------------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!-----------------------------------------------------------------------------------------------------------
        by1  = fil1*(fil1+ay1)
        by3  = fil3*(fil3+ay3)
!-----------------------------------------------------------------------------------------------------------
        PEx  = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
!-----------------------------------------------------------------------------------------------------------
        PE   = PE + hss*( PEx + PEy + PEz )/2.0D0
!-----------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 5-3
!-----------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!-----------------------------------------------------------------------------------------------------------
        DxDx = p(i3,1) - p(i5,1)
        DyDx = p(i3,2) - p(i5,2)
        DzDx = p(i3,3) - p(i5,3)
!-----------------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!-----------------------------------------------------------------------------------------------------------
        yvx1 = p(i5,1) - x00
        yvy1 = p(i5,2) - y00
        yvz1 = p(i5,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!-----------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!-----------------------------------------------------------------------------------------------------------
        yvx3 = p(i3,1) - x00
        yvy3 = p(i3,2) - y00
        yvz3 = p(i3,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!-----------------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!-----------------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - (bvx3*(yvx3/fil3**3)) )/2.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - (bvy3*(yvx3/fil3**3)) )/2.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - (bvz3*(yvx3/fil3**3)) )/2.0D0
```

```fortran
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - (bvx3*(yvy3/fil3**3)) )/2.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - (bvy3*(yvy3/fil3**3)) )/2.0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - (bvx3*(yvz3/fil3**3)) )/2.0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - (bvy3*(yvz3/fil3**3)) )/2.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.0D0
!-----------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!-----------------------------------------------------------------------------------------------------
        by1  = fil1*(fil1+ay1)
        by3  = fil3*(fil3+ay3)
!-----------------------------------------------------------------------------------------------------
        PEx  = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
!-----------------------------------------------------------------------------------------------------
        PE   = PE + hss*( PEx + PEy + PEz )/2.0D0
!-----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3-6
!-----------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!-----------------------------------------------------------------------------------------------------
        DxDx = p(i6,1) - p(i3,1)
        DyDx = p(i6,2) - p(i3,2)
        DzDx = p(i6,3) - p(i3,3)
!-----------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!-----------------------------------------------------------------------------------------------------
        yvx1 = p(i3,1) - x00
        yvy1 = p(i3,2) - y00
        yvz1 = p(i3,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!-----------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!-----------------------------------------------------------------------------------------------------
        yvx3 = p(i6,1) - x00
        yvy3 = p(i6,2) - y00
        yvz3 = p(i6,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!-----------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!-----------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - (bvx3*(yvx3/fil3**3)) )/2.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - (bvy3*(yvx3/fil3**3)) )/2.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - (bvz3*(yvx3/fil3**3)) )/2.0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - (bvx3*(yvy3/fil3**3)) )/2.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - (bvy3*(yvy3/fil3**3)) )/2.0D0
```



```fortran
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - (bvx3*(yvz3/fil3**3)) )/2.0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - (bvy3*(yvz3/fil3**3)) )/2.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.0D0
!-----------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!-----------------------------------------------------------------------------------------------------
        by1  = fil1*(fil1+ay1)
        by3  = fil3*(fil3+ay3)
!-----------------------------------------------------------------------------------------------------
        PEx  = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
!-----------------------------------------------------------------------------------------------------
        PE   = PE + hss*( PEx + PEy + PEz )/2.0D0
!-----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 6-1
!-----------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!-----------------------------------------------------------------------------------------------------
        DxDx = p(i1,1) - p(i6,1)
        DyDx = p(i1,2) - p(i6,2)
        DzDx = p(i1,3) - p(i6,3)
!-----------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!-----------------------------------------------------------------------------------------------------
        yvx1 = p(i6,1) - x00
        yvy1 = p(i6,2) - y00
        yvz1 = p(i6,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!-----------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!-----------------------------------------------------------------------------------------------------
        yvx3 = p(i1,1) - x00
        yvy3 = p(i1,2) - y00
        yvz3 = p(i1,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!-----------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!-----------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - (bvx3*(yvx3/fil3**3)) )/2.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - (bvy3*(yvx3/fil3**3)) )/2.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - (bvz3*(yvx3/fil3**3)) )/2.0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - (bvx3*(yvy3/fil3**3)) )/2.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - (bvy3*(yvy3/fil3**3)) )/2.0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - (bvx3*(yvz3/fil3**3)) )/2.0D0
```

```fortran
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - (bvy3*(yvz3/fil3**3)) )/2.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.0D0
!-----------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!-----------------------------------------------------------------------------------------------------
        by1  = fil1*(fil1+ay1)
        by3  = fil3*(fil3+ay3)
!-----------------------------------------------------------------------------------------------------
        PEx  = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
!-----------------------------------------------------------------------------------------------------
        PE   = PE + hss*( PEx + PEy + PEz )/2.0D0
!        Close(9)
!-----------------------------------------------------------------------------------------------------
    TExx = 2.0D0*( QExx + cpi -PE )
    TExy = 2.0D0*( QExy )
    TExz = 2.0D0*( QExz )
    TEyx = 2.0D0*( QEyx )
    TEyy = 2.0D0*( QEyy + cpi -PE )
    TEyz = 2.0D0*( QEyz )
    TEzx = 2.0D0*( QEzx )
    TEzy = 2.0D0*( QEzy )
    TEzz = 2.0D0*( QEzz + cpi -PE )
!-----------------------------------------------------------------------------------------------------
END SUBROUTINE intr_lin_sing_s4s
!=====================================================================================================
    SUBROUTINE intr_lin_sing_s5(x10,   y10,   z10,    &
                            &  k,                  &
                            &  TExx, TExy, TExz,   &
                            &  TEyx, TEyy, TEyz,   &
                            &  TEzx, TEzy, TEzz    )
!=====================================================================================================
!   This subroutine is a new version stokeslet Subroutine.
!Compute:
!   *The value of the Stokeslet over each singular element
!Now, (March/ 09 / 2015) this subroutine was made.
!=====================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog, pi, eps, x0, y0, z0
!=====================================================================================================
    IMPLICIT NONE
!=====================================================================================================
!   Variables
!-----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN) :: x10,   y10,   z10            !singulatity coordinates
    INTEGER,           INTENT(IN)  :: k                          !number of element
    REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TExz          !value of stresslet in the singular element
    REAL (KIND = DBL), INTENT(OUT) :: TEyx, TEyy, TEyz          !value of stresslet in the singular element
    REAL (KIND = DBL), INTENT(OUT) :: TEzx, TEzy, TEzz          !value of stresslet in the singular element
!=====================================================================================================
!   Variables inside the subroutine
!-----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                 !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6              !indices to obtain node numbers from each element
!-----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: cf, fil1, fil2, fil3           !integration weigth coefficients
    REAL (KIND = DBL) :: DxDx, DyDx, DzDx               !tangential vector around the triangle (xi,eta)
    REAL (KIND = DBL) :: hss                            !weigth
    REAL (KIND = DBL) :: cpi                            !Selection to add the solid angle
    REAL (KIND = DBL) :: modx0                          !weigth
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &           !vectoreial product
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  QExx, QExy, QExz, &           !Iq tensor
    &                     QEyx, QEyy, QEyz, &
    &                     QEzx, QEzy, QEzz
    REAL (KIND = DBL) ::  PEx, PEy, PEz, PE             !Ip tensor
    REAL (KIND = DBL) ::  yvx1, yvy1, yvz1                 !vector y1
    REAL (KIND = DBL) ::  yvx2, yvy2, yvz2                 !vector y2
    REAL (KIND = DBL) ::  yvx3, yvy3, yvz3                 !vector y3
```



```fortran
    REAL (KIND = DBL) ::  yvx4, yvy4, yvz4                 !vector y4
    REAL (KIND = DBL) ::  yvx5, yvy5, yvz5                 !vector y5
    REAL (KIND = DBL) ::  yvx6, yvy6, yvz6                 !vector y6
    REAL (KIND = DBL) ::  zvx1, zvy1, zvz1                 !vector (z1-x0)
    REAL (KIND = DBL) ::  zvx2, zvy2, zvz2                 !vector (z2-x0)
    REAL (KIND = DBL) ::  zvx3, zvy3, zvz3                 !vector (z3-x0)
    REAL (KIND = DBL) ::  zvx4, zvy4, zvz4                 !vector (z4-x0)
    REAL (KIND = DBL) ::  zvx5, zvy5, zvz5                 !vector (z5-x0)
    REAL (KIND = DBL) ::  zvx6, zvy6, zvz6                 !vector (z6-x0)
    REAL (KIND = DBL) ::  zcx1, zcy1, zcz1                 !vector zi ^ zi+1
    REAL (KIND = DBL) ::  zcx2, zcy2, zcz2                 !vector zi ^ zi+1
    REAL (KIND = DBL) ::  zcx3, zcy3, zcz3                 !vector zi ^ zi+1
    REAL (KIND = DBL) ::  zcx4, zcy4, zcz4                 !vector zi ^ zi+1
    REAL (KIND = DBL) ::  zcx5, zcy5, zcz5                 !vector zi ^ zi+1
    REAL (KIND = DBL) ::  zcx6, zcy6, zcz6                 !vector zi ^ zi+1
    REAL (KIND = DBL) ::  zd1, zd2, zd3, zd4, zd5, zd6     !vector zi . zi+1
    REAL (KIND = DBL) ::  px2, py2, pz2                  !preasure vector
    REAL (KIND = DBL) ::  a1, a2, a3                     !vector a
    REAL (KIND = DBL) ::  al1, al2, al3, al4, al5, al6  !alpha angle between zi and zi+1 from x0
    REAL (KIND = DBL) ::  th1, th2, th3, th4, th5, th6  !theta angle between vector a and mi vector
    REAL (KIND = DBL) ::  e1, e2, e3, e4, e5, e6        !constant
    REAL (KIND = DBL) ::  f1, f2, f3, f4, f5, f6        !constant
    REAL (KIND = DBL) ::  g1, g2, g3, g4, g5, g6        !constant
    REAL (KIND = DBL) ::  ay1, ay2, ay3                  !vector dot product a*y1, a*y2
    REAL (KIND = DBL) ::  by1, by2, by3                  !product yi*(yi+a.yi)
!=====================================================================================================
!   Initialize
!-----------------------------------------------------------------------------------------------------
    TExx = 0.0D0
    TExy = 0.0D0
    TExz = 0.0D0
    TEyx = 0.0D0
    TEyy = 0.0D0
    TEyz = 0.0D0
    TEzx = 0.0D0
    TEzy = 0.0D0
    TEzz = 0.0D0

    QExx = 0.0D0
    QExy = 0.0D0
    QExz = 0.0D0
    QEyx = 0.0D0
    QEyy = 0.0D0
    QEyz = 0.0D0
    QEzx = 0.0D0
    QEzy = 0.0D0
    QEzz = 0.0D0

    PEx = 0.0D0
    PEy = 0.0D0
    PEz = 0.0D0
    PE  = 0.0D0

    modx0= DSQRT((x0(k)-x10)**2+(y0(k)-y10)**2+(z0(k)-z10)**2)

    IF (modx0 >= 1.0D0 + eps) THEN
        cpi = 0.0D0
    ELSE IF (modx0 <= 1.0D0 + eps) THEN
        cpi = -8.0D0*pi
    ELSE
        cpi = -4.0D0*pi
    end if

    IF((ABS(x10-x0(k)) <= eps) .AND. &
    &  (ABS(y10-y0(k)) <= eps) .AND. &
    &  (ABS(z10-z0(k)) <= eps)) THEN
        a1   = x10
        a2   = y10
        a3   = z10
    ELSE
        a1   = x10-x0(k)
        a2   = y10-y0(k)
```

```fortran
        a3   = z10-z0(k)
    END IF

    !a1   = x00-x0(k)
    !a2   = y00-y0(k)
    !a3   = z00-z0(k)
    modx0= DSQRT(a1**2+a2**2+a3**2)
    a1   = a1/modx0
    a2   = a2/modx0
    a3   = a3/modx0


    !OPEN (9,file="TE.out")
!----------------------------------------------------------------------------------------------------
!   vertices of the kth triangle
!----------------------------------------------------------------------------------------------------
    i1 = n(k,1)
    i2 = n(k,2)
    i3 = n(k,3)
    i4 = n(k,4)
    i5 = n(k,5)
    i6 = n(k,6)
!----------------------------------------------------------------------------------------------------
        cf  = 0.0D0
!----------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
! rule formula
!----------------------------------------------------------------------------------------------------
!   Firstly, to compute all vectors and angles involved.
!----------------------------------------------------------------------------------------------------
    yvx1 = p(i1,1)-x10
    yvy1 = p(i1,2)-y10
    yvz1 = p(i1,3)-z10
    yvx2 = p(i2,1)-x10
    yvy2 = p(i2,2)-y10
    yvz2 = p(i2,3)-z10
    yvx3 = p(i3,1)-x10
    yvy3 = p(i3,2)-y10
    yvz3 = p(i3,3)-z10
    yvx4 = p(i4,1)-x10
    yvy4 = p(i4,2)-y10
    yvz4 = p(i4,3)-z10
    yvx5 = p(i5,1)-x10
    yvy5 = p(i5,2)-y10
    yvz5 = p(i5,3)-z10
    yvx6 = p(i6,1)-x10
    yvy6 = p(i6,2)-y10
    yvz6 = p(i6,3)-z10
!----------------------------------------------------------------------------------------------------
    zvx1 = yvx1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
    zvy1 = yvy1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
    zvz1 = yvz1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
    zvx2 = yvx2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
    zvy2 = yvy2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
    zvz2 = yvz2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
    zvx3 = yvx3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
    zvy3 = yvy3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
    zvz3 = yvz3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
    zvx4 = yvx4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
    zvy4 = yvy4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
    zvz4 = yvz4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
    zvx5 = yvx5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
    zvy5 = yvy5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
    zvz5 = yvz5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
    zvx6 = yvx6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
    zvy6 = yvy6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
    zvz6 = yvz6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
!----------------------------------------------------------------------------------------------------
!   Cross products.
!----------------------------------------------------------------------------------------------------
    zcx1 = zvy1*zvz4 - zvz1*zvy4
    zcy1 = zvz1*zvx4 - zvx1*zvz4
```



```fortran
    zcz1 = zvx1*zvy4 - zvy1*zvx4
    zcx2 = zvy4*zvz2 - zvz4*zvy2
    zcy2 = zvz4*zvx2 - zvx4*zvz2
    zcz2 = zvx4*zvy2 - zvy4*zvx2
    zcx3 = zvy2*zvz5 - zvz2*zvy5
    zcy3 = zvz2*zvx5 - zvx2*zvz5
    zcz3 = zvx2*zvy5 - zvy2*zvx5
    zcx4 = zvy5*zvz3 - zvz5*zvy3
    zcy4 = zvz5*zvx3 - zvx5*zvz3
    zcz4 = zvx5*zvy3 - zvy5*zvx3
    zcx5 = zvy3*zvz6 - zvz3*zvy6
    zcy5 = zvz3*zvx6 - zvx3*zvz6
    zcz5 = zvx3*zvy6 - zvy3*zvx6
    zcx6 = zvy6*zvz1 - zvz6*zvy1
    zcy6 = zvz6*zvx1 - zvx6*zvz1
    zcz6 = zvx6*zvy1 - zvy6*zvx1
!----------------------------------------------------------------------------------------------------
!   Dot products.
!----------------------------------------------------------------------------------------------------
    zd1 = zvx1*zvx4 +zvy1*zvy4 +zvx1*zvy4
    zd2 = zvx4*zvx2 +zvy4*zvy2 +zvx4*zvy2
    zd3 = zvx2*zvx5 +zvy2*zvy5 +zvx2*zvy5
    zd4 = zvx5*zvx3 +zvy5*zvy3 +zvx5*zvy3
    zd5 = zvx3*zvx6 +zvy3*zvy6 +zvx3*zvy6
    zd6 = zvx6*zvx1 +zvy6*zvy1 +zvx6*zvy1
!----------------------------------------------------------------------------------------------------
!   Angles
!----------------------------------------------------------------------------------------------------
    al1 = DACOS( zvx1*zvx4 + zvy1*zvy4 + zvz1*zvz4 )
    al2 = DACOS( zvx4*zvx2 + zvy4*zvy2 + zvz4*zvz2 )
    al3 = DACOS( zvx2*zvx5 + zvy2*zvy5 + zvz2*zvz5 )
    al4 = DACOS( zvx5*zvx3 + zvy5*zvy3 + zvz5*zvz3 )
    al5 = DACOS( zvx3*zvx6 + zvy3*zvy6 + zvz3*zvz6 )
    al6 = DACOS( zvx6*zvx1 + zvy6*zvy1 + zvz6*zvz1 )
!----------------------------------------------------------------------------------------------------
    th1 = DACOS( a1*zvx1 + a2*zvy1 + a3*zvz1 )
    th2 = DACOS( a1*zvx4 + a2*zvy4 + a3*zvz4 )
    th3 = DACOS( a1*zvx2 + a2*zvy2 + a3*zvz2 )
    th4 = DACOS( a1*zvx5 + a2*zvy5 + a3*zvz5 )
    th5 = DACOS( a1*zvx3 + a2*zvy3 + a3*zvz3 )
    th6 = DACOS( a1*zvx6 + a2*zvy6 + a3*zvz6 )
!----------------------------------------------------------------------------------------------------
!   e and g constants
!----------------------------------------------------------------------------------------------------
    e1 = ( DCOS( th2 ) - DCOS( th1 )*DCOS( al1 ) )/DSIN( al1 )
    e2 = ( DCOS( th3 ) - DCOS( th2 )*DCOS( al2 ) )/DSIN( al2 )
    e3 = ( DCOS( th4 ) - DCOS( th3 )*DCOS( al3 ) )/DSIN( al3 )
    e4 = ( DCOS( th5 ) - DCOS( th4 )*DCOS( al4 ) )/DSIN( al4 )
    e5 = ( DCOS( th6 ) - DCOS( th5 )*DCOS( al5 ) )/DSIN( al5 )
    e6 = ( DCOS( th1 ) - DCOS( th6 )*DCOS( al6 ) )/DSIN( al6 )
    f1 = DCOS( th1 )
    f2 = DCOS( th2 )
    f3 = DCOS( th3 )
    f4 = DCOS( th4 )
    f5 = DCOS( th5 )
    f6 = DCOS( th6 )
    g1 = DSQRT( 1.0D0 - e1**2 - f1**2 )
    g2 = DSQRT( 1.0D0 - e2**2 - f2**2 )
    g3 = DSQRT( 1.0D0 - e3**2 - f3**2 )
    g4 = DSQRT( 1.0D0 - e4**2 - f4**2 )
    g5 = DSQRT( 1.0D0 - e5**2 - f5**2 )
    g6 = DSQRT( 1.0D0 - e6**2 - f6**2 )
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1
!----------------------------------------------------------------------------------------------------
        QExx  = ( (zvx1 + zvx4)*zcx1 ) / ( 1.0D0 + zd1 )
        QExy  = ( (zvx1 + zvx4)*zcy1 ) / ( 1.0D0 + zd1 )
        QExz  = ( (zvx1 + zvx4)*zcz1 ) / ( 1.0D0 + zd1 )
        QEyx  = ( (zvy1 + zvy4)*zcx1 ) / ( 1.0D0 + zd1 )
        QEyy  = ( (zvy1 + zvy4)*zcy1 ) / ( 1.0D0 + zd1 )
        QEyz  = ( (zvy1 + zvy4)*zcz1 ) / ( 1.0D0 + zd1 )
        QEzx  = ( (zvz1 + zvz4)*zcx1 ) / ( 1.0D0 + zd1 )
```

```fortran
        QEzy  = ( (zvz1 + zvz4)*zcy1 ) / ( 1.0D0 + zd1 )
        QEzz  = ( (zvz1 + zvz4)*zcz1 ) / ( 1.0D0 + zd1 )
!----------------------------------------------------------------------------------------------------
        PEx  = ( 2.0D0*a1*zcx1/g1 )*( (DATAN(( (1.0D0 - f1)*DTAN(al1/2) + e1 )/ g1)) - DATAN(e1/g1))
        PEy  = ( 2.0D0*a2*zcy1/g1 )*( (DATAN(( (1.0D0 - f1)*DTAN(al1/2) + e1 )/ g1)) - DATAN(e1/g1))
        PEz  = ( 2.0D0*a3*zcz1/g1 )*( (DATAN(( (1.0D0 - f1)*DTAN(al1/2) + e1 )/ g1)) - DATAN(e1/g1))
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 4
!----------------------------------------------------------------------------------------------------
        QExx  = QExx + ( (zvx4 + zvx2)*zcx2 ) / ( 1.0D0 + zd2 )
        QExy  = QExy + ( (zvx4 + zvx2)*zcy2 ) / ( 1.0D0 + zd2 )
        QExz  = QExz + ( (zvx4 + zvx2)*zcz2 ) / ( 1.0D0 + zd2 )
        QEyx  = QEyx + ( (zvy4 + zvy2)*zcx2 ) / ( 1.0D0 + zd2 )
        QEyy  = QEyy + ( (zvy4 + zvy2)*zcy2 ) / ( 1.0D0 + zd2 )
        QEyz  = QEyz + ( (zvy4 + zvy2)*zcz2 ) / ( 1.0D0 + zd2 )
        QEzx  = QEzx + ( (zvz4 + zvz2)*zcx2 ) / ( 1.0D0 + zd2 )
        QEzy  = QEzy + ( (zvz4 + zvz2)*zcy2 ) / ( 1.0D0 + zd2 )
        QEzz  = QEzz + ( (zvz4 + zvz2)*zcz2 ) / ( 1.0D0 + zd2 )
!----------------------------------------------------------------------------------------------------
        PEx  = PEx + ( 2.0D0*a1*zcx2/g2 )*( (DATAN(( (1.0D0 - f2)*DTAN(al2/2) + e2 )/ g2)) - DATAN(e2/g2))
        PEy  = PEy + ( 2.0D0*a2*zcy2/g2 )*( (DATAN(( (1.0D0 - f2)*DTAN(al2/2) + e2 )/ g2)) - DATAN(e2/g2))
        PEz  = PEz + ( 2.0D0*a3*zcz2/g2 )*( (DATAN(( (1.0D0 - f2)*DTAN(al2/2) + e2 )/ g2)) - DATAN(e2/g2))
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2
!----------------------------------------------------------------------------------------------------
        QExx  = QExx + ( (zvx2 + zvx5)*zcx3 ) / ( 1.0D0 + zd3 )
        QExy  = QExy + ( (zvx2 + zvx5)*zcy3 ) / ( 1.0D0 + zd3 )
        QExz  = QExz + ( (zvx2 + zvx5)*zcz3 ) / ( 1.0D0 + zd3 )
        QEyx  = QEyx + ( (zvy2 + zvy5)*zcx3 ) / ( 1.0D0 + zd3 )
        QEyy  = QEyy + ( (zvy2 + zvy5)*zcy3 ) / ( 1.0D0 + zd3 )
        QEyz  = QEyz + ( (zvy2 + zvy5)*zcz3 ) / ( 1.0D0 + zd3 )
        QEzx  = QEzx + ( (zvz2 + zvz5)*zcx3 ) / ( 1.0D0 + zd3 )
        QEzy  = QEzy + ( (zvz2 + zvz5)*zcy3 ) / ( 1.0D0 + zd3 )
        QEzz  = QEzz + ( (zvz2 + zvz5)*zcz3 ) / ( 1.0D0 + zd3 )
!----------------------------------------------------------------------------------------------------
        PEx  = PEx + ( 2.0D0*a1*zcx3/g3 )*( (DATAN(( (1.0D0 - f3)*DTAN(al3/3) + e3 )/ g3)) - DATAN(e3/g3))
        PEy  = PEy + ( 2.0D0*a2*zcy3/g3 )*( (DATAN(( (1.0D0 - f3)*DTAN(al3/3) + e3 )/ g3)) - DATAN(e3/g3))
        PEz  = PEz + ( 2.0D0*a3*zcz3/g3 )*( (DATAN(( (1.0D0 - f3)*DTAN(al3/3) + e3 )/ g3)) - DATAN(e3/g3))
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 5
!----------------------------------------------------------------------------------------------------
        QExx  = QExx + ( (zvx5 + zvx3)*zcx4 ) / ( 1.0D0 + zd4 )
        QExy  = QExy + ( (zvx5 + zvx3)*zcy4 ) / ( 1.0D0 + zd4 )
        QExz  = QExz + ( (zvx5 + zvx3)*zcz4 ) / ( 1.0D0 + zd4 )
        QEyx  = QEyx + ( (zvy5 + zvy3)*zcx4 ) / ( 1.0D0 + zd4 )
        QEyy  = QEyy + ( (zvy5 + zvy3)*zcy4 ) / ( 1.0D0 + zd4 )
        QEyz  = QEyz + ( (zvy5 + zvy3)*zcz4 ) / ( 1.0D0 + zd4 )
        QEzx  = QEzx + ( (zvz5 + zvz3)*zcx4 ) / ( 1.0D0 + zd4 )
        QEzy  = QEzy + ( (zvz5 + zvz3)*zcy4 ) / ( 1.0D0 + zd4 )
        QEzz  = QEzz + ( (zvz5 + zvz3)*zcz4 ) / ( 1.0D0 + zd4 )
!----------------------------------------------------------------------------------------------------
        PEx  = PEx + ( 2.0D0*a1*zcx4/g4 )*( (DATAN(( (1.0D0 - f4)*DTAN(al4/2) + e4 )/ g4)) - DATAN(e4/g4))
        PEy  = PEy + ( 2.0D0*a2*zcy4/g4 )*( (DATAN(( (1.0D0 - f4)*DTAN(al4/2) + e4 )/ g4)) - DATAN(e4/g4))
        PEz  = PEz + ( 2.0D0*a3*zcz4/g4 )*( (DATAN(( (1.0D0 - f4)*DTAN(al4/2) + e4 )/ g4)) - DATAN(e4/g4))
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3
!----------------------------------------------------------------------------------------------------
        QExx  = QExx + ( (zvx3 + zvx6)*zcx5 ) / ( 1.0D0 + zd5 )
        QExy  = QExy + ( (zvx3 + zvx6)*zcy5 ) / ( 1.0D0 + zd5 )
        QExz  = QExz + ( (zvx3 + zvx6)*zcz5 ) / ( 1.0D0 + zd5 )
        QEyx  = QEyx + ( (zvy3 + zvy6)*zcx5 ) / ( 1.0D0 + zd5 )
        QEyy  = QEyy + ( (zvy3 + zvy6)*zcy5 ) / ( 1.0D0 + zd5 )
        QEyz  = QEyz + ( (zvy3 + zvy6)*zcz5 ) / ( 1.0D0 + zd5 )
        QEzx  = QEzx + ( (zvz3 + zvz6)*zcx5 ) / ( 1.0D0 + zd5 )
        QEzy  = QEzy + ( (zvz3 + zvz6)*zcy5 ) / ( 1.0D0 + zd5 )
        QEzz  = QEzz + ( (zvz3 + zvz6)*zcz5 ) / ( 1.0D0 + zd5 )
!----------------------------------------------------------------------------------------------------
        PEx  = PEx + ( 2.0D0*a1*zcx5/g5 )*( (DATAN(( (1.0D0 - f5)*DTAN(al5/2) + e5 )/ g5)) - DATAN(e5/g5))
        PEy  = PEy + ( 2.0D0*a2*zcy5/g5 )*( (DATAN(( (1.0D0 - f5)*DTAN(al5/2) + e5 )/ g5)) - DATAN(e5/g5))
        PEz  = PEz + ( 2.0D0*a3*zcz5/g5 )*( (DATAN(( (1.0D0 - f5)*DTAN(al5/2) + e5 )/ g5)) - DATAN(e5/g5))
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 6
```



```fortran
!----------------------------------------------------------------------------------------------------
        QExx  = QExx + ( (zvx6 + zvx1)*zcx6 ) / ( 1.0D0 + zd6 )
        QExy  = QExy + ( (zvx6 + zvx1)*zcy6 ) / ( 1.0D0 + zd6 )
        QExz  = QExz + ( (zvx6 + zvx1)*zcz6 ) / ( 1.0D0 + zd6 )
        QEyx  = QEyx + ( (zvy6 + zvy1)*zcx6 ) / ( 1.0D0 + zd6 )
        QEyy  = QEyy + ( (zvy6 + zvy1)*zcy6 ) / ( 1.0D0 + zd6 )
        QEyz  = QEyz + ( (zvy6 + zvy1)*zcz6 ) / ( 1.0D0 + zd6 )
        QEzx  = QEzx + ( (zvz6 + zvz1)*zcx6 ) / ( 1.0D0 + zd6 )
        QEzy  = QEzy + ( (zvz6 + zvz1)*zcy6 ) / ( 1.0D0 + zd6 )
        QEzz  = QEzz + ( (zvz6 + zvz1)*zcz6 ) / ( 1.0D0 + zd6 )
!----------------------------------------------------------------------------------------------------
        PEx  = PEx + ( 2.0D0*a1*zcx6/g6 )*( (DATAN(( (1.0D0 - f6)*DTAN(al6/2) + e6 )/ g6)) - DATAN(e6/g6))
        PEy  = PEy + ( 2.0D0*a2*zcy6/g6 )*( (DATAN(( (1.0D0 - f6)*DTAN(al6/2) + e6 )/ g6)) - DATAN(e6/g6))
        PEz  = PEz + ( 2.0D0*a3*zcz6/g6 )*( (DATAN(( (1.0D0 - f6)*DTAN(al6/2) + e6 )/ g6)) - DATAN(e6/g6))
!----------------------------------------------------------------------------------------------------
    TExx = 2.0D0*( QExx + cpi + PE )
    TExy = 2.0D0*( QExy )
    TExz = 2.0D0*( QExz )
    TEyx = 2.0D0*( QEyx )
    TEyy = 2.0D0*( QEyy + cpi + PE )
    TEyz = 2.0D0*( QEyz )
    TEzx = 2.0D0*( QEzx )
    TEzy = 2.0D0*( QEzy )
    TEzz = 2.0D0*( QEzz + cpi + PE )
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE intr_lin_sing_s5
!====================================================================================================
    SUBROUTINE intr_lin_sing_s6(x00,   y00,   z00,    &
                            &  k,                 &
                            &  TExx, TExy, TExz,   &
                            &  TEyx, TEyy, TEyz,   &
                            &  TEzx, TEzy, TEzz    )
!====================================================================================================
!   This subroutine is a new version stokeslet Subroutine.
!Compute:
!   *The value of the Stokeslet over each singular element
!Now, (March/ 09 / 2015) this subroutine was made.
!====================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog, pi, eps, vnx0, vny0, vnz0, x0, y0, z0
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN) :: x00,   y00,   z00            !singulatity coordinates
    INTEGER,           INTENT(IN)  :: k                         !number of element
    REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TExz          !value of stresslet in the singular element
    REAL (KIND = DBL), INTENT(OUT) :: TEyx, TEyy, TEyz          !value of stresslet in the singular element
    REAL (KIND = DBL), INTENT(OUT) :: TEzx, TEzy, TEzz          !value of stresslet in the singular element
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                    !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                  !indices to obtain node numbers from each element
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: cf, fil1, fil2, fil3           !integration weigth coefficients
    REAL (KIND = DBL) :: DxDx, DyDx, DzDx               !tangential vector around the triangle (xi,eta)
    REAL (KIND = DBL) :: hss                            !weigth
    REAL (KIND = DBL) :: cpi                            !Selection to add the solid angle
    REAL (KIND = DBL) :: modx0                          !weigth
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &           !vectoreial product
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  QExx, QExy, QExz, &           !Iq tensor
    &                     QEyx, QEyy, QEyz, &
    &                     QEzx, QEzy, QEzz
    REAL (KIND = DBL) ::  PEx, PEy, PEz, PE             !Ip tensor
    REAL (KIND = DBL) ::  yvx1, yvy1, yvz1                 !vector (y1-x0)
    REAL (KIND = DBL) ::  yvx2, yvy2, yvz2                 !vector (y2-x0)
    REAL (KIND = DBL) ::  yvx3, yvy3, yvz3                 !vector (y2-x0)
    REAL (KIND = DBL) ::  px2, py2, pz2                 !vector (y2-x0)
```

```fortran
    REAL (KIND = DBL) ::  a1, a2, a3                     !vector a
    REAL (KIND = DBL) ::  ay1, ay2, ay3                  !vector dot product a*y1, a*y2
    REAL (KIND = DBL) ::  by1, by2, by3                  !product yi*(yi+a.yi)
!===============================================================================================================
!   Initialize
!---------------------------------------------------------------------------------------------------------------
    TExx = 0.0D0
    TExy = 0.0D0
    TExz = 0.0D0
    TEyx = 0.0D0
    TEyy = 0.0D0
    TEyz = 0.0D0
    TEzx = 0.0D0
    TEzy = 0.0D0
    TEzz = 0.0D0

    QExx = 0.0D0
    QExy = 0.0D0
    QExz = 0.0D0
    QEyx = 0.0D0
    QEyy = 0.0D0
    QEyz = 0.0D0
    QEzx = 0.0D0
    QEzy = 0.0D0
    QEzz = 0.0D0

    PEx = 0.0D0
    PEy = 0.0D0
    PEz = 0.0D0
    PE  = 0.0D0

    modx0= DSQRT((x0(k)-x00)**2+(y0(k)-y00)**2+(z0(k)-z00)**2)

    IF (modx0 >= 1.0D0 + eps) THEN
        cpi = 0.0D0
    ELSE IF (modx0 <= 1.0D0 + eps) THEN
        cpi = -8.0D0*pi
    ELSE
        cpi = -4.0D0*pi
    end if

    IF((ABS(x00-x0(k)) <= eps) .AND. &
    &  (ABS(y00-y0(k)) <= eps) .AND. &
    &  (ABS(z00-z0(k)) <= eps)) THEN
        a1   = x00
        a2   = y00
        a3   = z00
    ELSE
        a1   = x00-x0(k)
        a2   = y00-y0(k)
        a3   = z00-z0(k)
    END IF
    modx0= DSQRT(a1**2+a2**2+a3**2)
    a1   = a1/modx0
    a2   = a2/modx0
    a3   = a3/modx0

    !OPEN (9,file="TE.out")
!---------------------------------------------------------------------------------------------------------------
!   vertices of the kth triangle
!---------------------------------------------------------------------------------------------------------------
    i1 = n(k,1)
    i2 = n(k,2)
    i3 = n(k,3)
    i4 = n(k,4)
    i5 = n(k,5)
    i6 = n(k,6)
!---------------------------------------------------------------------------------------------------------------
        cf  = 0.0D0
!---------------------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
! rule formula
```



```fortran
!---------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!---------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-6
!---------------------------------------------------------------------------------------------------------------
        DxDx = p(i6,1) - p(i1,1)
        DyDx = p(i6,2) - p(i1,2)
        DzDx = p(i6,3) - p(i1,3)
!---------------------------------------------------------------------------------------------------------------
        px2  = p(i1,1) + 0.5D0*DxDx
        py2  = p(i1,2) + 0.5D0*DyDx
        pz2  = p(i1,3) + 0.5D0*DzDx
!---------------------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!---------------------------------------------------------------------------------------------------------------
        yvx1 = p(i1,1) - x00
        yvy1 = p(i1,2) - y00
        yvz1 = p(i1,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!---------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!---------------------------------------------------------------------------------------------------------------
        yvx2 = px2 - x00
        yvy2 = py2 - y00
        yvz2 = pz2 - z00
        fil2 = DSQRT(yvx2**2 + yvy2**2 +yvz2**2)
!---------------------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz2 - DzDx*yvy2)
        bvy2 = (DzDx*yvx2 - DxDx*yvz2)
        bvz2 = (DxDx*yvy2 - DyDx*yvx2)
!---------------------------------------------------------------------------------------------------------------
        yvx3 = p(i6,1) - x00
        yvy3 = p(i6,2) - y00
        yvz3 = p(i6,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!---------------------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!---------------------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - 4.0D0*(bvx2*(yvx2/fil2**3)) - (bvx3*(yvx3/fil3**3)) )/3.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - 4.0D0*(bvy2*(yvx2/fil2**3)) - (bvy3*(yvx3/fil3**3)) )/3.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - 4.0D0*(bvz2*(yvx2/fil2**3)) - (bvz3*(yvx3/fil3**3)) )/3.0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - 4.0D0*(bvx2*(yvy2/fil2**3)) - (bvx3*(yvy3/fil3**3)) )/3.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - 4.0D0*(bvy2*(yvy2/fil2**3)) - (bvy3*(yvy3/fil3**3)) )/3.0D0
```

```fortran
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - 4.0D0*(bvz2*(yvy2/fil2**3)) - (bvz3*(yvy3/fil3**3)) )/3.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - 4.0D0*(bvx2*(yvz2/fil2**3)) - (bvx3*(yvz3/fil3**3)) )/3.0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - 4.0D0*(bvy2*(yvz2/fil2**3)) - (bvy3*(yvz3/fil3**3)) )/3.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - 4.0D0*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.0D0
!------------------------------------------------------------------------------------------------------------
        !a1   = (bvx1 + bvx2 + bvx3)/3.0D0
        !a2   = (bvy1 + bvy2 + bvy3)/3.0D0
        !a3   = (bvz1 + bvz2 + bvz3)/3.0D0
        !modx0= DSQRT(a1**2+a2**2+a3**2)
        !a1   = a1/modx0
        !a2   = a2/modx0
        !a3   = a3/modx0
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
        !by1  = DSQRT(( fil1*(fil1+ay1) + 0.3 )**2)
        !by2  = DSQRT(( fil2*(fil2+ay2) + 0.3 )**2)
        !by3  = DSQRT(( fil3*(fil3+ay3) + 0.3 )**2)
        by1  = DSQRT(( fil1*(fil1+ay1) )**2)
        by2  = DSQRT(( fil2*(fil2+ay2) )**2)
        by3  = DSQRT(( fil3*(fil3+ay3) )**2)
        !WRITE (9,*) by1
        !WRITE (9,*) by2
        !WRITE (9,*) by3
        PEx  = ( (a1*bvx1/by1) + 4.0D0*(a2*bvx2/by2) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + 4.0D0*(a2*bvy2/by2) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + 4.0D0*(a2*bvz2/by2) + (a3*bvz3/by3) )*a3
        !ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        !ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        !ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
        !by1  = fil1*(fil1+ay1)
        !by2  = fil2*(fil2+ay2)
        !by3  = fil3*(fil3+ay3)
        !PEx  = ( (bvx1/by1) + 4.0D0*(bvx2/by2) + (bvx3/by3) )*a1
        !PEy  = ( (bvy1/by1) + 4.0D0*(bvy2/by2) + (bvy3/by3) )*a2
        !PEz  = ( (bvz1/by1) + 4.0D0*(bvz2/by2) + (bvz3/by3) )*a3
!        PEx  = ( (bvx1**2/(fil1**2)) + 4.0D0*(bvx2**2/(fil2**2)) + (bvx3**2/(fil3**2)) )
!        PEy  = ( (bvy1**2/(fil1**2)) + 4.0D0*(bvy2**2/(fil2**2)) + (bvy3**2/(fil3**2)) )
!        PEz  = ( (bvz1**2/(fil1**2)) + 4.0D0*(bvz2**2/(fil2**2)) + (bvz3**2/(fil3**2)) )
        PE   =  hss*( PEx + PEy + PEz )/3.0D0
!------------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 6-3
!------------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!------------------------------------------------------------------------------------------------------------
        DxDx = p(i3,1) - p(i6,1)
        DyDx = p(i3,2) - p(i6,2)
        DzDx = p(i3,3) - p(i6,3)
!------------------------------------------------------------------------------------------------------------
        px2  = p(i6,1) + 0.5D0*DxDx
        py2  = p(i6,2) + 0.5D0*DyDx
```



```fortran
        pz2  = p(i6,3) + 0.5D0*DzDx
!------------------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!------------------------------------------------------------------------------------------------------------
        yvx1 = p(i6,1) - x00
        yvy1 = p(i6,2) - y00
        yvz1 = p(i6,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!------------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!------------------------------------------------------------------------------------------------------------
        yvx2 = px2 - x00
        yvy2 = py2 - y00
        yvz2 = pz2 - z00
        fil2 = DSQRT(yvx2**2 + yvy2**2 +yvz2**2)
!------------------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz2 - DzDx*yvy2)
        bvy2 = (DzDx*yvx2 - DxDx*yvz2)
        bvz2 = (DxDx*yvy2 - DyDx*yvx2)
!------------------------------------------------------------------------------------------------------------
        yvx3 = p(i3,1) - x00
        yvy3 = p(i3,2) - y00
        yvz3 = p(i3,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!------------------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!------------------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - 4.0D0*(bvx2*(yvx2/fil2**3)) - (bvx3*(yvx3/fil3**3)) )/3.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - 4.0D0*(bvy2*(yvx2/fil2**3)) - (bvy3*(yvx3/fil3**3)) )/3.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - 4.0D0*(bvz2*(yvx2/fil2**3)) - (bvz3*(yvx3/fil3**3)) )/3.0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - 4.0D0*(bvx2*(yvy2/fil2**3)) - (bvx3*(yvy3/fil3**3)) )/3.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - 4.0D0*(bvy2*(yvy2/fil2**3)) - (bvy3*(yvy3/fil3**3)) )/3.0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - 4.0D0*(bvz2*(yvy2/fil2**3)) - (bvz3*(yvy3/fil3**3)) )/3.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - 4.0D0*(bvx2*(yvz2/fil2**3)) - (bvx3*(yvz3/fil3**3)) )/3.0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - 4.0D0*(bvy2*(yvz2/fil2**3)) - (bvy3*(yvz3/fil3**3)) )/3.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - 4.0D0*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.0D0
!------------------------------------------------------------------------------------------------------------
        !a1   = (bvx1 + bvx2 + bvx3)/3.0D0
        !a2   = (bvy1 + bvy2 + bvy3)/3.0D0
        !a3   = (bvz1 + bvz2 + bvz3)/3.0D0
        !modx0= DSQRT(a1**2+a2**2+a3**2)
        !a1   = a1/modx0
        !a2   = a2/modx0
        !a3   = a3/modx0
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
        !by1  = DSQRT(( fil1*(fil1+ay1) + 0.3 )**2)
        !by2  = DSQRT(( fil2*(fil2+ay2) + 0.3 )**2)
        !by3  = DSQRT(( fil3*(fil3+ay3) + 0.3 )**2)
        by1  = DSQRT(( fil1*(fil1+ay1) )**2)
        by2  = DSQRT(( fil2*(fil2+ay2) )**2)
        by3  = DSQRT(( fil3*(fil3+ay3) )**2)
        !WRITE (9,*) by1
        !WRITE (9,*) by2
        !WRITE (9,*) by3
```

```
        PEx  = ( (a1*bvx1/by1) + 4.0D0*(a2*bvx2/by2) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + 4.0D0*(a2*bvy2/by2) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + 4.0D0*(a2*bvz2/by2) + (a3*bvz3/by3) )*a3
        !ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        !ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        !ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
        !by1  = fil1*(fil1+ay1)
        !by2  = fil2*(fil2+ay2)
        !by3  = fil3*(fil3+ay3)
        !PEx  = ( (bvx1/by1) + 4.0D0*(bvx2/by2) + (bvx3/by3) )*a1
        !PEy  = ( (bvy1/by1) + 4.0D0*(bvy2/by2) + (bvy3/by3) )*a2
        !PEz  = ( (bvz1/by1) + 4.0D0*(bvz2/by2) + (bvz3/by3) )*a3
!        PEx  = ( (bvx1**2/(fil1**2)) + 4.0D0*(bvx2**2/(fil2**2)) + (bvx3**2/(fil3**2)) )
!        PEy  = ( (bvy1**2/(fil1**2)) + 4.0D0*(bvy2**2/(fil2**2)) + (bvy3**2/(fil3**2)) )
!        PEz  = ( (bvz1**2/(fil1**2)) + 4.0D0*(bvz2**2/(fil2**2)) + (bvz3**2/(fil3**2)) )
        PE   = PE + hss*( PEx + PEy + PEz )/3.0D0
!----------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-5
!----------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!----------------------------------------------------------------------------------------------------------
        DxDx = p(i5,1) - p(i3,1)
        DyDx = p(i5,2) - p(i3,2)
        DzDx = p(i5,3) - p(i3,3)
!----------------------------------------------------------------------------------------------------------
        px2  = p(i3,1) + 0.5D0*DxDx
        py2  = p(i3,2) + 0.5D0*DyDx
        pz2  = p(i3,3) + 0.5D0*DzDx
!----------------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!----------------------------------------------------------------------------------------------------------
        yvx1 = p(i3,1) - x00
        yvy1 = p(i3,2) - y00
        yvz1 = p(i3,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!----------------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!----------------------------------------------------------------------------------------------------------
        yvx2 = px2 - x00
        yvy2 = py2 - y00
        yvz2 = pz2 - z00
        fil2 = DSQRT(yvx2**2 + yvy2**2 +yvz2**2)
!----------------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz2 - DzDx*yvy2)
        bvy2 = (DzDx*yvx2 - DxDx*yvz2)
        bvz2 = (DxDx*yvy2 - DyDx*yvx2)
!----------------------------------------------------------------------------------------------------------
        yvx3 = p(i5,1) - x00
        yvy3 = p(i5,2) - y00
        yvz3 = p(i5,3) - z00
```



```
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!----------------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!----------------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - 4.0D0*(bvx2*(yvx2/fil2**3)) - (bvx3*(yvx3/fil3**3)) )/3.
    0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - 4.0D0*(bvy2*(yvx2/fil2**3)) - (bvy3*(yvx3/fil3**3)) )/3.
    0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - 4.0D0*(bvz2*(yvx2/fil2**3)) - (bvz3*(yvx3/fil3**3)) )/3.
    0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - 4.0D0*(bvx2*(yvy2/fil2**3)) - (bvx3*(yvy3/fil3**3)) )/3.
    0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - 4.0D0*(bvy2*(yvy2/fil2**3)) - (bvy3*(yvy3/fil3**3)) )/3.
    0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - 4.0D0*(bvz2*(yvy2/fil2**3)) - (bvz3*(yvy3/fil3**3)) )/3.
    0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - 4.0D0*(bvx2*(yvz2/fil2**3)) - (bvx3*(yvz3/fil3**3)) )/3.
    0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - 4.0D0*(bvy2*(yvz2/fil2**3)) - (bvy3*(yvz3/fil3**3)) )/3.
    0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - 4.0D0*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
    0D0
!----------------------------------------------------------------------------------------------------------
        !a1   = (bvx1 + bvx2 + bvx3)/3.0D0
        !a2   = (bvy1 + bvy2 + bvy3)/3.0D0
        !a3   = (bvz1 + bvz2 + bvz3)/3.0D0
        !modx0= DSQRT(a1**2+a2**2+a3**2)
        !a1   = a1/modx0
        !a2   = a2/modx0
        !a3   = a3/modx0
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
        !by1  = DSQRT(( fil1*(fil1+ay1) + 0.3 )**2)
        !by2  = DSQRT(( fil2*(fil2+ay2) + 0.3 )**2)
        !by3  = DSQRT(( fil3*(fil3+ay3) + 0.3 )**2)
        by1  = DSQRT(( fil1*(fil1+ay1) )**2)
        by2  = DSQRT(( fil2*(fil2+ay2) )**2)
        by3  = DSQRT(( fil3*(fil3+ay3) )**2)
        !WRITE (9,*) by1
        !WRITE (9,*) by2
        !WRITE (9,*) by3
        PEx  = ( (a1*bvx1/by1) + 4.0D0*(a2*bvx2/by2) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + 4.0D0*(a2*bvy2/by2) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + 4.0D0*(a2*bvz2/by2) + (a3*bvz3/by3) )*a3
        !ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        !ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        !ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
        !by1  = fil1*(fil1+ay1)
        !by2  = fil2*(fil2+ay2)
        !by3  = fil3*(fil3+ay3)
        !PEx  = ( (bvx1/by1) + 4.0D0*(bvx2/by2) + (bvx3/by3) )*a1
        !PEy  = ( (bvy1/by1) + 4.0D0*(bvy2/by2) + (bvy3/by3) )*a2
        !PEz  = ( (bvz1/by1) + 4.0D0*(bvz2/by2) + (bvz3/by3) )*a3
!        PEx  = ( (bvx1**2/(fil1**2)) + 4.0D0*(bvx2**2/(fil2**2)) + (bvx3**2/(fil3**2)) )
!        PEy  = ( (bvy1**2/(fil1**2)) + 4.0D0*(bvy2**2/(fil2**2)) + (bvy3**2/(fil3**2)) )
!        PEz  = ( (bvz1**2/(fil1**2)) + 4.0D0*(bvz2**2/(fil2**2)) + (bvz3**2/(fil3**2)) )
        PE   = PE + hss*( PEx + PEy + PEz )/3.0D0
!----------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 5-2
!----------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
```

```fortran
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!-----------------------------------------------------------------------------------------------------
        DxDx = p(i2,1) - p(i5,1)
        DyDx = p(i2,2) - p(i5,2)
        DzDx = p(i2,3) - p(i5,3)
!-----------------------------------------------------------------------------------------------------
        px2  = p(i5,1) + 0.5D0*DxDx
        py2  = p(i5,2) + 0.5D0*DyDx
        pz2  = p(i5,3) + 0.5D0*DzDx
!-----------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!-----------------------------------------------------------------------------------------------------
        yvx1 = p(i5,1) - x00
        yvy1 = p(i5,2) - y00
        yvz1 = p(i5,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!-----------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!-----------------------------------------------------------------------------------------------------
        yvx2 = px2 - x00
        yvy2 = py2 - y00
        yvz2 = pz2 - z00
        fil2 = DSQRT(yvx2**2 + yvy2**2 +yvz2**2)
!-----------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz2 - DzDx*yvy2)
        bvy2 = (DzDx*yvx2 - DxDx*yvz2)
        bvz2 = (DxDx*yvy2 - DyDx*yvx2)
!-----------------------------------------------------------------------------------------------------
        yvx3 = p(i2,1) - x00
        yvy3 = p(i2,2) - y00
        yvz3 = p(i2,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!-----------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!-----------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - 4.0D0*(bvx2*(yvx2/fil2**3)) - (bvx3*(yvx3/fil3**3)) )/3. ↙
    0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - 4.0D0*(bvy2*(yvx2/fil2**3)) - (bvy3*(yvx3/fil3**3)) )/3. ↙
    0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - 4.0D0*(bvz2*(yvx2/fil2**3)) - (bvz3*(yvx3/fil3**3)) )/3. ↙
    0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - 4.0D0*(bvx2*(yvy2/fil2**3)) - (bvx3*(yvy3/fil3**3)) )/3. ↙
    0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - 4.0D0*(bvy2*(yvy2/fil2**3)) - (bvy3*(yvy3/fil3**3)) )/3. ↙
    0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - 4.0D0*(bvz2*(yvy2/fil2**3)) - (bvz3*(yvy3/fil3**3)) )/3. ↙
    0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - 4.0D0*(bvx2*(yvz2/fil2**3)) - (bvx3*(yvz3/fil3**3)) )/3. ↙
    0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - 4.0D0*(bvy2*(yvz2/fil2**3)) - (bvy3*(yvz3/fil3**3)) )/3. ↙
    0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - 4.0D0*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3. ↙
    0D0
!-----------------------------------------------------------------------------------------------------
        !a1   = (bvx1 + bvx2 + bvx3)/3.0D0
        !a2   = (bvy1 + bvy2 + bvy3)/3.0D0
        !a3   = (bvz1 + bvz2 + bvz3)/3.0D0
```



```fortran
        !modx0= DSQRT(a1**2+a2**2+a3**2)
        !a1   = a1/modx0
        !a2   = a2/modx0
        !a3   = a3/modx0
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
        !by1  = DSQRT(( fil1*(fil1+ay1) + 0.3 )**2)
        !by2  = DSQRT(( fil2*(fil2+ay2) + 0.3 )**2)
        !by3  = DSQRT(( fil3*(fil3+ay3) + 0.3 )**2)
        by1  = DSQRT(( fil1*(fil1+ay1) )**2)
        by2  = DSQRT(( fil2*(fil2+ay2) )**2)
        by3  = DSQRT(( fil3*(fil3+ay3) )**2)
        !WRITE (9,*) by1
        !WRITE (9,*) by2
        !WRITE (9,*) by3
        PEx  = ( (a1*bvx1/by1) + 4.0D0*(a2*bvx2/by2) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + 4.0D0*(a2*bvy2/by2) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + 4.0D0*(a2*bvz2/by2) + (a3*bvz3/by3) )*a3
        !ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        !ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        !ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
        !by1  = fil1*(fil1+ay1)
        !by2  = fil2*(fil2+ay2)
        !by3  = fil3*(fil3+ay3)
        !PEx  = ( (bvx1/by1) + 4.0D0*(bvx2/by2) + (bvx3/by3) )*a1
        !PEy  = ( (bvy1/by1) + 4.0D0*(bvy2/by2) + (bvy3/by3) )*a2
        !PEz  = ( (bvz1/by1) + 4.0D0*(bvz2/by2) + (bvz3/by3) )*a3
!        PEx  = ( (bvx1**2/(fil1**2)) + 4.0D0*(bvx2**2/(fil2**2)) + (bvx3**2/(fil3**2)) )
!        PEy  = ( (bvy1**2/(fil1**2)) + 4.0D0*(bvy2**2/(fil2**2)) + (bvy3**2/(fil3**2)) )
!        PEz  = ( (bvz1**2/(fil1**2)) + 4.0D0*(bvz2**2/(fil2**2)) + (bvz3**2/(fil3**2)) )
        PE   = PE + hss*( PEx + PEy + PEz )/3.0D0
!-----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-4
!-----------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!-----------------------------------------------------------------------------------------------------
        DxDx = p(i4,1) - p(i2,1)
        DyDx = p(i4,2) - p(i2,2)
        DzDx = p(i4,3) - p(i2,3)
!-----------------------------------------------------------------------------------------------------
        px2  = p(i2,1) + 0.5D0*DxDx
        py2  = p(i2,2) + 0.5D0*DyDx
        pz2  = p(i2,3) + 0.5D0*DzDx
!-----------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!-----------------------------------------------------------------------------------------------------
        yvx1 = p(i2,1) - x00
        yvy1 = p(i2,2) - y00
        yvz1 = p(i2,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!-----------------------------------------------------------------------------------------------------
```

```fortran
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!-----------------------------------------------------------------------------------------------------
        yvx2 = px2 - x00
        yvy2 = py2 - y00
        yvz2 = pz2 - z00
        fil2 = DSQRT(yvx2**2 + yvy2**2 +yvz2**2)
!-----------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz2 - DzDx*yvy2)
        bvy2 = (DzDx*yvx2 - DxDx*yvz2)
        bvz2 = (DxDx*yvy2 - DyDx*yvx2)
!-----------------------------------------------------------------------------------------------------
        yvx3 = p(i4,1) - x00
        yvy3 = p(i4,2) - y00
        yvz3 = p(i4,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!-----------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!-----------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - 4.0D0*(bvx2*(yvx2/fil2**3)) - (bvx3*(yvx3/fil3**3)) )/3.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - 4.0D0*(bvy2*(yvx2/fil2**3)) - (bvy3*(yvx3/fil3**3)) )/3.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - 4.0D0*(bvz2*(yvx2/fil2**3)) - (bvz3*(yvx3/fil3**3)) )/3.0D0
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - 4.0D0*(bvx2*(yvy2/fil2**3)) - (bvx3*(yvy3/fil3**3)) )/3.0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - 4.0D0*(bvy2*(yvy2/fil2**3)) - (bvy3*(yvy3/fil3**3)) )/3.0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - 4.0D0*(bvz2*(yvy2/fil2**3)) - (bvz3*(yvy3/fil3**3)) )/3.0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - 4.0D0*(bvx2*(yvz2/fil2**3)) - (bvx3*(yvz3/fil3**3)) )/3.0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - 4.0D0*(bvy2*(yvz2/fil2**3)) - (bvy3*(yvz3/fil3**3)) )/3.0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - 4.0D0*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.0D0
!-----------------------------------------------------------------------------------------------------
        !a1   = (bvx1 + bvx2 + bvx3)/3.0D0
        !a2   = (bvy1 + bvy2 + bvy3)/3.0D0
        !a3   = (bvz1 + bvz2 + bvz3)/3.0D0
        !modx0= DSQRT(a1**2+a2**2+a3**2)
        !a1   = a1/modx0
        !a2   = a2/modx0
        !a3   = a3/modx0
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
        !by1  = DSQRT(( fil1*(fil1+ay1) + 0.3 )**2)
        !by2  = DSQRT(( fil2*(fil2+ay2) + 0.3 )**2)
        !by3  = DSQRT(( fil3*(fil3+ay3) + 0.3 )**2)
        by1  = DSQRT(( fil1*(fil1+ay1) )**2)
        by2  = DSQRT(( fil2*(fil2+ay2) )**2)
        by3  = DSQRT(( fil3*(fil3+ay3) )**2)
        !WRITE (9,*) by1
        !WRITE (9,*) by2
        !WRITE (9,*) by3
        PEx  = ( (a1*bvx1/by1) + 4.0D0*(a2*bvx2/by2) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + 4.0D0*(a2*bvy2/by2) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + 4.0D0*(a2*bvz2/by2) + (a3*bvz3/by3) )*a3
        !ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        !ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        !ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
        !by1  = fil1*(fil1+ay1)
        !by2  = fil2*(fil2+ay2)
        !by3  = fil3*(fil3+ay3)
        !PEx  = ( (bvx1/by1) + 4.0D0*(bvx2/by2) + (bvx3/by3) )*a1
        !PEy  = ( (bvy1/by1) + 4.0D0*(bvy2/by2) + (bvy3/by3) )*a2
        !PEz  = ( (bvz1/by1) + 4.0D0*(bvz2/by2) + (bvz3/by3) )*a3
```



```fortran
!        PEx  = ( (bvx1**2/(fil1**2)) + 4.0D0*(bvx2**2/(fil2**2)) + (bvx3**2/(fil3**2)) )
!        PEy  = ( (bvy1**2/(fil1**2)) + 4.0D0*(bvy2**2/(fil2**2)) + (bvy3**2/(fil3**2)) )
!        PEz  = ( (bvz1**2/(fil1**2)) + 4.0D0*(bvz2**2/(fil2**2)) + (bvz3**2/(fil3**2)) )
        PE   = PE + hss*( PEx + PEy + PEz )/3.0D0
!-----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 4-1
!-----------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
!-----------------------------------------------------------------------------------------------------
        DxDx = p(i1,1) - p(i4,1)
        DyDx = p(i1,2) - p(i4,2)
        DzDx = p(i1,3) - p(i4,3)
!-----------------------------------------------------------------------------------------------------
        px2  = p(i4,1) + 0.5D0*DxDx
        py2  = p(i4,2) + 0.5D0*DyDx
        pz2  = p(i4,3) + 0.5D0*DzDx
!-----------------------------------------------------------------------------------------------------
        hss  = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
        DxDx = DxDx/hss
        DyDx = DyDx/hss
        DzDx = DzDx/hss
!-----------------------------------------------------------------------------------------------------
        yvx1 = p(i4,1) - x00
        yvy1 = p(i4,2) - y00
        yvz1 = p(i4,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!-----------------------------------------------------------------------------------------------------
        bvx1 = (DyDx*yvz1 - DzDx*yvy1)
        bvy1 = (DzDx*yvx1 - DxDx*yvz1)
        bvz1 = (DxDx*yvy1 - DyDx*yvx1)
!-----------------------------------------------------------------------------------------------------
        yvx2 = px2 - x00
        yvy2 = py2 - y00
        yvz2 = pz2 - z00
        fil2 = DSQRT(yvx2**2 + yvy2**2 +yvz2**2)
!-----------------------------------------------------------------------------------------------------
        bvx2 = (DyDx*yvz2 - DzDx*yvy2)
        bvy2 = (DzDx*yvx2 - DxDx*yvz2)
        bvz2 = (DxDx*yvy2 - DyDx*yvx2)
!-----------------------------------------------------------------------------------------------------
        yvx3 = p(i1,1) - x00
        yvy3 = p(i1,2) - y00
        yvz3 = p(i1,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!-----------------------------------------------------------------------------------------------------
        bvx3 = (DyDx*yvz3 - DzDx*yvy3)
        bvy3 = (DzDx*yvx3 - DxDx*yvz3)
        bvz3 = (DxDx*yvy3 - DyDx*yvx3)
!-----------------------------------------------------------------------------------------------------
        QExx  = QExx + hss*( - (bvx1*(yvx1/fil1**3)) - 4.0D0*(bvx2*(yvx2/fil2**3)) - (bvx3*(yvx3/fil3**3)) )/3.0D0
        QExy  = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - 4.0D0*(bvy2*(yvx2/fil2**3)) - (bvy3*(yvx3/fil3**3)) )/3.0D0
        QExz  = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - 4.0D0*(bvz2*(yvx2/fil2**3)) - (bvz3*(yvx3/fil3**3)) )/3.0D0
```

```fortran
        QEyx  = QEyx + hss*( - (bvx1*(yvy1/fil1**3)) - 4.0D0*(bvx2*(yvy2/fil2**3)) - (bvx3*(yvy3/fil3**3)) )/3.
    0D0
        QEyy  = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - 4.0D0*(bvy2*(yvy2/fil2**3)) - (bvy3*(yvy3/fil3**3)) )/3.
    0D0
        QEyz  = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - 4.0D0*(bvz2*(yvy2/fil2**3)) - (bvz3*(yvy3/fil3**3)) )/3.
    0D0
        QEzx  = QEzx + hss*( - (bvx1*(yvz1/fil1**3)) - 4.0D0*(bvx2*(yvz2/fil2**3)) - (bvx3*(yvz3/fil3**3)) )/3.
    0D0
        QEzy  = QEzy + hss*( - (bvy1*(yvz1/fil1**3)) - 4.0D0*(bvy2*(yvz2/fil2**3)) - (bvy3*(yvz3/fil3**3)) )/3.
    0D0
        QEzz  = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - 4.0D0*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
    0D0
!---------------------------------------------------------------------------------------------------------------
        !a1   = (bvx1 + bvx2 + bvx3)/3.0D0
        !a2   = (bvy1 + bvy2 + bvy3)/3.0D0
        !a3   = (bvz1 + bvz2 + bvz3)/3.0D0
        !modx0= DSQRT(a1**2+a2**2+a3**2)
        !a1   = a1/modx0
        !a2   = a2/modx0
        !a3   = a3/modx0
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
        !by1  = DSQRT(( fil1*(fil1+ay1) + 0.3 )**2)
        !by2  = DSQRT(( fil2*(fil2+ay2) + 0.3 )**2)
        !by3  = DSQRT(( fil3*(fil3+ay3) + 0.3 )**2)
        by1  = DSQRT(( fil1*(fil1+ay1) )**2)
        by2  = DSQRT(( fil2*(fil2+ay2) )**2)
        by3  = DSQRT(( fil3*(fil3+ay3) )**2)
        !WRITE (9,*) by1
        !WRITE (9,*) by2
        !WRITE (9,*) by3
        PEx  = ( (a1*bvx1/by1) + 4.0D0*(a2*bvx2/by2) + (a3*bvx3/by3) )*a1
        PEy  = ( (a1*bvy1/by1) + 4.0D0*(a2*bvy2/by2) + (a3*bvy3/by3) )*a2
        PEz  = ( (a1*bvz1/by1) + 4.0D0*(a2*bvz2/by2) + (a3*bvz3/by3) )*a3
        !ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        !ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        !ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
        !by1  = fil1*(fil1+ay1)
        !by2  = fil2*(fil2+ay2)
        !by3  = fil3*(fil3+ay3)
        !PEx  = ( (bvx1/by1) + 4.0D0*(bvx2/by2) + (bvx3/by3) )*a1
        !PEy  = ( (bvy1/by1) + 4.0D0*(bvy2/by2) + (bvy3/by3) )*a2
        !PEz  = ( (bvz1/by1) + 4.0D0*(bvz2/by2) + (bvz3/by3) )*a3
!        PEx  = ( (bvx1**2/(fil1**2)) + 4.0D0*(bvx2**2/(fil2**2)) + (bvx3**2/(fil3**2)) )
!        PEy  = ( (bvy1**2/(fil1**2)) + 4.0D0*(bvy2**2/(fil2**2)) + (bvy3**2/(fil3**2)) )
!        PEz  = ( (bvz1**2/(fil1**2)) + 4.0D0*(bvz2**2/(fil2**2)) + (bvz3**2/(fil3**2)) )
        PE   = PE + hss*( PEx + PEy + PEz )/3.0D0
!        Close(9)
!---------------------------------------------------------------------------------------------------------------
    TExx = 2.0D0*( -QExx + cpi -PE )
    TExy = 2.0D0*( -QExy )
    TExz = 2.0D0*( -QExz )
    TEyx = 2.0D0*( -QEyx )
    TEyy = 2.0D0*( -QEyy + cpi -PE )
    TEyz = 2.0D0*( -QEyz )
    TEzx = 2.0D0*( -QEzx )
    TEzy = 2.0D0*( -QEzy )
    TEzz = 2.0D0*( -QEzz + cpi -PE )
!---------------------------------------------------------------------------------------------------------------
    END SUBROUTINE intr_lin_sing_s6
!===============================================================================================================
    SUBROUTINE intr_lin_sing_s7(x10,   y10,   z10,    &
                           &  k,                  &
                           &  TExx, TExy, TExz,   &
                           &  TEyx, TEyy, TEyz,   &
                           &  TEzx, TEzy, TEzz    )
!===============================================================================================================
!   This subroutine is a new version stokeslet Subroutine.
!Compute:
!   *The value of the Stokeslet over each singular element
!Now, (March/ 09 / 2015) this subroutine was made.
```



```fortran
!===============================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog, pi, eps, x0, y0, z0
!===============================================================================================================
    IMPLICIT NONE
!===============================================================================================================
!   Variables
!---------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN) :: x10,   y10,   z10             !singulatity coordinates
    INTEGER,           INTENT(IN)  :: k                        !number of element
    REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TExz          !value of stresslet in the singular element
    REAL (KIND = DBL), INTENT(OUT) :: TEyx, TEyy, TEyz          !value of stresslet in the singular element
    REAL (KIND = DBL), INTENT(OUT) :: TEzx, TEzy, TEzz          !value of stresslet in the singular element
!===============================================================================================================
!   Variables inside the subroutine
!---------------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                   !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6             !indices to obtain node numbers from each element
!---------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: cf, fil1, fil2, fil3          !integration weigth coefficients
    REAL (KIND = DBL) :: DxDx, DyDx, DzDx              !tangential vector around the triangle (xi,eta)
    REAL (KIND = DBL) :: hss                           !weigth
    REAL (KIND = DBL) :: cpi                           !Selection to add the solid angle
    REAL (KIND = DBL) :: modx0                         !weigth
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &          !vectoreial product
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  QExx, QExy, QExz, &          !Iq tensor
    &                     QEyx, QEyy, QEyz, &
    &                     QEzx, QEzy, QEzz
    REAL (KIND = DBL) ::  PEx, PEy, PEz, PE            !Ip tensor
    REAL (KIND = DBL) ::  yvx1, yvy1, yvz1                !vector y1
    REAL (KIND = DBL) ::  yvx2, yvy2, yvz2                !vector y2
    REAL (KIND = DBL) ::  yvx3, yvy3, yvz3                !vector y3
    REAL (KIND = DBL) ::  yvx4, yvy4, yvz4                !vector y4
    REAL (KIND = DBL) ::  yvx5, yvy5, yvz5                !vector y5
    REAL (KIND = DBL) ::  yvx6, yvy6, yvz6                !vector y6
    REAL (KIND = DBL) ::  zvx1, zvy1, zvz1                !vector (z1-x0)
    REAL (KIND = DBL) ::  zvx2, zvy2, zvz2                !vector (z2-x0)
    REAL (KIND = DBL) ::  zvx3, zvy3, zvz3                !vector (z3-x0)
    REAL (KIND = DBL) ::  zvx4, zvy4, zvz4                !vector (z4-x0)
    REAL (KIND = DBL) ::  zvx5, zvy5, zvz5                !vector (z5-x0)
    REAL (KIND = DBL) ::  zvx6, zvy6, zvz6                !vector (z6-x0)
    REAL (KIND = DBL) ::  zcx1, zcy1, zcz1                !vector zi ^ zi+1
    REAL (KIND = DBL) ::  zcx2, zcy2, zcz2                !vector zi ^ zi+1
    REAL (KIND = DBL) ::  zcx3, zcy3, zcz3                !vector zi ^ zi+1
    REAL (KIND = DBL) ::  zcx4, zcy4, zcz4                !vector zi ^ zi+1
    REAL (KIND = DBL) ::  zcx5, zcy5, zcz5                !vector zi ^ zi+1
    REAL (KIND = DBL) ::  zcx6, zcy6, zcz6                !vector zi ^ zi+1
    REAL (KIND = DBL) ::  zd1, zd2, zd3, zd4, zd5, zd6    !vector zi . zi+1
    REAL (KIND = DBL) ::  px2, py2, pz2                !preasure vector
    REAL (KIND = DBL) ::  a1, a2, a3                   !vector a
    REAL (KIND = DBL) ::  al1, al2, al3, al4, al5, al6  !alpha angle between zi and zi+1 from x0
    REAL (KIND = DBL) ::  th1, th2, th3, th4, th5, th6  !theta angle between vector a and mi vector
    REAL (KIND = DBL) ::  e1, e2, e3, e4, e5, e6       !constant
    REAL (KIND = DBL) ::  f1, f2, f3, f4, f5, f6       !constant
    REAL (KIND = DBL) ::  g1, g2, g3, g4, g5, g6       !constant
    REAL (KIND = DBL) ::  ay1, ay2, ay3                !vector dot product a*y1, a*y2
    REAL (KIND = DBL) ::  by1, by2, by3                !product yi*(yi+a.yi)
!===============================================================================================================
!   Initialize
!---------------------------------------------------------------------------------------------------------------
    TExx = 0.0D0
    TExy = 0.0D0
    TExz = 0.0D0
    TEyx = 0.0D0
    TEyy = 0.0D0
    TEyz = 0.0D0
    TEzx = 0.0D0
    TEzy = 0.0D0
    TEzz = 0.0D0

```

```
    QExx = 0.0D0
    QExy = 0.0D0
    QExz = 0.0D0
    QEyx = 0.0D0
    QEyy = 0.0D0
    QEyz = 0.0D0
    QEzx = 0.0D0
    QEzy = 0.0D0
    QEzz = 0.0D0

    PEx = 0.0D0
    PEy = 0.0D0
    PEz = 0.0D0
    PE  = 0.0D0

    modx0= DSQRT((x0(k)-x10)**2+(y0(k)-y10)**2+(z0(k)-z10)**2)

    IF (modx0 >= 1.0D0 + eps) THEN
        cpi = 0.0D0
    ELSE IF (modx0 <= 1.0D0 + eps) THEN
        cpi = -8.0D0*pi
    ELSE
        cpi = -4.0D0*pi
    end if

    IF((ABS(x10-x0(k)) <= eps) .AND. &
    &  (ABS(y10-y0(k)) <= eps) .AND. &
    &  (ABS(z10-z0(k)) <= eps)) THEN
        a1   = x10
        a2   = y10
        a3   = z10
    ELSE
        a1   = x10-x0(k)
        a2   = y10-y0(k)
        a3   = z10-z0(k)
    END IF

    !a1   = x00-x0(k)
    !a2   = y00-y0(k)
    !a3   = z00-z0(k)
    modx0= DSQRT(a1**2+a2**2+a3**2)
    a1   = a1/modx0
    a2   = a2/modx0
    a3   = a3/modx0

    !OPEN (9,file="TE.out")
!-----------------------------------------------------------------------------------------------------------
!   vertices of the kth triangle
!-----------------------------------------------------------------------------------------------------------
    i1 = n(k,1)
    i2 = n(k,2)
    i3 = n(k,3)
    i4 = n(k,4)
    i5 = n(k,5)
    i6 = n(k,6)
!-----------------------------------------------------------------------------------------------------------
        cf  = 0.0D0
!-----------------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
! rule formula
!-----------------------------------------------------------------------------------------------------------
!   Firstly, to compute all vectors and angles involved.
!-----------------------------------------------------------------------------------------------------------
    yvx1 = p(i1,1)-x10
    yvy1 = p(i1,2)-y10
    yvz1 = p(i1,3)-z10
    yvx2 = p(i2,1)-x10
    yvy2 = p(i2,2)-y10
    yvz2 = p(i2,3)-z10
    yvx3 = p(i3,1)-x10
    yvy3 = p(i3,2)-y10
    yvz3 = p(i3,3)-z10
```



```
    yvx4 = p(i4,1)-x10
    yvy4 = p(i4,2)-y10
    yvz4 = p(i4,3)-z10
    yvx5 = p(i5,1)-x10
    yvy5 = p(i5,2)-y10
    yvz5 = p(i5,3)-z10
    yvx6 = p(i6,1)-x10
    yvy6 = p(i6,2)-y10
    yvz6 = p(i6,3)-z10
!-----------------------------------------------------------------------------------------------------------
    zvx1 = yvx1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
    zvy1 = yvy1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
    zvz1 = yvz1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
    zvx2 = yvx2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
    zvy2 = yvy2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
    zvz2 = yvz2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
    zvx3 = yvx3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
    zvy3 = yvy3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
    zvz3 = yvz3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
    zvx4 = yvx4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
    zvy4 = yvy4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
    zvz4 = yvz4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
    zvx5 = yvx5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
    zvy5 = yvy5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
    zvz5 = yvz5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
    zvx6 = yvx6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
    zvy6 = yvy6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
    zvz6 = yvz6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
!-----------------------------------------------------------------------------------------------------------
!   Cross products.
!-----------------------------------------------------------------------------------------------------------
    zcx1 = zvy1*zvz6 - zvz1*zvy6
    zcy1 = zvz1*zvx6 - zvx1*zvz6
    zcz1 = zvx1*zvy6 - zvy1*zvx6
    zcx2 = zvy6*zvz3 - zvz6*zvy3
    zcy2 = zvz6*zvx3 - zvx6*zvz3
    zcz2 = zvx6*zvy3 - zvy6*zvx3
    zcx3 = zvy3*zvz5 - zvz3*zvy5
    zcy3 = zvz3*zvx5 - zvx3*zvz5
    zcz3 = zvx3*zvy5 - zvy3*zvx5
    zcx4 = zvy5*zvz2 - zvz5*zvy2
    zcy4 = zvz5*zvx2 - zvx5*zvz2
    zcz4 = zvx5*zvy2 - zvy5*zvx2
    zcx5 = zvy2*zvz4 - zvz2*zvy4
    zcy5 = zvz2*zvx4 - zvx2*zvz4
    zcz5 = zvx2*zvy4 - zvy2*zvx4
    zcx6 = zvy4*zvz1 - zvz4*zvy1
    zcy6 = zvz4*zvx1 - zvx4*zvz1
    zcz6 = zvx4*zvy1 - zvy4*zvx1
!-----------------------------------------------------------------------------------------------------------
!   Dot products.
!-----------------------------------------------------------------------------------------------------------
    zd1 = zvx1*zvx6 +zvy1*zvy6 +zvx1*zvy6
    zd2 = zvx6*zvx3 +zvy6*zvy3 +zvx6*zvy3
    zd3 = zvx3*zvx5 +zvy3*zvy5 +zvx3*zvy5
    zd4 = zvx5*zvx2 +zvy5*zvy2 +zvx5*zvy2
    zd5 = zvx2*zvx4 +zvy2*zvy4 +zvx2*zvy4
    zd6 = zvx4*zvx1 +zvy4*zvy1 +zvx4*zvy1
!-----------------------------------------------------------------------------------------------------------
!   Angles
!-----------------------------------------------------------------------------------------------------------
    al1 = DACOS( zvx1*zvx6 + zvy1*zvy6 + zvz1*zvz6 )
    al2 = DACOS( zvx6*zvx3 + zvy6*zvy3 + zvz6*zvz3 )
    al3 = DACOS( zvx3*zvx5 + zvy3*zvy5 + zvz3*zvz5 )
    al4 = DACOS( zvx5*zvx2 + zvy5*zvy2 + zvz5*zvz2 )
    al5 = DACOS( zvx2*zvx4 + zvy2*zvy4 + zvz2*zvz4 )
    al6 = DACOS( zvx4*zvx1 + zvy4*zvy1 + zvz4*zvz1 )
!-----------------------------------------------------------------------------------------------------------
    th1 = DACOS( a1*zvx1 + a2*zvy1 + a3*zvz1 )
    th2 = DACOS( a1*zvx6 + a2*zvy6 + a3*zvz6 )
    th3 = DACOS( a1*zvx3 + a2*zvy3 + a3*zvz3 )
    th4 = DACOS( a1*zvx5 + a2*zvy5 + a3*zvz5 )
```

```fortran
    th5 = DACOS( a1*zvx2 + a2*zvy2 + a3*zvz2 )
    th6 = DACOS( a1*zvx4 + a2*zvy4 + a3*zvz4 )
!-----------------------------------------------------------------------------------------------
!   e and g constants
!-----------------------------------------------------------------------------------------------
    e1 = ( DCOS( th2 ) - DCOS( th1 )*DCOS( al1 ) )/DSIN( al1 )
    e2 = ( DCOS( th3 ) - DCOS( th2 )*DCOS( al2 ) )/DSIN( al2 )
    e3 = ( DCOS( th4 ) - DCOS( th3 )*DCOS( al3 ) )/DSIN( al3 )
    e4 = ( DCOS( th5 ) - DCOS( th4 )*DCOS( al4 ) )/DSIN( al4 )
    e5 = ( DCOS( th6 ) - DCOS( th5 )*DCOS( al5 ) )/DSIN( al5 )
    e6 = ( DCOS( th1 ) - DCOS( th6 )*DCOS( al6 ) )/DSIN( al6 )
    f1 = DCOS( th1 )
    f2 = DCOS( th2 )
    f3 = DCOS( th3 )
    f4 = DCOS( th4 )
    f5 = DCOS( th5 )
    f6 = DCOS( th6 )
    g1 = DSQRT( 1.0D0 - e1**2 - f1**2 )
    g2 = DSQRT( 1.0D0 - e2**2 - f2**2 )
    g3 = DSQRT( 1.0D0 - e3**2 - f3**2 )
    g4 = DSQRT( 1.0D0 - e4**2 - f4**2 )
    g5 = DSQRT( 1.0D0 - e5**2 - f5**2 )
    g6 = DSQRT( 1.0D0 - e6**2 - f6**2 )
!-----------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1
!-----------------------------------------------------------------------------------------------
        QExx  = ( (zvx1 + zvx6)*zcx1 ) / ( 1.0D0 + zd1 )
        QExy  = ( (zvx1 + zvx6)*zcy1 ) / ( 1.0D0 + zd1 )
        QExz  = ( (zvx1 + zvx6)*zcz1 ) / ( 1.0D0 + zd1 )
        QEyx  = ( (zvy1 + zvy6)*zcx1 ) / ( 1.0D0 + zd1 )
        QEyy  = ( (zvy1 + zvy6)*zcy1 ) / ( 1.0D0 + zd1 )
        QEyz  = ( (zvy1 + zvy6)*zcz1 ) / ( 1.0D0 + zd1 )
        QEzx  = ( (zvz1 + zvz6)*zcx1 ) / ( 1.0D0 + zd1 )
        QEzy  = ( (zvz1 + zvz6)*zcy1 ) / ( 1.0D0 + zd1 )
        QEzz  = ( (zvz1 + zvz6)*zcz1 ) / ( 1.0D0 + zd1 )
!-----------------------------------------------------------------------------------------------
        PEx  = ( 2.0D0*a1*zcx1/g1 )*( (DATAN(( (1.0D0 - f1)*DTAN(al1/2) + e1 )/ g1)) - DATAN(e1/g1))
        PEy  = ( 2.0D0*a2*zcy1/g1 )*( (DATAN(( (1.0D0 - f1)*DTAN(al1/2) + e1 )/ g1)) - DATAN(e1/g1))
        PEz  = ( 2.0D0*a3*zcz1/g1 )*( (DATAN(( (1.0D0 - f1)*DTAN(al1/2) + e1 )/ g1)) - DATAN(e1/g1))
!-----------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 6
!-----------------------------------------------------------------------------------------------
        QExx  = QExx + ( (zvx6 + zvx3)*zcx2 ) / ( 1.0D0 + zd2 )
        QExy  = QExy + ( (zvx6 + zvx3)*zcy2 ) / ( 1.0D0 + zd2 )
        QExz  = QExz + ( (zvx6 + zvx3)*zcz2 ) / ( 1.0D0 + zd2 )
        QEyx  = QEyx + ( (zvy6 + zvy3)*zcx2 ) / ( 1.0D0 + zd2 )
        QEyy  = QEyy + ( (zvy6 + zvy3)*zcy2 ) / ( 1.0D0 + zd2 )
        QEyz  = QEyz + ( (zvy6 + zvy3)*zcz2 ) / ( 1.0D0 + zd2 )
        QEzx  = QEzx + ( (zvz6 + zvz3)*zcx2 ) / ( 1.0D0 + zd2 )
        QEzy  = QEzy + ( (zvz6 + zvz3)*zcy2 ) / ( 1.0D0 + zd2 )
        QEzz  = QEzz + ( (zvz6 + zvz3)*zcz2 ) / ( 1.0D0 + zd2 )
!-----------------------------------------------------------------------------------------------
        PEx  = PEx + ( 2.0D0*a1*zcx2/g2 )*( (DATAN(( (1.0D0 - f2)*DTAN(al2/2) + e2 )/ g2)) - DATAN(e2/g2))
        PEy  = PEy + ( 2.0D0*a2*zcy2/g2 )*( (DATAN(( (1.0D0 - f2)*DTAN(al2/2) + e2 )/ g2)) - DATAN(e2/g2))
        PEz  = PEz + ( 2.0D0*a3*zcz2/g2 )*( (DATAN(( (1.0D0 - f2)*DTAN(al2/2) + e2 )/ g2)) - DATAN(e2/g2))
!-----------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3
!-----------------------------------------------------------------------------------------------
        QExx  = QExx + ( (zvx3 + zvx5)*zcx3 ) / ( 1.0D0 + zd3 )
        QExy  = QExy + ( (zvx3 + zvx5)*zcy3 ) / ( 1.0D0 + zd3 )
        QExz  = QExz + ( (zvx3 + zvx5)*zcz3 ) / ( 1.0D0 + zd3 )
        QEyx  = QEyx + ( (zvy3 + zvy5)*zcx3 ) / ( 1.0D0 + zd3 )
        QEyy  = QEyy + ( (zvy3 + zvy5)*zcy3 ) / ( 1.0D0 + zd3 )
        QEyz  = QEyz + ( (zvy3 + zvy5)*zcz3 ) / ( 1.0D0 + zd3 )
        QEzx  = QEzx + ( (zvz3 + zvz5)*zcx3 ) / ( 1.0D0 + zd3 )
        QEzy  = QEzy + ( (zvz3 + zvz5)*zcy3 ) / ( 1.0D0 + zd3 )
        QEzz  = QEzz + ( (zvz3 + zvz5)*zcz3 ) / ( 1.0D0 + zd3 )
!-----------------------------------------------------------------------------------------------
        PEx  = PEx + ( 2.0D0*a1*zcx3/g3 )*( (DATAN(( (1.0D0 - f3)*DTAN(al3/3) + e3 )/ g3)) - DATAN(e3/g3))
        PEy  = PEy + ( 2.0D0*a2*zcy3/g3 )*( (DATAN(( (1.0D0 - f3)*DTAN(al3/3) + e3 )/ g3)) - DATAN(e3/g3))
        PEz  = PEz + ( 2.0D0*a3*zcz3/g3 )*( (DATAN(( (1.0D0 - f3)*DTAN(al3/3) + e3 )/ g3)) - DATAN(e3/g3))
!-----------------------------------------------------------------------------------------------
```



```fortran
! computation of curvature line integral along segment 5
!-----------------------------------------------------------------------------------------------
        QExx  = QExx + ( (zvx5 + zvx2)*zcx4 ) / ( 1.0D0 + zd4 )
        QExy  = QExy + ( (zvx5 + zvx2)*zcy4 ) / ( 1.0D0 + zd4 )
        QExz  = QExz + ( (zvx5 + zvx2)*zcz4 ) / ( 1.0D0 + zd4 )
        QEyx  = QEyx + ( (zvy5 + zvy2)*zcx4 ) / ( 1.0D0 + zd4 )
        QEyy  = QEyy + ( (zvy5 + zvy2)*zcy4 ) / ( 1.0D0 + zd4 )
        QEyz  = QEyz + ( (zvy5 + zvy2)*zcz4 ) / ( 1.0D0 + zd4 )
        QEzx  = QEzx + ( (zvz5 + zvz2)*zcx4 ) / ( 1.0D0 + zd4 )
        QEzy  = QEzy + ( (zvz5 + zvz2)*zcy4 ) / ( 1.0D0 + zd4 )
        QEzz  = QEzz + ( (zvz5 + zvz2)*zcz4 ) / ( 1.0D0 + zd4 )
!-----------------------------------------------------------------------------------------------
        PEx  = PEx + ( 2.0D0*a1*zcx4/g4 )*( (DATAN(( (1.0D0 - f4)*DTAN(al4/2) + e4 )/ g4)) - DATAN(e4/g4))
        PEy  = PEy + ( 2.0D0*a2*zcy4/g4 )*( (DATAN(( (1.0D0 - f4)*DTAN(al4/2) + e4 )/ g4)) - DATAN(e4/g4))
        PEz  = PEz + ( 2.0D0*a3*zcz4/g4 )*( (DATAN(( (1.0D0 - f4)*DTAN(al4/2) + e4 )/ g4)) - DATAN(e4/g4))
!-----------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2
!-----------------------------------------------------------------------------------------------
        QExx  = QExx + ( (zvx2 + zvx4)*zcx5 ) / ( 1.0D0 + zd5 )
        QExy  = QExy + ( (zvx2 + zvx4)*zcy5 ) / ( 1.0D0 + zd5 )
        QExz  = QExz + ( (zvx2 + zvx4)*zcz5 ) / ( 1.0D0 + zd5 )
        QEyx  = QEyx + ( (zvy2 + zvy4)*zcx5 ) / ( 1.0D0 + zd5 )
        QEyy  = QEyy + ( (zvy2 + zvy4)*zcy5 ) / ( 1.0D0 + zd5 )
        QEyz  = QEyz + ( (zvy2 + zvy4)*zcz5 ) / ( 1.0D0 + zd5 )
        QEzx  = QEzx + ( (zvz2 + zvz4)*zcx5 ) / ( 1.0D0 + zd5 )
        QEzy  = QEzy + ( (zvz2 + zvz4)*zcy5 ) / ( 1.0D0 + zd5 )
        QEzz  = QEzz + ( (zvz2 + zvz4)*zcz5 ) / ( 1.0D0 + zd5 )
!-----------------------------------------------------------------------------------------------
        PEx  = PEx + ( 2.0D0*a1*zcx5/g5 )*( (DATAN(( (1.0D0 - f5)*DTAN(al5/2) + e5 )/ g5)) - DATAN(e5/g5))
        PEy  = PEy + ( 2.0D0*a2*zcy5/g5 )*( (DATAN(( (1.0D0 - f5)*DTAN(al5/2) + e5 )/ g5)) - DATAN(e5/g5))
        PEz  = PEz + ( 2.0D0*a3*zcz5/g5 )*( (DATAN(( (1.0D0 - f5)*DTAN(al5/2) + e5 )/ g5)) - DATAN(e5/g5))
!-----------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 4
!-----------------------------------------------------------------------------------------------
        QExx  = QExx + ( (zvx4 + zvx1)*zcx6 ) / ( 1.0D0 + zd6 )
        QExy  = QExy + ( (zvx4 + zvx1)*zcy6 ) / ( 1.0D0 + zd6 )
        QExz  = QExz + ( (zvx4 + zvx1)*zcz6 ) / ( 1.0D0 + zd6 )
        QEyx  = QEyx + ( (zvy4 + zvy1)*zcx6 ) / ( 1.0D0 + zd6 )
        QEyy  = QEyy + ( (zvy4 + zvy1)*zcy6 ) / ( 1.0D0 + zd6 )
        QEyz  = QEyz + ( (zvy4 + zvy1)*zcz6 ) / ( 1.0D0 + zd6 )
        QEzx  = QEzx + ( (zvz4 + zvz1)*zcx6 ) / ( 1.0D0 + zd6 )
        QEzy  = QEzy + ( (zvz4 + zvz1)*zcy6 ) / ( 1.0D0 + zd6 )
        QEzz  = QEzz + ( (zvz4 + zvz1)*zcz6 ) / ( 1.0D0 + zd6 )
!-----------------------------------------------------------------------------------------------
        PEx  = PEx + ( 2.0D0*a1*zcx6/g6 )*( (DATAN(( (1.0D0 - f6)*DTAN(al6/2) + e6 )/ g6)) - DATAN(e6/g6))
        PEy  = PEy + ( 2.0D0*a2*zcy6/g6 )*( (DATAN(( (1.0D0 - f6)*DTAN(al6/2) + e6 )/ g6)) - DATAN(e6/g6))
        PEz  = PEz + ( 2.0D0*a3*zcz6/g6 )*( (DATAN(( (1.0D0 - f6)*DTAN(al6/2) + e6 )/ g6)) - DATAN(e6/g6))
!-----------------------------------------------------------------------------------------------
    TExx = 2.0D0*( QExx + cpi + PE )
    TExy = 2.0D0*( QExy )
    TExz = 2.0D0*( QExz )
    TEyx = 2.0D0*( QEyx )
    TEyy = 2.0D0*( QEyy + cpi + PE )
    TEyz = 2.0D0*( QEyz )
    TEzx = 2.0D0*( QEzx )
    TEzy = 2.0D0*( QEzy )
    TEzz = 2.0D0*( QEzz + cpi + PE )
!-----------------------------------------------------------------------------------------------
    END SUBROUTINE intr_lin_sing_s7
!===============================================================================================
    SUBROUTINE intr_lin_sing_s8(x0e,   y0e,   z0e,    &
                            &  x00,   y00,   z00,  &
                            &  k,                  &
                            &  TExx, TExy, TExz,   &
                            &  TEyx, TEyy, TEyz,   &
                            &  TEzx, TEzy, TEzz    )
!===============================================================================================
!   This subroutine is a new version stokeslet Subroutine.
!Compute:
!   *The value of the Stokeslet over each singular element
!Now, (March/ 09 / 2015) this subroutine was made.
!===============================================================================================
```

```fortran
    USE Mod_Nodal_Interp
    USE Mod_Prtcl_3D_Geo, ONLY: abc
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog,&
    &                         alphaQ,  betaQ,  gammaQ, &
    &                         arel,         crvmel,    &
    &                         vnx0,     vny0,    vnz0, &
    &                         ZZ,           WW,        &
    &                         xiq, etq, wq, pi, eps, x0, y0, z0
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN) :: x00,   y00,   z00             !singulatity coordinates
    REAL (KIND = DBL), INTENT(IN) :: x0e,   y0e,   z0e             !singulatity coordinates element
    INTEGER,           INTENT(IN)  :: k                          !number of element
    REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TExz          !value of stresslet in the singular element
    REAL (KIND = DBL), INTENT(OUT) :: TEyx, TEyy, TEyz          !value of stresslet in the singular element
    REAL (KIND = DBL), INTENT(OUT) :: TEzx, TEzy, TEzz          !value of stresslet in the singular element
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                      !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                    !indices to obtain node numbers from each element
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: cf, fil1, fil2, fil3           !integration weigth coefficients
    REAL (KIND = DBL) :: hss                            !weigth
    REAL (KIND = DBL) :: cpi                            !Selection to add the solid angle
    REAL (KIND = DBL) :: modx0                          !weigth
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &           !vectoreial product
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  QExx, QExy, QExz, &           !Iq tensor
    &                     QEyx, QEyy, QEyz, &
    &                     QEzx, QEzy, QEzz
    REAL (KIND = DBL) ::  PEx, PEy, PEz, PE             !Ip tensor
    REAL (KIND = DBL) ::  yvx1, yvy1, yvz1                 !vector (y1-x0)
    REAL (KIND = DBL) ::  yvx2, yvy2, yvz2                 !vector (y2-x0)
    REAL (KIND = DBL) ::  yvx3, yvy3, yvz3                 !vector (y2-x0)
    REAL (KIND = DBL) ::  px2, py2, pz2                 !vector (y2-x0)
    REAL (KIND = DBL) ::  a1, a2, a3                    !vector a
    REAL (KIND = DBL) ::  ay1, ay2, ay3                 !vector dot product a*y1, a*y2
    REAL (KIND = DBL) ::  by1, by2, by3                 !product yi*(yi+a.yi)
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: xi, eta                        !variables of weigth to integrate over a triangle
    REAL (KIND = DBL) :: x, y, z                        !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi            !coordinates of the tangential vector over the xi axis
    REAL (KIND = DBL) :: DxDet, DyDet, DzDet            !coordinates of the tangential vector over the eta axis
    REAL (KIND = DBL) :: vnx, vny, vnz                  !normal vector coordinates of the element
    REAL (KIND = DBL) :: hs, xs, es                     !surface metric on a triangle
    REAL (KIND = DBL) :: al, be, ga, alc, bec, gac      !integration weigth coefficients
    REAL (KIND = DBL) :: fil                        !integration weigth coefficients
    REAL (KIND = DBL), DIMENSION(6) ::xxi, eet          !variables of weigth over in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDx, DyDx, DzDx  !tangential vector over xi axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDe, DyDe, DzDe  !tangential vector over eta axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz       !normal vector in triangle (xi,eta)
!====================================================================================================
!   Initialize
!----------------------------------------------------------------------------------------------------
    TExx = 0.0D0
    TExy = 0.0D0
    TExz = 0.0D0
    TEyx = 0.0D0
    TEyy = 0.0D0
    TEyz = 0.0D0
    TEzx = 0.0D0
    TEzy = 0.0D0
    TEzz = 0.0D0
!----------------------------------------------------------------------------------------------------
    QExx = 0.0D0
    QExy = 0.0D0
    QExz = 0.0D0
```



```fortran
    QEyx = 0.0D0
    QEyy = 0.0D0
    QEyz = 0.0D0
    QEzx = 0.0D0
    QEzy = 0.0D0
    QEzz = 0.0D0
!----------------------------------------------------------------------------------------------------
    PEx = 0.0D0
    PEy = 0.0D0
    PEz = 0.0D0
    PE  = 0.0D0
!----------------------------------------------------------------------------------------------------
    a1   = x0e-x00
    a2   = y0e-y00
    a3   = z0e-z00
    modx0= DSQRT(a1**2+a2**2+a3**2)
!----------------------------------------------------------------------------------------------------
    IF (modx0 >= 1.0D0 + eps) THEN
        cpi = 0.0D0
    ELSE IF (modx0 <= 1.0D0 - eps) THEN
        cpi = -8.0D0*pi
    ELSE
        cpi = -4.0D0*pi
    end if
!----------------------------------------------------------------------------------------------------
    a1   = x00-x0e
    a2   = y00-y0e
    a3   = z00-z0e
    modx0= DSQRT(a1**2+a2**2+a3**2)
!----------------------------------------------------------------------------------------------------
    a1   = a1/modx0
    a2   = a2/modx0
    a3   = a3/modx0
!----------------------------------------------------------------------------------------------------
    !OPEN (9,file="TE.out")
!----------------------------------------------------------------------------------------------------
!   vertices of the kth triangle
!----------------------------------------------------------------------------------------------------
    i1 = n(k,1)
    i2 = n(k,2)
    i3 = n(k,3)
    i4 = n(k,4)
    i5 = n(k,5)
    i6 = n(k,6)
    CALL abc(p(i1,1)-x00,   p(i1,2)-y00,  p(i1,3)-z00,    &
    &        p(i2,1)-x00,   p(i2,2)-y00,  p(i2,3)-z00,    &
    &        p(i3,1)-x00,   p(i3,2)-y00,  p(i3,3)-z00,    &
    &        p(i4,1)-x00,   p(i4,2)-y00,  p(i4,3)-z00,    &
    &        p(i5,1)-x00,   p(i5,2)-y00,  p(i5,3)-z00,    &
    &        p(i6,1)-x00,   p(i6,2)-y00,  p(i6,3)-z00,    &
    &        al, be, ga)
    alc = 1.0D0-al
    bec = 1.0D0-be
    gac = 1.0D0-ga
!----------------------------------------------------------------------------------------------------
        cf  = 0.0D0
!----------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
! rule formula
!----------------------------------------------------------------------------------------------------
        DO i = 1, 13
            xi  = xiq(i)
            eta = etq(i)
            CALL interp_p(p(i1,1)-x00,   p(i1,2)-y00,  p(i1,3)-z00,    &
            &             p(i2,1)-x00,   p(i2,2)-y00,  p(i2,3)-z00,    &
            &             p(i3,1)-x00,   p(i3,2)-y00,  p(i3,3)-z00,    &
            &             p(i4,1)-x00,   p(i4,2)-y00,  p(i4,3)-z00,    &
            &             p(i5,1)-x00,   p(i5,2)-y00,  p(i5,3)-z00,    &
            &             p(i6,1)-x00,   p(i6,2)-y00,  p(i6,3)-z00,    &
            &             al,        be,       ga,         &
            &             xi,        eta,                  &
            &             x,         y,        z,          &
```

```
           &            DxDxi,     DyDxi,    DzDxi,      &
           &            DxDet,     DyDet,    DzDet,      &
           &            vnx,       vny,      vnz,        &
           &            hs)
           cf = hs*wq(i)
       END DO
!       cf=0.5D0*cf
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-4
!formula (4.2.10) of Pozrikidis (1997)
!----------------------------------------------------------------------------------------------------
       xxi(1) = 0.0D0
       eet(1) = 0.0D0
       xxi(2) = 1.0D0
       eet(2) = 0.0D0
       xxi(3) = 0.0D0
       eet(3) = 1.0D0
       xxi(4) = al
       eet(4) = 0.0D0
       xxi(5) = ga
       eet(5) = gac
       xxi(6) = 0.0D0
       eet(6) = be
       DO i = 1, 6
           xi  = xxi(i)
           eta = eet(i)
           CALL interp_p(p(i1,1)-x00,   p(i1,2)-y00,  p(i1,3)-z00,    &
           &            p(i2,1)-x00,   p(i2,2)-y00,  p(i2,3)-z00,    &
           &            p(i3,1)-x00,   p(i3,2)-y00,  p(i3,3)-z00,    &
           &            p(i4,1)-x00,   p(i4,2)-y00,  p(i4,3)-z00,    &
           &            p(i5,1)-x00,   p(i5,2)-y00,  p(i5,3)-z00,    &
           &            p(i6,1)-x00,   p(i6,2)-y00,  p(i6,3)-z00,    &
           &            al,        be,       ga,         &
           &            xi,        eta,                  &
           &            x,         y,        z,          &
           &            DxDx(i),   DyDx(i),  DzDx(i),    &
           &            DxDe(i),   DyDe(i),  DzDe(i),    &
           &            vx(i),     vy(i),    vz(i),      &
           &            hs)
       END DO
!----------------------------------------------------------------------------------------------------
       bvx1 = 0.0D0
       bvy1 = 0.0D0
       bvz1 = 0.0D0
       bvx2 = 0.0D0
       bvy2 = 0.0D0
       bvz2 = 0.0D0
       bvx3 = 0.0D0
       bvy3 = 0.0D0
       bvz3 = 0.0D0
       yvx1 = 0.0D0
       yvy1 = 0.0D0
       yvz1 = 0.0D0
       yvx2 = 0.0D0
       yvy2 = 0.0D0
       yvz2 = 0.0D0
       yvx3 = 0.0D0
       yvy3 = 0.0D0
       yvz3 = 0.0D0
       fil1 = 0.0D0
       fil2 = 0.0D0
       fil3 = 0.0D0
       ay1  = 0.0D0
       ay2  = 0.0D0
       ay3  = 0.0D0
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-4-2
!----------------------------------------------------------------------------------------------------
       yvx1 = p(i1,1) - x00
       yvy1 = p(i1,2) - y00
       yvz1 = p(i1,3) - z00
       fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
```



```
!----------------------------------------------------------------------------------------------------
       yvx2 = p(i4,1) - x00
       yvy2 = p(i4,2) - y00
       yvz2 = p(i4,3) - z00
       fil2 = DSQRT(yvx2**2 + yvy2**2 +yvz2**2)
!----------------------------------------------------------------------------------------------------
       yvx3 = p(i2,1) - x00
       yvy3 = p(i2,2) - y00
       yvz3 = p(i2,3) - z00
       fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!----------------------------------------------------------------------------------------------------
       ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
       ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
       ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!----------------------------------------------------------------------------------------------------
       by1  = ( fil1*(fil1+ay1) )
       by2  = ( fil2*(fil2+ay2) )
       by3  = ( fil3*(fil3+ay3) )
!----------------------------------------------------------------------------------------------------
       bvx1 = yvy1*DzDx(1)-yvz1*DyDx(1)
       bvy1 = yvz1*DxDx(1)-yvx1*DzDx(1)
       bvz1 = yvx1*DyDx(1)-yvy1*DxDx(1)
!----------------------------------------------------------------------------------------------------
       bvx2 = yvy2*DzDx(4)-yvz2*DyDx(4)
       bvy2 = yvz2*DxDx(4)-yvx2*DzDx(4)
       bvz2 = yvx2*DyDx(4)-yvy2*DxDx(4)
!----------------------------------------------------------------------------------------------------
       bvx3 = yvy3*DzDx(2)-yvz3*DyDx(2)
       bvy3 = yvz3*DxDx(2)-yvx3*DzDx(2)
       bvz3 = yvx3*DyDx(2)-yvy3*DxDx(2)
!----------------------------------------------------------------------------------------------------
       QExx = ((al*yvx1*bvx1)/(fil1**3)) + ((yvx2*bvx2)/(fil2**3)) + ((alc*yvx3*bvx3)/(fil3**3))
       QExy = ((al*yvx1*bvy1)/(fil1**3)) + ((yvx2*bvy2)/(fil2**3)) + ((alc*yvx3*bvy3)/(fil3**3))
       QExz = ((al*yvx1*bvz1)/(fil1**3)) + ((yvx2*bvz2)/(fil2**3)) + ((alc*yvx3*bvz3)/(fil3**3))
       QEyx = ((al*yvy1*bvx1)/(fil1**3)) + ((yvy2*bvx2)/(fil2**3)) + ((alc*yvy3*bvx3)/(fil3**3))
       QEyy = ((al*yvy1*bvy1)/(fil1**3)) + ((yvy2*bvy2)/(fil2**3)) + ((alc*yvy3*bvy3)/(fil3**3))
       QEyz = ((al*yvy1*bvz1)/(fil1**3)) + ((yvy2*bvz2)/(fil2**3)) + ((alc*yvy3*bvz3)/(fil3**3))
       QEzx = ((al*yvz1*bvx1)/(fil1**3)) + ((yvz2*bvx2)/(fil2**3)) + ((alc*yvz3*bvx3)/(fil3**3))
       QEzy = ((al*yvz1*bvy1)/(fil1**3)) + ((yvz2*bvy2)/(fil2**3)) + ((alc*yvz3*bvy3)/(fil3**3))
       QEzz = ((al*yvz1*bvz1)/(fil1**3)) + ((yvz2*bvz2)/(fil2**3)) + ((alc*yvz3*bvz3)/(fil3**3))
!----------------------------------------------------------------------------------------------------
       PEx = ((al*a1*bvx1)/(by1)) + ((a1*bvx2)/(by2)) + ((alc*a1*bvx3)/(by3))
       PEy = ((al*a2*bvy1)/(by1)) + ((a2*bvy2)/(by2)) + ((alc*a2*bvy3)/(by3))
       PEz = ((al*a3*bvz1)/(by1)) + ((a3*bvz2)/(by2)) + ((alc*a3*bvz3)/(by3))
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-5-3
!----------------------------------------------------------------------------------------------------
       yvx1 = p(i2,1) - x00
       yvy1 = p(i2,2) - y00
       yvz1 = p(i2,3) - z00
       fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!----------------------------------------------------------------------------------------------------
       yvx2 = p(i5,1) - x00
       yvy2 = p(i5,2) - y00
       yvz2 = p(i5,3) - z00
       fil2 = DSQRT(yvx2**2 + yvy2**2 +yvz2**2)
!----------------------------------------------------------------------------------------------------
       yvx3 = p(i3,1) - x00
       yvy3 = p(i3,2) - y00
       yvz3 = p(i3,3) - z00
       fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!----------------------------------------------------------------------------------------------------
       ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
       ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
       ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!----------------------------------------------------------------------------------------------------
       by1  = ( fil1*(fil1+ay1) )
       by2  = ( fil2*(fil2+ay2) )
       by3  = ( fil3*(fil3+ay3) )
!----------------------------------------------------------------------------------------------------
       bvx1 = yvy1*DzDx(2)-yvz1*DyDx(2)
       bvy1 = yvz1*DxDx(2)-yvx1*DzDx(2)
```

```
        bvz1 = yvx1*DyDx(2)-yvy1*DxDx(2)
!----------------------------------------------------------------------------------------------------
        bvx2 = yvy2*DzDx(5)-yvz2*DyDx(5)
        bvy2 = yvz2*DxDx(5)-yvx2*DzDx(5)
        bvz2 = yvx2*DyDx(5)-yvy2*DxDx(5)
!----------------------------------------------------------------------------------------------------
        bvx3 = yvy3*DzDx(3)-yvz3*DyDx(3)
        bvy3 = yvz3*DxDx(3)-yvx3*DzDx(3)
        bvz3 = yvx3*DyDx(3)-yvy3*DxDx(3)
!----------------------------------------------------------------------------------------------------
        QExx = QExx - ((gac*yvx1*bvx1)/(fil1**3)) - ((yvx2*bvx2)/(fil2**3)) - ((ga*yvx3*bvx3)/(fil3**3))
        QExy = QExy - ((gac*yvx1*bvy1)/(fil1**3)) - ((yvx2*bvy2)/(fil2**3)) - ((ga*yvx3*bvy3)/(fil3**3))
        QExz = QExz - ((gac*yvx1*bvz1)/(fil1**3)) - ((yvx2*bvz2)/(fil2**3)) - ((ga*yvx3*bvz3)/(fil3**3))
        QEyx = QEyx - ((gac*yvy1*bvx1)/(fil1**3)) - ((yvy2*bvx2)/(fil2**3)) - ((ga*yvy3*bvx3)/(fil3**3))
        QEyy = QEyy - ((gac*yvy1*bvy1)/(fil1**3)) - ((yvy2*bvy2)/(fil2**3)) - ((ga*yvy3*bvy3)/(fil3**3))
        QEyz = QEyz - ((gac*yvy1*bvz1)/(fil1**3)) - ((yvy2*bvz2)/(fil2**3)) - ((ga*yvy3*bvz3)/(fil3**3))
        QEzx = QEzx - ((gac*yvz1*bvx1)/(fil1**3)) - ((yvz2*bvx2)/(fil2**3)) - ((ga*yvz3*bvx3)/(fil3**3))
        QEzy = QEzy - ((gac*yvz1*bvy1)/(fil1**3)) - ((yvz2*bvy2)/(fil2**3)) - ((ga*yvz3*bvy3)/(fil3**3))
        QEzz = QEzz - ((gac*yvz1*bvz1)/(fil1**3)) - ((yvz2*bvz2)/(fil2**3)) - ((ga*yvz3*bvz3)/(fil3**3))
!----------------------------------------------------------------------------------------------------
        !QExx = QExx + ((gac*yvx1*bvx1)/(fil1**3)) + ((yvx2*bvx2)/(fil2**3)) + ((ga*yvx3*bvx3)/(fil3**3))
        !QExy = QExy + ((gac*yvx1*bvy1)/(fil1**3)) + ((yvx2*bvy2)/(fil2**3)) + ((ga*yvx3*bvy3)/(fil3**3))
        !QExz = QExz + ((gac*yvx1*bvz1)/(fil1**3)) + ((yvx2*bvz2)/(fil2**3)) + ((ga*yvx3*bvz3)/(fil3**3))
        !QEyx = QEyx + ((gac*yvy1*bvx1)/(fil1**3)) + ((yvy2*bvx2)/(fil2**3)) + ((ga*yvy3*bvx3)/(fil3**3))
        !QEyy = QEyy + ((gac*yvy1*bvy1)/(fil1**3)) + ((yvy2*bvy2)/(fil2**3)) + ((ga*yvy3*bvy3)/(fil3**3))
        !QEyz = QEyz + ((gac*yvy1*bvz1)/(fil1**3)) + ((yvy2*bvz2)/(fil2**3)) + ((ga*yvy3*bvz3)/(fil3**3))
        !QEzx = QEzx + ((gac*yvz1*bvx1)/(fil1**3)) + ((yvz2*bvx2)/(fil2**3)) + ((ga*yvz3*bvx3)/(fil3**3))
        !QEzy = QEzy + ((gac*yvz1*bvy1)/(fil1**3)) + ((yvz2*bvy2)/(fil2**3)) + ((ga*yvz3*bvy3)/(fil3**3))
        !QEzz = QEzz + ((gac*yvz1*bvz1)/(fil1**3)) + ((yvz2*bvz2)/(fil2**3)) + ((ga*yvz3*bvz3)/(fil3**3))
!----------------------------------------------------------------------------------------------------
        PEx = PEx - ((gac*a1*bvx1)/(by1)) - ((a1*bvx2)/(by2)) - ((ga*a1*bvx3)/(by3))
        PEy = PEy - ((gac*a2*bvy1)/(by1)) - ((a2*bvy2)/(by2)) - ((ga*a2*bvy3)/(by3))
        PEz = PEz - ((gac*a3*bvz1)/(by1)) - ((a3*bvz2)/(by2)) - ((ga*a3*bvz3)/(by3))
!----------------------------------------------------------------------------------------------------
        !PEx = PEx + ((gac*a1*bvx1)/(by1)) + ((a1*bvx2)/(by2)) + ((ga*a1*bvx3)/(by3))
        !PEy = PEy + ((gac*a2*bvy1)/(by1)) + ((a2*bvy2)/(by2)) + ((ga*a2*bvy3)/(by3))
        !PEz = PEz + ((gac*a3*bvz1)/(by1)) + ((a3*bvz2)/(by2)) + ((ga*a3*bvz3)/(by3))
!----------------------------------------------------------------------------------------------------
        bvx1 = yvy1*DzDe(2)-yvz1*DyDe(2)
        bvy1 = yvz1*DxDe(2)-yvx1*DzDe(2)
        bvz1 = yvx1*DyDe(2)-yvy1*DxDe(2)
!----------------------------------------------------------------------------------------------------
        bvx2 = yvy2*DzDe(5)-yvz2*DyDe(5)
        bvy2 = yvz2*DxDe(5)-yvx2*DzDe(5)
        bvz2 = yvx2*DyDe(5)-yvy2*DxDe(5)
!----------------------------------------------------------------------------------------------------
        bvx3 = yvy3*DzDe(3)-yvz3*DyDe(3)
        bvy3 = yvz3*DxDe(3)-yvx3*DzDe(3)
        bvz3 = yvx3*DyDe(3)-yvy3*DxDe(3)
!----------------------------------------------------------------------------------------------------
        QExx = QExx + ((gac*yvx1*bvx1)/(fil1**3)) + ((yvx2*bvx2)/(fil2**3)) + ((ga*yvx3*bvx3)/(fil3**3))
        QExy = QExy + ((gac*yvx1*bvy1)/(fil1**3)) + ((yvx2*bvy2)/(fil2**3)) + ((ga*yvx3*bvy3)/(fil3**3))
        QExz = QExz + ((gac*yvx1*bvz1)/(fil1**3)) + ((yvx2*bvz2)/(fil2**3)) + ((ga*yvx3*bvz3)/(fil3**3))
        QEyx = QEyx + ((gac*yvy1*bvx1)/(fil1**3)) + ((yvy2*bvx2)/(fil2**3)) + ((ga*yvy3*bvx3)/(fil3**3))
        QEyy = QEyy + ((gac*yvy1*bvy1)/(fil1**3)) + ((yvy2*bvy2)/(fil2**3)) + ((ga*yvy3*bvy3)/(fil3**3))
        QEyz = QEyz + ((gac*yvy1*bvz1)/(fil1**3)) + ((yvy2*bvz2)/(fil2**3)) + ((ga*yvy3*bvz3)/(fil3**3))
        QEzx = QEzx + ((gac*yvz1*bvx1)/(fil1**3)) + ((yvz2*bvx2)/(fil2**3)) + ((ga*yvz3*bvx3)/(fil3**3))
        QEzy = QEzy + ((gac*yvz1*bvy1)/(fil1**3)) + ((yvz2*bvy2)/(fil2**3)) + ((ga*yvz3*bvy3)/(fil3**3))
        QEzz = QEzz + ((gac*yvz1*bvz1)/(fil1**3)) + ((yvz2*bvz2)/(fil2**3)) + ((ga*yvz3*bvz3)/(fil3**3))
!----------------------------------------------------------------------------------------------------
        !QExx = QExx - ((gac*yvx1*bvx1)/(fil1**3)) - ((yvx2*bvx2)/(fil2**3)) - ((ga*yvx3*bvx3)/(fil3**3))
        !QExy = QExy - ((gac*yvx1*bvy1)/(fil1**3)) - ((yvx2*bvy2)/(fil2**3)) - ((ga*yvx3*bvy3)/(fil3**3))
        !QExz = QExz - ((gac*yvx1*bvz1)/(fil1**3)) - ((yvx2*bvz2)/(fil2**3)) - ((ga*yvx3*bvz3)/(fil3**3))
        !QEyx = QEyx - ((gac*yvy1*bvx1)/(fil1**3)) - ((yvy2*bvx2)/(fil2**3)) - ((ga*yvy3*bvx3)/(fil3**3))
        !QEyy = QEyy - ((gac*yvy1*bvy1)/(fil1**3)) - ((yvy2*bvy2)/(fil2**3)) - ((ga*yvy3*bvy3)/(fil3**3))
        !QEyz = QEyz - ((gac*yvy1*bvz1)/(fil1**3)) - ((yvy2*bvz2)/(fil2**3)) - ((ga*yvy3*bvz3)/(fil3**3))
        !QEzx = QEzx - ((gac*yvz1*bvx1)/(fil1**3)) - ((yvz2*bvx2)/(fil2**3)) - ((ga*yvz3*bvx3)/(fil3**3))
        !QEzy = QEzy - ((gac*yvz1*bvy1)/(fil1**3)) - ((yvz2*bvy2)/(fil2**3)) - ((ga*yvz3*bvy3)/(fil3**3))
        !QEzz = QEzz - ((gac*yvz1*bvz1)/(fil1**3)) - ((yvz2*bvz2)/(fil2**3)) - ((ga*yvz3*bvz3)/(fil3**3))
!----------------------------------------------------------------------------------------------------
        PEx = PEx + ((gac*a1*bvx1)/(by1)) + ((a1*bvx2)/(by2)) + ((ga*a1*bvx3)/(by3))
        PEy = PEy + ((gac*a2*bvy1)/(by1)) + ((a2*bvy2)/(by2)) + ((ga*a2*bvy3)/(by3))
```



```
        PEz = PEz + ((gac*a3*bvz1)/(by1)) + ((a3*bvz2)/(by2)) + ((ga*a3*bvz3)/(by3))
!----------------------------------------------------------------------------------------------------
        !PEx = PEx - ((gac*a1*bvx1)/(by1)) - ((a1*bvx2)/(by2)) - ((ga*a1*bvx3)/(by3))
        !PEy = PEy - ((gac*a2*bvy1)/(by1)) - ((a2*bvy2)/(by2)) - ((ga*a2*bvy3)/(by3))
        !PEz = PEz - ((gac*a3*bvz1)/(by1)) - ((a3*bvz2)/(by2)) - ((ga*a3*bvz3)/(by3))
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3-6-1
!----------------------------------------------------------------------------------------------------
        yvx1 = p(i3,1) - x00
        yvy1 = p(i3,2) - y00
        yvz1 = p(i3,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!----------------------------------------------------------------------------------------------------
        yvx2 = p(i6,1) - x00
        yvy2 = p(i6,2) - y00
        yvz2 = p(i6,3) - z00
        fil2 = DSQRT(yvx2**2 + yvy2**2 +yvz2**2)
!----------------------------------------------------------------------------------------------------
        yvx3 = p(i1,1) - x00
        yvy3 = p(i1,2) - y00
        yvz3 = p(i1,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!----------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!----------------------------------------------------------------------------------------------------
        by1  = DSQRT(( fil1*(fil1+ay1) )**2)
        by2  = DSQRT(( fil2*(fil2+ay2) )**2)
        by3  = DSQRT(( fil3*(fil3+ay3) )**2)
!----------------------------------------------------------------------------------------------------
        bvx1 = yvy1*DzDe(3)-yvz1*DyDe(3)
        bvy1 = yvz1*DxDe(3)-yvx1*DzDe(3)
        bvz1 = yvx1*DyDe(3)-yvy1*DxDe(3)
!----------------------------------------------------------------------------------------------------
        bvx2 = yvy2*DzDe(6)-yvz2*DyDe(6)
        bvy2 = yvz2*DxDe(6)-yvx2*DzDe(6)
        bvz2 = yvx2*DyDe(6)-yvy2*DxDe(6)
!----------------------------------------------------------------------------------------------------
        bvx3 = yvy3*DzDe(1)-yvz3*DyDe(1)
        bvy3 = yvz3*DxDe(1)-yvx3*DzDe(1)
        bvz3 = yvx3*DyDe(1)-yvy3*DxDe(1)
!----------------------------------------------------------------------------------------------------
        QExx = QExx - ((be*yvx1*bvx1)/(fil1**3)) - ((yvx2*bvx2)/(fil2**3)) - ((bec*yvx3*bvx3)/(fil3**3))
        QExy = QExy - ((be*yvx1*bvy1)/(fil1**3)) - ((yvx2*bvy2)/(fil2**3)) - ((bec*yvx3*bvy3)/(fil3**3))
        QExz = QExz - ((be*yvx1*bvz1)/(fil1**3)) - ((yvx2*bvz2)/(fil2**3)) - ((bec*yvx3*bvz3)/(fil3**3))
        QEyx = QEyx - ((be*yvy1*bvx1)/(fil1**3)) - ((yvy2*bvx2)/(fil2**3)) - ((bec*yvy3*bvx3)/(fil3**3))
        QEyy = QEyy - ((be*yvy1*bvy1)/(fil1**3)) - ((yvy2*bvy2)/(fil2**3)) - ((bec*yvy3*bvy3)/(fil3**3))
        QEyz = QEyz - ((be*yvy1*bvz1)/(fil1**3)) - ((yvy2*bvz2)/(fil2**3)) - ((bec*yvy3*bvz3)/(fil3**3))
        QEzx = QEzx - ((be*yvz1*bvx1)/(fil1**3)) - ((yvz2*bvx2)/(fil2**3)) - ((bec*yvz3*bvx3)/(fil3**3))
        QEzy = QEzy - ((be*yvz1*bvy1)/(fil1**3)) - ((yvz2*bvy2)/(fil2**3)) - ((bec*yvz3*bvy3)/(fil3**3))
        QEzz = QEzz - ((be*yvz1*bvz1)/(fil1**3)) - ((yvz2*bvz2)/(fil2**3)) - ((bec*yvz3*bvz3)/(fil3**3))
!----------------------------------------------------------------------------------------------------
        !QExx = QExx + ((be*yvx1*bvx1)/(fil1**3)) + ((yvx2*bvx2)/(fil2**3)) + ((bec*yvx3*bvx3)/(fil3**3))
        !QExy = QExy + ((be*yvx1*bvy1)/(fil1**3)) + ((yvx2*bvy2)/(fil2**3)) + ((bec*yvx3*bvy3)/(fil3**3))
        !QExz = QExz + ((be*yvx1*bvz1)/(fil1**3)) + ((yvx2*bvz2)/(fil2**3)) + ((bec*yvx3*bvz3)/(fil3**3))
        !QEyx = QEyx + ((be*yvy1*bvx1)/(fil1**3)) + ((yvy2*bvx2)/(fil2**3)) + ((bec*yvy3*bvx3)/(fil3**3))
        !QEyy = QEyy + ((be*yvy1*bvy1)/(fil1**3)) + ((yvy2*bvy2)/(fil2**3)) + ((bec*yvy3*bvy3)/(fil3**3))
        !QEyz = QEyz + ((be*yvy1*bvz1)/(fil1**3)) + ((yvy2*bvz2)/(fil2**3)) + ((bec*yvy3*bvz3)/(fil3**3))
        !QEzx = QEzx + ((be*yvz1*bvx1)/(fil1**3)) + ((yvz2*bvx2)/(fil2**3)) + ((bec*yvz3*bvx3)/(fil3**3))
        !QEzy = QEzy + ((be*yvz1*bvy1)/(fil1**3)) + ((yvz2*bvy2)/(fil2**3)) + ((bec*yvz3*bvy3)/(fil3**3))
        !QEzz = QEzz + ((be*yvz1*bvz1)/(fil1**3)) + ((yvz2*bvz2)/(fil2**3)) + ((bec*yvz3*bvz3)/(fil3**3))
!----------------------------------------------------------------------------------------------------
        PEx = PEx - ((be*a1*bvx1)/(by1)) - ((a1*bvx2)/(by2)) - ((bec*a1*bvx3)/(by3))
        PEy = PEy - ((be*a2*bvy1)/(by1)) - ((a2*bvy2)/(by2)) - ((bec*a2*bvy3)/(by3))
        PEz = PEz - ((be*a3*bvz1)/(by1)) - ((a3*bvz2)/(by2)) - ((bec*a3*bvz3)/(by3))
!----------------------------------------------------------------------------------------------------
!        PEx = PEx + ((be*a1*bvx1)/(by1)) + ((a1*bvx2)/(by2)) + ((bec*a1*bvx3)/(by3))
!        PEy = PEy + ((be*a2*bvy1)/(by1)) + ((a2*bvy2)/(by2)) + ((bec*a2*bvy3)/(by3))
!        PEz = PEz + ((be*a3*bvz1)/(by1)) + ((a3*bvz2)/(by2)) + ((bec*a3*bvz3)/(by3))
!----------------------------------------------------------------------------------------------------
    TExx = 2.0D0*( QExx + cpi -PE )/(cf)
```

```fortran
    TExy = 2.0D0*( QExy )/(2.0D0*cf)
    TExz = 2.0D0*( QExz )/(2.0D0*cf)
    TEyx = 2.0D0*( QEyx )/(2.0D0*cf)
    TEyy = 2.0D0*( QEyy + cpi -PE )/(cf)
    TEyz = 2.0D0*( QEyz )/(2.0D0*cf)
    TEzx = 2.0D0*( QEzx )/(2.0D0*cf)
    TEzy = 2.0D0*( QEzy )/(2.0D0*cf)
    TEzz = 2.0D0*( QEzz + cpi -PE )/(cf)
!    WRITE(*,*) PEx, PEy, PEz
    !WRITE(*,*) TEyx, TEyy, TEyz
    !WRITE(*,*) TEzx, TEzy, TEzz
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE intr_lin_sing_s8
!====================================================================================================
!====================================================================================================
    SUBROUTINE intr_lin_sing_s9(x0e,   y0e,   z0e,    &
                            &  x00,   y00,   z00,  &
                            &  k,                  &
                            &  TExx, TExy, TExz,   &
                            &  TEyx, TEyy, TEyz,   &
                            &  TEzx, TEzy, TEzz    )
!====================================================================================================
!   This subroutine is a new version stokeslet Subroutine.
!Compute:
!   *The value of the Stokeslet over each singular element
!Now, (March/ 09 / 2015) this subroutine was made.
!====================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_Prtcl_3D_Geo, ONLY: abc
    USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog,&
    &                         alphaQ,  betaQ,  gammaQ, &
    &                         arel,         crvmel,    &
    &                         vnx0,     vny0,    vnz0, &
    &                         ZZ,           WW,        &
    &                         xiq, etq, wq, pi, eps, x0, y0, z0
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN) :: x00,   y00,   z00             !singulatity coordinates
    REAL (KIND = DBL), INTENT(IN) :: x0e,   y0e,   z0e             !singulatity coordinates element
    INTEGER,           INTENT(IN)  :: k                         !number of element
    REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TExz          !value of stresslet in the singular element
    REAL (KIND = DBL), INTENT(OUT) :: TEyx, TEyy, TEyz          !value of stresslet in the singular element
    REAL (KIND = DBL), INTENT(OUT) :: TEzx, TEzy, TEzz          !value of stresslet in the singular element
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                  !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                !indices to obtain node numbers from each element
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: cf, fil1, fil2, fil3           !integration weigth coefficients
    REAL (KIND = DBL) :: hss                            !weigth
    REAL (KIND = DBL) :: cpi                            !Selection to add the solid angle
    REAL (KIND = DBL) :: modx0                          !weigth
    REAL (KIND = DBL) ::  bvx1, bvy1, bvz1, &           !vectoreial product
    &                     bvx2, bvy2, bvz2, &
    &                     bvx3, bvy3, bvz3
    REAL (KIND = DBL) ::  QExx, QExy, QExz, &           !Iq tensor
    &                     QEyx, QEyy, QEyz, &
    &                     QEzx, QEzy, QEzz
    REAL (KIND = DBL) ::  PEx, PEy, PEz, PE             !Ip tensor
    REAL (KIND = DBL) ::  yvx1, yvy1, yvz1                 !vector (y1-x0)
    REAL (KIND = DBL) ::  yvx2, yvy2, yvz2                 !vector (y2-x0)
    REAL (KIND = DBL) ::  yvx3, yvy3, yvz3                 !vector (y2-x0)
    REAL (KIND = DBL) ::  px2, py2, pz2                 !vector (y2-x0)
    REAL (KIND = DBL) ::  a1, a2, a3                    !vector a
    REAL (KIND = DBL) ::  ay1, ay2, ay3                 !vector dot product a*y1, a*y2
    REAL (KIND = DBL) ::  by1, by2, by3                 !product yi*(yi+a.yi)
!----------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: xi, eta                        !variables of weigth to integrate over a triangle
```



```fortran
    REAL (KIND = DBL) :: x, y, z                        !coordinates of the f(x,y,z)= F(xi,eta)
    REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi            !coordinates of the tangential vector over the xi axis
    REAL (KIND = DBL) :: DxDet, DyDet, DzDet            !coordinates of the tangential vector over the eta axis
    REAL (KIND = DBL) :: vnx, vny, vnz                  !normal vector coordinates of the element
    REAL (KIND = DBL) :: hs, xs, es                     !surface metric on a triangle
    REAL (KIND = DBL) :: al, be, ga, alc, bec, gac      !integration weigth coefficients
    REAL (KIND = DBL) :: fil                        !integration weigth coefficients
    REAL (KIND = DBL), DIMENSION(6) ::xxi, eet          !variables of weigth over in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDx, DyDx, DzDx  !tangential vector over xi axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) ::DxDe, DyDe, DzDe  !tangential vector over eta axis in triangle (xi,eta)
    REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz       !normal vector in triangle (xi,eta)
!====================================================================================================
!   Initialize
!----------------------------------------------------------------------------------------------------
    TExx = 0.0D0
    TExy = 0.0D0
    TExz = 0.0D0
    TEyx = 0.0D0
    TEyy = 0.0D0
    TEyz = 0.0D0
    TEzx = 0.0D0
    TEzy = 0.0D0
    TEzz = 0.0D0
!----------------------------------------------------------------------------------------------------
    QExx = 0.0D0
    QExy = 0.0D0
    QExz = 0.0D0
    QEyx = 0.0D0
    QEyy = 0.0D0
    QEyz = 0.0D0
    QEzx = 0.0D0
    QEzy = 0.0D0
    QEzz = 0.0D0
!----------------------------------------------------------------------------------------------------
    PEx = 0.0D0
    PEy = 0.0D0
    PEz = 0.0D0
    PE  = 0.0D0
!----------------------------------------------------------------------------------------------------
    a1   = -x00
    a2   = -y00
    a3   = -z00
    modx0= DSQRT(a1**2+a2**2+a3**2)
!----------------------------------------------------------------------------------------------------
    cpi = 4.0D0*pi
!----------------------------------------------------------------------------------------------------
    a1   = a1/modx0
    a2   = a2/modx0
    a3   = a3/modx0
!----------------------------------------------------------------------------------------------------
    !OPEN (9,file="TE.out")
!----------------------------------------------------------------------------------------------------
!   vertices of the kth triangle
!----------------------------------------------------------------------------------------------------
    i1 = n(k,1)
    i2 = n(k,2)
    i3 = n(k,3)
    i4 = n(k,4)
    i5 = n(k,5)
    i6 = n(k,6)
    CALL abc(p(i1,1)-x00,   p(i1,2)-y00,  p(i1,3)-z00,    &
    &        p(i2,1)-x00,   p(i2,2)-y00,  p(i2,3)-z00,    &
    &        p(i3,1)-x00,   p(i3,2)-y00,  p(i3,3)-z00,    &
    &        p(i4,1)-x00,   p(i4,2)-y00,  p(i4,3)-z00,    &
    &        p(i5,1)-x00,   p(i5,2)-y00,  p(i5,3)-z00,    &
    &        p(i6,1)-x00,   p(i6,2)-y00,  p(i6,3)-z00,    &
    &        al, be, ga)
    alc = 1.0D0-al
    bec = 1.0D0-be
    gac = 1.0D0-ga
!----------------------------------------------------------------------------------------------------
        cf  = 0.0D0
```

```
!-----------------------------------------------------------------------------------------------------------
! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
! rule formula
!-----------------------------------------------------------------------------------------------------------
        DO i = 1, 13
            xi  = xiq(i)
            eta = etq(i)
            CALL interp_p(p(i1,1)-x00,   p(i1,2)-y00,  p(i1,3)-z00,    &
            &             p(i2,1)-x00,   p(i2,2)-y00,  p(i2,3)-z00,    &
            &             p(i3,1)-x00,   p(i3,2)-y00,  p(i3,3)-z00,    &
            &             p(i4,1)-x00,   p(i4,2)-y00,  p(i4,3)-z00,    &
            &             p(i5,1)-x00,   p(i5,2)-y00,  p(i5,3)-z00,    &
            &             p(i6,1)-x00,   p(i6,2)-y00,  p(i6,3)-z00,    &
            &             al,        be,       ga,         &
            &             xi,        eta,                  &
            &             x,         y,         z,         &
            &             DxDxi,     DyDxi,    DzDxi,      &
            &             DxDet,     DyDet,    DzDet,      &
            &             vnx,       vny,      vnz,        &
            &             hs)
            cf = hs*wq(i)
        END DO
!        cf=0.5D0*cf
!-----------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-4
!formula (4.2.10) of Pozrikidis (1997)
!-----------------------------------------------------------------------------------------------------------
        xxi(1) = 0.0D0
        eet(1) = 0.0D0
        xxi(2) = 1.0D0
        eet(2) = 0.0D0
        xxi(3) = 0.0D0
        eet(3) = 1.0D0
        xxi(4) = al
        eet(4) = 0.0D0
        xxi(5) = ga
        eet(5) = gac
        xxi(6) = 0.0D0
        eet(6) = be
        DO i = 1, 6
            xi  = xxi(i)
            eta = eet(i)
            CALL interp_p(p(i1,1)-x00,   p(i1,2)-y00,  p(i1,3)-z00,    &
            &             p(i2,1)-x00,   p(i2,2)-y00,  p(i2,3)-z00,    &
            &             p(i3,1)-x00,   p(i3,2)-y00,  p(i3,3)-z00,    &
            &             p(i4,1)-x00,   p(i4,2)-y00,  p(i4,3)-z00,    &
            &             p(i5,1)-x00,   p(i5,2)-y00,  p(i5,3)-z00,    &
            &             p(i6,1)-x00,   p(i6,2)-y00,  p(i6,3)-z00,    &
            &             al,        be,       ga,         &
            &             xi,        eta,                  &
            &             x,         y,        z,          &
            &             DxDx(i),   DyDx(i),  DzDx(i),    &
            &             DxDe(i),   DyDe(i),  DzDe(i),    &
            &             vx(i),     vy(i),    vz(i),      &
            &             hs)
        END DO
!-----------------------------------------------------------------------------------------------------------
        bvx1 = 0.0D0
        bvy1 = 0.0D0
        bvz1 = 0.0D0
        bvx2 = 0.0D0
        bvy2 = 0.0D0
        bvz2 = 0.0D0
        bvx3 = 0.0D0
        bvy3 = 0.0D0
        bvz3 = 0.0D0
        yvx1 = 0.0D0
        yvy1 = 0.0D0
        yvz1 = 0.0D0
        yvx2 = 0.0D0
        yvy2 = 0.0D0
        yvz2 = 0.0D0
```



```
        yvx3 = 0.0D0
        yvy3 = 0.0D0
        yvz3 = 0.0D0
        fil1 = 0.0D0
        fil2 = 0.0D0
        fil3 = 0.0D0
        ay1  = 0.0D0
        ay2  = 0.0D0
        ay3  = 0.0D0
!-----------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 1-4-2
!-----------------------------------------------------------------------------------------------------------
        yvx1 = p(i1,1) - x00
        yvy1 = p(i1,2) - y00
        yvz1 = p(i1,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!-----------------------------------------------------------------------------------------------------------
        yvx2 = p(i4,1) - x00
        yvy2 = p(i4,2) - y00
        yvz2 = p(i4,3) - z00
        fil2 = DSQRT(yvx2**2 + yvy2**2 +yvz2**2)
!-----------------------------------------------------------------------------------------------------------
        yvx3 = p(i2,1) - x00
        yvy3 = p(i2,2) - y00
        yvz3 = p(i2,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!-----------------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!-----------------------------------------------------------------------------------------------------------
        by1  = ( fil1*(fil1+ay1) )
        by2  = ( fil2*(fil2+ay2) )
        by3  = ( fil3*(fil3+ay3) )
!-----------------------------------------------------------------------------------------------------------
        bvx1 = yvy1*DzDx(1)-yvz1*DyDx(1)
        bvy1 = yvz1*DxDx(1)-yvx1*DzDx(1)
        bvz1 = yvx1*DyDx(1)-yvy1*DxDx(1)
!-----------------------------------------------------------------------------------------------------------
        bvx2 = yvy2*DzDx(4)-yvz2*DyDx(4)
        bvy2 = yvz2*DxDx(4)-yvx2*DzDx(4)
        bvz2 = yvx2*DyDx(4)-yvy2*DxDx(4)
!-----------------------------------------------------------------------------------------------------------
        bvx3 = yvy3*DzDx(2)-yvz3*DyDx(2)
        bvy3 = yvz3*DxDx(2)-yvx3*DzDx(2)
        bvz3 = yvx3*DyDx(2)-yvy3*DxDx(2)
!-----------------------------------------------------------------------------------------------------------
        QExx = ((al*yvx1*bvx1)/(fil1**3)) + ((yvx2*bvx2)/(fil2**3)) + ((alc*yvx3*bvx3)/(fil3**3))
        QExy = ((al*yvx1*bvy1)/(fil1**3)) + ((yvx2*bvy2)/(fil2**3)) + ((alc*yvx3*bvy3)/(fil3**3))
        QExz = ((al*yvx1*bvz1)/(fil1**3)) + ((yvx2*bvz2)/(fil2**3)) + ((alc*yvx3*bvz3)/(fil3**3))
        QEyx = ((al*yvy1*bvx1)/(fil1**3)) + ((yvy2*bvx2)/(fil2**3)) + ((alc*yvy3*bvx3)/(fil3**3))
        QEyy = ((al*yvy1*bvy1)/(fil1**3)) + ((yvy2*bvy2)/(fil2**3)) + ((alc*yvy3*bvy3)/(fil3**3))
        QEyz = ((al*yvy1*bvz1)/(fil1**3)) + ((yvy2*bvz2)/(fil2**3)) + ((alc*yvy3*bvz3)/(fil3**3))
        QEzx = ((al*yvz1*bvx1)/(fil1**3)) + ((yvz2*bvx2)/(fil2**3)) + ((alc*yvz3*bvx3)/(fil3**3))
        QEzy = ((al*yvz1*bvy1)/(fil1**3)) + ((yvz2*bvy2)/(fil2**3)) + ((alc*yvz3*bvy3)/(fil3**3))
        QEzz = ((al*yvz1*bvz1)/(fil1**3)) + ((yvz2*bvz2)/(fil2**3)) + ((alc*yvz3*bvz3)/(fil3**3))
!-----------------------------------------------------------------------------------------------------------
        PEx = ((al*a1*bvx1)/(by1)) + ((a1*bvx2)/(by2)) + ((alc*a1*bvx3)/(by3))
        PEy = ((al*a2*bvy1)/(by1)) + ((a2*bvy2)/(by2)) + ((alc*a2*bvy3)/(by3))
        PEz = ((al*a3*bvz1)/(by1)) + ((a3*bvz2)/(by2)) + ((alc*a3*bvz3)/(by3))
!-----------------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 2-5-3
!-----------------------------------------------------------------------------------------------------------
        yvx1 = p(i2,1) - x00
        yvy1 = p(i2,2) - y00
        yvz1 = p(i2,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!-----------------------------------------------------------------------------------------------------------
        yvx2 = p(i5,1) - x00
        yvy2 = p(i5,2) - y00
        yvz2 = p(i5,3) - z00
        fil2 = DSQRT(yvx2**2 + yvy2**2 +yvz2**2)
```

```fortran
!----------------------------------------------------------------------------------------------------
        yvx3 = p(i3,1) - x00
        yvy3 = p(i3,2) - y00
        yvz3 = p(i3,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!----------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!----------------------------------------------------------------------------------------------------
        by1  = ( fil1*(fil1+ay1) )
        by2  = ( fil2*(fil2+ay2) )
        by3  = ( fil3*(fil3+ay3) )
!----------------------------------------------------------------------------------------------------
        bvx1 = yvy1*DzDx(2)-yvz1*DyDx(2)
        bvy1 = yvz1*DxDx(2)-yvx1*DzDx(2)
        bvz1 = yvx1*DyDx(2)-yvy1*DxDx(2)
!----------------------------------------------------------------------------------------------------
        bvx2 = yvy2*DzDx(5)-yvz2*DyDx(5)
        bvy2 = yvz2*DxDx(5)-yvx2*DzDx(5)
        bvz2 = yvx2*DyDx(5)-yvy2*DxDx(5)
!----------------------------------------------------------------------------------------------------
        bvx3 = yvy3*DzDx(3)-yvz3*DyDx(3)
        bvy3 = yvz3*DxDx(3)-yvx3*DzDx(3)
        bvz3 = yvx3*DyDx(3)-yvy3*DxDx(3)
!----------------------------------------------------------------------------------------------------
        QExx = QExx - ((gac*yvx1*bvx1)/(fil1**3)) - ((yvx2*bvx2)/(fil2**3)) - ((ga*yvx3*bvx3)/(fil3**3))
        QExy = QExy - ((gac*yvx1*bvy1)/(fil1**3)) - ((yvx2*bvy2)/(fil2**3)) - ((ga*yvx3*bvy3)/(fil3**3))
        QExz = QExz - ((gac*yvx1*bvz1)/(fil1**3)) - ((yvx2*bvz2)/(fil2**3)) - ((ga*yvx3*bvz3)/(fil3**3))
        QEyx = QEyx - ((gac*yvy1*bvx1)/(fil1**3)) - ((yvy2*bvx2)/(fil2**3)) - ((ga*yvy3*bvx3)/(fil3**3))
        QEyy = QEyy - ((gac*yvy1*bvy1)/(fil1**3)) - ((yvy2*bvy2)/(fil2**3)) - ((ga*yvy3*bvy3)/(fil3**3))
        QEyz = QEyz - ((gac*yvy1*bvz1)/(fil1**3)) - ((yvy2*bvz2)/(fil2**3)) - ((ga*yvy3*bvz3)/(fil3**3))
        QEzx = QEzx - ((gac*yvz1*bvx1)/(fil1**3)) - ((yvz2*bvx2)/(fil2**3)) - ((ga*yvz3*bvx3)/(fil3**3))
        QEzy = QEzy - ((gac*yvz1*bvy1)/(fil1**3)) - ((yvz2*bvy2)/(fil2**3)) - ((ga*yvz3*bvy3)/(fil3**3))
        QEzz = QEzz - ((gac*yvz1*bvz1)/(fil1**3)) - ((yvz2*bvz2)/(fil2**3)) - ((ga*yvz3*bvz3)/(fil3**3))
!----------------------------------------------------------------------------------------------------
!        QExx = QExx + ((gac*yvx1*bvx1)/(fil1**3)) + ((yvx2*bvx2)/(fil2**3)) + ((ga*yvx3*bvx3)/(fil3**3))
!        QExy = QExy + ((gac*yvx1*bvy1)/(fil1**3)) + ((yvx2*bvy2)/(fil2**3)) + ((ga*yvx3*bvy3)/(fil3**3))
!        QExz = QExz + ((gac*yvx1*bvz1)/(fil1**3)) + ((yvx2*bvz2)/(fil2**3)) + ((ga*yvx3*bvz3)/(fil3**3))
!        QEyx = QEyx + ((gac*yvy1*bvx1)/(fil1**3)) + ((yvy2*bvx2)/(fil2**3)) + ((ga*yvy3*bvx3)/(fil3**3))
!        QEyy = QEyy + ((gac*yvy1*bvy1)/(fil1**3)) + ((yvy2*bvy2)/(fil2**3)) + ((ga*yvy3*bvy3)/(fil3**3))
!        QEyz = QEyz + ((gac*yvy1*bvz1)/(fil1**3)) + ((yvy2*bvz2)/(fil2**3)) + ((ga*yvy3*bvz3)/(fil3**3))
!        QEzx = QEzx + ((gac*yvz1*bvx1)/(fil1**3)) + ((yvz2*bvx2)/(fil2**3)) + ((ga*yvz3*bvx3)/(fil3**3))
!        QEzy = QEzy + ((gac*yvz1*bvy1)/(fil1**3)) + ((yvz2*bvy2)/(fil2**3)) + ((ga*yvz3*bvy3)/(fil3**3))
!        QEzz = QEzz + ((gac*yvz1*bvz1)/(fil1**3)) + ((yvz2*bvz2)/(fil2**3)) + ((ga*yvz3*bvz3)/(fil3**3))
!----------------------------------------------------------------------------------------------------
        PEx = PEx - ((gac*a1*bvx1)/(by1)) - ((a1*bvx2)/(by2)) - ((ga*a1*bvx3)/(by3))
        PEy = PEy - ((gac*a2*bvy1)/(by1)) - ((a2*bvy2)/(by2)) - ((ga*a2*bvy3)/(by3))
        PEz = PEz - ((gac*a3*bvz1)/(by1)) - ((a3*bvz2)/(by2)) - ((ga*a3*bvz3)/(by3))
!----------------------------------------------------------------------------------------------------
!        PEx = PEx + ((gac*a1*bvx1)/(by1)) + ((a1*bvx2)/(by2)) + ((ga*a1*bvx3)/(by3))
!        PEy = PEy + ((gac*a2*bvy1)/(by1)) + ((a2*bvy2)/(by2)) + ((ga*a2*bvy3)/(by3))
!        PEz = PEz + ((gac*a3*bvz1)/(by1)) + ((a3*bvz2)/(by2)) + ((ga*a3*bvz3)/(by3))
!----------------------------------------------------------------------------------------------------
        bvx1 = yvy1*DzDe(2)-yvz1*DyDe(2)
        bvy1 = yvz1*DxDe(2)-yvx1*DzDe(2)
        bvz1 = yvx1*DyDe(2)-yvy1*DxDe(2)
!----------------------------------------------------------------------------------------------------
        bvx2 = yvy2*DzDe(5)-yvz2*DyDe(5)
        bvy2 = yvz2*DxDe(5)-yvx2*DzDe(5)
        bvz2 = yvx2*DyDe(5)-yvy2*DxDe(5)
!----------------------------------------------------------------------------------------------------
        bvx3 = yvy3*DzDe(3)-yvz3*DyDe(3)
        bvy3 = yvz3*DxDe(3)-yvx3*DzDe(3)
        bvz3 = yvx3*DyDe(3)-yvy3*DxDe(3)
!----------------------------------------------------------------------------------------------------
        QExx = QExx + ((gac*yvx1*bvx1)/(fil1**3)) + ((yvx2*bvx2)/(fil2**3)) + ((ga*yvx3*bvx3)/(fil3**3))
        QExy = QExy + ((gac*yvx1*bvy1)/(fil1**3)) + ((yvx2*bvy2)/(fil2**3)) + ((ga*yvx3*bvy3)/(fil3**3))
        QExz = QExz + ((gac*yvx1*bvz1)/(fil1**3)) + ((yvx2*bvz2)/(fil2**3)) + ((ga*yvx3*bvz3)/(fil3**3))
        QEyx = QEyx + ((gac*yvy1*bvx1)/(fil1**3)) + ((yvy2*bvx2)/(fil2**3)) + ((ga*yvy3*bvx3)/(fil3**3))
        QEyy = QEyy + ((gac*yvy1*bvy1)/(fil1**3)) + ((yvy2*bvy2)/(fil2**3)) + ((ga*yvy3*bvy3)/(fil3**3))
        QEyz = QEyz + ((gac*yvy1*bvz1)/(fil1**3)) + ((yvy2*bvz2)/(fil2**3)) + ((ga*yvy3*bvz3)/(fil3**3))
```



```fortran
        QEzx = QEzx + ((gac*yvz1*bvx1)/(fil1**3)) + ((yvz2*bvx2)/(fil2**3)) + ((ga*yvz3*bvx3)/(fil3**3))
        QEzy = QEzy + ((gac*yvz1*bvy1)/(fil1**3)) + ((yvz2*bvy2)/(fil2**3)) + ((ga*yvz3*bvy3)/(fil3**3))
        QEzz = QEzz + ((gac*yvz1*bvz1)/(fil1**3)) + ((yvz2*bvz2)/(fil2**3)) + ((ga*yvz3*bvz3)/(fil3**3))
!----------------------------------------------------------------------------------------------------
!        QExx = QExx - ((gac*yvx1*bvx1)/(fil1**3)) - ((yvx2*bvx2)/(fil2**3)) - ((ga*yvx3*bvx3)/(fil3**3))
!        QExy = QExy - ((gac*yvx1*bvy1)/(fil1**3)) - ((yvx2*bvy2)/(fil2**3)) - ((ga*yvx3*bvy3)/(fil3**3))
!        QExz = QExz - ((gac*yvx1*bvz1)/(fil1**3)) - ((yvx2*bvz2)/(fil2**3)) - ((ga*yvx3*bvz3)/(fil3**3))
!        QEyx = QEyx - ((gac*yvy1*bvx1)/(fil1**3)) - ((yvy2*bvx2)/(fil2**3)) - ((ga*yvy3*bvx3)/(fil3**3))
!        QEyy = QEyy - ((gac*yvy1*bvy1)/(fil1**3)) - ((yvy2*bvy2)/(fil2**3)) - ((ga*yvy3*bvy3)/(fil3**3))
!        QEyz = QEyz - ((gac*yvy1*bvz1)/(fil1**3)) - ((yvy2*bvz2)/(fil2**3)) - ((ga*yvy3*bvz3)/(fil3**3))
!        QEzx = QEzx - ((gac*yvz1*bvx1)/(fil1**3)) - ((yvz2*bvx2)/(fil2**3)) - ((ga*yvz3*bvx3)/(fil3**3))
!        QEzy = QEzy - ((gac*yvz1*bvy1)/(fil1**3)) - ((yvz2*bvy2)/(fil2**3)) - ((ga*yvz3*bvy3)/(fil3**3))
!        QEzz = QEzz - ((gac*yvz1*bvz1)/(fil1**3)) - ((yvz2*bvz2)/(fil2**3)) - ((ga*yvz3*bvz3)/(fil3**3))
!----------------------------------------------------------------------------------------------------
        PEx = PEx + ((gac*a1*bvx1)/(by1)) + ((a1*bvx2)/(by2)) + ((ga*a1*bvx3)/(by3))
        PEy = PEy + ((gac*a2*bvy1)/(by1)) + ((a2*bvy2)/(by2)) + ((ga*a2*bvy3)/(by3))
        PEz = PEz + ((gac*a3*bvz1)/(by1)) + ((a3*bvz2)/(by2)) + ((ga*a3*bvz3)/(by3))
!----------------------------------------------------------------------------------------------------
        !PEx = PEx - ((gac*a1*bvx1)/(by1)) - ((a1*bvx2)/(by2)) - ((ga*a1*bvx3)/(by3))
        !PEy = PEy - ((gac*a2*bvy1)/(by1)) - ((a2*bvy2)/(by2)) - ((ga*a2*bvy3)/(by3))
        !PEz = PEz - ((gac*a3*bvz1)/(by1)) - ((a3*bvz2)/(by2)) - ((ga*a3*bvz3)/(by3))
!----------------------------------------------------------------------------------------------------
! computation of curvature line integral along segment 3-6-1
!----------------------------------------------------------------------------------------------------
        yvx1 = p(i3,1) - x00
        yvy1 = p(i3,2) - y00
        yvz1 = p(i3,3) - z00
        fil1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
!----------------------------------------------------------------------------------------------------
        yvx2 = p(i6,1) - x00
        yvy2 = p(i6,2) - y00
        yvz2 = p(i6,3) - z00
        fil2 = DSQRT(yvx2**2 + yvy2**2 +yvz2**2)
!----------------------------------------------------------------------------------------------------
        yvx3 = p(i1,1) - x00
        yvy3 = p(i1,2) - y00
        yvz3 = p(i1,3) - z00
        fil3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
!----------------------------------------------------------------------------------------------------
        ay1  = (a1*yvx1 + a2*yvy1 + a3*yvz1)
        ay2  = (a1*yvx2 + a2*yvy2 + a3*yvz2)
        ay3  = (a1*yvx3 + a2*yvy3 + a3*yvz3)
!----------------------------------------------------------------------------------------------------
        by1  = DSQRT(( fil1*(fil1+ay1) )**2)
        by2  = DSQRT(( fil2*(fil2+ay2) )**2)
        by3  = DSQRT(( fil3*(fil3+ay3) )**2)
!----------------------------------------------------------------------------------------------------
        bvx1 = yvy1*DzDe(3)-yvz1*DyDe(3)
        bvy1 = yvz1*DxDe(3)-yvx1*DzDe(3)
        bvz1 = yvx1*DyDe(3)-yvy1*DxDe(3)
!----------------------------------------------------------------------------------------------------
        bvx2 = yvy2*DzDe(6)-yvz2*DyDe(6)
        bvy2 = yvz2*DxDe(6)-yvx2*DzDe(6)
        bvz2 = yvx2*DyDe(6)-yvy2*DxDe(6)
!----------------------------------------------------------------------------------------------------
        bvx3 = yvy3*DzDe(1)-yvz3*DyDe(1)
        bvy3 = yvz3*DxDe(1)-yvx3*DzDe(1)
        bvz3 = yvx3*DyDe(1)-yvy3*DxDe(1)
!----------------------------------------------------------------------------------------------------
        QExx = QExx - ((be*yvx1*bvx1)/(fil1**3)) - ((yvx2*bvx2)/(fil2**3)) - ((bec*yvx3*bvx3)/(fil3**3))
        QExy = QExy - ((be*yvx1*bvy1)/(fil1**3)) - ((yvx2*bvy2)/(fil2**3)) - ((bec*yvx3*bvy3)/(fil3**3))
        QExz = QExz - ((be*yvx1*bvz1)/(fil1**3)) - ((yvx2*bvz2)/(fil2**3)) - ((bec*yvx3*bvz3)/(fil3**3))
        QEyx = QEyx - ((be*yvy1*bvx1)/(fil1**3)) - ((yvy2*bvx2)/(fil2**3)) - ((bec*yvy3*bvx3)/(fil3**3))
        QEyy = QEyy - ((be*yvy1*bvy1)/(fil1**3)) - ((yvy2*bvy2)/(fil2**3)) - ((bec*yvy3*bvy3)/(fil3**3))
        QEyz = QEyz - ((be*yvy1*bvz1)/(fil1**3)) - ((yvy2*bvz2)/(fil2**3)) - ((bec*yvy3*bvz3)/(fil3**3))
        QEzx = QEzx - ((be*yvz1*bvx1)/(fil1**3)) - ((yvz2*bvx2)/(fil2**3)) - ((bec*yvz3*bvx3)/(fil3**3))
        QEzy = QEzy - ((be*yvz1*bvy1)/(fil1**3)) - ((yvz2*bvy2)/(fil2**3)) - ((bec*yvz3*bvy3)/(fil3**3))
        QEzz = QEzz - ((be*yvz1*bvz1)/(fil1**3)) - ((yvz2*bvz2)/(fil2**3)) - ((bec*yvz3*bvz3)/(fil3**3))
!----------------------------------------------------------------------------------------------------
        !QExx = QExx + ((be*yvx1*bvx1)/(fil1**3)) + ((yvx2*bvx2)/(fil2**3)) + ((bec*yvx3*bvx3)/(fil3**3))
        !QExy = QExy + ((be*yvx1*bvy1)/(fil1**3)) + ((yvx2*bvy2)/(fil2**3)) + ((bec*yvx3*bvy3)/(fil3**3))
        !QExz = QExz + ((be*yvx1*bvz1)/(fil1**3)) + ((yvx2*bvz2)/(fil2**3)) + ((bec*yvx3*bvz3)/(fil3**3))
```

```
        !QEyx = QEyx + ((be*yvy1*bvx1)/(fil1**3)) + ((yvy2*bvx2)/(fil2**3)) + ((bec*yvy3*bvx3)/(fil3**3))
        !QEyy = QEyy + ((be*yvy1*bvy1)/(fil1**3)) + ((yvy2*bvy2)/(fil2**3)) + ((bec*yvy3*bvy3)/(fil3**3))
        !QEyz = QEyz + ((be*yvy1*bvz1)/(fil1**3)) + ((yvy2*bvz2)/(fil2**3)) + ((bec*yvy3*bvz3)/(fil3**3))
        !QEzx = QEzx + ((be*yvz1*bvx1)/(fil1**3)) + ((yvz2*bvx2)/(fil2**3)) + ((bec*yvz3*bvx3)/(fil3**3))
        !QEzy = QEzy + ((be*yvz1*bvy1)/(fil1**3)) + ((yvz2*bvy2)/(fil2**3)) + ((bec*yvz3*bvy3)/(fil3**3))
        !QEzz = QEzz + ((be*yvz1*bvz1)/(fil1**3)) + ((yvz2*bvz2)/(fil2**3)) + ((bec*yvz3*bvz3)/(fil3**3))
!-----------------------------------------------------------------------------------------------------------
        PEx = PEx - ((be*a1*bvx1)/(by1)) - ((a1*bvx2)/(by2)) - ((bec*a1*bvx3)/(by3))
        PEy = PEy - ((be*a2*bvy1)/(by1)) - ((a2*bvy2)/(by2)) - ((bec*a2*bvy3)/(by3))
        PEz = PEz - ((be*a3*bvz1)/(by1)) - ((a3*bvz2)/(by2)) - ((bec*a3*bvz3)/(by3))
!-----------------------------------------------------------------------------------------------------------
        !PEx = PEx + ((be*a1*bvx1)/(by1)) + ((a1*bvx2)/(by2)) + ((bec*a1*bvx3)/(by3))
        !PEy = PEy + ((be*a2*bvy1)/(by1)) + ((a2*bvy2)/(by2)) + ((bec*a2*bvy3)/(by3))
        !PEz = PEz + ((be*a3*bvz1)/(by1)) + ((a3*bvz2)/(by2)) + ((bec*a3*bvz3)/(by3))
!-----------------------------------------------------------------------------------------------------------
    TExx = 2.0D0*( QExx + cpi -PE )/(cf)
    TExy = 2.0D0*( QExy )/(2.0D0*cf)
    TExz = 2.0D0*( QExz )/(2.0D0*cf)
    TEyx = 2.0D0*( QEyx )/(2.0D0*cf)
    TEyy = 2.0D0*( QEyy + cpi -PE )/(cf)
    TEyz = 2.0D0*( QEyz )/(2.0D0*cf)
    TEzx = 2.0D0*( QEzx )/(2.0D0*cf)
    TEzy = 2.0D0*( QEzy )/(2.0D0*cf)
    TEzz = 2.0D0*( QEzz + cpi -PE )/(cf)
!    WRITE(*,*) PEx, PEy, PEz
    !WRITE(*,*) TEyx, TEyy, TEyz
    !WRITE(*,*) TEzx, TEzy, TEzz
!-----------------------------------------------------------------------------------------------------------
    END SUBROUTINE intr_lin_sing_s9
!===========================================================================================================
    END MODULE Mod_Prtcl3D_DLP
```

```fortran
MODULE Mod_sgf_3d_sfs
!==============================================================================================================
!   Version: 0.8 created on 22 / 03 / 2012
!   Version: 0.9 created on 03 / 09 / 2012
!   Version: 1.0 created on 13 / 11 / 2012
!                                                                                        Alfredo Sanjuan Sanjuan
!==============================================================================================================
    CONTAINS
!==============================================================================================================
    SUBROUTINE sgf_3d_sfs( x,    y,    z,     &
                        &  x0,  y0,  z0,    &
                        &  Txxx,Txxy,Txxz,  &
                        &  Txyx,Txyy,Txyz,  &
                        &  Txzx,Txzy,Txzz,  &
                        &  Tyxx,Tyxy,Tyxz,  &
                        &  Tyyx,Tyyy,Tyyz,  &
                        &  Tyzx,Tyzy,Tyzz,  &
                        &  Tzxx,Tzxy,Tzxz,  &
                        &  Tzyx,Tzyy,Tzyz,  &
                        &  Tzzx,Tzzy,Tzzz   )
!==============================================================================================================
!   Free-space Green's function: Stresslet Pozrikidis (1992, p. 23)
!   This new version of sfg_3d_sfs only calculates the Green's function: Stresslet.
!==============================================================================================================
    USE Mod_SharedVars, ONLY: DBL, ULog
    IMPLICIT NONE
!==============================================================================================================
!   Variables
!--------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), INTENT(IN)  :: x,   y,   z             !coordinates of collocation point of the element
    REAL (KIND = DBL), INTENT(IN)  :: x0, y0, z0            !coordinates of the singularity
    REAL (KIND = DBL), INTENT(OUT) :: Txxx,Txxy,Txxz, &     !Free-space Green´s function of Stokeslet.
    &                                  Txyx,Txyy,Txyz, &     !Integrate over ijk component over the element
    &                                  Txzx,Txzy,Txzz, &
    &                                  Tyxx,Tyxy,Tyxz, &
    &                                  Tyyx,Tyyy,Tyyz, &
    &                                  Tyzx,Tyzy,Tyzz, &
    &                                  Tzxx,Tzxy,Tzxz, &
    &                                  Tzyx,Tzyy,Tzyz, &
    &                                  Tzzx,Tzzy,Tzzz
!==============================================================================================================
!   Variables inside the subroutine
!--------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL) :: dx, dy, dz                 !differences between the (x,y,z) and (x0,y0,z0) coordinates
    REAL (KIND = DBL) :: dxx, dxy, dxz, &           !square distance di*dj=dij
    &                    dyy, dyz, dzz
    REAL (KIND = DBL) :: r                          !distance r between the (x,y,z) point and (x0,y0,z0)
    REAL (KIND = DBL) :: ri5                         !stresslet coefficient
!==============================================================================================================
!   Initialize
    dx = 0.0D0
    dy = 0.0D0
    dz = 0.0D0
!--------------------------------------------------------------------------------------------------------------
    dxx = 0.0D0
    dxy = 0.0D0
    dxz = 0.0D0
    dyy = 0.0D0
    dyz = 0.0D0
    dzz = 0.0D0
!--------------------------------------------------------------------------------------------------------------
    Txxx = 0.0D0
    Txxy = 0.0D0
    Txxz = 0.0D0
    Txyx = 0.0D0
    Txyy = 0.0D0
    Txyz = 0.0D0
    Txzx = 0.0D0
    Txzy = 0.0D0
    Txzz = 0.0D0
!--------------------------------------------------------------------------------------------------------------
    Tyxx = 0.0D0
```



```fortran
    Tyxy = 0.0D0
    Tyxz = 0.0D0
    Tyyx = 0.0D0
    Tyyy = 0.0D0
    Tyyz = 0.0D0
    Tyzx = 0.0D0
    Tyzy = 0.0D0
    Tyzz = 0.0D0
!--------------------------------------------------------------------------------------------------------------
    Tzxx = 0.0D0
    Tzxy = 0.0D0
    Tzxz = 0.0D0
    Tzyx = 0.0D0
    Tzyy = 0.0D0
    Tzyz = 0.0D0
    Tzzx = 0.0D0
    Tzzy = 0.0D0
    Tzzz = 0.0D0
!--------------------------------------------------------------------------------------------------------------
    dx = x-x0
    dy = y-y0
    dz = z-z0
!--------------------------------------------------------------------------------------------------------------
    dxx = dx*dx
    dxy = dx*dy
    dxz = dx*dz
    dyy = dy*dy
    dyz = dy*dz
    dzz = dz*dz
!--------------------------------------------------------------------------------------------------------------
    r = DSQRT(dxx + dyy + dzz)
!--------------------------------------------------------------------------------------------------------------
!   compute the stress tensor
!--------------------------------------------------------------------------------------------------------------
    ri5 = -6.0D0/r**5
!--------------------------------------------------------------------------------------------------------------
    Txxx = dxx*dx * ri5
    Txxy = dxx*dy * ri5
    Txxz = dxx*dz * ri5
    Txyx = Txxy
    Txyy = dxy*dy * ri5
    Txyz = dxy*dz * ri5
    Txzx = Txxz
    Txzy = Txyz
    Txzz = dxz*dz * ri5
!--------------------------------------------------------------------------------------------------------------
    Tyxx = Txxy
    Tyxy = Txyy
    Tyxz = Txyz
    Tyyx = Txyy
    Tyyy = dyy*dy * ri5
    Tyyz = dyy*dz * ri5
    Tyzx = Txyz
    Tyzy = Tyyz
    Tyzz = dyz*dz * ri5
!--------------------------------------------------------------------------------------------------------------
    Tzxx = Txxz
    Tzxy = Txyz
    Tzxz = Txzz
    Tzyx = Txyz
    Tzyy = Tyyz
    Tzyz = Tyzz
    Tzzx = Txzz
    Tzzy = Tyzz
    Tzzz = dzz*dz * ri5
!--------------------------------------------------------------------------------------------------------------
END SUBROUTINE sgf_3d_sfs
!==============================================================================================================
END MODULE Mod_sgf_3d_sfs
```

```
MODULE Mod_Velocity_Menu
!==========================================================================================================
!   Version: 0.9 created on 01 / 03 / 2013
!                                                                                   Alfredo Sanjuan Sanjuan
!==========================================================================================================
    CONTAINS
!==========================================================================================================
    SUBROUTINE Infinity_Flow(NVF, alpha, gi, Uoo, stride, steps, nelm, h)
!==========================================================================================================
!   This Subroutine obtains the value of velocity field in the interface.
! SYMBOLS:
!==========================================================================================================
!==========================================================================================================
    USE Mod_SharedVars, ONLY: DBL, x0, y0, z0, Ulog
    USE Mod_VelFieldTRM
!==========================================================================================================
!==========================================================================================================
    IMPLICIT NONE
!==========================================================================================================
!==========================================================================================================
!   Variables
    INTEGER, INTENT(INOUT)   :: NVF                                     !selection of velocity field
    INTEGER, INTENT(IN) :: nelm                                         !number of elements
    INTEGER, INTENT(IN) :: stride, steps                                       !number of elements
    REAL(KIND = DBL), INTENT(IN) :: alpha, h                            !flow parameter
    REAL(KIND = DBL), INTENT(IN) :: gi                                  !intensity of flow
    REAL (KIND = DBL) ::tobi                                             !dummy integrate arc constants
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: Uoo  !arrays of infinity velocity field
!==========================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------------
    INTEGER  :: i                                                       !Counters
    tobi = gi*DSQRT(1.0D0)
!==========================================================================================================
!   The next step is use a Case statement to obtain the coeficientes.
    SELECT CASE(NVF)
!----------------------------------------------------------------------------------------------------------
    CASE (1)
        FORALL (i=1:nelm)
        Uoo(i)= 0.0D0
        Uoo(i+nelm)= 0.0D0
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!----------------------------------------------------------------------------------------------------------
    CASE (2)
        FORALL (i=1:nelm)
        Uoo(i)= 0.0D0
        Uoo(i+nelm)= 1.0D0
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!----------------------------------------------------------------------------------------------------------
    CASE (3)
        FORALL (i=1:nelm)
        Uoo(i)= tobi/DSQRT(2.0D0)
        Uoo(i+nelm)= tobi/DSQRT(2.0D0)
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!----------------------------------------------------------------------------------------------------------
    CASE (4)
        FORALL (i=1:nelm)
        Uoo(i)= 2.0D0*tobi/DSQRT(5.0D0)
        Uoo(i+nelm)= tobi/DSQRT(5.0D0)
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!----------------------------------------------------------------------------------------------------------
    CASE (5) !It is not normalized
        FORALL (i=1:nelm)
        Uoo(i)= tobi*z0(i)
        Uoo(i+nelm)= -tobi*z0(i)
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!----------------------------------------------------------------------------------------------------------
```



```
    CASE (6)
        FORALL (i=1:nelm)
        Uoo(i)= -tobi*y0(i)
        Uoo(i+nelm)= tobi*x0(i)
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!----------------------------------------------------------------------------------------------------------
    CASE (7)
        !FORALL (i=1:nelm)
        !Uoo(i)= ((1.0D0- DEXP((-100.0D0*DBLE(stride))/DBLE(steps)))*gi*x0(i)*(alpha - 2.0D0))/(2.0D0*DSQRT(4.0D0
    - 2.0D0*alpha + alpha**2))
        !Uoo(i+nelm)= ((1.0D0- DEXP((-100.0D0*DBLE(stride))/DBLE(steps)))*gi*y0(i))/(DSQRT(4.0D0 - 2.0D0*alpha +
    alpha**2))
        !Uoo(i+nelm+nelm)= ((1.0D0- DEXP((-100.0D0*DBLE(stride))/DBLE(steps)))*-alpha*gi*z0(i))/(2.0D0*DSQRT(4.0D0
     - 2.0D0*alpha + alpha**2))
        !END FORALL
        FORALL (i=1:nelm)
        Uoo(i)= (tobi*x0(i)*(alpha - 2.0D0))/(2.0D0*DSQRT(4.0D0 - 2.0D0*alpha + alpha**2))
        Uoo(i+nelm)= (tobi*y0(i))/(DSQRT(4.0D0 - 2.0D0*alpha + alpha**2))
        Uoo(i+nelm+nelm)= (-alpha*tobi*z0(i))/(2.0D0*DSQRT(4.0D0 - 2.0D0*alpha + alpha**2))
        END FORALL
!----------------------------------------------------------------------------------------------------------
    CASE (8)
        FORALL (i=1:nelm)
        Uoo(i)= 0.0D0
        Uoo(i+nelm)= tobi*(1.0D0 +(z0(i)**2))
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!----------------------------------------------------------------------------------------------------------
    CASE (9)
        FORALL (i=1:nelm)
        Uoo(i)= (tobi/(DSQRT(2.0D0)*(1.0D0+alpha)))*( (1.0D0+alpha)*x0(i) + (1.0D0-alpha)*y0(i) )
        Uoo(i+nelm)= (tobi/(DSQRT(2.0D0)*(1.0D0+alpha)))*( (-1.0D0+alpha)*x0(i) + (-1.0D0-alpha)*y0(i) )
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!----------------------------------------------------------------------------------------------------------
    CASE (10)
        !FORALL (i=1:nelm)
        !Uoo(i)= gi*(1.0D0- DEXP((-100.0D0*DBLE(stride))/DBLE(steps)))*y0(i)/(1.0D0+alpha)
        !Uoo(i+nelm)=  gi*(1.0D0- DEXP((-100.0D0*DBLE(stride))/DBLE(steps)))*alpha*x0(i)/(1.0D0+alpha)
        !Uoo(i+nelm+nelm)= 0.0D0
        !END FORALL
        FORALL (i=1:nelm)
        Uoo(i)= tobi*y0(i)/(1.0D0+alpha)
        Uoo(i+nelm)=  tobi*alpha*x0(i)/(1.0D0+alpha)
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!----------------------------------------------------------------------------------------------------------
    CASE (11)
        FORALL (i=1:nelm)
        Uoo(i)= DSIN(z0(i)) + DCOS(y0(i))
        Uoo(i+nelm)= DSIN(x0(i))+DCOS(z0(i))
        Uoo(i+nelm+nelm)= DSIN(y0(i))+DCOS(x0(i))
        END FORALL
!----------------------------------------------------------------------------------------------------------
    CASE (12)
        FORALL (i=1:nelm)
        Uoo(i)= tobi*y0(i)*(1.0D0 - ((-0.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride))))
        Uoo(i+nelm)= tobi*x0(i)*(1.0D0 + ((-0.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride))))
        !Uoo(i)= gi*y0(i)*(1.0D0 - ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride))))
        !Uoo(i+nelm)= gi*x0(i)*(1.0D0 + ((-2.0D0*alpha) + (((4.0D0*alpha)/DBLE(steps))*DBLE(stride))))
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!----------------------------------------------------------------------------------------------------------
    CASE (13)
        FORALL (i=1:nelm)
        Uoo(i)= tobi*(y0(i))*(1.0D0 - ((2.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride))))
        Uoo(i+nelm)= tobi*x0(i)*(1.0D0 + ((2.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride))))
        !Uoo(i)= gi*(y0(i))*(1.0D0 - ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride))))
        !Uoo(i+nelm)= gi*x0(i)*(1.0D0 + ((4.0D0*alpha) - (((4.0D0*alpha)/DBLE(steps))*DBLE(stride))))
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
```

```
!------------------------------------------------------------------------------------------------------------
    CASE (14)
!$omp parallel
!$omp do
        DO i = 1, nelm
            CALL VelPsi(x0(i), y0(i), Uoo(i), Uoo(i+nelm))
            Uoo(i+nelm+nelm)= 0.0D0
        END DO
!$omp end do
!$omp end parallel
!------------------------------------------------------------------------------------------------------------
    CASE (15)
        FORALL (i=1:nelm)
        Uoo(i)= 0.5D0*tobi*y0(i)/(1.0D0+alpha)
        Uoo(i+nelm)= 0.5D0*tobi*alpha*x0(i)/(1.0D0+alpha)
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!------------------------------------------------------------------------------------------------------------
    CASE (16)
        FORALL (i=1:nelm)
        Uoo(i)= tobi*y0(i)/(1.0D0+alpha)
        !Uoo(i+nelm)= gi*x0(i)*((-0.5D0*alpha) + (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))/(1.0D0+alpha)
        Uoo(i+nelm)= tobi*x0(i)*alpha/(1.0D0+alpha)
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!------------------------------------------------------------------------------------------------------------
    CASE (17)
        FORALL (i=1:nelm)
        Uoo(i)= tobi*y0(i)/(1.0D0+alpha)
        !Uoo(i+nelm)= gi*x0(i)*((2.5D0*alpha) - (((2.5D0*alpha)/DBLE(steps))*DBLE(stride)))/(1.0D0+alpha)
        Uoo(i+nelm)= tobi*x0(i)*alpha/(1.0D0+alpha)
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!------------------------------------------------------------------------------------------------------------
    CASE (18)
        FORALL (i=1:nelm)
        Uoo(i)= tobi*(1.0D0- DEXP((-100.0D0*DBLE(stride))/DBLE(steps)))*y0(i)*(1.0D0 - alpha)
        Uoo(i+nelm)= tobi*(1.0D0- DEXP((-100.0D0*DBLE(stride))/DBLE(steps)))*x0(i)*(1.0D0 + alpha)
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!------------------------------------------------------------------------------------------------------------
    CASE (71)
        FORALL (i=1:nelm)
        Uoo(i)= (tobi*x0(i)*(alpha - 2.0D0))/(2.0D0*DSQRT(4.0D0 - 2.0D0*alpha + alpha**2))
        Uoo(i+nelm)= (tobi*y0(i))/(DSQRT(4.0D0 - 2.0D0*alpha + alpha**2))
        Uoo(i+nelm+nelm)= (-alpha*gi*z0(i))/(2.0D0*DSQRT(4.0D0 - 2.0D0*alpha + alpha**2))
        END FORALL
!------------------------------------------------------------------------------------------------------------
    CASE (72)
        FORALL (i=1:nelm)
        Uoo(i)= (tobi*x0(i)*(alpha - 2.0D0))/(2.0D0*DSQRT(4.0D0 - 2.0D0*alpha + alpha**2))
        Uoo(i+nelm)= (tobi*y0(i))/(DSQRT(4.0D0 - 2.0D0*alpha + alpha**2))
        Uoo(i+nelm+nelm)= (-alpha*gi*z0(i))/(2.0D0*DSQRT(4.0D0 - 2.0D0*alpha + alpha**2))
        END FORALL
!------------------------------------------------------------------------------------------------------------
    CASE DEFAULT
        NVF = 0
        FORALL (i=1:nelm)
        Uoo(i)= 0.0D0
        Uoo(i+nelm)= 0.0D0
        Uoo(i+nelm+nelm)= 0.0D0
        END FORALL
!------------------------------------------------------------------------------------------------------------
        WRITE (ULog,*)
        WRITE (ULog,*) '    Velocity menu:'
        WRITE (ULog,*) '    Chosen velocity field is not available; It was taken the zero velocity field.'
    END SELECT
    END SUBROUTINE Infinity_Flow
!============================================================================================================
END MODULE Mod_Velocity_Menu
```

```
MODULE Mod_VelFieldTRM
!====================================================================================================
! : Version: 0.7 created on  -- / III /2010
!                                                                               Marco Antonio Reyes Huesca
!----------------------------------------------------------------------------------------------------
!   Version: 1.0 created on 12 / 11 / 2012
!                                                                                 Alfredo Sanjuan Sanjuan
!====================================================================================================
!   This module was made to evaluate the solution of Two Roll Mill on surface.
!====================================================================================================
CONTAINS
!====================================================================================================
SUBROUTINE VelPsi(x00, y00, Velx, Vely)
!====================================================================================================
!   This subroutine computes the velocity field of analitical solution of TRM.
!====================================================================================================
    USE Mod_SharedVars , ONLY: DBL,  NCs, R, w, vR, ORy, StPnt, a, de, g, M1, M2, A0, B0, C0, D0, Kc, An,   &
                            &  Bn, Cn, Dn, lambda, shrate
!    USE Mod_TRMCoefs
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
    REAL (KIND = DBL), INTENT(IN)  :: x00, y00            !Point's coordinates
    REAL (KIND = DBL), INTENT(OUT) :: Velx, Vely        !Velocities
!====================================================================================================
!   Variables inside the subroutine
    INTEGER :: i                                        !number of points
    REAL (KIND = DBL)  :: u, v                          !Components of velocity vector
    REAL (KIND = DBL)  :: co1, co2                      !Complex componentes, co1, real, co2 imaginary
    REAL (KIND = DBL)  :: SumA, SumB                    !Sumatories to compute the velocity
    REAL (KIND = DBL)  :: x, y                          !Coordinates of p
    !    REAL (KIND = DBL)  :: VelX, VelY                    !Components of velocity at the end of computation
    COMPLEX (KIND = DBL)  :: Cmp                        !Complex number
!====================================================================================================
!   Initialize
    x    = x00
    y    = y00
    SumA = x**2 + (y - a)**2
    co1 = (x**2 + y**2 - a**2)
    co2 = 2*a*x
!----------------------------------------------------------------------------------------------------
    Cmp = DCMPLX(co1,co2)/SumA
!----------------------------------------------------------------------------------------------------
    v   = DLOG((co1**2 + co2**2)/SumA**2)/2
    u   = DIMAG(CDLOG(Cmp))
!   u   = DLOG(DATAN(co2/co1))
!----------------------------------------------------------------------------------------------------
    SumA = 0.0D0
    SumB = 0.0D0
    VelX = 0.0D0
    VelY = 0.0D0
!====================================================================================================
   IF (v < vR(1) .AND. v > vR(2)) THEN
!!      !DEC$ PARALLEL
      DO i = NCs, 2, -1
         SumA = SumA +                                                &
         &  (COS(i*u)*SIN(u) + i*(COSH(v) - COS(u))*SIN(i*u))*  &
         &    (An(i)*COSH((1 + i)*v) + Bn(i)*COSH((-1 + i)*v) + &
         &     Cn(i)*SINH((1 + i)*v) + Dn(i)*SINH((-1 + i)*v))

         SumB = SumB + COS(i*u)*((COSH(v) - COS(u))*    &
         &   ((i + 1)*An(i)*SINH((i + 1)*v) +           &
         &    (i - 1)*Bn(i)*SINH((i - 1)*v) +           &
         &    (i + 1)*Cn(i)*COSH((i + 1)*v) +           &
         &    (i - 1)*Dn(i)*COSH((i - 1)*v)) -          &
         &   SINH(v)*(An(i)*COSH((i + 1)*v) +           &
         &            Bn(i)*COSH((i - 1)*v) +           &
         &            Cn(i)*SINH((i + 1)*v) +           &
         &            Dn(i)*SINH((i - 1)*v)))
      END DO
!====================================================================================================
```



```
      VelX = -((SIN(u)*SINH(v)* &
      &  (-(SIN(u)*(Kc*COS(u) + (A0 - Kc + Bn(1))*COSH(v) +   &
      &  (C0 + D0*v)*SINH(v) + (An(1)*(COSH(v) + COSH(3*v)) + &
      &   Cn(1)*(SINH(v) + SINH(3*v)))/2)) -                  &
      &          SumA))/(COSH(v) - COS(u))**2) +             &
      &   ((-1 + COS(u)*COSH(v))*                             &
      &    (B0*(COS(u) - COSH(v)) - 2*Cn(1)*COS(u)*           &
      &      COSH(2*v) - (A0 + D0)*SINH(v) - Kc*SINH(v) -     &
      &      COSH(v)*(C0 + D0*v + 4*An(1)*COS(u)*             &
      &      SINH(v)) + (SINH(v)*(A0*COSH(v) +                &
      &      C0*SINH(v) + D0*v*SINH(v) +                      &
      &      COS(u)*(Bn(1) + An(1)*COSH(2*v) +                &
      &      Cn(1)*SINH(2*v))))/(-COS(u) + COSH(v)) +         &
      &        SumB/(COS(u) - COSH(v))))/(COSH(v) - COS(u));
!====================================================================================================
      VelY = ((1 - COS(u)*COSH(v))*                          &
      &  (-(SIN(u)*(Kc*COS(u) + (A0 - Kc + Bn(1))*COSH(v) +  &
      &  (C0 + D0*v)*SINH(v) +                               &
      &   (An(1)*(COSH(v) + COSH(3*v)) +                     &
      &    Cn(1)*(SINH(v) + SINH(3*v)))/2)) -                &
      &          SumA)/(COSH(v) - COS(u))**2) -             &
      &   (SIN(u)*SINH(v)*(B0*(COS(u) - COSH(v)) -           &
      &    2*Cn(1)*COS(u)*COSH(2*v) - (A0 + D0)*SINH(v) -    &
      &       Kc*SINH(v) - COSH(v)*(C0 + D0*v +              &
      &      4*An(1)*COS(u)*SINH(v)) +                       &
      &     (SINH(v)*(A0*COSH(v) + C0*SINH(v) +              &
      &      D0*v*SINH(v) + COS(u)*(Bn(1) +                  &
      &      An(1)*COSH(2*v) + Cn(1)*SINH(2*v))))/           &
      &      (-COS(u) + COSH(v)) +                           &
      &        SumB/(COS(u) - COSH(v))))/(COSH(v) - COS(u));
!====================================================================================================
   ELSE IF (v >= vR(1)) THEN
      VelX = -(y - ORy(1))*w(1)
      VelY = x*w(1)
   ELSE IF (v <= vR(2)) THEN
      VelX = -(y - ORy(2))*w(2)
      VelY = x*w(2)
   END IF
!====================================================================================================
 !  VelP = (/  VelX, VelY  /)
!====================================================================================================
END SUBROUTINE VelPsi
!====================================================================================================
!FUNCTION TRMill_velx (x)
!!   This function computes the velocity component x based on analitical solution of TRM velocity field.
!!===================================================================================================
!    USE Mod_SharedVars, ONLY: DBL
!    USE Mod_TRMCoefs
!!===================================================================================================
!    IMPLICIT NONE
!!===================================================================================================
!!   Variables
!    REAL (KIND = DBL), DIMENSION(2), INTENT(IN)  :: P           !Point's coordinates
!    REAL (KIND = DBL), DIMENSION(2), INTENT(OUT) :: VelP        !Velocities
!!===================================================================================================
!!   Variables inside the subroutine
!    INTEGER :: n                                       !number of points
!    REAL (KIND = DBL)  :: u, v                         !Components of velocity vector
!    REAL (KIND = DBL)  :: co1, co2                     !Complex componentes, co1, real, co2 imaginary
!    REAL (KIND = DBL)  :: SumA, SumB                   !Sumatories to compute the velocity
!    REAL (KIND = DBL)  :: x, y                         !Coordinates of p
!    REAL (KIND = DBL)  :: VelX, VelY                   !Components of velocity at the end of computation
!    COMPLEX (KIND = DBL)  :: Cmp                       !Complex number
!!===================================================================================================
!!   Initialize
!   x    = P(1)
!   y    = P(2)
!   SumA = x**2 + (y - a)**2
!   co1 = (x**2 + y**2 - a**2)
!   co2 = 2*a*x
!
!   Cmp = DCMPLX(co1,co2)/SumA
```

```
!
!   v   = DLOG((co1**2 + co2**2)/SumA**2)/2
!   u   = DIMAG(CDLOG(Cmp))
!!   u   = DLOG(DATAN(co2/co1))
!
!   SumA = 0.0D0
!   SumB = 0.0D0
!   VelX = 0.0D0
!   VelY = 0.0D0
!
!   IF (v < vR(1) .AND. v > vR(2)) THEN
!!      !DEC$ PARALLEL
!      DO n = NCs, 2, -1
!         SumA = SumA +                                          &
!         &  (COS(n*u)*SIN(u) + n*(COSH(v) - COS(u))*SIN(n*u))*  &
!         &    (An(n)*COSH((1 + n)*v) + Bn(n)*COSH((-1 + n)*v) + &
!         &     Cn(n)*SINH((1 + n)*v) + Dn(n)*SINH((-1 + n)*v))
!
!         SumB = SumB + COS(n*u)*((COSH(v) - COS(u))*    &
!         &   ((n + 1)*An(n)*SINH((n + 1)*v) +           &
!         &    (n - 1)*Bn(n)*SINH((n - 1)*v) +           &
!         &    (n + 1)*Cn(n)*COSH((n + 1)*v) +           &
!         &    (n - 1)*Dn(n)*COSH((n - 1)*v)) -          &
!         &   SINH(v)*(An(n)*COSH((n + 1)*v) +           &
!         &            Bn(n)*COSH((n - 1)*v) +           &
!         &            Cn(n)*SINH((n + 1)*v) +           &
!         &            Dn(n)*SINH((n - 1)*v)))
!      END DO
!
!      VelX = -((SIN(u)*SINH(v)* &
!      &  (-(SIN(u)*(Kc*COS(u) + (A0 - Kc + Bn(1))*COSH(v) +   &
!      &  (C0 + D0*v)*SINH(v) + (An(1)*(COSH(v) + COSH(3*v)) + &
!      &   Cn(1)*(SINH(v) + SINH(3*v)))/2)) -                  &
!      &          SumA))/(COSH(v) - COS(u))**2) +             &
!      &   ((-1 + COS(u)*COSH(v))*                             &
!      &    (B0*(COS(u) - COSH(v)) - 2*Cn(1)*COS(u)*           &
!      &      COSH(2*v) - (A0 + D0)*SINH(v) - Kc*SINH(v) -     &
!      &      COSH(v)*(C0 + D0*v + 4*An(1)*COS(u)*             &
!      &      SINH(v)) + (SINH(v)*(A0*COSH(v) +                &
!      &      C0*SINH(v) + D0*v*SINH(v) +                      &
!      &      COS(u)*(Bn(1) + An(1)*COSH(2*v) +                &
!      &      Cn(1)*SINH(2*v))))/(-COS(u) + COSH(v)) +         &
!      &        SumB/(COS(u) - COSH(v))))/(COSH(v) - COS(u));
!
!      VelY = ((1 - COS(u)*COSH(v))*                          &
!      &  (-(SIN(u)*(Kc*COS(u) + (A0 - Kc + Bn(1))*COSH(v) +  &
!      &  (C0 + D0*v)*SINH(v) +                               &
!      &   (An(1)*(COSH(v) + COSH(3*v)) +                     &
!      &    Cn(1)*(SINH(v) + SINH(3*v)))/2)) -                &
!      &          SumA)/(COSH(v) - COS(u))**2) -             &
!      &   (SIN(u)*SINH(v)*(B0*(COS(u) - COSH(v)) -           &
!      &    2*Cn(1)*COS(u)*COSH(2*v) - (A0 + D0)*SINH(v) -    &
!      &       Kc*SINH(v) - COSH(v)*(C0 + D0*v +              &
!      &      4*An(1)*COS(u)*SINH(v)) +                       &
!      &     (SINH(v)*(A0*COSH(v) + C0*SINH(v) +              &
!      &      D0*v*SINH(v) + COS(u)*(Bn(1) +                  &
!      &      An(1)*COSH(2*v) + Cn(1)*SINH(2*v))))/           &
!      &      (-COS(u) + COSH(v)) +                           &
!      &        SumB/(COS(u) - COSH(v))))/(COSH(v) - COS(u));
!
!   ELSE IF (v >= vR(1)) THEN
!      VelX = -(y - ORy(1))*w(1)
!      VelY = x*w(1)
!   ELSE IF (v <= vR(2)) THEN
!      VelX = -(y - ORy(2))*w(2)
!      VelY = x*w(2)
!   END IF
!
!   VelP = (/  VelX, VelY  /)
!END FUNCTION TRMill_velx
!
!==============================================================================================================
```



```
END MODULE  Mod_VelFieldTRM
```

```
MODULE Mod_SNEDOS
!====================================================================================================
!Version 1.0            07 / January / 2013
!                                                                 Alfredo Sanjuan Sanjuan, in others words, me :)
!====================================================================================================
!   This module have the subroutines to obtain the position of the dorp in a new time t1.
!----------------------------------------------------------------------------------------------------
CONTAINS
    SUBROUTINE Euler_Method(p, Vel, h, npts, vna)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars , ONLY:DBL
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: npts                                               !number of nodes
    REAL (KIND = DBL), INTENT(IN) :: h                                        !step of time
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: p  !table of position vector of all points
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: Vel   !arrays of the components of
                                                                        !velocity over all points
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: vna   !arrays of the components of normal
                                                                        !vector over all points
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                                !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: p1        !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: f     !Dummy array
        REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0     !Dummy array

!====================================================================================================
   !Initialize
    ALLOCATE(f0(npts),f(npts,3), p1(npts,3))
    f0 = 0.0D0
    f  = 0.0D0
    p1 = 0.0D0
!----------------------------------------------------------------------------------------------------
    FORALL (i=1:npts)
        !f0(i,1)= (0.0D0*vna(i,1))+(0.0D0*vna(i,2))+(0.0D0*vna(i,3))
        f0(i)=((Vel(i,1)*vna(i,1))+(Vel(i,2)*vna(i,2))+(Vel(i,3)*vna(i,3)))
    END FORALL
!----------------------------------------------------------------------------------------------------
    FORALL (i=1:npts)
        f(i,:)= h*vna(i,:)*f0(i)
        !f(i,2)= h*vna(i,2)*f0(i)
        !f(i,3)= h*vna(i,3)*f0(i)
    END FORALL
!----------------------------------------------------------------------------------------------------
    !f=h*Vel
!----------------------------------------------------------------------------------------------------
    p1= p + f
    p = p1

!----------------------------------------------------------------------------------------------------
    END SUBROUTINE Euler_Method
    SUBROUTINE Euler_Method2(x0, y0, z0,&
                            & Vel1x, Vel1y, Vel1z, &
                            &h, nelm)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars , ONLY:DBL, vnx0, vny0, vnz0
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
```



```
    INTEGER, INTENT(IN) :: nelm                                               !number of elements
    REAL (KIND = DBL), INTENT(IN) :: h                                        !step of time
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0, y0, z0  !arrays of the components of
                                                                               !position vectors
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) ::Vel1x, Vel1y, Vel1z !arrays of the components of
                                                                               !velocity over collocation points
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                                !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01        !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fx, fy, fz     !Dummy array
        REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0     !Dummy array

!====================================================================================================
   !Initialize
    ALLOCATE(f0(nelm),fx(nelm),fy(nelm),fz(nelm), x01(nelm), y01(nelm), z01(nelm))
    f0 = 0.0D0
    fx = 0.0D0
    fy = 0.0D0
    fz = 0.0D0
    x01 = 0.0D0
    y01 = 0.0D0
    z01 = 0.0D0
!====================================================================================================
!   EULER METHOD
!====================================================================================================
    FORALL (i=1:nelm)
        f0(i)= (Vel1x(i)*vnx0(i) + Vel1y(i)*vny0(i) + Vel1z(i)*vnz0(i))
    END FORALL
!----------------------------------------------------------------------------------------------------
    FORALL (i=1:nelm)
        fx(i) = h*vnx0(i)*f0(i)
        fy(i) = h*vny0(i)*f0(i)
        fz(i) = h*vnz0(i)*f0(i)
    END FORALL
!====================================================================================================
!   LAGRANGE METHOD
!====================================================================================================
    !FORALL (i=1:nelm)
    !    fx(i) = h*Vel1x(i)
    !    fy(i) = h*Vel1y(i)
    !    fz(i) = h*Vel1z(i)
    !END FORALL
!====================================================================================================
    x01= x0 + fx
    y01= y0 + fy
    z01= z0 + fz
!----------------------------------------------------------------------------------------------------
    x0= x01
    y0= y01
    z0= z01
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE Euler_Method2
!====================================================================================================
!====================================================================================================
    SUBROUTINE Euler_Method3(x0, y0, z0,&
                            & Vel1x, Vel1y, Vel1z, &
                            &h, nelm,tr)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars , ONLY:DBL, vnx0, vny0, vnz0, farel
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: nelm                                               !number of elements
    REAL (KIND = DBL), INTENT(IN) :: h                                        !step of time
    REAL (KIND = DBL), INTENT(IN) :: tr                                       !factor to mix eulerian and lagrangian
```

```
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0, y0, z0  !arrays of the components of
                                                                              !position vectors
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) ::Vel1x, Vel1y, Vel1z !arrays of the components of
                                                                              !velocity over collocation points
!==============================================================================================================
!   Variables inside the subroutine
!--------------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                                  !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01         !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe, fye, fze     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxl, fyl, fzl     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0     !Dummy array

!==============================================================================================================
   !Initialize
    ALLOCATE(f0(nelm),fxe(nelm),fye(nelm),fze(nelm),fxl(nelm),fyl(nelm),fzl(nelm), x01(nelm), y01(nelm), z01(nelm))
    f0  = 0.0D0
    fxe = 0.0D0
    fye = 0.0D0
    fze = 0.0D0
    fxl = 0.0D0
    fyl = 0.0D0
    fzl = 0.0D0
    x01 = 0.0D0
    y01 = 0.0D0
    z01 = 0.0D0
    farel = 0.0D0
!==============================================================================================================
!   EULER METHOD
!==============================================================================================================
    FORALL (i=1:nelm)
        f0(i)= (Vel1x(i)*vnx0(i) + Vel1y(i)*vny0(i) + Vel1z(i)*vnz0(i))
    END FORALL
    farel = f0
!--------------------------------------------------------------------------------------------------------------
    FORALL (i=1:nelm)
        fxe(i) = h*vnx0(i)*f0(i)
        fye(i) = h*vny0(i)*f0(i)
        fze(i) = h*vnz0(i)*f0(i)
    END FORALL
!==============================================================================================================
!   LAGRANGE METHOD
!==============================================================================================================
    FORALL (i=1:nelm)
        fxl(i) = h*Vel1x(i)
        fyl(i) = h*Vel1y(i)
        fzl(i) = h*Vel1z(i)
    END FORALL
!==============================================================================================================
    x01= x0 + (tr*fxl) + (1.0D0-tr)*fxe
    y01= y0 + (tr*fyl) + (1.0D0-tr)*fye
    z01= z0 + (tr*fzl) + (1.0D0-tr)*fze
!--------------------------------------------------------------------------------------------------------------
    x0= x01
    y0= y01
    z0= z01
!--------------------------------------------------------------------------------------------------------------
    END SUBROUTINE Euler_Method3
!==============================================================================================================
    SUBROUTINE Euler_Method4(p, h, npts,tr)
!==============================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!==============================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars , ONLY:DBL, Vel, vna
!==============================================================================================================
    IMPLICIT NONE
!==============================================================================================================
!   Variables
!--------------------------------------------------------------------------------------------------------------
```



```
    INTEGER, INTENT(IN) :: npts                                       !number of nodes
    REAL (KIND = DBL), INTENT(IN) :: h                                !step of time
    REAL (KIND = DBL), INTENT(IN) :: tr                               !factor to mix eulerian and lagrangian

    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: p  !arrays of the components of
!==============================================================================================================
!   Variables inside the subroutine
!--------------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                                  !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: px1, py1, pz1         !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe, fye, fze     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxl, fyl, fzl     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0     !Dummy array

!==============================================================================================================
   !Initialize
    ALLOCATE(f0(npts),fxe(npts),fye(npts),fze(npts),fxl(npts),fyl(npts),fzl(npts),px1(npts),py1(npts),pz1(npts))
    f0  = 0.0D0
    fxe = 0.0D0
    fye = 0.0D0
    fze = 0.0D0
    fxl = 0.0D0
    fyl = 0.0D0
    fzl = 0.0D0
    px1 = 0.0D0
    py1 = 0.0D0
    pz1 = 0.0D0
!==============================================================================================================
!   EULER METHOD
!==============================================================================================================
    FORALL (i=1:npts)
        f0(i)= (Vel(i,1)*vna(i,1) + Vel(i,2)*vna(i,2) + Vel(i,3)*vna(i,3))
    END FORALL
!--------------------------------------------------------------------------------------------------------------
        fxe(:) = h*vna(:,1)*f0(:)
        fye(:) = h*vna(:,2)*f0(:)
        fze(:) = h*vna(:,3)*f0(:)
!==============================================================================================================
!   LAGRANGE METHOD
!==============================================================================================================
        fxl(:) = h*Vel(:,1)
        fyl(:) = h*Vel(:,2)
        fzl(:) = h*Vel(:,3)
!==============================================================================================================
    px1= p(:,1) + (tr*fxl) + (1.0D0-tr)*fxe
    py1= p(:,2) + (tr*fyl) + (1.0D0-tr)*fye
    pz1= p(:,3) + (tr*fzl) + (1.0D0-tr)*fze
!--------------------------------------------------------------------------------------------------------------
    p(:,1)= px1
    p(:,2)= py1
    p(:,3)= pz1
!--------------------------------------------------------------------------------------------------------------
    END SUBROUTINE Euler_Method4
!==============================================================================================================
    SUBROUTINE Euler_MethodM(x0, y0, z0,&
                            & Vel1x, Vel1y, Vel1z, &
                            & Vel2x, Vel2y, Vel2z, &
                            &h, nelm,tr)
!==============================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!==============================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars , ONLY:DBL, vnx0, vny0, vnz0, vnx02, vny02, vnz02, farel
!==============================================================================================================
    IMPLICIT NONE
!==============================================================================================================
!   Variables
!--------------------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: nelm                                       !number of elements
    REAL (KIND = DBL), INTENT(IN) :: h                                !step of time
    REAL (KIND = DBL), INTENT(IN) :: tr                               !factor to mix eulerian and lagrangian
```

```
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0, y0, z0  !arrays of the components of
                                                                        !position vectors
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) ::Vel1x, Vel1y, Vel1z !arrays of the components of
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) ::Vel2x, Vel2y, Vel2z !arrays of the components of
                                                                        !velocity over collocation points
!==========================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                                !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01        !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe1, fye1, fze1     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxl1, fyl1, fzl1     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f01     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe2, fye2, fze2     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxl2, fyl2, fzl2     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f02     !Dummy array

!==========================================================================================================
   !Initialize
    ALLOCATE(f01(nelm),f02(nelm),fxe1(nelm),fye1(nelm),fze1(nelm),fxl1(nelm),fyl1(nelm),fzl1(nelm),fxe2(nelm),
    fye2(nelm),fze2(nelm),fxl2(nelm),fyl2(nelm),fzl2(nelm),x01(nelm), y01(nelm), z01(nelm))
    f01  = 0.0D0
    fxe1 = 0.0D0
    fye1 = 0.0D0
    fze1 = 0.0D0
    fxl1 = 0.0D0
    fyl1 = 0.0D0
    fzl1 = 0.0D0
    f02  = 0.0D0
    fxe2 = 0.0D0
    fye2 = 0.0D0
    fze2 = 0.0D0
    fxl2 = 0.0D0
    fyl2 = 0.0D0
    fzl2 = 0.0D0
    x01  = 0.0D0
    y01  = 0.0D0
    z01  = 0.0D0
    farel = 0.0D0
!==========================================================================================================
!   EULER METHOD
!==========================================================================================================
    FORALL (i=1:nelm)
        f01(i)= (Vel1x(i)*vnx0(i) + Vel1y(i)*vny0(i) + Vel1z(i)*vnz0(i))
        f02(i)= (Vel2x(i)*vnx02(i) + Vel2y(i)*vny02(i) + Vel2z(i)*vnz02(i))
    END FORALL
    farel = 0.5D0*(f01+f02)
!----------------------------------------------------------------------------------------------------------
    FORALL (i=1:nelm)
        fxe1(i) = 0.5D0*h*vnx0(i)*f01(i)
        fye1(i) = 0.5D0*h*vny0(i)*f01(i)
        fze1(i) = 0.5D0*h*vnz0(i)*f01(i)
        fxe2(i) = 0.5D0*h*vnx02(i)*f02(i)
        fye2(i) = 0.5D0*h*vny02(i)*f02(i)
        fze2(i) = 0.5D0*h*vnz02(i)*f02(i)
    END FORALL
!==========================================================================================================
!   LAGRANGE METHOD
!==========================================================================================================
    FORALL (i=1:nelm)
        fxl1(i) = 0.5D0*h*Vel1x(i)
        fyl1(i) = 0.5D0*h*Vel1y(i)
        fzl1(i) = 0.5D0*h*Vel1z(i)
        fxl2(i) = 0.5D0*h*Vel2x(i)
        fyl2(i) = 0.5D0*h*Vel2y(i)
        fzl2(i) = 0.5D0*h*Vel2z(i)
    END FORALL
!==========================================================================================================
    x01= x0 + (tr*(fxl1+fxl2)) + (1.0D0-tr)*(fxe1+fxe2)
    y01= y0 + (tr*(fyl1+fyl2)) + (1.0D0-tr)*(fye1+fye2)
    z01= z0 + (tr*(fzl1+fzl2)) + (1.0D0-tr)*(fze1+fze2)
!----------------------------------------------------------------------------------------------------------
```



```
    x0= x01
    y0= y01
    z0= z01
!----------------------------------------------------------------------------------------------------------
    END SUBROUTINE Euler_MethodM
!==========================================================================================================
!==========================================================================================================
    SUBROUTINE RK3(x0,      y0,    z0,&
               & Vel1x, Vel1y, Vel1z,&
               & h,      nelm        )
!==========================================================================================================
!   This subroutine computes the normal vector of every node on the surface using a Runge-Kutta of order 3

!==========================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars , ONLY:DBL, vnx0, vny0, vnz0
!==========================================================================================================
    IMPLICIT NONE
!==========================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: nelm                                         !number of elements
    REAL (KIND = DBL), INTENT(IN) :: h                                  !step of time
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0, y0, z0  !arrays of the components of
                                                                        !position vectors
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) ::Vel1x, Vel1y, Vel1z !arrays of the components of
                                                                        !velocity over collocation points
!==========================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                                !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01   !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fx, fy, fz      !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: k1x, k2x, k3x   !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: k1y, k2y, k3y   !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: k1z, k2z, k3z   !Dummy array

    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0              !Dummy array

!==========================================================================================================
   !Initialize
    ALLOCATE(f0(nelm),fx(nelm),fy(nelm),fz(nelm), x01(nelm), y01(nelm), z01(nelm))
    f0 = 0.0D0
    fx = 0.0D0
    fy = 0.0D0
    fz = 0.0D0
    x01 = 0.0D0
    y01 = 0.0D0
    z01 = 0.0D0
    ALLOCATE(k1x(nelm), k2x(nelm), k3x(nelm),k1y(nelm), k2y(nelm), k3y(nelm),k1z(nelm), k2z(nelm), k3z(nelm))
    k1x = 0.0D0
    k2x = 0.0D0
    k3x = 0.0D0
    k1y = 0.0D0
    k2y = 0.0D0
    k3y = 0.0D0
    k1z = 0.0D0
    k2z = 0.0D0
    k3z = 0.0D0
!==========================================================================================================
!   RUNGE-KUTTA OF THIRD ORDER RK3
!   EULERIAN METHOD
!==========================================================================================================
!    FORALL (i=1:nelm)
!        f0(i)= (Vel1x(i)*vnx0(i) + Vel1y(i)*vny0(i) + Vel1z(i)*vnz0(i))
!    END FORALL
!!---------------------------------------------------------------------------------------------------------
!    FORALL (i=1:nelm)
!        fx(i) = h*vnx0(i)*f0(i)
!        fy(i) = h*vny0(i)*f0(i)
!        fz(i) = h*vnz0(i)*f0(i)
!    END FORALL
```

```fortran
!====================================================================================================
!   LAGRANGIAN METHOD
!====================================================================================================
!First we calculate K1
!----------------------------------------------------------------------------------------------------
    FORALL (i=1:nelm)
        k1x(i) = Vel1x(i)
        k1y(i) = Vel1y(i)
        k1z(i) = Vel1z(i)
    END FORALL
!====================================================================================================
!Then we calculate K2
!----------------------------------------------------------------------------------------------------
    FORALL (i=1:nelm)
        k2x(i) = (x0(i)) + ((h/3.0D0)*k1x(i))
        k2y(i) = (y0(i)) + ((h/3.0D0)*k1y(i))
        k2z(i) = (z0(i)) + ((h/3.0D0)*k1z(i))
    END FORALL
!====================================================================================================
!Finaly we calculate K3
!----------------------------------------------------------------------------------------------------
    FORALL (i=1:nelm)
        k3x(i) = (x0(i)) + ((h*2.0D0/3.0D0)*k2x(i))
        k3y(i) = (y0(i)) + ((h*2.0D0/3.0D0)*k2y(i))
        k3z(i) = (z0(i)) + ((h*2.0D0/3.0D0)*k2z(i))
    END FORALL
!====================================================================================================
    x01= x0 + ((h/4.0D0)*(k1x+k3x))
    y01= y0 + ((h/4.0D0)*(k1y+k3y))
    z01= z0 + ((h/4.0D0)*(k1z+k3z))
!----------------------------------------------------------------------------------------------------
    x0= x01
    y0= y01
    z0= z01
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE RK3
!====================================================================================================
    SUBROUTINE Euler_Met5(p, h, K1, npts,tr)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars , ONLY:DBL, vna
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: npts                                         !number of nodes
    REAL (KIND = DBL), INTENT(IN) :: h                                  !step of time
    REAL (KIND = DBL), INTENT(IN) :: tr                                 !factor to mix eulerian and lagrangian

    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: K1  !arrays of the components of
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: p  !arrays of the components of
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                            !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: px1, py1, pz1        !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe, fye, fze     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxl, fyl, fzl     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0     !Dummy array

!====================================================================================================
   !Initialize
    ALLOCATE(f0(npts),fxe(npts),fye(npts),fze(npts),fxl(npts),fyl(npts),fzl(npts),px1(npts),py1(npts),pz1(npts))
    f0  = 0.0D0
    fxe = 0.0D0
    fye = 0.0D0
    fze = 0.0D0
    fxl = 0.0D0
    fyl = 0.0D0
```



```fortran
    fzl = 0.0D0
    px1 = 0.0D0
    py1 = 0.0D0
    pz1 = 0.0D0
!====================================================================================================
!   EULER METHOD
!====================================================================================================
    FORALL (i=1:npts)
        f0(i)= (K1(i,1)*vna(i,1) + K1(i,2)*vna(i,2) + K1(i,3)*vna(i,3))
    END FORALL
!----------------------------------------------------------------------------------------------------
        fxe(:) = h*vna(:,1)*f0(:)
        fye(:) = h*vna(:,2)*f0(:)
        fze(:) = h*vna(:,3)*f0(:)
!====================================================================================================
!   LAGRANGE METHOD
!====================================================================================================
        fxl(:) = h*K1(:,1)
        fyl(:) = h*K1(:,2)
        fzl(:) = h*K1(:,3)
!====================================================================================================
    px1= p(:,1) + (tr*fxl) + (1.0D0-tr)*fxe
    py1= p(:,2) + (tr*fyl) + (1.0D0-tr)*fye
    pz1= p(:,3) + (tr*fzl) + (1.0D0-tr)*fze
!----------------------------------------------------------------------------------------------------
    p(:,1)= px1
    p(:,2)= py1
    p(:,3)= pz1
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE Euler_Met5
!====================================================================================================
!====================================================================================================
    SUBROUTINE Euler_Met6(p, h, K1, K2, npts,tr)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars , ONLY:DBL, vna
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: npts                                         !number of nodes
    REAL (KIND = DBL), INTENT(IN) :: h                                  !step of time
    REAL (KIND = DBL), INTENT(IN) :: tr                                 !factor to mix eulerian and lagrangian

    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: K1, K2 !arrays of the components of
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: p  !arrays of the components of
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                            !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: px1, py1, pz1        !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe, fye, fze     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxl, fyl, fzl     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: Vel     !Dummy array

!====================================================================================================
   !Initialize
    ALLOCATE(f0(npts),fxe(npts),fye(npts),fze(npts),fxl(npts),fyl(npts),fzl(npts),px1(npts),py1(npts),pz1(npts),
    Vel(npts,3))
    f0  = 0.0D0
    fxe = 0.0D0
    fye = 0.0D0
    fze = 0.0D0
    fxl = 0.0D0
    fyl = 0.0D0
    fzl = 0.0D0
    px1 = 0.0D0
    py1 = 0.0D0
```

```fortran
    pz1 = 0.0D0
    Vel= 2*K2-K1
!====================================================================================================
!   EULER METHOD
!====================================================================================================
    FORALL (i=1:npts)
        f0(i)= (Vel(i,1)*vna(i,1) + Vel(i,2)*vna(i,2) + Vel(i,3)*vna(i,3))
    END FORALL
!----------------------------------------------------------------------------------------------------
        fxe(:) = h*vna(:,1)*f0(:)
        fye(:) = h*vna(:,2)*f0(:)
        fze(:) = h*vna(:,3)*f0(:)
!====================================================================================================
!   LAGRANGE METHOD
!====================================================================================================
        fxl(:) = h*Vel(:,1)
        fyl(:) = h*Vel(:,2)
        fzl(:) = h*Vel(:,3)
!====================================================================================================
    px1= p(:,1) + (tr*fxl) + (1.0D0-tr)*fxe
    py1= p(:,2) + (tr*fyl) + (1.0D0-tr)*fye
    pz1= p(:,3) + (tr*fzl) + (1.0D0-tr)*fze
!----------------------------------------------------------------------------------------------------
    p(:,1)= px1
    p(:,2)= py1
    p(:,3)= pz1
!----------------------------------------------------------------------------------------------------
    END SUBROUTINE Euler_Met6
!====================================================================================================
    SUBROUTINE KMaker(xel, yel, zel, &
                & K1, K2, K3, K4, nelm, h)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
    USE Mod_SharedVars , ONLY:DBL !, vnx0, vny0, vnz0, farel
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: nelm                                         !number of elements
    REAL (KIND = DBL), INTENT(IN) :: h                                  !step of time
!    REAL (KIND = DBL), INTENT(IN) :: tr                                 !factor to mix eulerian and lagrangian

    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: xel, yel, zel  !arrays of the components of
                                                                        !position vectors
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: K1, K2, K3, K4 !arrays of the components of
                                                                        !position vectors
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                            !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01        !Dummy array
!====================================================================================================
!
    ALLOCATE(x01(nelm),y01(nelm),z01(nelm))
    x01 = 0.0D0
    y01 = 0.0D0
    z01 = 0.0D0
!----------------------------------------------------------------------------------------------------
    FORALL (i=1:nelm)
    x01(i) = xel(i)+ (h/6.0D0)*(K1(i)          +2.0D0*K2(i)          +2.0D0*K3(i)          +K4(i))
    y01(i) = yel(i)+ (h/6.0D0)*(K1(i+nelm)     +2.0D0*K2(i+nelm)     +2.0D0*K3(i+nelm)     +K4(i+nelm))
    z01(i) = zel(i)+ (h/6.0D0)*(K1(i+nelm+nelm)+2.0D0*K2(i+nelm+nelm)+2.0D0*K3(i+nelm+nelm)+K4(i+nelm+nelm))
    END FORALL
!----------------------------------------------------------------------------------------------------
    xel = x01
    yel = y01
    zel = z01
                END SUBROUTINE KMaker
!====================================================================================================
    SUBROUTINE KMK3(pel, &
```



```fortran
                    & K1, K2, K3, npts, h)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
    USE Mod_SharedVars , ONLY:DBL !, vnx0, vny0, vnz0, farel
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: npts                                         !number of elements
    REAL (KIND = DBL), INTENT(IN) :: h                                  !step of time
!    REAL (KIND = DBL), INTENT(IN) :: tr                                 !factor to mix eulerian and lagrangian

    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: pel  !arrays of the components of
                                                                        !position vectors
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: K1, K2, K3 !arrays of the components of
                                                                        !position vectors
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                            !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: p01        !Dummy array
!====================================================================================================
!
    ALLOCATE(p01(npts,3))
    p01 = 0.0D0
!----------------------------------------------------------------------------------------------------
    FORALL (i=1:npts)
    p01(i,:) = pel(i,:) + (h/6.0D0)*(K1(i,:)+4.0D0*K2(i,:)+K3(i,:))
    END FORALL
!----------------------------------------------------------------------------------------------------
    pel = p01
    END SUBROUTINE KMK3
!====================================================================================================
    SUBROUTINE RK31(x0, y0, z0, h, K1, nelm,tr)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars , ONLY:DBL, vnx0, vny0, vnz0
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: nelm                                         !number of nodes
    REAL (KIND = DBL), INTENT(IN) :: h                                  !step of time
    REAL (KIND = DBL), INTENT(IN) :: tr                                 !factor to mix eulerian and lagrangian

    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: K1  !arrays of the components of
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0, y0, z0  !arrays of the components of
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                            !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: px1, py1, pz1        !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe, fye, fze     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxl, fyl, fzl     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0     !Dummy array

!====================================================================================================
   !Initialize
    ALLOCATE(f0(nelm),fxe(nelm),fye(nelm),fze(nelm),fxl(nelm),fyl(nelm),fzl(nelm),px1(nelm),py1(nelm),pz1(nelm))
    f0  = 0.0D0
    fxe = 0.0D0
    fye = 0.0D0
    fze = 0.0D0
    fxl = 0.0D0
    fyl = 0.0D0
    fzl = 0.0D0
    px1 = 0.0D0
```

```fortran
    py1 = 0.0D0
    pz1 = 0.0D0
!===================================================================================================
!   EULER METHOD
!===================================================================================================
    FORALL (i=1:nelm)
        f0(i)= (K1(i)*vnx0(i) + K1(i+nelm)*vny0(i) + K1(i+nelm+nelm)*vnz0(i))
    END FORALL
!---------------------------------------------------------------------------------------------------
        fxe(:) = h*vnx0(:)*f0(:)
        fye(:) = h*vny0(:)*f0(:)
        fze(:) = h*vnz0(:)*f0(:)
!===================================================================================================
!   LAGRANGE METHOD
!===================================================================================================
    FORALL (i=1:nelm)
        fxl(i) = h*K1(i)
        fyl(i) = h*K1(i+nelm)
        fzl(i) = h*K1(i+nelm+nelm)
    END FORALL
!===================================================================================================
    px1= x0 + (tr*fxl) + (1.0D0-tr)*fxe
    py1= y0 + (tr*fyl) + (1.0D0-tr)*fye
    pz1= z0 + (tr*fzl) + (1.0D0-tr)*fze
!---------------------------------------------------------------------------------------------------
    x0= px1
    y0= py1
    z0= pz1
!---------------------------------------------------------------------------------------------------
    END SUBROUTINE RK31
!===================================================================================================
!===================================================================================================
    SUBROUTINE RK32(x0, y0, z0, h, K1, K2, nelm,tr)
!===================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!===================================================================================================
    USE Mod_Nodal_Interp
    USE Mod_SharedVars , ONLY:DBL, vnx0, vny0, vnz0
!===================================================================================================
    IMPLICIT NONE
!===================================================================================================
!   Variables
!---------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: nelm                                              !number of nodes
    REAL (KIND = DBL), INTENT(IN) :: h                                       !step of time
    REAL (KIND = DBL), INTENT(IN) :: tr                                      !factor to mix eulerian and lagrangian   ↙

    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: K1, K2 !arrays of the components of
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0,y0,z0  !arrays of the components of
!===================================================================================================
!   Variables inside the subroutine
!---------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                                !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: px1, py1, pz1        !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe, fye, fze     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxl, fyl, fzl     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: Vel     !Dummy array

!===================================================================================================
   !Initialize
    ALLOCATE(f0(nelm),fxe(nelm),fye(nelm),fze(nelm),fxl(nelm),fyl(nelm),fzl(nelm),px1(nelm),py1(nelm),pz1(nelm),  ↙
    Vel(3*nelm))
    f0  = 0.0D0
    fxe = 0.0D0
    fye = 0.0D0
    fze = 0.0D0
    fxl = 0.0D0
    fyl = 0.0D0
    fzl = 0.0D0
    px1 = 0.0D0
    py1 = 0.0D0
```



```fortran
    pz1 = 0.0D0
    Vel= 2*K2-K1
!===================================================================================================
!   EULER METHOD
!===================================================================================================
    FORALL (i=1:nelm)
        f0(i)= (Vel(i)*vnx0(i) + Vel(i+nelm)*vny0(i) + Vel(i+nelm+nelm)*vnz0(i))
    END FORALL
!---------------------------------------------------------------------------------------------------
        fxe(:) = h*vnx0(:)*f0(:)
        fye(:) = h*vny0(:)*f0(:)
        fze(:) = h*vnz0(:)*f0(:)
!===================================================================================================
!   LAGRANGE METHOD
!===================================================================================================
    FORALL (i=1:nelm)
        fxl(i) = h*Vel(i)
        fyl(i) = h*Vel(i+nelm)
        fzl(i) = h*Vel(i+nelm+nelm)
    END FORALL
!===================================================================================================
    px1= x0 + (tr*fxl) + (1.0D0-tr)*fxe
    py1= y0 + (tr*fyl) + (1.0D0-tr)*fye
    pz1= z0 + (tr*fzl) + (1.0D0-tr)*fze
!---------------------------------------------------------------------------------------------------
    x0= px1
    y0= py1
    z0= pz1
!---------------------------------------------------------------------------------------------------
    END SUBROUTINE RK32
!===================================================================================================
!===================================================================================================
    SUBROUTINE RMK3(x0, y0, z0, &
                    & K1, K2, K3, nelm, h)
!===================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!===================================================================================================
    USE Mod_SharedVars , ONLY:DBL !, vnx0, vny0, vnz0, farel
!===================================================================================================
    IMPLICIT NONE
!===================================================================================================
!   Variables
!---------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: nelm                                              !number of elements
    REAL (KIND = DBL), INTENT(IN) :: h                                       !step of time
!     REAL (KIND = DBL), INTENT(IN) :: tr                                     !factor to mix eulerian and lagrangian   ↙

    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0,y0,z0  !arrays of the components of
                                                                             !position vectors
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: K1, K2, K3 !arrays of the components of
                                                                             !position vectors
!===================================================================================================
!   Variables inside the subroutine
!---------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                                !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01       !Dummy array
!===================================================================================================
!
    ALLOCATE(x01(nelm), y01(nelm), z01(nelm))
    x01 = 0.0D0
    y01 = 0.0D0
    z01 = 0.0D0
!---------------------------------------------------------------------------------------------------
    FORALL (i=1:nelm)
        x01(i) = x0(i) + (h/6.0D0)*(K1(i)+4.0D0*K2(i)+K3(i))
        y01(i) = y0(i) + (h/6.0D0)*(K1(i+nelm)+4.0D0*K2(i+nelm)+K3(i+nelm))
        z01(i) = z0(i) + (h/6.0D0)*(K1(i+nelm+nelm)+4.0D0*K2(i+nelm+nelm)+K3(i+nelm+nelm))
    END FORALL
!---------------------------------------------------------------------------------------------------
    x0 = x01
    y0 = y01
    z0 = z01
```

```fortran
 !----------------------------------------------------------------------------------------------------
   END SUBROUTINE RMK3
!====================================================================================================
!====================================================================================================

    SUBROUTINE ABM4tem(x0, y0, z0, &
                    & Vel1x1, Vel1y1, Vel1z1, &
                    & Vel1x2, Vel1y2, Vel1z2, &
                    & Vel1x3, Vel1y3, Vel1z3, &
                    & Vel1x4, Vel1y4, Vel1z4, nelm, h)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
    USE Mod_SharedVars , ONLY:DBL !, vnx0, vny0, vnz0, farel
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: nelm                                             !number of elements
    REAL (KIND = DBL), INTENT(IN) :: h                                      !step of time
!    REAL (KIND = DBL), INTENT(IN) :: tr                                     !factor to mix eulerian and lagrangian

    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0, y0, z0  !arrays of the components of
                                                                            !position vectors
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) ::   Vel1x1, Vel1y1, Vel1z1, &
                                                                & Vel1x2, Vel1y2, Vel1z2, &
                                                                & Vel1x3, Vel1y3, Vel1z3, &
                                                                & Vel1x4, Vel1y4, Vel1z4
!arrays of the components of
                                                                            !position vectors
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                                    !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01        !Dummy array
!====================================================================================================
!
    ALLOCATE(x01(nelm),y01(nelm),z01(nelm))
    x01 = 0.0D0
    y01 = 0.0D0
    z01 = 0.0D0
!----------------------------------------------------------------------------------------------------
    FORALL (i=1:nelm)
    x01(i) = x0(i)+ (h/24.0D0)*(-9.0D0*Vel1x1(i) +37.0D0*Vel1x2(i) -59.0D0*Vel1x3(i) +55.0D0*Vel1x4(i))
    y01(i) = y0(i)+ (h/24.0D0)*(-9.0D0*Vel1y1(i) +37.0D0*Vel1y2(i) -59.0D0*Vel1y3(i) +55.0D0*Vel1y4(i))
    z01(i) = z0(i)+ (h/24.0D0)*(-9.0D0*Vel1z1(i) +37.0D0*Vel1z2(i) -59.0D0*Vel1z3(i) +55.0D0*Vel1z4(i))
    END FORALL
!----------------------------------------------------------------------------------------------------
    x0 = x01
    y0 = y01
    z0 = z01
    END SUBROUTINE ABM4tem
!====================================================================================================
    SUBROUTINE ABM4(x0, y0, z0, &
                    & Vel1x1, Vel1y1, Vel1z1, &
                    & Vel1x2, Vel1y2, Vel1z2, &
                    & Vel1x3, Vel1y3, Vel1z3, &
                    & Vel1x4, Vel1y4, Vel1z4, nelm, h)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
    USE Mod_SharedVars , ONLY:DBL !, vnx0, vny0, vnz0, farel
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: nelm                                             !number of elements
    REAL (KIND = DBL), INTENT(IN) :: h                                      !step of time
!    REAL (KIND = DBL), INTENT(IN) :: tr                                     !factor to mix eulerian and lagrangian
```



```fortran
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0, y0, z0  !arrays of the components of
                                                                            !position vectors
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) ::   Vel1x1, Vel1y1, Vel1z1, &
                                                                & Vel1x2, Vel1y2, Vel1z2, &
                                                                & Vel1x3, Vel1y3, Vel1z3, &
                                                                & Vel1x4, Vel1y4, Vel1z4
!arrays of the components of
                                                                            !position vectors
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                                    !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01        !Dummy array
!====================================================================================================
!
    ALLOCATE(x01(nelm),y01(nelm),z01(nelm))
    x01 = 0.0D0
    y01 = 0.0D0
    z01 = 0.0D0
!----------------------------------------------------------------------------------------------------
    FORALL (i=1:nelm)
    x01(i) = x0(i)+ (h/24.0D0)*(Vel1x1(i) -5.0D0*Vel1x2(i) +19.0D0*Vel1x3(i) +9.0D0*Vel1x4(i))
    y01(i) = y0(i)+ (h/24.0D0)*(Vel1y1(i) -5.0D0*Vel1y2(i) +19.0D0*Vel1y3(i) +9.0D0*Vel1y4(i))
    z01(i) = z0(i)+ (h/24.0D0)*(Vel1z1(i) -5.0D0*Vel1z2(i) +19.0D0*Vel1z3(i) +9.0D0*Vel1z4(i))
    END FORALL
!----------------------------------------------------------------------------------------------------
    x0 = x01
    y0 = y01
    z0 = z01
    END SUBROUTINE ABM4
!====================================================================================================
    SUBROUTINE ABM4vel(Vel1x, Vel1y, Vel1z, &
                    &   nelm,     h,      tr)
!====================================================================================================
!   This subroutine computes the normal vector of every node on the surface
!====================================================================================================
!====================================================================================================
    USE Mod_SharedVars , ONLY:DBL , vnx0, vny0, vnz0 !, farel
!====================================================================================================
    IMPLICIT NONE
!====================================================================================================
!   Variables
!----------------------------------------------------------------------------------------------------
    INTEGER, INTENT(IN) :: nelm                                             !number of elements
    REAL (KIND = DBL), INTENT(IN) :: h                                      !step of time
    REAL (KIND = DBL), INTENT(IN) :: tr                                     !factor to mix eulerian and lagrangian

    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) ::   Vel1x, Vel1y, Vel1z
!====================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                                    !Counters
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01, f0        !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe, fye, fze     !Dummy array
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxl, fyl, fzl     !Dummy array
!====================================================================================================
    ALLOCATE(x01(nelm),y01(nelm),z01(nelm))
    x01 = 0.0D0
    y01 = 0.0D0
    z01 = 0.0D0
!====================================================================================================
   !Initialize
    ALLOCATE(f0(nelm),fxe(nelm),fye(nelm),fze(nelm),fxl(nelm),fyl(nelm),fzl(nelm))
    f0  = 0.0D0
    fxe = 0.0D0
    fye = 0.0D0
    fze = 0.0D0
    fxl = 0.0D0
    fyl = 0.0D0
    fzl = 0.0D0
!====================================================================================================
!   EULER METHOD
```

```
!================================================================================================================
    FORALL (i = 1:nelm)
        f0(i)= (Vel1x(i)*vnx0(i) + Vel1y(i)*vny0(i) + Vel1z(i)*vnz0(i))
    END FORALL
!================================================================================================================
    FORALL (i = 1:nelm)
        fxe(i) = vnx0(i)*f0(i)
        fye(i) = vny0(i)*f0(i)
        fze(i) = vnz0(i)*f0(i)
    END FORALL
!================================================================================================================
!   LAGRANGE METHOD
!================================================================================================================
    FORALL (i = 1:nelm)
        fxl(i) = Vel1x(i)
        fyl(i) = Vel1y(i)
        fzl(i) = Vel1z(i)
    END FORALL
!================================================================================================================
    x01= (tr*fxl) + (1.0D0-tr)*fxe
    y01= (tr*fyl) + (1.0D0-tr)*fye
    z01= (tr*fzl) + (1.0D0-tr)*fze
!================================================================================================================
    Vel1x = x01
    Vel1y = y01
    Vel1z = z01
!================================================================================================================
    END SUBROUTINE ABM4vel

END MODULE Mod_SNEDOS
```

```
MODULE Mod_Correction
!==============================================================================================================
!Version 1.0            27 / November / 2013
!                                                                     Alfredo Sanjuan Sanjuan, in others words, me :)
!==============================================================================================================
!   The first action is to generate a correction of drop'e volume.
!==============================================================================================================
CONTAINS
!==============================================================================================================
    SUBROUTINE Corrector_vol(p, npts, prt_vlm_n, cx, cy, cz)
!==============================================================================================================
!   This subroutine
!==============================================================================================================
    USE Mod_SharedVars , ONLY:DBL
!==============================================================================================================
    IMPLICIT NONE
!==============================================================================================================
!   Variables
!--------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: p  !The matrix with the information of
    INTEGER, INTENT(IN) :: npts                                         !number of nodes
    REAL (KIND = DBL), INTENT(IN)  :: prt_vlm_n
    REAL (KIND = DBL), INTENT(IN) :: cx,   cy,   cz                    !drop's centroid coordinates
!==============================================================================================================
!   Variables inside the subroutine
!--------------------------------------------------------------------------------------------------------------
    INTEGER ::   i                                         !Counters
!--------------------------------------------------------------------------------------------------------------
   REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: p0, p1 !Dummies array to adjust the position vectors.
   REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: r0, r1 !Dummies array to adjust the position vectors.
!==============================================================================================================
!   The adjust beggins here
    ALLOCATE(p0(npts,3), p1(npts,3),r0(npts), r1(npts))
    p0 = 0.0D0
    p1 = 0.0D0
    r0 = 0.0D0
    r1 = 0.0D0
!--------------------------------------------------------------------------------------------------------------
    FORALL (i=1:npts)
        p0(i,1)=  (p(i,1) - cx)
        p0(i,2)=  (p(i,2) - cy)
        p0(i,3)=  (p(i,3) - cz)
    END FORALL
    FORALL (i=1:npts)
        r0(i)= DSQRT( p0(i,1)**2 + p0(i,2)**2 + p0(i,3)**2 )
    END FORALL
    FORALL (i=1:npts)
        !r1(i)= r0(i) / ((prt_vlm_n)**(1.0D0/3.0D0))
        p1(i,:)= p0(i,:) / ((prt_vlm_n)**(1.0D0/3.0D0))
    END FORALL
    !FORALL (i=1:npts)
    !    p1(i,:)= p0(i,:)*r1(i)
    !END FORALL
    !FORALL (i=1:npts)
    !    p(i,1)= (p1(i,1))/((prt_vlm_n)**(1.0D0/3.0D0)) !+cx
    !    p(i,2)= (p1(i,2))/((prt_vlm_n)**(1.0D0/3.0D0)) !+cy
    !    p(i,3)= (p1(i,3))/((prt_vlm_n)**(1.0D0/3.0D0)) !+cz
    !END FORALL
    !p0=0.0D0
    !FORALL (i=1:npts)
    !    p0(i,1)=  p(i,1) +cx
    !    p0(i,2)=  p(i,2) +cy
    !    p0(i,3)=  p(i,3) +cz
    !END FORALL
        p=p1
!==============================================================================================================
    END SUBROUTINE Corrector_vol
!==============================================================================================================
    SUBROUTINE Corrector_vol2(vel1, nelm)
!==============================================================================================================
!   This subroutine
!==============================================================================================================
```



```
    USE Mod_SharedVars , ONLY:DBL, vnx0, vny0, vnz0, arel
!==============================================================================================================
    IMPLICIT NONE
!==============================================================================================================
!   Variables
!--------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: vel1  !The matrix with the information of
    INTEGER, INTENT(IN) :: nelm                                          !number of nodes
    !REAL (KIND = DBL), INTENT(IN)  :: prt_vlm_n
    !REAL (KIND = DBL), INTENT(IN) :: cx,   cy,   cz                     !drop's centroid coordinates
!==============================================================================================================
!   Variables inside the subroutine
!--------------------------------------------------------------------------------------------------------------
    INTEGER ::   i                                         !Counters
!--------------------------------------------------------------------------------------------------------------
   REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: vel, val !Dummies array to adjust the position vectors.
   REAL (KIND = DBL) :: areas, surface  !Dummies array to adjust the position vectors.

!==============================================================================================================
!   The adjust beggins here
    ALLOCATE(val(nelm), vel(3*nelm))
   vel = 0.0D0
   val = 0.0D0
   surface = 0.0D0
   areas = 0.0D0
!--------------------------------------------------------------------------------------------------------------
    FORALL (i=1:nelm)
        val(i)=  Vel1(i)*vnx0(i) + Vel1(i+nelm)*vny0(i) + Vel1(i+nelm+nelm)*vnz0(i)
    END FORALL
    FORALL (i=1:nelm)
        vel(i)          =  val(i)*vnx0(i)
        vel(i+nelm)     =  val(i)*vny0(i)
        vel(i+nelm+nelm)=  val(i)*vnz0(i)
    END FORALL

    areas = SUM(arel)
    surface = SUM(val)

    WRITE(*,*) areas
    WRITE(*,*) surface
    WRITE(*,*) surface/areas
    WRITE(*,*)

    !FORALL (i=1:nelm)
    !    Vel1(i)=  Vel(i)
    !END FORALL
!==============================================================================================================
    END SUBROUTINE Corrector_vol2
!==============================================================================================================
     SUBROUTINE D_taylor(p, npts, DT, DTm, DTmm, lmajor, bminor, wminor, angulo, cx, cy, cz, req)
!==============================================================================================================
!   This subroutine
!==============================================================================================================
    USE Mod_SharedVars , ONLY:DBL, pi
!==============================================================================================================
    IMPLICIT NONE
!==============================================================================================================
!   Variables
!--------------------------------------------------------------------------------------------------------------
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: p  !The matrix with the information of
    INTEGER, INTENT(IN) :: npts                                      !number of nodes
    REAL (KIND = DBL), INTENT(IN) :: cx,   cy,   cz,  req                     !drop's centroid coordinates and r0
    REAL (KIND = DBL), INTENT(OUT)  :: DT, DTm, DTmm, lmajor, bminor, wminor, angulo
!==============================================================================================================
!   Variables inside the subroutine
!--------------------------------------------------------------------------------------------------------------
    INTEGER ::   i, j                                        !Counters
!--------------------------------------------------------------------------------------------------------------
   REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: xp !Dummies array to adjust the position vectors.
!==============================================================================================================
!   The adjust beggins here
    ALLOCATE(xp(npts,3))
```

```fortran
    xp = 0.0D0
    lmajor = 0.0D0
    bminor = 0.0D0
    wminor = 0.0D0
    angulo = 0.0D0
    DT  = 0.0D0
    DTm = 0.0D0
!------------------------------------------------------------------------------------------------------------
    FORALL (i=1:npts)
        xp(i,1) =  DSQRT( (p(i,1)-cx)**2 + (p(i,2)-cy)**2 + (p(i,3)-cz)**2)
        xp(i,2) =  DSQRT(p(i,3)-cz)**2
        xp(i,3) =  DATAN((p(i,2)-cy)/(p(i,1)-cx))
    END FORALL

    lmajor  = (MAXVAL(xp(:,1)))/req
    bminor  = (MINVAL(xp(:,1)))/req
    wminor  = (MAXVAL(xp(:,2)))/req

    angulo  = (xp(MAXLOC(xp(:,1),1),3)*180.0D0)/pi
    DT      = ((lmajor - bminor)/(lmajor + bminor))
    DTm     = ((lmajor - wminor)/(lmajor + wminor))
    DTmm    = ((wminor - bminor)/(wminor + bminor))
!============================================================================================================
     END SUBROUTINE D_taylor
!============================================================================================================
END MODULE Mod_Correction
```

```
MODULE Mod_Data_Files
!==============================================================================================================
!   Version: 0.9 created on 21 / 10 / 2013
!                                                                                        Alfredo Sanjuan Sanjuan
!   Version: 1.0 created on 14 / 01 / 2015
!                                                                                        Alfredo Sanjuan Sanjuan
!==============================================================================================================
CONTAINS
!==============================================================================================================
    SUBROUTINE write_datan(nelm, stride)
!==============================================================================================================
!   This subroutine write the data of nodes of the mesh in a file
!   This subroutine applies to every step of time.
!==============================================================================================================
    USE Mod_SharedVars, ONLY: DBL, p, n
!==============================================================================================================
    IMPLICIT NONE
!==============================================================================================================
!   Variables
!   Variables inside the subroutine
!--------------------------------------------------------------------------------------------------------------
    INTEGER,           INTENT(IN)  :: nelm                  !number of points
    INTEGER,           INTENT(IN)  :: stride                !counter
!==============================================================================================================
!   Variables inside the subroutine
!--------------------------------------------------------------------------------------------------------------
    CHARACTER (len=35) :: fname
    CHARACTER (len=19) :: dname
    CHARACTER (len= 4) :: ename
    CHARACTER (len= 7) :: num
!--------------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                    !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                  !
    INTEGER  :: select                                  !Selection
    INTEGER  :: fstatus                                 !FSTATUS
!==============================================================================================================
!   First, the information of nelm and stride are transformed into character variables.
    j = 1000000 + stride
    WRITE(num,'(I7)') j
    ename = '.dat'
!--------------------------------------------------------------------------------------------------------------
!   Then, DOOM SUBROUTINE is called.
!    CALL doom(ddname)
    dname = TRIM('Data\Position\nodes')
!--------------------------------------------------------------------------------------------------------------
!   The name of file is made.
!--------------------------------------------------------------------------------------------------------------
    fname = TRIM(TRIM(dname)//TRIM(num)//ename)

    OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)

    IF (fstatus == 0) THEN
        WRITE(500,*) nelm
        WRITE(500,*) j

        DO i = 1, nelm
            i1 = n(i,1)
            i2 = n(i,2)
            i3 = n(i,3)
            i4 = n(i,4)
            i5 = n(i,5)
            i6 = n(i,6)
            WRITE (500,103) p(i1,1),p(i1,2),p(i1,3)
            WRITE (500,103) p(i2,1),p(i2,2),p(i2,3)
            WRITE (500,103) p(i3,1),p(i3,2),p(i3,3)
            WRITE (500,103) p(i4,1),p(i4,2),p(i4,3)
            WRITE (500,103) p(i5,1),p(i5,2),p(i5,3)
            WRITE (500,103) p(i6,1),p(i6,2),p(i6,3)
        END DO
    END IF

!--------------------------------------------------------------------------------------------------------------
```



```
    CLOSE(UNIT = 500)
!--------------------------------------------------------------------------------------------------------------
    103 FORMAT(10(1x,ES24.16))
!==============================================================================================================
    END SUBROUTINE write_datan
!==============================================================================================================
!==============================================================================================================
    SUBROUTINE write_datac(nelm, stride)
!==============================================================================================================
!   This subroutine write the data of collocation points if the mesh in a file
!   This subroutine applies to every step of time.
!==============================================================================================================
    USE Mod_SharedVars, ONLY: DBL, x0, y0, z0
!==============================================================================================================
    IMPLICIT NONE
!==============================================================================================================
!   Variables
!   Variables inside the subroutine
!--------------------------------------------------------------------------------------------------------------
    INTEGER,           INTENT(IN)  :: nelm                  !number of points
    INTEGER,           INTENT(IN)  :: stride                !counter
!==============================================================================================================
!   Variables inside the subroutine
!--------------------------------------------------------------------------------------------------------------
    CHARACTER (len=35) :: fname
    CHARACTER (len=19) :: dname
    CHARACTER (len= 4) :: ename
    CHARACTER (len= 7) :: num
!--------------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                    !Counters
    INTEGER  :: i1, i2, i3, i4, i5, i6                  !
    INTEGER  :: select                                  !Selection
    INTEGER  :: fstatus                                 !FSTATUS
!==============================================================================================================
!   First, the information of nelm and stride are transformed into character variables.
    j = 1000000 + stride
    WRITE(num,'(I7)') j
    ename = '.dat'
!--------------------------------------------------------------------------------------------------------------
    dname = TRIM('Data\Positioc\collc')
!--------------------------------------------------------------------------------------------------------------
!   The name of file is made.
!--------------------------------------------------------------------------------------------------------------
    fname = TRIM(TRIM(dname)//TRIM(num)//ename)

    OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)

    IF (fstatus == 0) THEN
        WRITE(500,*) nelm
        WRITE(500,*) j
!--------------------------------------------------------------------------------------------------------------
        DO i = 1, nelm
            WRITE (500,103)  x0(i), y0(i), z0(i)
        END DO
    END IF
!--------------------------------------------------------------------------------------------------------------
    CLOSE(UNIT = 500)
!--------------------------------------------------------------------------------------------------------------
    103 FORMAT(10(1x,ES24.16))
!==============================================================================================================
    END SUBROUTINE write_datac
!==============================================================================================================
!==============================================================================================================
    SUBROUTINE write_datak(nelm, stride)
!==============================================================================================================
!   This subroutine write the data of curvature in a file
!   This subroutine applies to every step of time.
!==============================================================================================================
    USE Mod_SharedVars, ONLY: DBL, crvmel
!==============================================================================================================
    IMPLICIT NONE
!==============================================================================================================
```

```fortran
!   Variables
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------------
    INTEGER,           INTENT(IN)  :: nelm                  !number of elements
    INTEGER,           INTENT(IN)  :: stride                !counter
!==========================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------------
    CHARACTER (len=35) :: fname
    CHARACTER (len=15) :: dname
    CHARACTER (len= 4) :: ename
    CHARACTER (len= 7) :: num
!----------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                 !Counters
    INTEGER  :: select                               !Selection
    INTEGER  :: fstatus                              !FSTATUS
!==========================================================================================================
!   First, the information of nelm and stride are transformed into character variables.
    j = 1000000 + stride
    WRITE(num,'(I7)') j
    ename = '.dat'
!----------------------------------------------------------------------------------------------------------
!   Then, DOOM SUBROUTINE is called.
    dname = TRIM('Data\Curv\nelm')
!----------------------------------------------------------------------------------------------------------
!   The name of file is made.
!----------------------------------------------------------------------------------------------------------
    fname = TRIM(TRIM(dname)//TRIM(num)//ename)

    OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)

    IF (fstatus == 0) THEN
        WRITE(500,*) nelm
        WRITE(500,*) j
!----------------------------------------------------------------------------------------------------------
        DO i = 1, nelm
            WRITE (500,*) crvmel(i)
        END DO
    END IF
!----------------------------------------------------------------------------------------------------------
    CLOSE(UNIT = 500)
!----------------------------------------------------------------------------------------------------------
    103 FORMAT(10(1x,ES24.16))
!==========================================================================================================
    END SUBROUTINE write_datak
!==========================================================================================================
!==========================================================================================================
    SUBROUTINE write_datar(nelm, stride)
!==========================================================================================================
!   This subroutine write the data of element area in a file
!   This subroutine applies to every step of time.
!==========================================================================================================
    USE Mod_SharedVars, ONLY: DBL, arel
!==========================================================================================================
    IMPLICIT NONE
!==========================================================================================================
!   Variables
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------------
    INTEGER,           INTENT(IN)  :: nelm                  !number of elements
    INTEGER,           INTENT(IN)  :: stride                !counter
!==========================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------------
    CHARACTER (len=35) :: fname
    CHARACTER (len=15) :: dname
    CHARACTER (len= 4) :: ename
    CHARACTER (len= 7) :: num
!----------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                 !Counters
!    INTEGER  :: select                               !Selection
    INTEGER  :: fstatus                              !FSTATUS
```



```fortran
!==========================================================================================================
!   First, the information of nelm and stride are transformed into character variables.
    j = 1000000 + stride
    WRITE(num,'(I7)') j
    ename = '.dat'
!----------------------------------------------------------------------------------------------------------
!   Then, DOOM SUBROUTINE is called.
    dname = TRIM('Data\Area\nelm')
!----------------------------------------------------------------------------------------------------------
!   The name of file is made.
!----------------------------------------------------------------------------------------------------------
    fname = TRIM(TRIM(dname)//TRIM(num)//ename)

    OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)

    IF (fstatus == 0) THEN
        WRITE(500,*) nelm
        WRITE(500,*) j
!----------------------------------------------------------------------------------------------------------
        DO i = 1, nelm
            WRITE (500,*) arel(i)
        END DO
    END IF
!----------------------------------------------------------------------------------------------------------
    CLOSE(UNIT = 500)
!----------------------------------------------------------------------------------------------------------
    103 FORMAT(10(1x,ES24.16))
!==========================================================================================================
    END SUBROUTINE write_datar
!==========================================================================================================
!==========================================================================================================
    SUBROUTINE write_datavl(nelm, stride, Vel1x, Vel1y, Vel1z)
!==========================================================================================================
!   This subroutine write the data of velocity on collocation points in a file
!   This subroutine applies to every step of time.
!==========================================================================================================
    USE Mod_SharedVars, ONLY: DBL
!==========================================================================================================
    IMPLICIT NONE
!==========================================================================================================
!   Variables
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------------
    INTEGER,           INTENT(IN)  :: nelm                  !number of elements
    INTEGER,           INTENT(IN)  :: stride                !counter
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: Vel1x, Vel1y, Vel1z !arrays of the components
                                                            ! of normal vectors over collocation points
!==========================================================================================================
!   Variables inside the subroutine
!----------------------------------------------------------------------------------------------------------
    CHARACTER (len=35) :: fname
    CHARACTER (len=15) :: dname
    CHARACTER (len= 4) :: ename
    CHARACTER (len= 7) :: num
!----------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                 !Counters
!    INTEGER  :: select                               !Selection
    INTEGER  :: fstatus                              !FSTATUS
!==========================================================================================================
!   First, the information of nelm and stride are transformed into character variables.
    j = 1000000 + stride
    WRITE(num,'(I7)') j
    ename = '.dat'
!----------------------------------------------------------------------------------------------------------
!   Then, DOOM SUBROUTINE is called.
    dname = TRIM('Data\Velo\city')
!----------------------------------------------------------------------------------------------------------
!   The name of file is made.
!----------------------------------------------------------------------------------------------------------
    fname = TRIM(TRIM(dname)//TRIM(num)//ename)

    OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
```

```fortran

    IF (fstatus == 0) THEN
        WRITE(500,*) nelm
        WRITE(500,*) j
!---------------------------------------------------------------------------------------------------------------
        DO i = 1, nelm
            WRITE (500,103)  Vel1x(i), Vel1y(i), Vel1z(i)
        END DO
    END IF
!---------------------------------------------------------------------------------------------------------------
    CLOSE(UNIT = 500)
!---------------------------------------------------------------------------------------------------------------
    103 FORMAT(10(1x,ES24.16))
!===============================================================================================================
    END SUBROUTINE write_datavl
!===============================================================================================================
!===============================================================================================================
    SUBROUTINE write_datakFG(nelm, stride, GM)
!===============================================================================================================
!   This subroutine write the data of curvature in a file
!   This subroutine applies to every step of time.
!===============================================================================================================
    USE Mod_SharedVars, ONLY: DBL, crvmel
!===============================================================================================================
    IMPLICIT NONE
!===============================================================================================================
!   Variables
!   Variables inside the subroutine
!---------------------------------------------------------------------------------------------------------------
    INTEGER,           INTENT(IN)  :: nelm                  !number of elements
    INTEGER,           INTENT(IN)  :: stride                !counter
    REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: GM
 !===============================================================================================================
!   Variables inside the subroutine
!---------------------------------------------------------------------------------------------------------------
    CHARACTER (len=35) :: fname
    CHARACTER (len=15) :: dname
    CHARACTER (len= 4) :: ename
    CHARACTER (len= 7) :: num
!---------------------------------------------------------------------------------------------------------------
    INTEGER  :: i, j                                !Counters
!    INTEGER  :: select                              !Selection
    INTEGER  :: fstatus                             !FSTATUS
!===============================================================================================================
!   First, the information of nelm and stride are transformed into character variables.
    j = 1000000 + stride
    WRITE(num,'(I7)') j
    ename = '.dat'
!---------------------------------------------------------------------------------------------------------------
!   Then, DOOM SUBROUTINE is called.
    dname = TRIM('Data\Curv\nelm')
!---------------------------------------------------------------------------------------------------------------
!   The name of file is made.
!---------------------------------------------------------------------------------------------------------------
    fname = TRIM(TRIM(dname)//TRIM(num)//ename)

    OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)

    IF (fstatus == 0) THEN
        WRITE(500,*) nelm
        WRITE(500,*) j
!---------------------------------------------------------------------------------------------------------------
        DO i = 1, nelm
            WRITE (500,103) GM(i), GM(i+nelm), GM(i+nelm+nelm), crvmel(i)
        END DO
    END IF
!---------------------------------------------------------------------------------------------------------------
    CLOSE(UNIT = 500)
!---------------------------------------------------------------------------------------------------------------
    103 FORMAT(10(1x,ES24.16))
!===============================================================================================================
    END SUBROUTINE write_datakFG
```



```fortran
!===============================================================================================================
!===============================================================================================================
!   SUBROUTINE doom(ddname)
!!==============================================================================================================
!!    This subroutine write the data in a file
!!    This subroutine applies to velocity and position of collocation points.
!!==============================================================================================================
!    USE Mod_SharedVars, ONLY: DBL, doomsday
!!==============================================================================================================
!    IMPLICIT NONE
!!==============================================================================================================
!!    Variables
!!    Variables inside the subroutine
!!--------------------------------------------------------------------------------------------------------------
!    CHARACTER (len=52),INTENT(INOUT) :: ddname                !semiroot
!!==============================================================================================================
!!   Select subroutine and there are three options
!!    1.- Alfredo PC
!!    2.- Cahn-Hilliard and Ladyzhenskaya Test_experiments
!!    3.- Cahn-Hilliard and Ladyzhenskaya second and third programs.
!!    Default.- Alfredo PC
!!==============================================================================================================

!!    The choice is made here padawan.
!!--------------------------------------------------------------------------------------------------------------
!    SELECT CASE(doomsday)
!!--------------------------------------------------------------------------------------------------------------
!    CASE (1)
!!--------------------------------------------------------------------------------------------------------------
!!     This option is for your PC
!    ddname = 'C:\Users\Alfredo\Desktop\prtcl_mkl\Test_experiments\'
!!--------------------------------------------------------------------------------------------------------------
!    CASE (2)
!!--------------------------------------------------------------------------------------------------------------
!!     This option is for Ladyzhenskaya
!    ddname = 'C:\Users\asanjuans\Desktop\Test_experiments\'
!!--------------------------------------------------------------------------------------------------------------
!    CASE (3)
!!--------------------------------------------------------------------------------------------------------------
!!     This option is for Ladyzhenskaya II
!    ddname = 'C:\Users\asanjuans\Desktop\Test_experimen_2\'
!!--------------------------------------------------------------------------------------------------------------
!    CASE (4)
!!--------------------------------------------------------------------------------------------------------------
!!     This option is for Ladyzhenskaya III
!    ddname = 'C:\Users\asanjuans\Desktop\Test_experimen_3\'
!!--------------------------------------------------------------------------------------------------------------
!    CASE DEFAULT
!!--------------------------------------------------------------------------------------------------------------
!!     This expretion is for your PC
!    ddname = 'C:\Users\asanjuans\Desktop\prtcl_mkl\Test_experiments\'
!    END SELECT
!!--------------------------------------------------------------------------------------------------------------
!    ddname = TRIM(ddname)
!!--------------------------------------------------------------------------------------------------------------
!   END SUBROUTINE doom
!!==============================================================================================================
!!==============================================================================================================
!    SUBROUTINE geometryprint(npts, stride)
!!==============================================================================================================
!!    This subroutine write the data of curvature in a file
!!    This subroutine applies to every step of time.
!!==============================================================================================================
!    USE Mod_SharedVars, ONLY: DBL, p
!!==============================================================================================================
!    IMPLICIT NONE
!!==============================================================================================================
!!    Variables
!!    Variables inside the subroutine
!!--------------------------------------------------------------------------------------------------------------
!    INTEGER,           INTENT(IN)  :: npts                  !number of points
!    INTEGER,           INTENT(IN)  :: stride                !counter
```

```
!!=================================================================================================
!!   Variables inside the subroutine
!!-------------------------------------------------------------------------------------------------
!    CHARACTER (len=25) :: fnum
!    CHARACTER (len=15) :: numch
!    CHARACTER (len=95) :: fname
!    CHARACTER (len=67) :: dname
!    CHARACTER (len=52) :: ddname
!    CHARACTER (len= 4) :: ename
!    CHARACTER (len= 5) :: num
!!-------------------------------------------------------------------------------------------------
!    INTEGER  :: i, j                                  !Counters
!    INTEGER  :: select                                !Selection
!    INTEGER  :: fstatus                               !FSTATUS
!!=================================================================================================
!!   First, the information of nelm and stride are transformed into character variables.
!    WRITE(numch,*) npts
!    fnum = TRIM('(' // TRIM(numch) // 'ES25.17)')
!    j = 10000 + stride
!    WRITE(num,'(I5)') j
!    ename = '.dat'
!!-------------------------------------------------------------------------------------------------
!!   Then, DOOM SUBROUTINE is called.
!    CALL doom(ddname)
!    dname = TRIM(TRIM(ddname)//'geometry\p')
!    dname = TRIM(dname)
!!-------------------------------------------------------------------------------------------------
!!   The name of file is made.
!!-------------------------------------------------------------------------------------------------
!    fname = TRIM(TRIM(dname)//TRIM(num)//ename)
!
!    OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
!
!    IF (fstatus == 0) THEN
!        WRITE(500,*) npts
!        WRITE(500,*) j
!!-------------------------------------------------------------------------------------------------
!        DO i = 1, npts
!            WRITE (500,103) p(i,1),p(i,2),p(i,3)
!        END DO
!    END IF
!!-------------------------------------------------------------------------------------------------
!    CLOSE(UNIT = 500)
!!-------------------------------------------------------------------------------------------------
!    103 FORMAT(10(1x,ES24.16))
!!=================================================================================================
!    END SUBROUTINE geometryprint
!!=================================================================================================
!!=================================================================================================
!    SUBROUTINE uvaprint(nelm, stride)
!!=================================================================================================
!!   This subroutine write the data of curvature in a file
!!   This subroutine applies to every step of time.
!!=================================================================================================
!    USE Mod_SharedVars, ONLY: DBL, farel
!!=================================================================================================
!    IMPLICIT NONE
!!=================================================================================================
!!   Variables
!!   Variables inside the subroutine
!!-------------------------------------------------------------------------------------------------
!    INTEGER,           INTENT(IN)  :: nelm                  !number of elements
!    INTEGER,           INTENT(IN)  :: stride                !counter
!!=================================================================================================
!!   Variables inside the subroutine
!!-------------------------------------------------------------------------------------------------
!    CHARACTER (len=25) :: fnum
!    CHARACTER (len=15) :: numch
!    CHARACTER (len=95) :: fname
!    CHARACTER (len=67) :: dname
!    CHARACTER (len=52) :: ddname
!    CHARACTER (len= 4) :: ename
```



```
!    CHARACTER (len= 5) :: num
!!-------------------------------------------------------------------------------------------------
!    INTEGER  :: i, j                                  !Counters
!    INTEGER  :: select                                !Selection
!    INTEGER  :: fstatus                               !FSTATUS
!!=================================================================================================
!!   First, the information of nelm and stride are transformed into character variables.
!    WRITE(numch,*) nelm
!    fnum = TRIM('(' // TRIM(numch) // 'ES25.17)')
!    j = 10000 + stride
!    WRITE(num,'(I5)') j
!    ename = '.dat'
!!-------------------------------------------------------------------------------------------------
!!   Then, DOOM SUBROUTINE is called.
!    CALL doom(ddname)
!    dname = TRIM(TRIM(ddname)//'nelm\uva')
!    dname = TRIM(dname)
!!-------------------------------------------------------------------------------------------------
!!   The name of file is made.
!!-------------------------------------------------------------------------------------------------
!    fname = TRIM(TRIM(dname)//TRIM(num)//ename)
!
!    OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
!
!    IF (fstatus == 0) THEN
!        WRITE(500,*) nelm
!        WRITE(500,*) j
!!-------------------------------------------------------------------------------------------------
!        DO i = 1, nelm
!            WRITE (500,*) farel(i)
!            !!WRITE (8,103) arel(i)
!        END DO
!    END IF
!!-------------------------------------------------------------------------------------------------
!    CLOSE(UNIT = 500)
!!-------------------------------------------------------------------------------------------------
!    103 FORMAT(10(1x,ES24.16))
!!=================================================================================================
!    END SUBROUTINE uvaprint
!!=================================================================================================
!================================================================================================
END MODULE Mod_Data_Files
```