



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



Universidad Nacional  
Autónoma de México

Dirección General de Bibliotecas de la UNAM

**Biblioteca Central**



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.





---

**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 than a micron are involved. Also, which may become most relevant, is the capture and sequestration of CO<sub>2</sub>—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 than 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



Universidad Nacional  
Autónoma de México

Dirección General de Bibliotecas de la UNAM

**Biblioteca Central**



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

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***

<b>ACKNOWLEDGMENTS .....</b>	<b>I</b>
<b>ABSTRACT .....</b>	<b>III</b>
<b>TABLE OF CONTENTS .....</b>	<b>V</b>
<b>LIST OF ILLUSTRATIONS .....</b>	<b>VII</b>
<b>CHAPTER 1. STUDY OF THE DEFORMATION OF A DROP .....</b>	<b>1</b>
<b>1.1 Background .....</b>	<b>6</b>
<b>1.2 Numerical method .....</b>	<b>10</b>
1.2.1 Numerical implementation of the computation of drop deformation in strong flows .....	13
1.2.2 Performance of the mesh used in the method in the time .....	14
1.2.3 Oriented parallel programming in the numerical scheme .....	17
1.2.4 Curvature in BEM 3D .....	18
1.2.5 Numerical Accuracy .....	21
<b>CHAPTER 2. CALIBRATION OF NUMERICAL METHOD .....</b>	<b>23</b>
<b>2.1 The numerical method vs previous numerical methods .....</b>	<b>24</b>
<b>2.2 Experiments in simple shear flow compared with experimental data .....</b>	<b>27</b>
<b>2.3 Comparisons of numerical vs. experimental data for extensional flow .....</b>	<b>30</b>
<b>CHAPTER 3. SHAPE OF A DROP IMMERSSED IN A FLUID UNDER AN ELONGATIONAL FLOW WITH VORTICITY; SMALL RATIOS OF VISCOSITY .....</b>	<b>35</b>
<b>3.1 Time evolution of drop deformation .....</b>	<b>36</b>
<b>3.2 Attained stationary states .....</b>	<b>36</b>
<b>CHAPTER 4. SHAPE OF A DROP IMMERSSED IN A FLUID UNDER AN ELONGATIONAL FLOW WITH VORTICITY; LARGE RATIOS OF VISCOSITY .....</b>	<b>45</b>
<b>4.1 Time evolution of drop deformation .....</b>	<b>46</b>



Universidad Nacional  
Autónoma de México

Dirección General de Bibliotecas de la UNAM

**Biblioteca Central**



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

4.2 Stationary states attained .....	47
--------------------------------------	----

<b>CHAPTER 5. CHARACTERISTIC TIMES OF DROP DEFORMATION UNDER AN ELONGATIONAL FLOW WITH VORTICITY .....</b>	<b>59</b>
--	-----------

5.1 Characteristic times obtained in drop deformation in an impulsive flow applied with small viscosity ratio.....	60
--	----

5.2 Characteristic time in the retraction of a drop.....	68
--	----

<b>CHAPTER 6. SHEAR RATE VS. CAPILLARY NUMBER. EFFECT ON THE STEADY FLOW FORM OF DROPS.....</b>	<b>87</b>
---	-----------

6.1 Consequences of shear rate $G$ in the stationary state of the drop for small value of rate of viscosity .....	88
---	----

6.2 Consequences of shear rate $G$ on the stationary state of the drop for large capillary number values.....	98
---	----

<b>CHAPTER 7. THE FORM OF A DROP IMMERSSED IN AN EXTENSIONAL FLOW... 111</b>
--

7.1 Drop deformation in uniaxial flow .....	112
---	-----

7.2 Drop deformation in extensional 2-D flow .....	120
--	-----

<b>CHAPTER 8. HYSTERESIS BEHAVIOR IN DROP DEFORMATION .....</b>	<b>123</b>
---	------------

8.1 Hysteresis in strong flows.....	125
-------------------------------------	-----

8.2 Hysteresis in extensional flows.....	132
--	-----

<b>CHAPTER 9. GENERAL CONCLUSIONS: DROP DEFORMATION IN STRONG FLOWS</b> 137
--

<b>BIBLIOGRAPHY .....</b>	<b>145</b>
---------------------------	------------

<b>APPENDIX A .....</b>	<b>151</b>
-------------------------	------------

<b>APPENDIX B .....</b>	<b>157</b>
-------------------------	------------

## **LIST OF ILLUSTRATIONS**

<i>Figure 1.1. Scheme of a two-phase fluid. Drop in stationary shape. <math>\lambda\mu = 0.012</math>, <math>\alpha = 0.13</math> and <math>Ca = 0.40</math>. (a) 3D-view of the drop within an applied 2D-flow shown on the <math>xy</math>-plane. (b) Conventional characterization of deformation on the <math>xy</math>-plane: <math>L</math> and <math>B</math>. (c) Cross section of the drop: <math>W</math> being the third deformation length scale. ....</i>	<i>7</i>
<i>Figure 1.2 The better known planar linear flows in 2D. Definition of weak flows and strong flows using the parameter <math>\alpha</math>.....</i>	<i>9</i>
<i>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 <math>k = 4</math>. ....</i>	<i>15</i>
<i>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. ....</i>	<i>15</i>
<i>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. ....</i>	<i>16</i>
<i>Figure 1.6 Mapping of curved triangles from 3D-space to 2D-space. ....</i>	<i>18</i>
<i>Figure 2.1 Stationary Taylor's Drop Deformation vs. Capillary Number. Simple shear flow, ... <math>\alpha = 0.0</math> and <math>\lambda\mu = 1.0</math>. ....</i>	<i>26</i>
<i>Figure 2.2 <math>B/W</math> ratio vs. Capillary Number for different <math>\lambda\mu</math> values.....</i>	<i>26</i>
<i>Figure 2.3 Principal axes of the drop vs. Capillary Number. <math>\lambda\mu = 1.4</math>, <math>\alpha = 0.0</math>.....</i>	<i>28</i>
<i>Figure 2.4 Comparison between Numerical Simulation of BEM3D and Guido's Experimental Data for a drop deformation. Simple shear flow, <math>\alpha = 0.0</math>, <math>\lambda\mu = 1.4</math>, <math>Ca = 0.46</math>. The experimental data were <math>D = 0.70</math>, and <math>\theta = 32.05^\circ</math>. The numerical values were <math>D = 0.67</math>, and <math>\theta = 34.8^\circ</math>.....</i>	<i>29</i>
<i>Figure 2.5 Deformation of the drop induced by a 2D-elongational flow. Elongation along <math>y</math>-axis, compression along <math>x</math>-axis and a neutral deformation for <math>z</math>-axis. <math>xy</math>-plane, above; <math>zy</math>-plane below. ....</i>	<i>31</i>
<i>Figure 2.6 Comparison of steady shape of deformation between experimental data vs numerical simulation of a drop in extensional 2D-flow with <math>\lambda\mu = 1.0</math>. a) <math>Ca = 0.05</math> and <math>DT = 0.08</math>; b) <math>Ca = 0.10</math> and <math>DT = 0.18</math>; and c) <math>Ca = 0.13</math>, and <math>DT = 0.31</math>. ....</i>	<i>33</i>
<i>Figure 3.1 Taylor's Deformation of the drop vs. Time. <math>\lambda\mu = 0.012</math>, <math>\alpha = 0.13</math>. ....</i>	<i>36</i>
<i>Figure 3.2 Evolution of the three principal axes of the drop in a flow with <math>\alpha = 0.13</math>, <math>Ca = 0.4</math> and total time <math>Gt = 15</math>. ....</i>	<i>37</i>
<i>Figure 3.3 Ratio of the <math>B</math> wrt. <math>W</math> axes in numerical simulations for a flow with <math>\alpha = 0.13</math> and total time <math>Gt = 15</math> for different values of <math>Ca</math>. ....</i>	<i>38</i>

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 .....	39
Figure 3.5 Deformation 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.....	40
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 .....	40
Figure 3.7 Deformation of the drop for numerical simulations for flow with $\alpha = 0.03$ , $\alpha = 0.05$ and $\alpha = 0.13$ and analytical prediction of Taylor Cox in simple shear flow $\alpha = 0.0$ for different values of Ca.....	41
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.....	42
Figure 4.1 Taylor's Deformation of drops in the time. $\lambda\mu = 16$ , $\alpha = 0.03$ .....	47
Figure 4.2 Ratio of the B wrt. W axes in numerical simulations for a flow with $\alpha=0.03$ and total time $Gt =75$ for different values of Ca.....	48
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 $Gt =75$ . .....	49
Figure 4.4 Stationary ratio of the B wrt. the W axis in numerical simulations for flow with $\alpha = 0.03$ , $\alpha = 0.05$ and $\alpha = 0.13$ for different values of Ca.....	50
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.....	51
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.....	53
Figure 4.7 Drop evolution in the time. $Ca = 0.87$ with $\lambda\mu = 16$ 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. ....	55
Figure 4.8 Polar plot of Taylor deformation vs. orientation angle for different capillary numbers with $\alpha = 0.03$ , $\lambda\mu = 16$ . .....	55
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.....	56
Figure 5.1 Time evolution of principal L axis which $\alpha = 0.13$ for different values of Ca. ....	61
Figure 5.2 Time evolution of principal B axis which $\alpha = 0.13$ for different values of Ca.....	61
Figure 5.3 Time evolution of principal W axis which $\alpha = 0.13$ for different values of Ca.....	62
Figure 5.4 Time evolution of principal L axis normalized wrt. stationary value, which $\alpha = 0.13$ for different values of Ca.....	62

Figure 5.5 Time evolution of principal B axis normalized wrt. stationary value, which $\alpha = 0.13$ for different values of Ca.....	63
Figure 5.6 Time evolution of principal W axis normalized wrt. stationary value, which $\alpha = 0.13$ for different values of Ca.....	63
Figure 5.7 Time evolution of principal axes normalized wrt. stationary value, which $\alpha=0.03$ for different values of Ca. Characteristic times in symbols. ....	64
Figure 5.8 Time evolution of principal axes normalized wrt. stationary value, which $\alpha=0.05$ for different values of Ca. Characteristic times in symbols. ....	65
Figure 5.9 Time evolution of principal axes normalized wrt. stationary value, which $\alpha=0.13$ for different values of Ca. Characteristic times in symbols. ....	66
Figure 5.10 Characteristic Times vs Capillary Number. ....	67
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.....	68
Figure 5.12 Taylor Deformation of the drop vs time. $\alpha = 0.13$ , $Ca = 0.40$ .....	71
Figure 5.13 Mo Shape Parameter vs Time. $\alpha = 0.13$ , $Ca = 0.40$ .....	72
Figure 5.14 Taylor Deformation vs Time (lines). Exponential Decay vs Time (dash-dot-dot). Characteristic time $\tau$ (diamonds). $\alpha = 0.13$ , $Ca = 0.40$ .....	73
Figure 5.15 Taylor Deformation vs Time (lines). Exponential Decay vs Time (dash-dot-dot). Characteristic time $\tau$ (diamonds). $\alpha=0.13$ , $Ca=0.40$ .....	74
Figure 5.16 Characteristic times $t$ vs Capillary Number which parameter $\alpha = 0.13$ .....	75
Figure 5.17 $\ln(DT/D0)$ vs. time. Drop retraction was analyzed with Taylor deformation which $Ca = 40$ and $\alpha = 0.13$ .....	76
Figure 5.18 Angle of Orientation vs. time in drop retraction which parameter $\alpha = 0.13$ . ....	77
Figure 5.19 Linear fit of $-\ln(DT/D0)$ for $5\tau$ of time. Drop retraction was analyzed with Taylor deformation which $Ca = 40$ and $\alpha = 0.13$ .....	78
Figure 5.20 Linear fit of $-\ln(DMo/D0)$ for $5\tau$ of time. Drop retraction was analyzed with Mo shape parameter which $Ca = 40$ and $\alpha = 0.13$ .....	78
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 $\tau$ .....	80
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 $\tau$ .....	80
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 $\tau$ .....	81
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 $\tau$ .....	82

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 $\tau$ . ...	83
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 $\tau$ . .....	84
Figure 6.1 Polar representation of Angle of Orientation vs. Taylor Deformation with $\alpha = 0.13$ and $\lambda\mu = 0.012$ . .....	88
Figure 6.2 Angle of Orientation vs. Taylor Deformation with $\alpha = 0.13$ and $\lambda\mu = 0.012$ . .....	89
Figure 6.3 Evolution of Taylor Deformation for the same capillary number with different values of $G$ . .....	90
Figure 6.4 Angle of Orientation vs Taylor Deformation for drops with the same Capillary Number with different values of $G$ .....	91
Figure 6.5 Angle of Orientation vs. Capillary Number for $Ca = 0.35$ and different values of $G$ . ...	92
Figure 6.6 Trembling effect in drop deformation for different capillary number with $G \gg 1$ , $\alpha = 0.13$ , $\lambda\mu = 0.01$ . .....	93
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 $G$ .....	95
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 $G = 3.96$	
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 (orange line). The velocity calculated using the normal and tangential component employed is observed in the blue line. $Ca = 0.04$ , $\alpha = 0.13$ and $\lambda\mu = 0.012$ . .....	97
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. .....	99
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. .....	100
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. .....	101
Figure 6.13 Comparison of the $L$ -axis when capillary number is increased as function of Parameter $G$ , with $\lambda\mu = 0.012$ , $\alpha = 0.13$ . .....	102
Figure 6.14 Comparison of the $W$ -axes when capillary number is increased as function of Parameter $G$ , which $\lambda\mu = 0.012$ , $\alpha = 0.13$ . .....	104
Figure 6.15 Comparison of the $W$ -axes when $Ca_{cr} \ll Ca$ , which $\lambda\mu = 0.012$ , $\alpha = 0.13$ . .....	104



Figure 6.16 Principal Axes evolution in the time for a drop whit $Ca_{cr} \ll Ca$ , $\lambda\mu = 0.012$ , $\alpha = 0.13$ .....	106
Figure 6.17 Taylor Deformation in different planes: $Ca = 0.40$ , $\lambda\mu = 0.012$ , $\alpha = 0.13$ .....	106
Figure 6.18 Taylor Deformation whit $Ca_{cr} \ll Ca$ , $\lambda\mu = 0.012$ , $\alpha = 0.13$ .....	107
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 $\theta = 23.85^\circ$ .....	108
Figure 6.20 Tank-treading and Trembling effect of a drop with $Ca = 87$ , $\lambda\mu = 16$ and $\alpha = 0.03$ . .....	109
Figure 7.1 Evolution of principal axes of a drop in uniaxial flow. $\lambda\mu = 0.01$ , $Ca = 0.31$ .....	112
Figure 7.2 Shapes of steady drop deformation, football balloon predicted by Acrivos and Lo. ..	113
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 $\lambda\mu = 0.01$ )......	114
Figure 7.4 Comparison of a drop in uniaxial flow in steady shape. Theoretical shape by Acrivos & Lo, left; and BEM3D, righth. $Ca = 0.26$ and $\lambda\mu = 0.01$ .....	116
Figure 7.5 Taylor Deformation vs Capillary number for a drop in extensional flow 3D.....	117
Figure 7.6 Steady state attain by a drop in extensional 3D-flow with $\lambda\mu = 1.0$ and $Ca = 0.1$ ..	119
Figure 7.7 Stationary deformation vs capillary numbers with different values of $\lambda\mu$ for extensional 2D-flow. Comparison with exp. data of $Ca_{cr}$ .....	120
Figure 7.8 Ratio of B- wrt. W-axis for different values of $\lambda\mu$ in pure extensional 2D-flow axis...	121
Figure 7.9 Lengths (normalized) of the principal axes of drops under stationary extensional 2D-flow with $\alpha = 1.0$ , and $\lambda\mu = 0.1, 1.3$ an $24.1$ , for different values of $Ca$ . .....	122
Figure 8.1 Evolution of the flow parameter $\alpha$ 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 $\alpha$ evolves as a ramp up and down; pure elongational flow occurs at $\alpha = 1$ .....	126
Figure 8.2 Evolution of principal axes length scales vs time; i.e., vs the parameter $\alpha$ . Time = 500 [Gt], $\lambda\mu = 16$ , from simple shear flow $\alpha = 0$ to extensional 2D-flow $\alpha = 1$ . Time: $100 < Gt < 500$ . .....	127
Figure 8.3 Principal axes vs flow parameter alpha $\alpha$ . With $\lambda\mu = 16$ , from simple shear flow $\alpha = 0$ to extensional 2D-flow $\alpha = 1$ . Time: $100 < Gt < 500$ .....	128
Figure 8.4 Taylor Deformations vs. flow parameter $\alpha$ . With $\lambda\mu = 16$ , from simple shear flow $\alpha = 0$ to extensional 2D-flow $\alpha = 1$ . Time period: $100 < Gt < 500$ . .....	130
Figure 8.5 Principal axes vs. flow parameter $\alpha$ . With $\lambda\mu = 16$ , from simple shear flow $\alpha = 0$ to extensional 2D-flow $\alpha = 1$ . Time: $20 < Gt < 100$ . .....	132
Figure 8.6 Evolution of $\beta$ 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. ....	133
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: $40 < Gt < 200$ .....	134



*Figure 8.8 Principal axes vs flow parameter betta  $\beta$ . With  $\lambda\mu = 16$ , from extensional 2D-flow  $\beta = 0$  to uniaxial flow  $\beta = 1$ . Time:  $40 < Gt < 200$ . ..... 135*

*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. .... 142*

# ***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



Universidad Nacional  
Autónoma de México

Dirección General de Bibliotecas de la UNAM

**Biblioteca Central**



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

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, Blawdziewicz, & Loewenberg, 2000), (Huo, Lowengrub, & Shelley, 2001), (Blawdziewicz, Cristini, & Loewenberg, 2003), (Kim & Lowengrub, 2004), (Bazhlekov, 2004), (Reyes, 2005), (Khismatullin, Renardy, & Renardy, 2006), (Subramanian & Koch, 2006), (Young, Blawdziewicz, 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*.

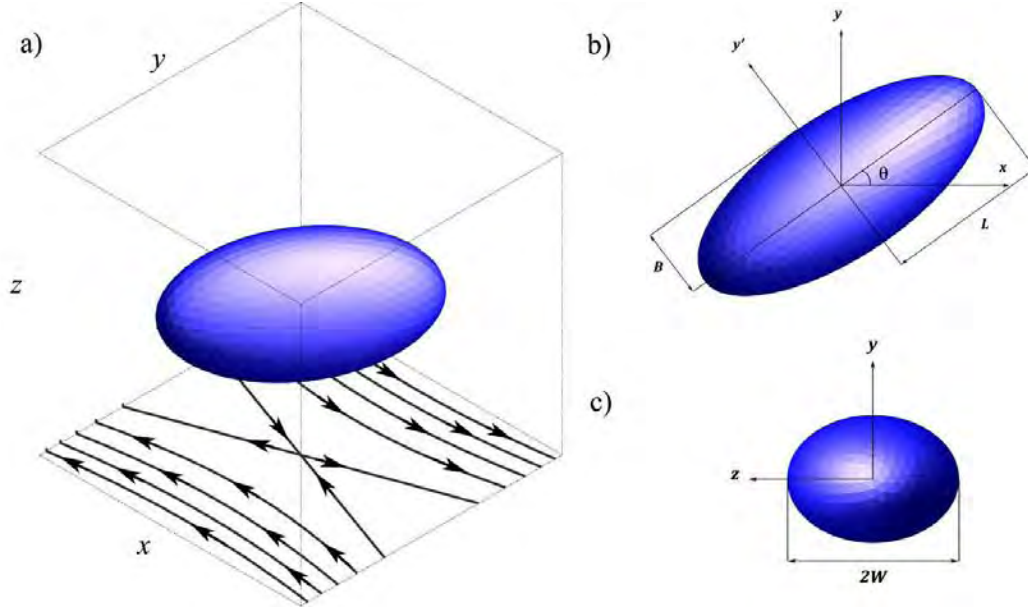
Chapter 9 contains my brief overview of the detailed dynamics of the deformation of a drop when embedded in a large class of flows, and a discussion of the results of this numerical study. The problems solved and unsolved by the method and new branches of study in the drop deformations in strong flows.

## **1.1 Background**

Properties of multiphase fluids are well characterized under static 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  $\mu_2$ , while the continuum phase viscosity is  $\mu_1$ . The viscosity ratio is  $\lambda_\mu = \mu_2/\mu_1$ , and the interfacial tension on the interface is  $\gamma$ . 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.

$$\mu \nabla^2 \mathbf{u} = \nabla p, \quad (1.1)$$

$$\nabla \cdot \mathbf{u} = 0. \quad (1.2)$$

The velocity field  $\mathbf{u}$  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:

$$\Delta \mathbf{f} = \mathbf{f}_1 - \mathbf{f}_2 = (\boldsymbol{\sigma}_1 - \boldsymbol{\sigma}_2) \cdot \hat{\mathbf{n}}, \quad (1.3)$$

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

$$\Delta \mathbf{f} = \gamma \hat{\mathbf{n}} \nabla \cdot \hat{\mathbf{n}} = 2\gamma \kappa_m \hat{\mathbf{n}}. \quad (1.4)$$

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

$$\mathbf{u}(x, y, z) = \frac{G}{1 + \alpha} (y, -\alpha x, 0) \text{ when } \mathbf{x}(x, y, z) \rightarrow \infty, \quad (1.5)$$

where  $G$  is the intensity of the rate of deformation tensor of  $\mathbf{u}(x, y, z)$ . 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

$$G = (|\mathbf{II}_{2D}|)^{1/2}, \quad (1.6)$$

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

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

$$\alpha = \frac{\|\mathbf{D}\| - \|\overline{\mathbf{W}}\|}{\|\mathbf{D}\| + \|\overline{\mathbf{W}}\|}. \quad (1.7)$$

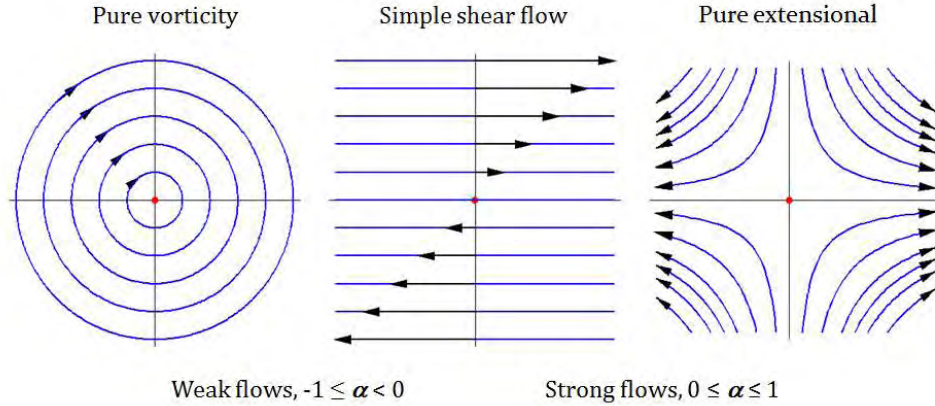
The term  $\mathbf{D}$  corresponds to the *rate deformation tensor*  $\mathbf{D} = \frac{1}{2}(\nabla \mathbf{u} + \nabla \mathbf{u}^T)$ , (Makosco, 1994).  $\overline{\mathbf{W}}$  is the *objective vorticity tensor* defined by Astarita  $\overline{\mathbf{W}} = \mathbf{W} - \boldsymbol{\Omega}$ , (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.*:

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

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  $\alpha$ -parameter:

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



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

Here, the parameter  $\alpha$  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*  $G = \dot{\epsilon}$ .

Thus, the dynamic of drop deformation appears to be characterize by three dimensionless numbers. The viscosity ratio  $\lambda_\mu$ , the capillary number  $Ca$ , Eq. (1.10) and the *flow-type* parameter  $\alpha$ , 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.

$$Ca = \frac{r_0 \mu G}{\gamma}. \quad (1.10)$$

The results presented in Chapter 3 and Chapter 4 were calculated for a range of values of the *flow-type* parameter  $\alpha$  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):

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

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

$$\nabla \cdot \boldsymbol{\sigma} = -\nabla p + \mu \nabla^2 \mathbf{u}, \quad (1.12)$$

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

Let us consider two unrelated Stokes flows with velocities  $\mathbf{u}$  and  $\mathbf{u}^*$  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

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

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

$$\iint_S (\mu \mathbf{u}^* \cdot \mathbf{f} - \mu^* \mathbf{u} \cdot \mathbf{f}^*) dS = 0, \quad (1.14)$$

where  $\mathbf{f} = \boldsymbol{\sigma} \cdot \hat{\mathbf{n}}$ , and  $\hat{\mathbf{n}}$  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).

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

were the vector  $\mathbf{b}$  is a constant vector, and  $\delta(\mathbf{x} - \mathbf{x}_0)$  is the tridimensional delta function centered at  $\mathbf{x}_0 \in S$ . 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  $\mathbf{x}_0$  (named *pole* or *source point*), along the direction and strength of  $\mathbf{b}$ ; 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

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

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

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

were  $\mathbf{G}(\mathbf{x}, \mathbf{x}_0)$ ,  $\mathbf{p}(\mathbf{x}, \mathbf{x}_0)$  and  $\mathbf{T}(\mathbf{x}, \mathbf{x}_0)$  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:

$$\mathbf{G}(\mathbf{x}, \mathbf{x}_0) = -\frac{\delta}{r} + \frac{\hat{\mathbf{x}}\hat{\mathbf{x}}}{r^3}, \quad (1.19)$$

$$\mathbf{p}(\mathbf{x}, \mathbf{x}_0) = 2\frac{\hat{\mathbf{x}}}{r^3} \quad \text{and} \quad (1.20)$$

$$\mathbf{T}(\mathbf{x}, \mathbf{x}_0) = -6\frac{\hat{\mathbf{x}}\hat{\mathbf{x}}\hat{\mathbf{x}}}{r^5}; \quad (1.21)$$

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

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:

$$\mathbf{u}^\infty(\mathbf{x}) - \frac{1}{8\pi\mu_1} \int_S \Delta \mathbf{f}(\mathbf{x}) \cdot \mathbf{G}(\mathbf{x}, \mathbf{x}_0) dS(\mathbf{x}) + \frac{1 - \lambda_\mu}{8\pi} \int_S \mathbf{u}_0(\mathbf{x}) \cdot \mathbf{T}(\mathbf{x}, \mathbf{x}_0) \cdot \hat{\mathbf{n}}(\mathbf{x}) dS(\mathbf{x}) = \begin{cases} \mathbf{u}_1(\mathbf{x}) & \text{1.22 a),} \\ \frac{1 + \lambda_\mu}{8\pi} \mathbf{u}_0(\mathbf{x}) & \text{1.22 b),} \\ \lambda_\mu \mathbf{u}_2(\mathbf{x}) & \text{or 1.22 c).} \end{cases}$$

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

The flow imposed on the continuum phase is determine by  $\mathbf{u}^\infty(\mathbf{x})$ : 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  $\mathbf{x}_0(t + \Delta t) \in S$ ) 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.

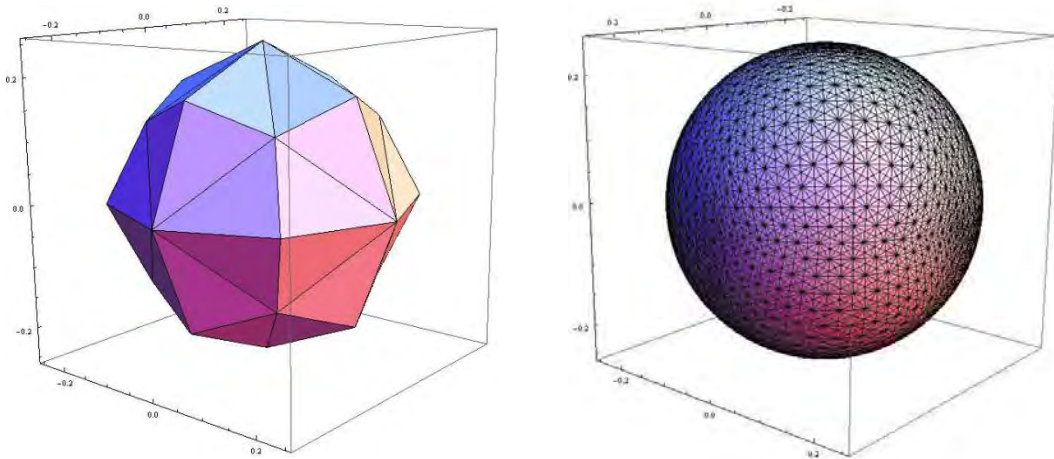
### ***1.2.2 Performance of the mesh used in the method in the time***

*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).

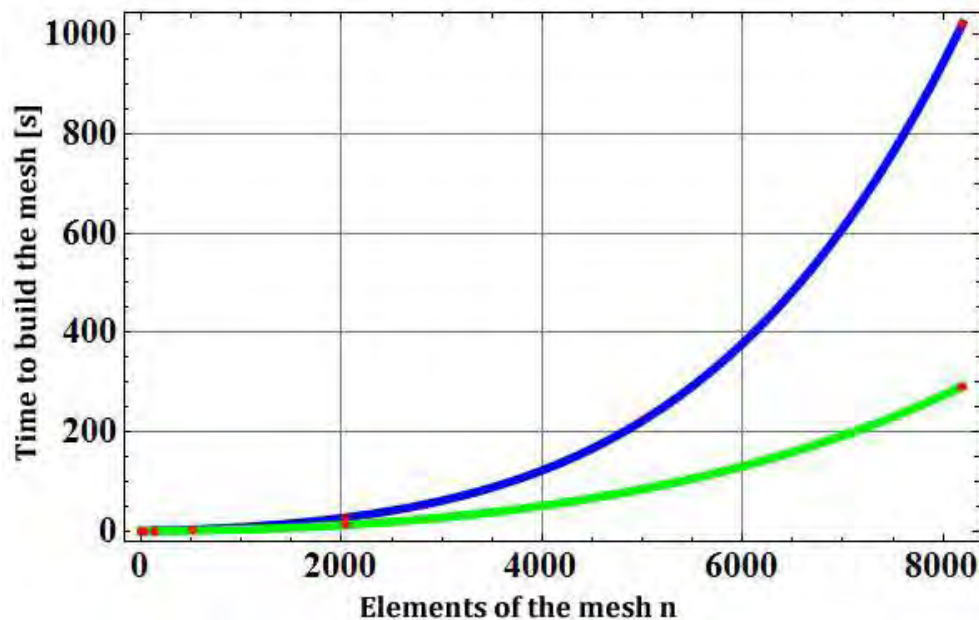


$$n = 8 * 4^k. \quad (1.23)$$



**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  $k = 4$ .

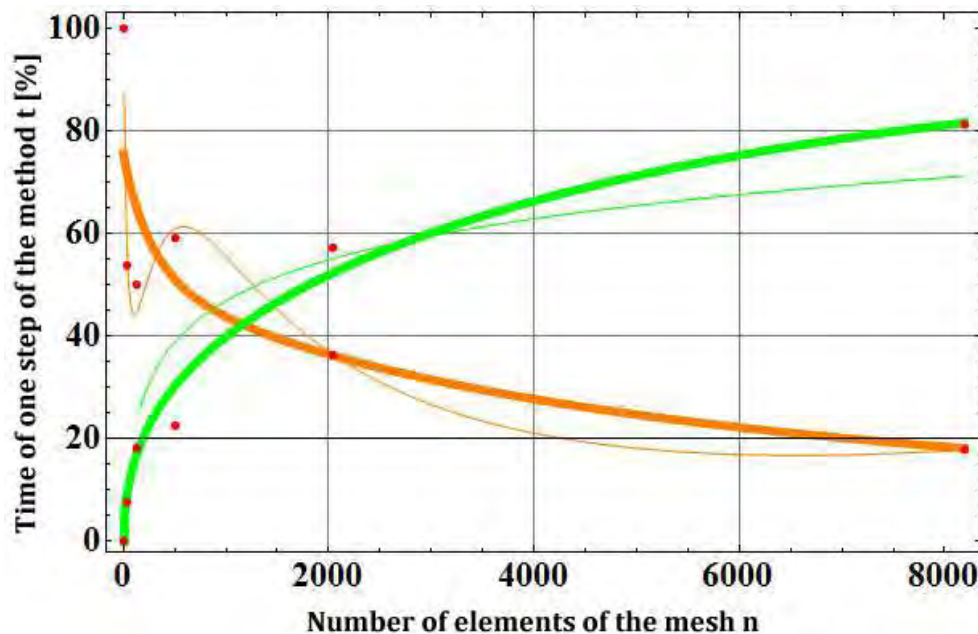
The  $k$  is the number of partition that is applied on the initial octahedron. When the partition  $k = 4$ , the number of elements is 2048. In Figure 1.3, a simple mesh with 8 elements (left) is shown, and a final mesh after the partitioning process (right) with 2048 elements.



**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 thread 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 solve 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 threads used in the algorithm. Most of the results presented here were carried out with 8 threads 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 programming 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.

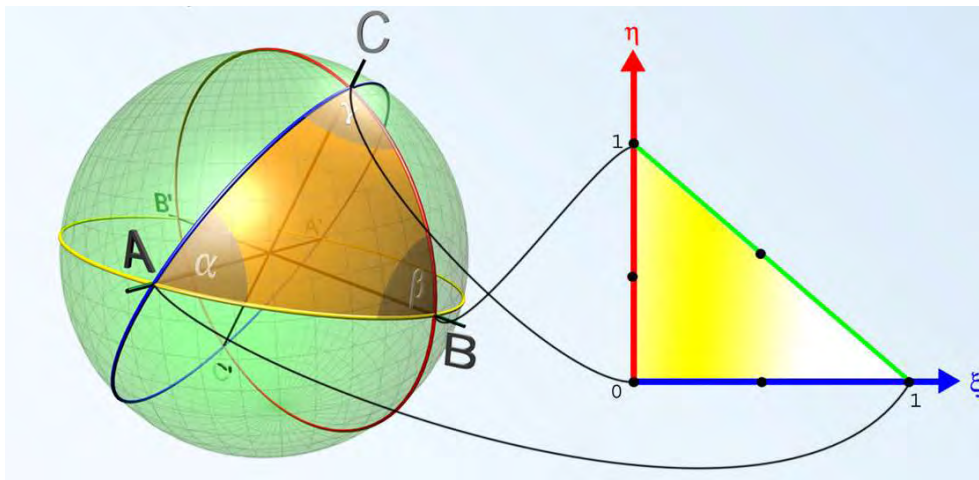
The parallel methods applied use the newest procedures of Fortran 2010 together with the OMP-Libraries of Intel. The final code can simulate different kinds of flows, covering the possibility to study all *2D-flows* from shear to *extensional flow* (strong flows), as well as elongational flows without 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 prove 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 is 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 where the evolution of cross section was measured in a clear manner.



Universidad Nacional  
Autónoma de México

Dirección General de Bibliotecas de la UNAM

**Biblioteca Central**



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

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.<sup>1</sup>

---

<sup>1</sup> 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.

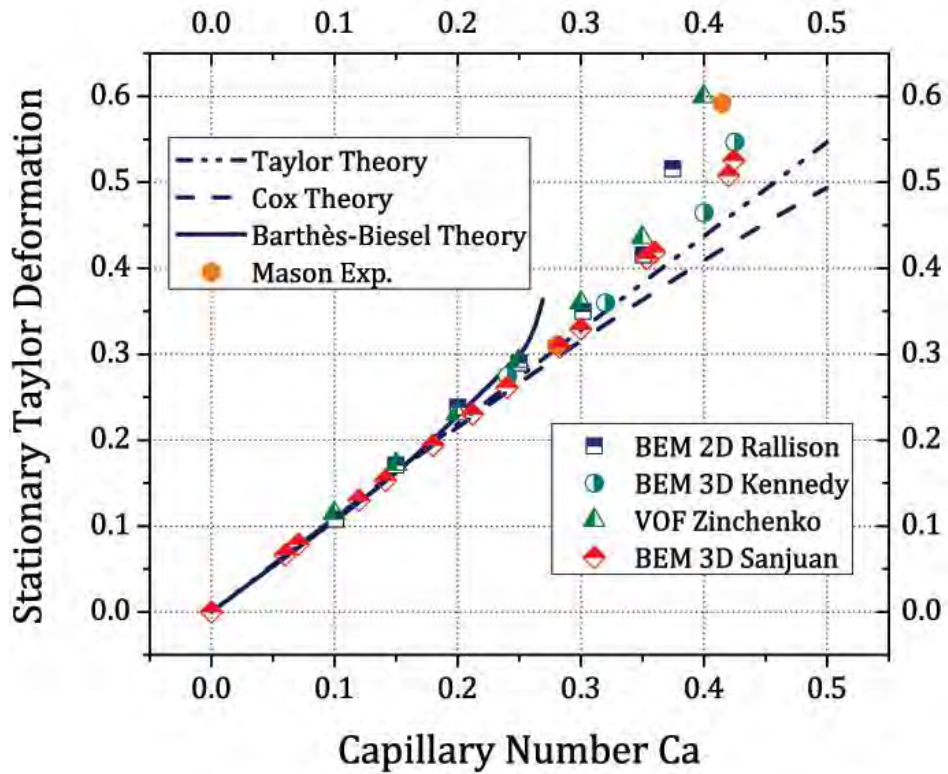


Figure 2.1 Stationary Taylor's Drop Deformation vs. Capillary Number. Simple shear flow,  $\alpha = 0.0$  and  $\lambda_\mu = 1.0$ .

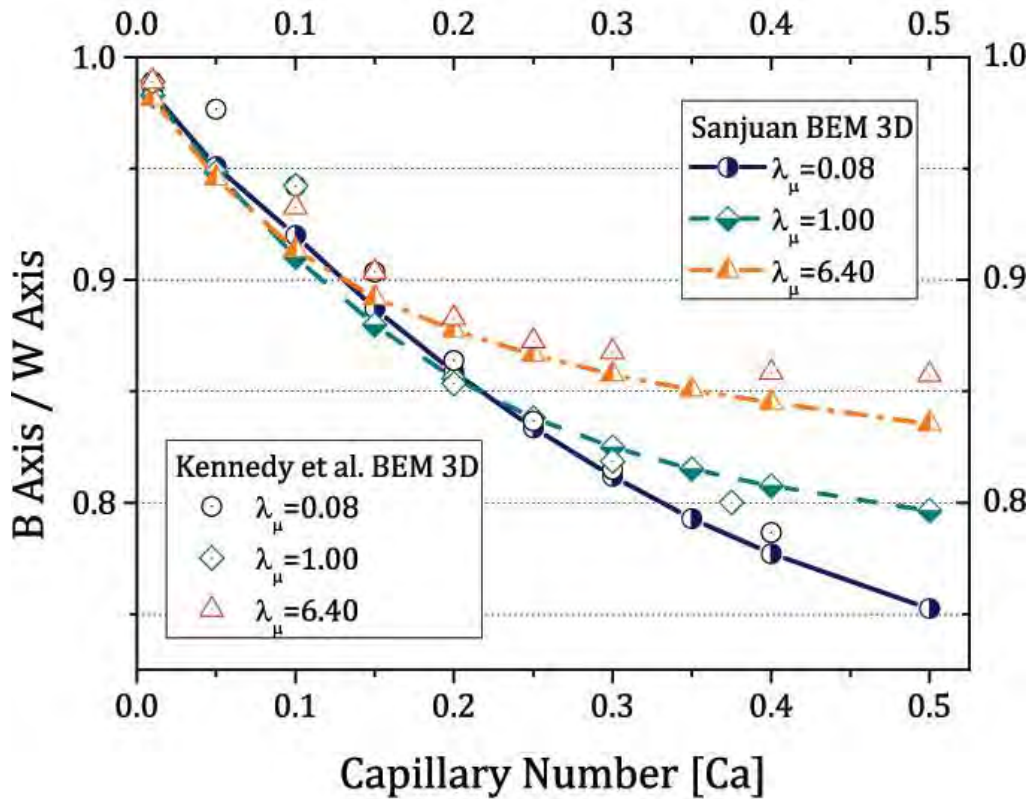


Figure 2.2 B/W ratio vs. Capillary Number for different  $\lambda_\mu$  values.

## 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  $\lambda_\mu$ <sup>2</sup>;

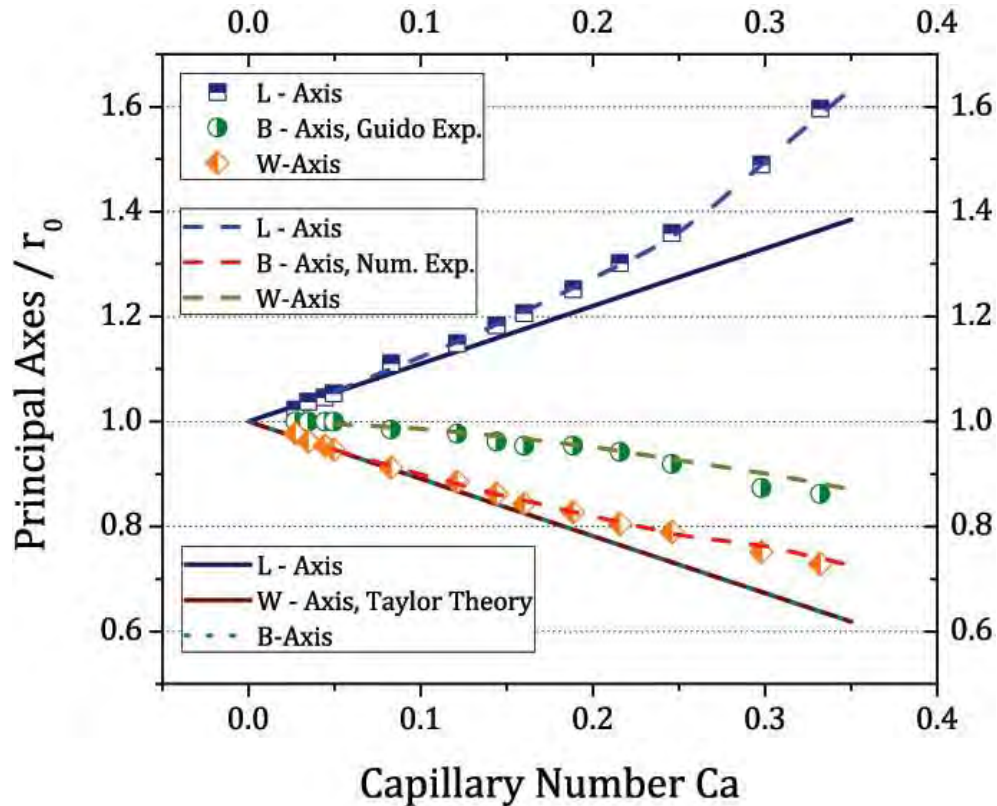
Figure 2.3 shows the values of the numerical simulation for a drop —of radius  $25 \mu m$  and  $\lambda_\mu = 1.4$ — versus the experimental data of Guido; for values of  $\lambda_\mu = 1.3$  and  $\lambda_\mu = 1.4$ . These *normalized values* of the 3 principal axes were obtained for steady states 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

---

<sup>2</sup> 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  $\lambda_\mu$ .



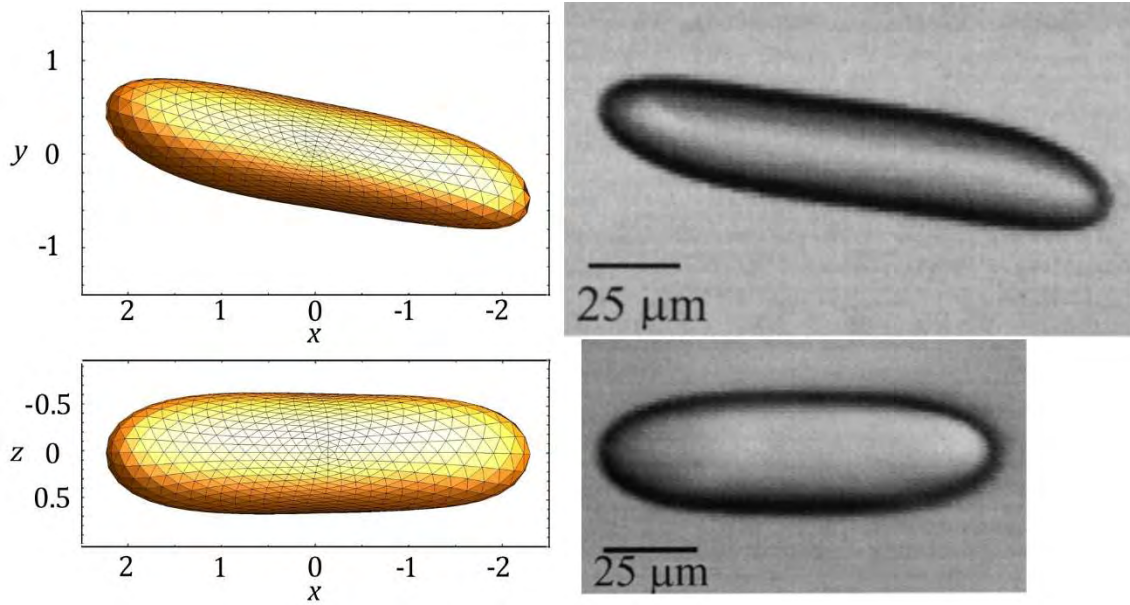
higher order on the shape form observed in the images. So, the information used to calibrate the code could not be the best option.



**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  $25 \mu\text{m}$ . 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  $\theta = 32.05^\circ$ . The numerical values were  $D = 0.67$ , and  $\theta = 34.8^\circ$ .

For the shapes on the left, these are the numerical result for a drop at  $Ca = 0.46$ . The radius of the drop is  $25\mu\text{m}$ . 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  $25\mu\text{m}$ , 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.

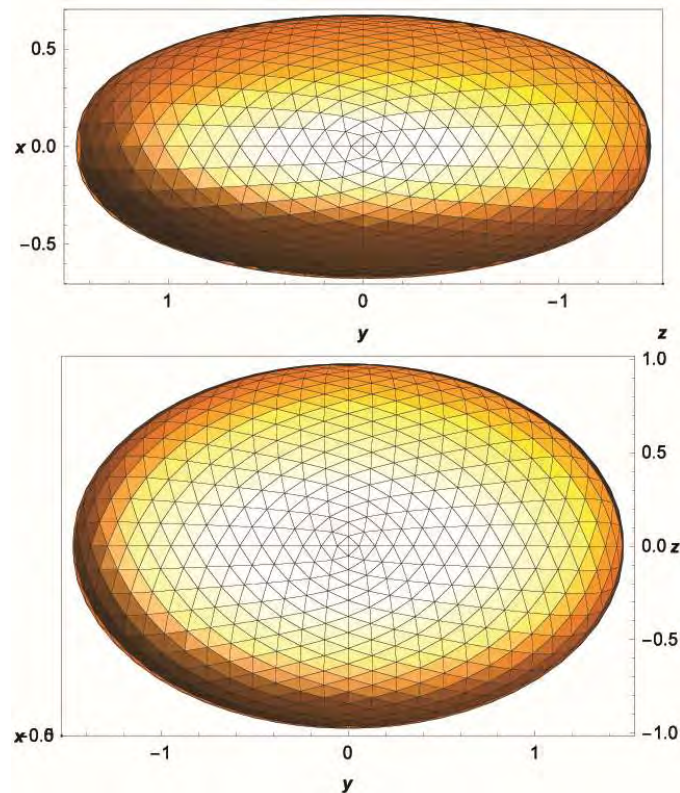
The *deformation of drops immersed in strong flows* has not being studied intensively. Most of the earlier works are those of Taylor and of Leal and collaborators (Bentley & Leal, 1986a) Stone, Milliken, etc., only the work of Rosas *et al.*, addressed these flows more recently. The assumption of  $L \neq B, B = W$  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*<sup>3</sup>. *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

---

<sup>3</sup> *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*.

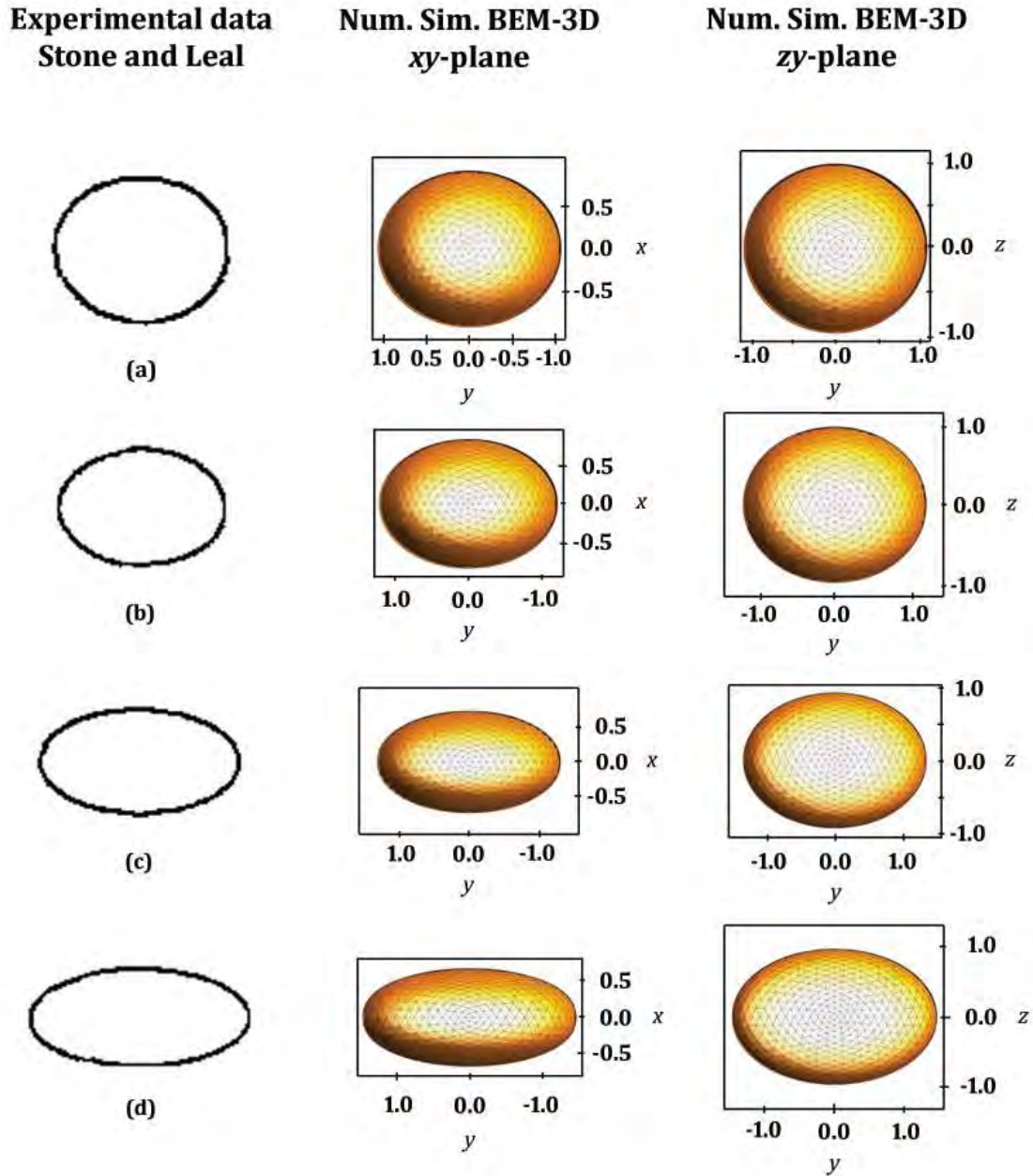
Taylor, Acrivos and Hinch elucidate theoretically the deformation dynamics of drops in *2D-elongational flow* using an asymptotic approximation to describe the form of the drop (Acrivos & Lo, 1978), (Hinch & Acrivos, 1979) and (Hinch & Acrivos, 1980). Furthermore, it is possible to compare as well my 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/ $r_0$ , BEM-3D	$W$ -axis/ $r_0$ , 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,  $\lambda_\mu$ , 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<sup>4</sup>. 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  $\alpha = 0.13$ .

---

<sup>4</sup> 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.





Universidad Nacional  
Autónoma de México

Dirección General de Bibliotecas de la UNAM

**Biblioteca Central**



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

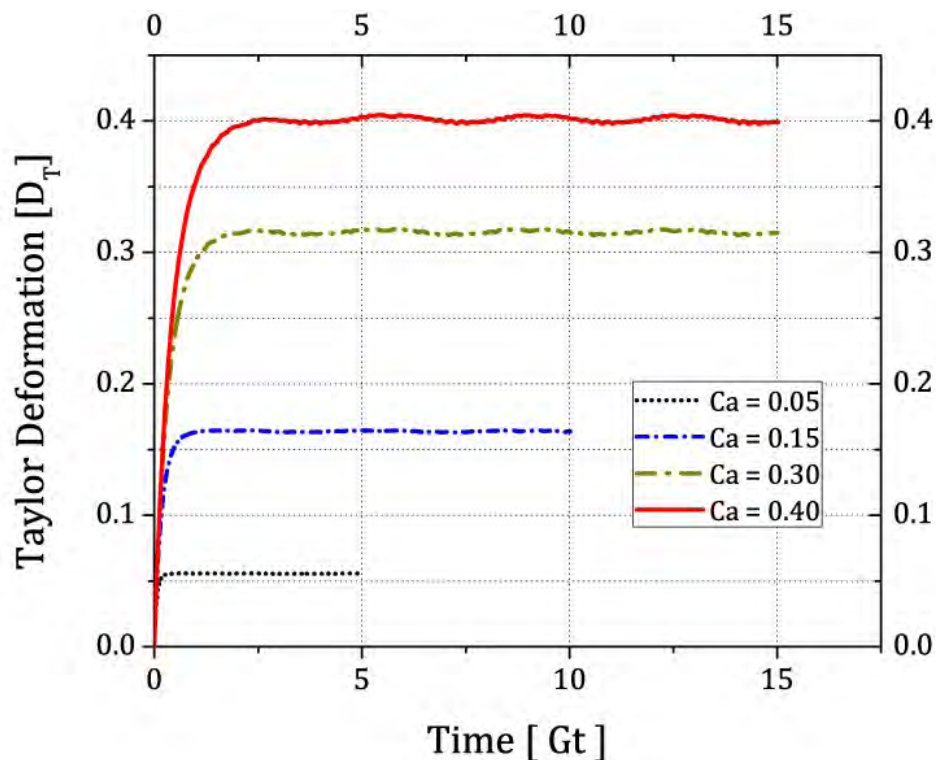
**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

### 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.  $\lambda_\mu = 0.012$ ,  $\alpha = 0.13$ .

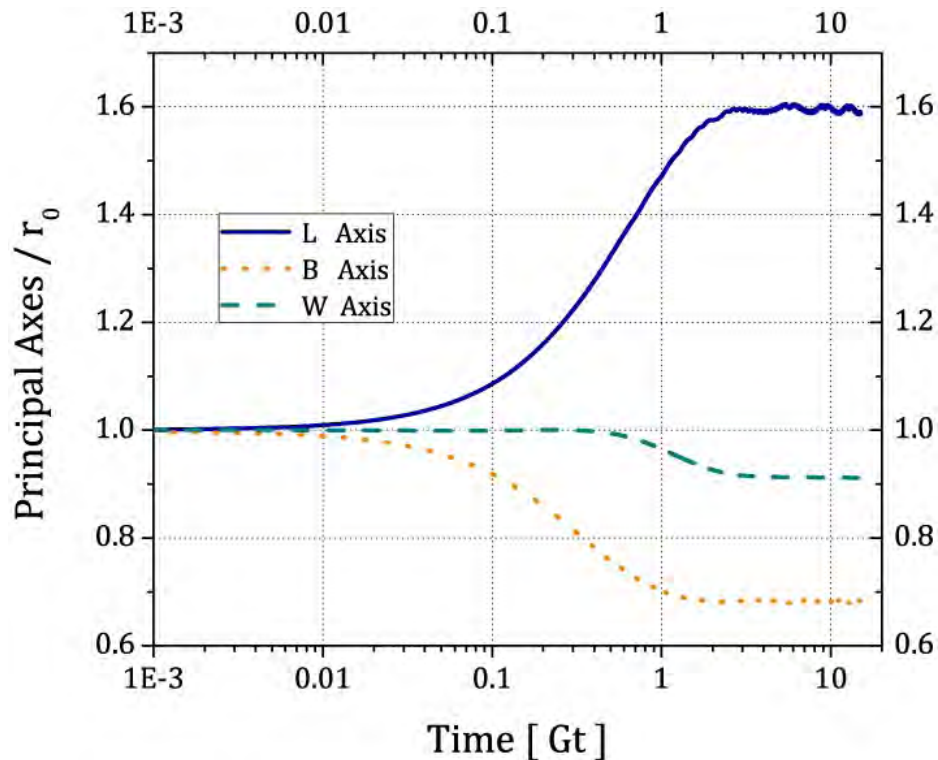
### 3.2 Attained stationary states

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  $0.3 \leq Ca \leq 0.4$ .

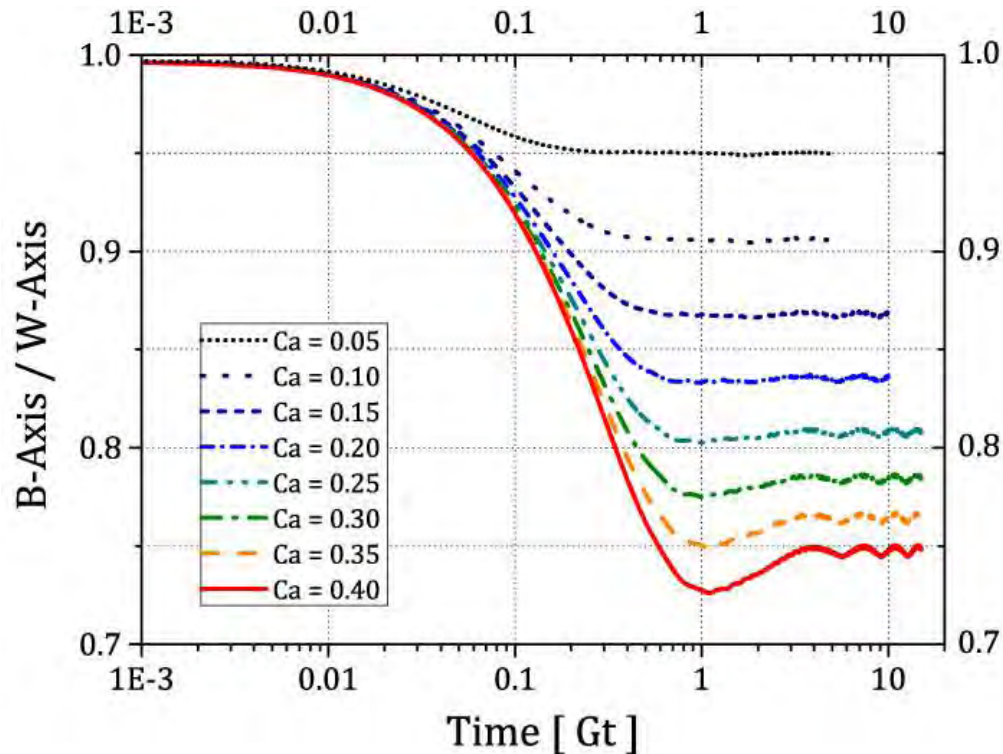
Figure 3.2 shows that the stationary state is reached as early as 10% of the total simulation time; corresponding to approximately 1 unit of dimensionless time; the drop remains in the stationary shape for the rest of the simulation. Although small oscillations in the stationary values persist, the amplitude 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.

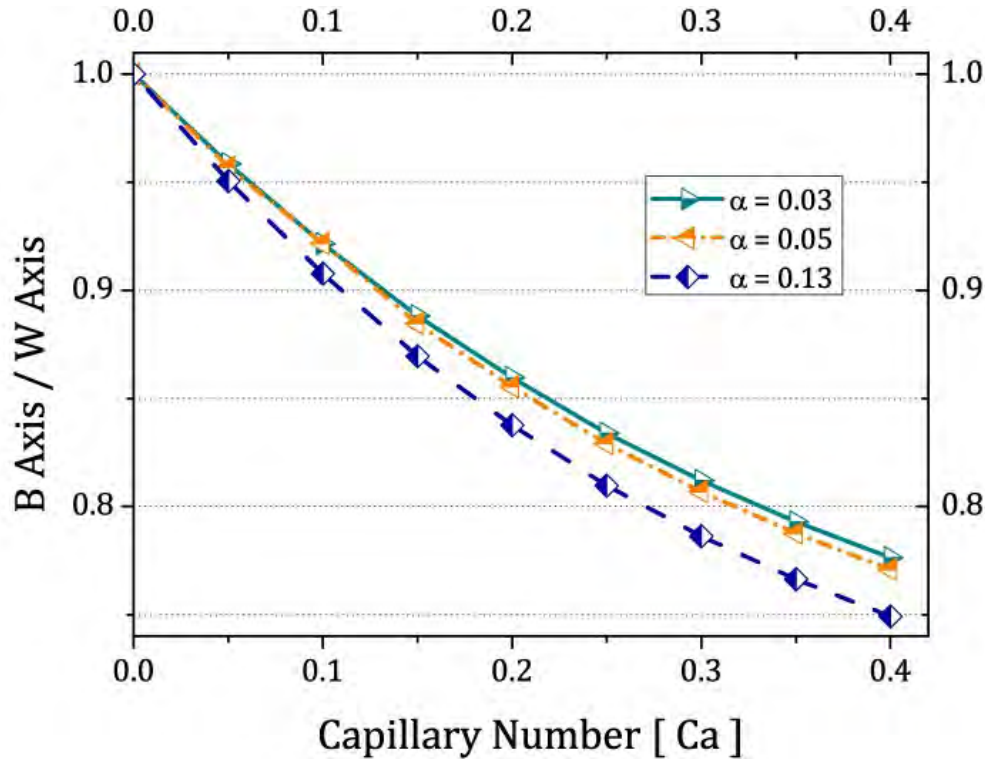


**Figure 3.3** Ratio of the  $B$  wrt.  $W$  axes in numerical simulations for a flow with  $\alpha = 0.13$  and total time  $Gt = 15$  for different values of  $Ca$ .

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  $\alpha$  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

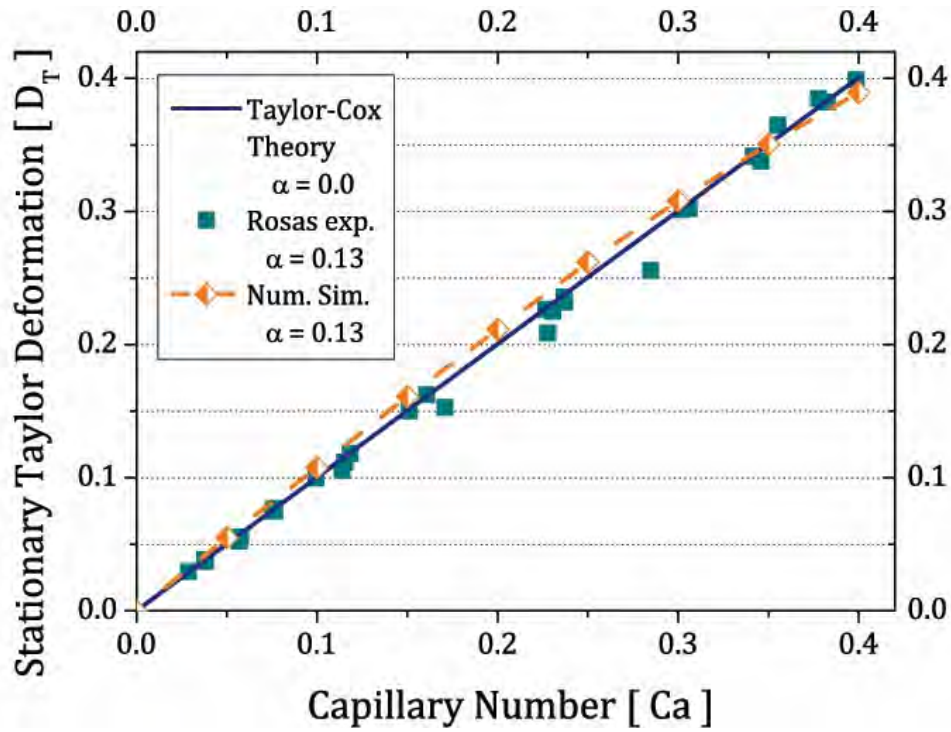
dominant effect that may persist even for pure *2D-elongational flow*. That is, when  $Ca = 0.4$ , and for flows with  $\alpha = 0.03$ , the deformation achieved is less than 85% of the values reached for flows of  $\alpha = 0.13$ .



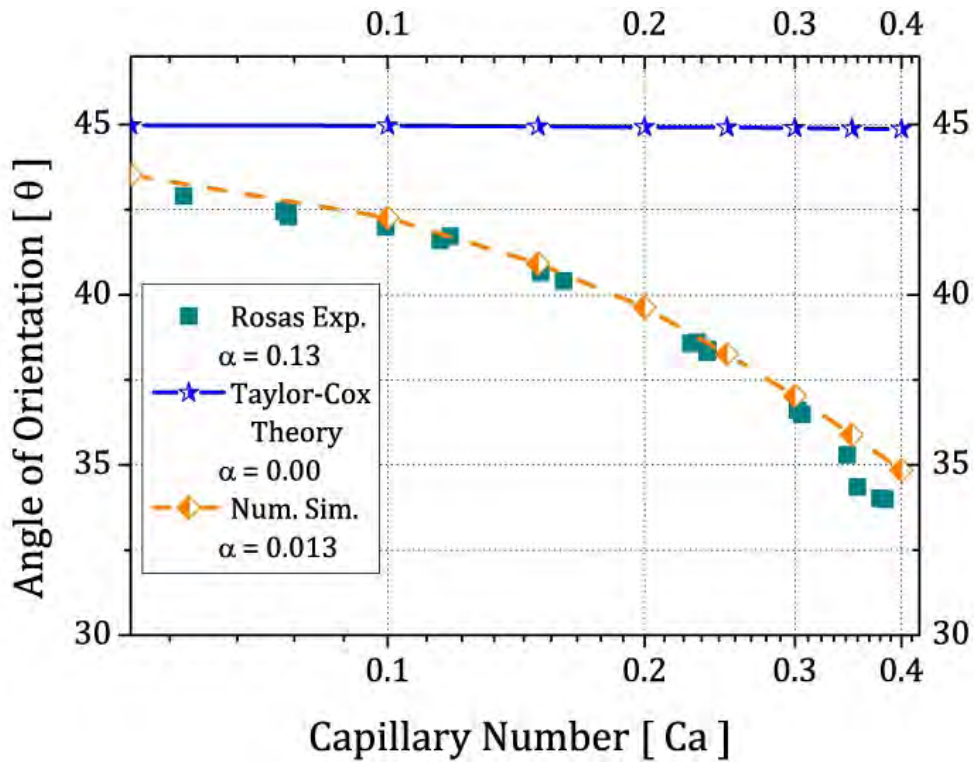
**Figure 3.4** Ratio of the B- wrt. the W-axes under stationary state conditions. Simulations traces correspond to different elongational flows with  $\alpha = 0.03$  ,  $\alpha = 0.05$  and  $\alpha = 0.13$  and for different values of  $Ca$

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).



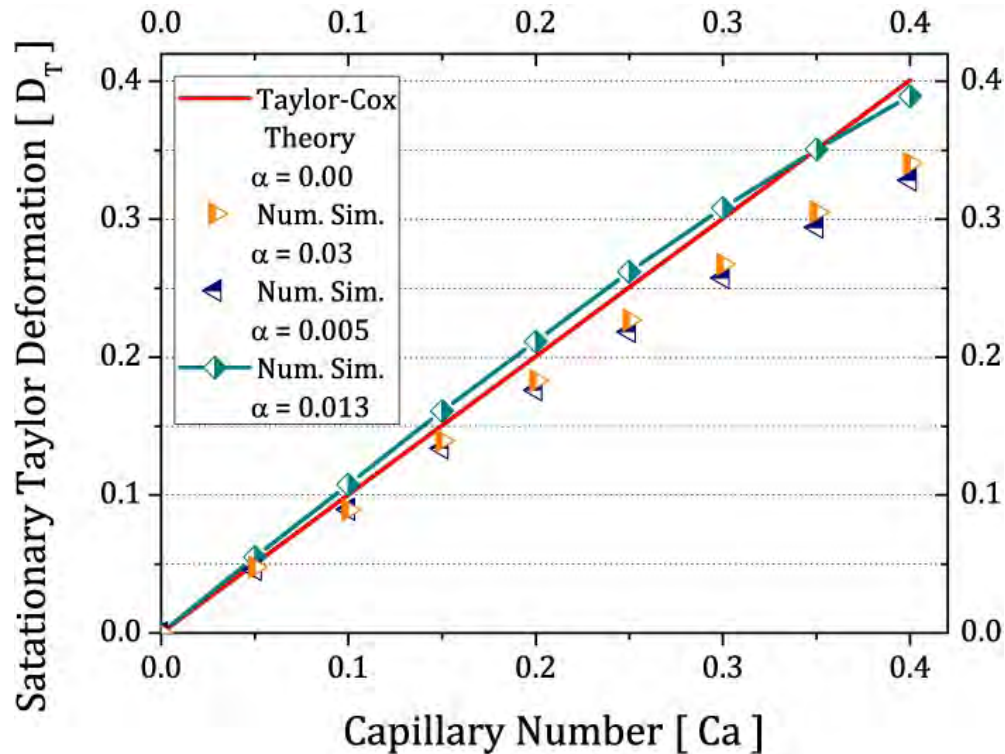
**Figure 3.5** Deformation of the drop for numerical simulations and experimental data for  $\alpha = 0.13$  and analytical prediction of Taylor Cox in simple shear flow  $\alpha = 0.0$  for different values of  $Ca$



**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.



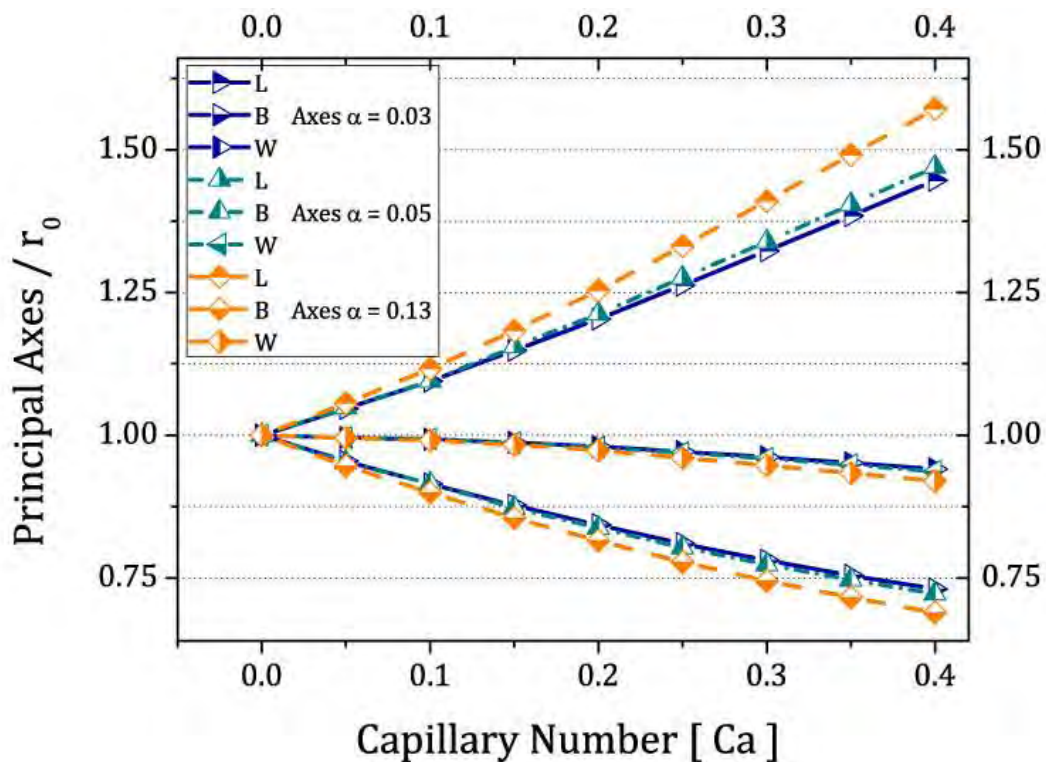
**Figure 3.7** Deformation of the drop for numerical simulations for flow with  $\alpha = 0.03$ ,  $\alpha = 0.05$  and  $\alpha = 0.13$  and analytical prediction of Taylor Cox in simple shear flow  $\alpha = 0.0$  for different values of *Ca*.

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  $\alpha$  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

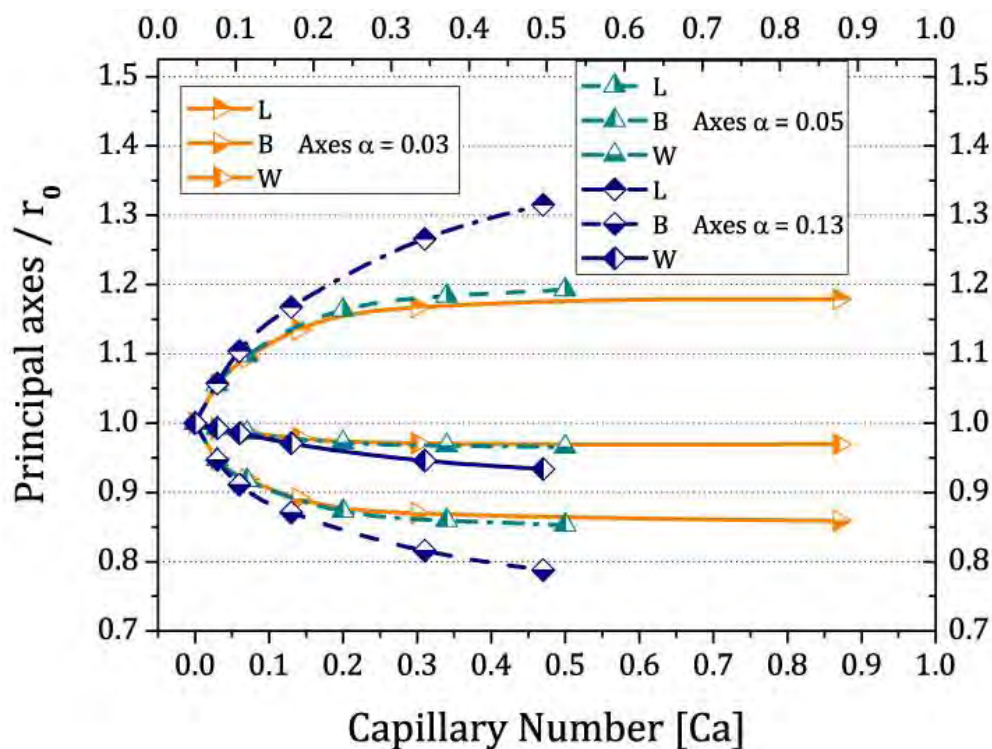
*simple shear flows*. This is likely the case for  $Ca < 0.4$ , and may be correct even if the capillary number is as small as  $Ca = 0.05$ . For  $Ca > 0.4$ , the deformation in the  $xy$ -plane continues to increase, but stronger than the numerical simulation shows, Taylor-Cox model overestimate the 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  $\alpha$  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  $angle = 1.34^\circ$  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,  $\lambda_\mu \gg 1$ , 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).



Universidad Nacional  
Autónoma de México

Dirección General de Bibliotecas de la UNAM

**Biblioteca Central**



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

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.

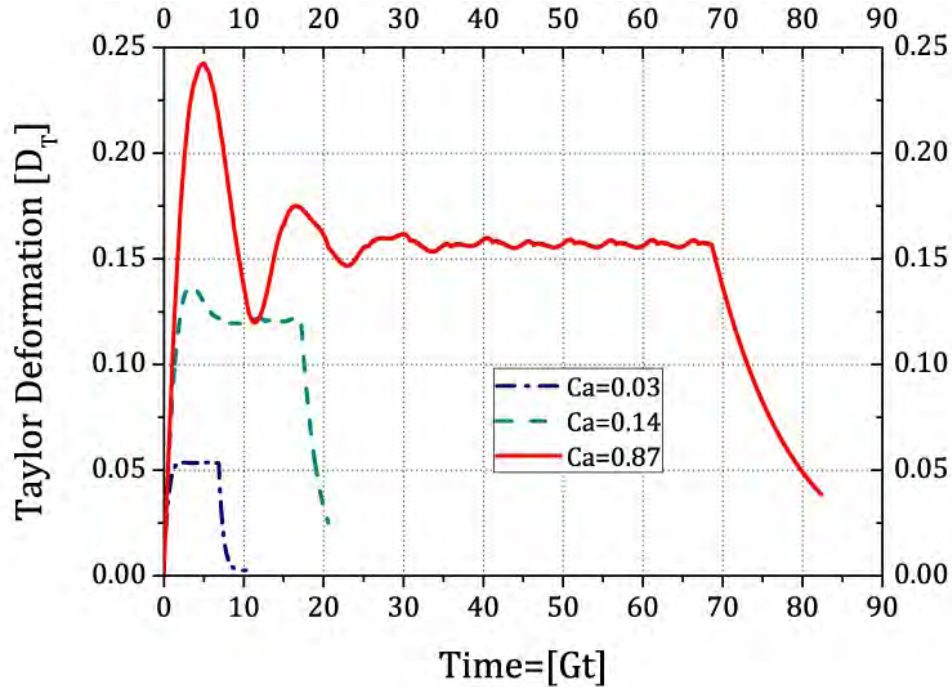
In order to understand the role played by the orientation angle and deformation on an equal footing, Cox presented polar plots of deformation  $D_T$  and orientation  $\theta$ . The time-parameterized numerical trajectories  $D_T$  *vs.*  $\theta$  of the drop here presented are similar as those predicted 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.  $\lambda_\mu = 16$ ,  $\alpha = 0.03$ .

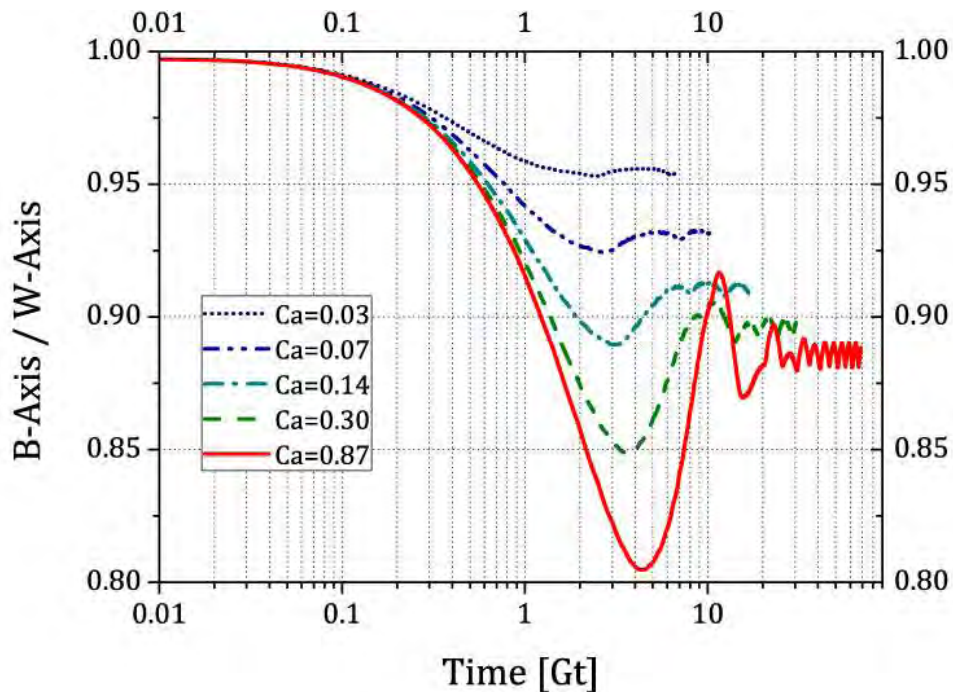
Figure 4.1 shows typical curves of deformation histories for  $Ca = 0.03$ ,  $Ca = 0.14$ , and  $Ca = 0.87$  with viscosity ratio  $\lambda_\mu = 16$ , and  $Gt = 83$ . The flow parameter values are the same as Rosas used with the Two-Roll Mill (TRM) device (Rosas, Reyes, Minzoni, & Geffroy, 2014). For 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  $\alpha=0.03$  and total time  $Gt=75$  for different values of  $Ca$ .

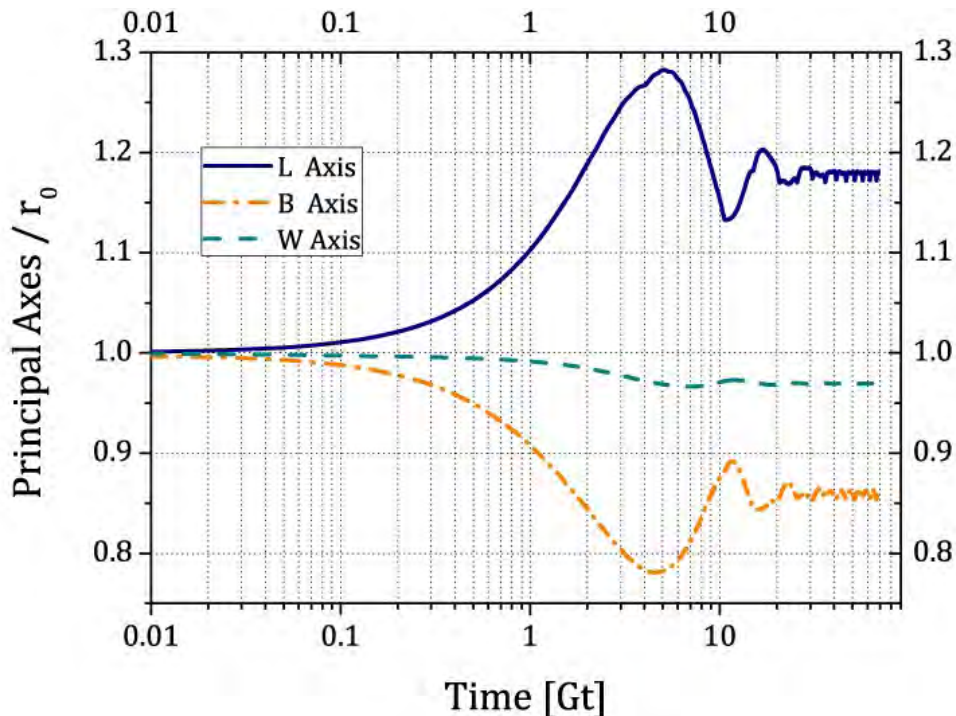
As shown in Figure 4.1, when  $Ca = 0.03$ , the drop deformations have a similar behavior in time as was observed in Chapter 3, but with a smaller length. The drop reaches a stationary shape, and if the flow is turned off, the drop returns to the spherical shape. When the capillary number increases a little more, the competition (as function of the rate of viscosity) is more evident. There is an overshoot 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.

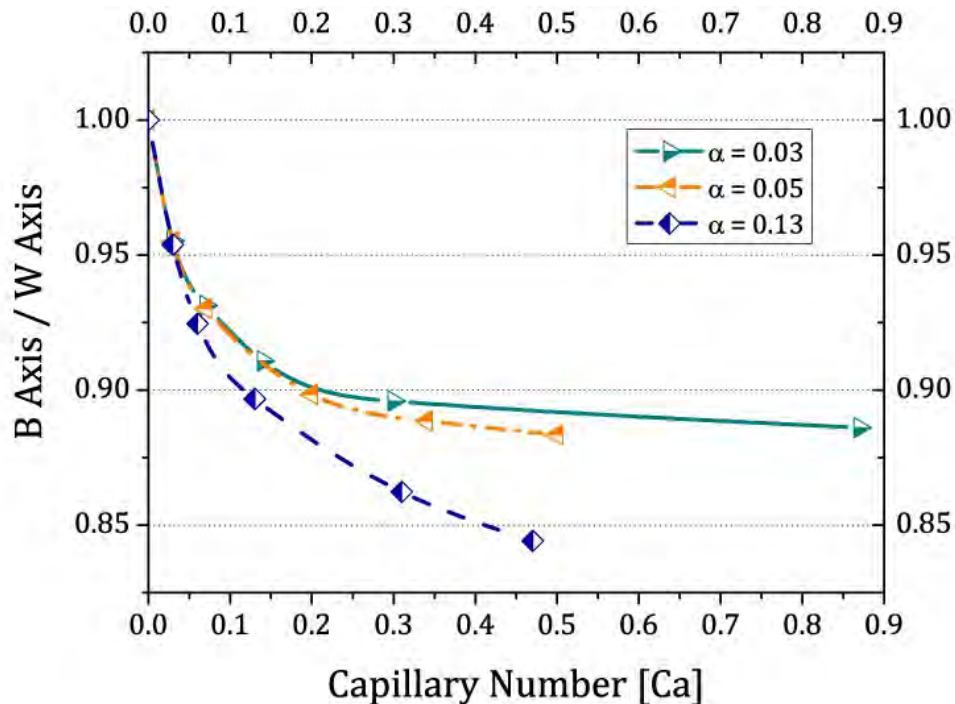
In a similar way as in Chapter 3, data for the  $B/W$ -axis, shown in Fig. 4.2, indicates that the cross section of the drop is not circular and is time dependent, and may pulsate slightly. In Fig. 4.2, during the initial 10% of the evolution of  $B/W$ , the main oscillation occurs, with the amplitude 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  $\alpha$  values, but the  $2D$ -character of the flow remain dominant.



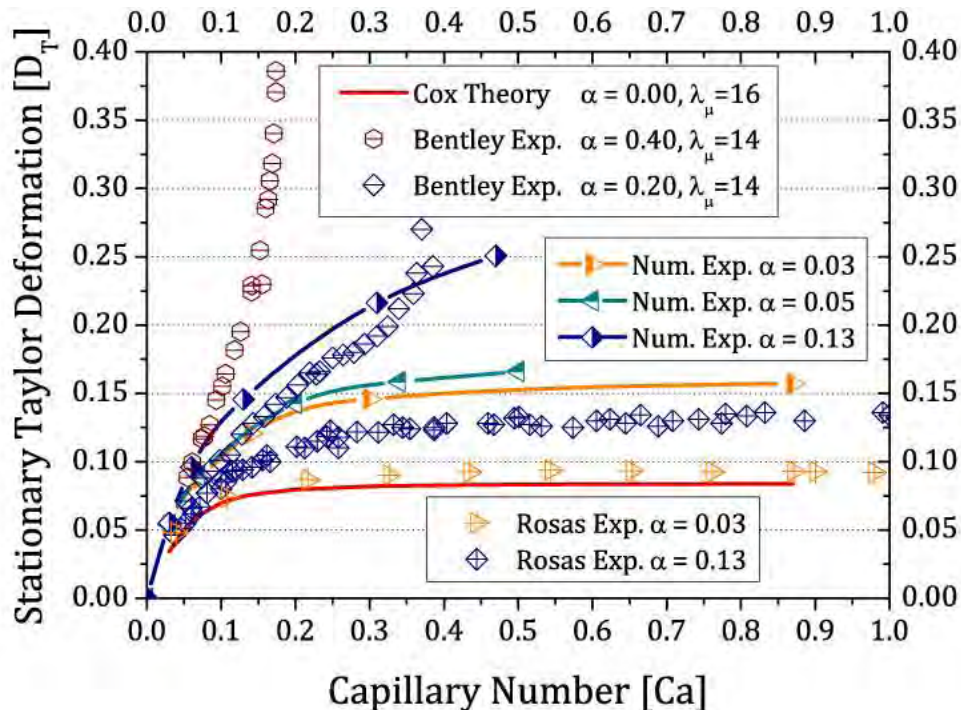
**Figure 4.4** Stationary ratio of the  $B$  wrt. the  $W$  axis in numerical simulations for flow with  $\alpha = 0.03$ ,  $\alpha = 0.05$  and  $\alpha = 0.13$  for different values of  $Ca$ .

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,  $\alpha$ . 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*  $\alpha$ . 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  $\alpha$  but also on the ratio of viscosity  $\lambda_\mu$ . 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  $\lambda_\mu$ .

Furthermore, at the critical limit conditions, the length scales of the deformation are more difficult to define because no steady elongation is 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.<sup>5</sup> 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

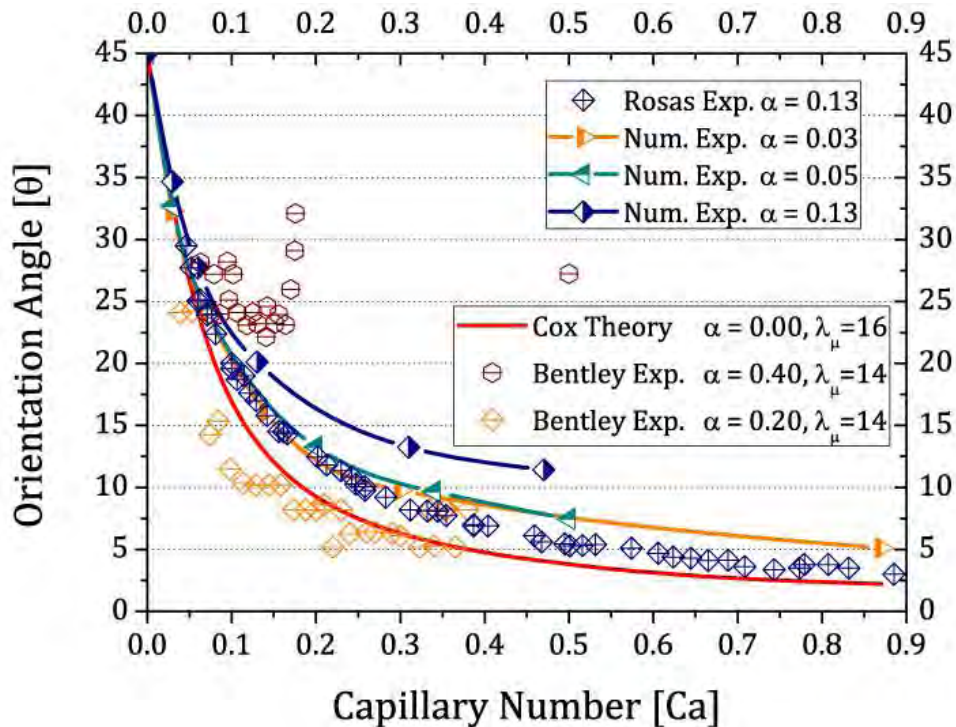
---

<sup>5</sup> 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  $\alpha$  goes near values of one. In the other extreme, as the *parameter*  $\alpha$  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:  $\alpha$  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.

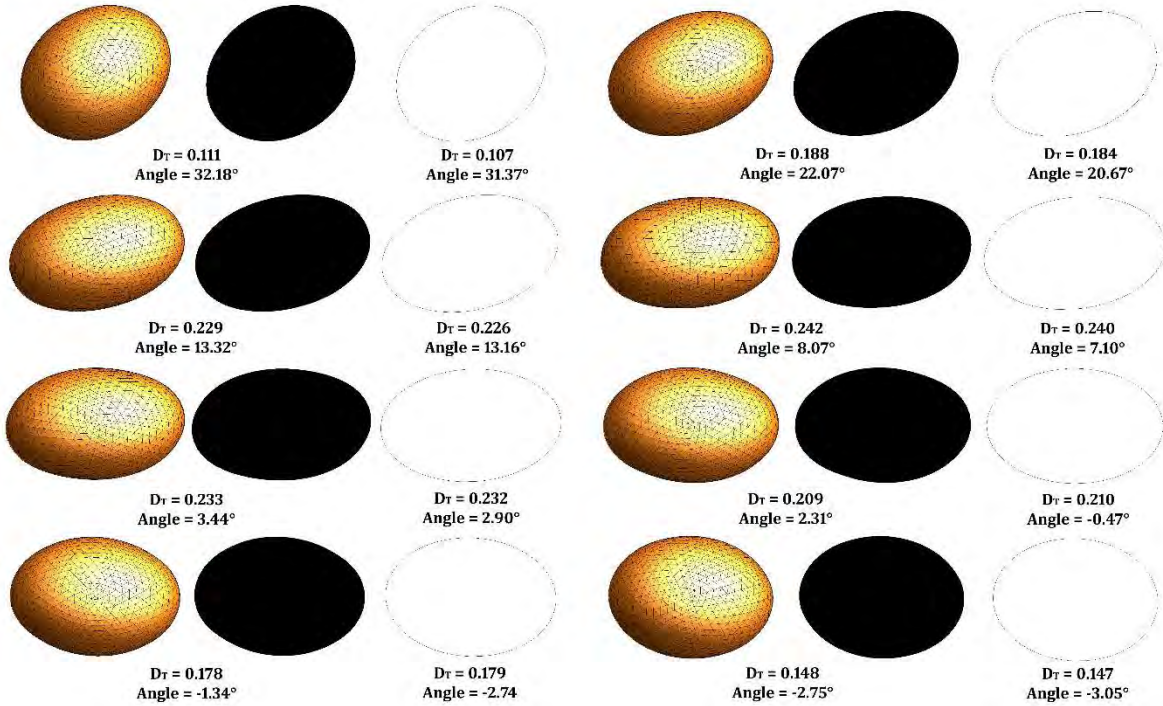


Figure 4.7 Drop evolution in the time.  $Ca = 0.87$  with  $\lambda_\mu = 16$  and  $\alpha = 0.03$ . The color images are the numerical shape obtained with BEM3D, the black pictures are the projection on the  $xy$ -plane; within these projections, ellipses were fitted.

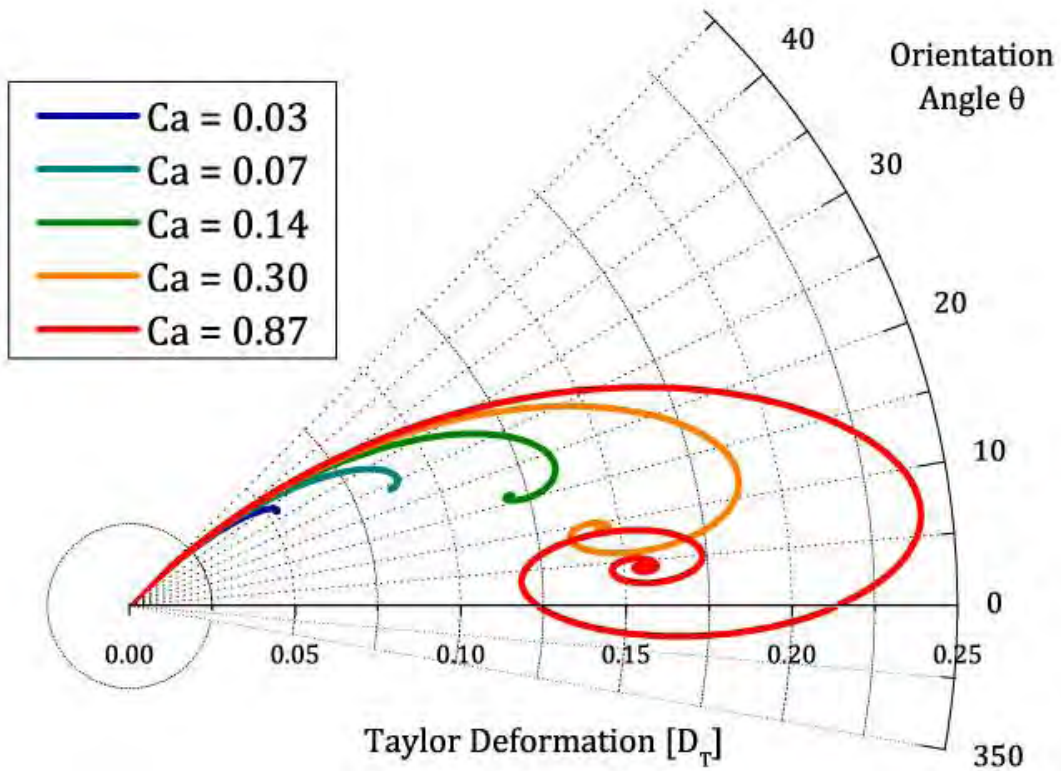


Figure 4.8 Polar plot of Taylor deformation vs. orientation angle for different capillary numbers with  $\alpha = 0.03$ ,  $\lambda_\mu = 16$ .

# **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





Universidad Nacional  
Autónoma de México

Dirección General de Bibliotecas de la UNAM

**Biblioteca Central**



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

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.

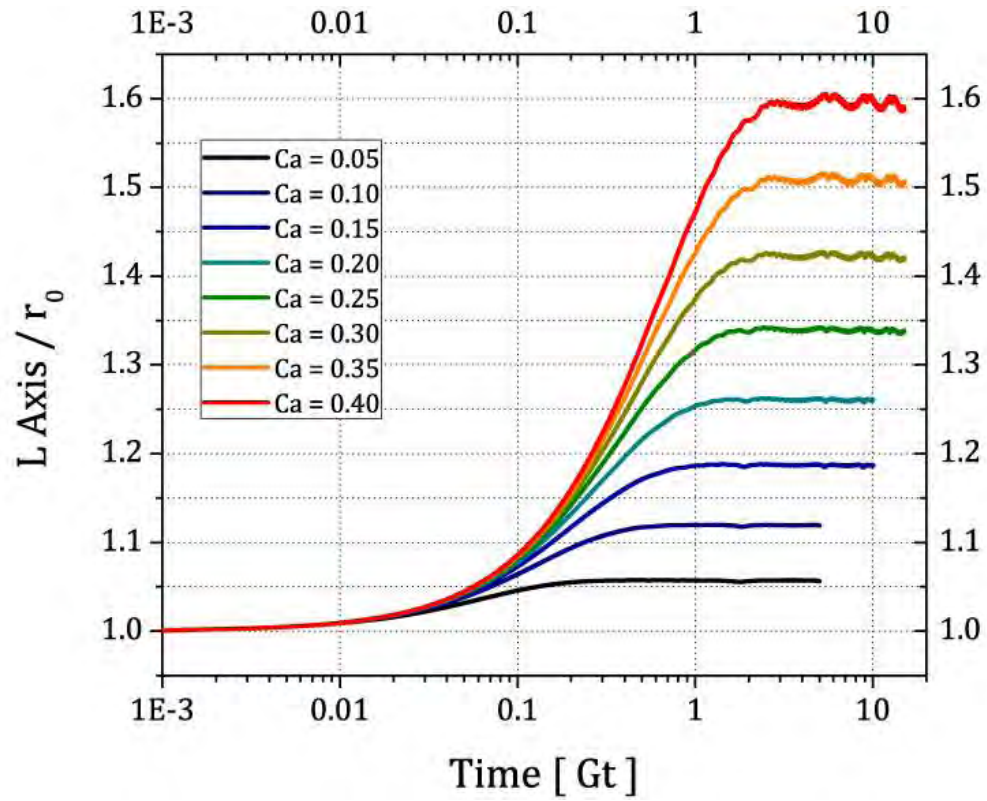


Figure 5.1 Time evolution of principal L axis which  $\alpha = 0.13$  for different values of Ca.

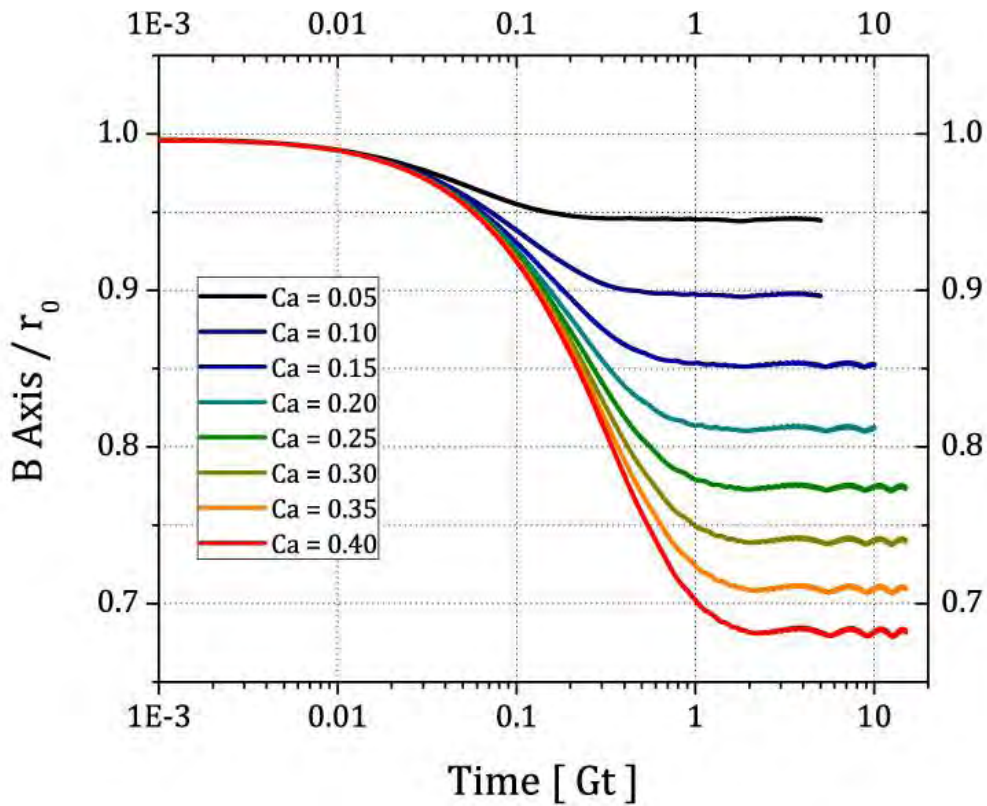


Figure 5.2 Time evolution of principal B axis which  $\alpha = 0.13$  for different values of Ca.

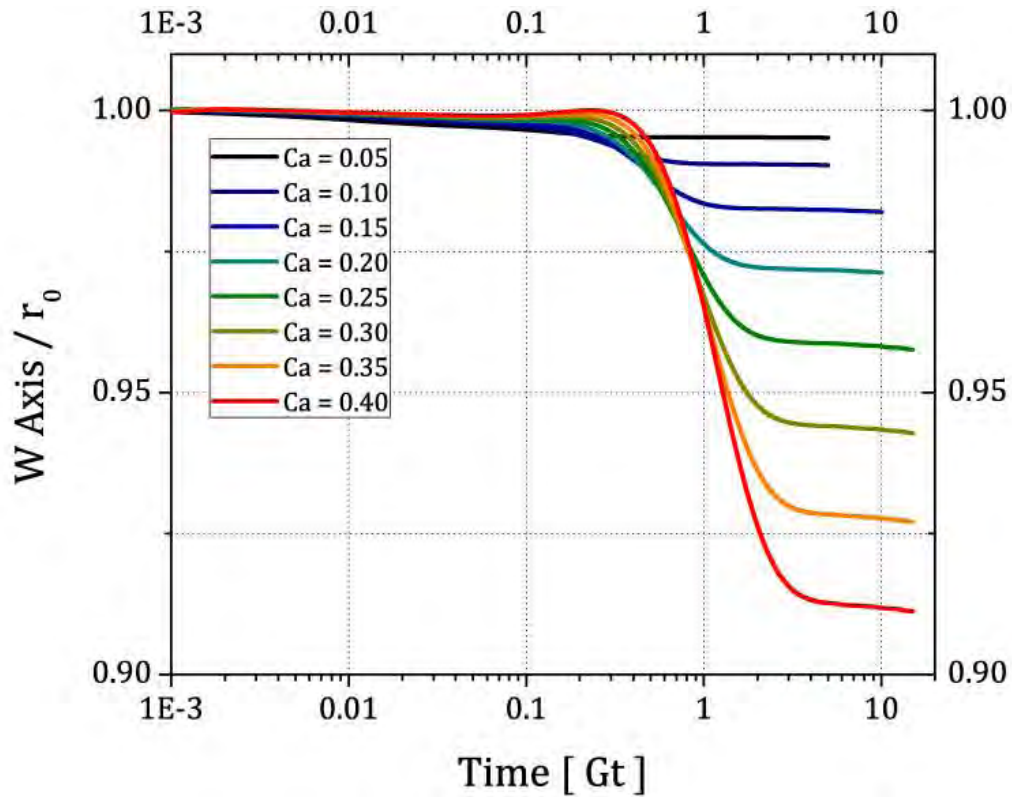


Figure 5.3 Time evolution of principal W axis which  $\alpha = 0.13$  for different values of Ca.

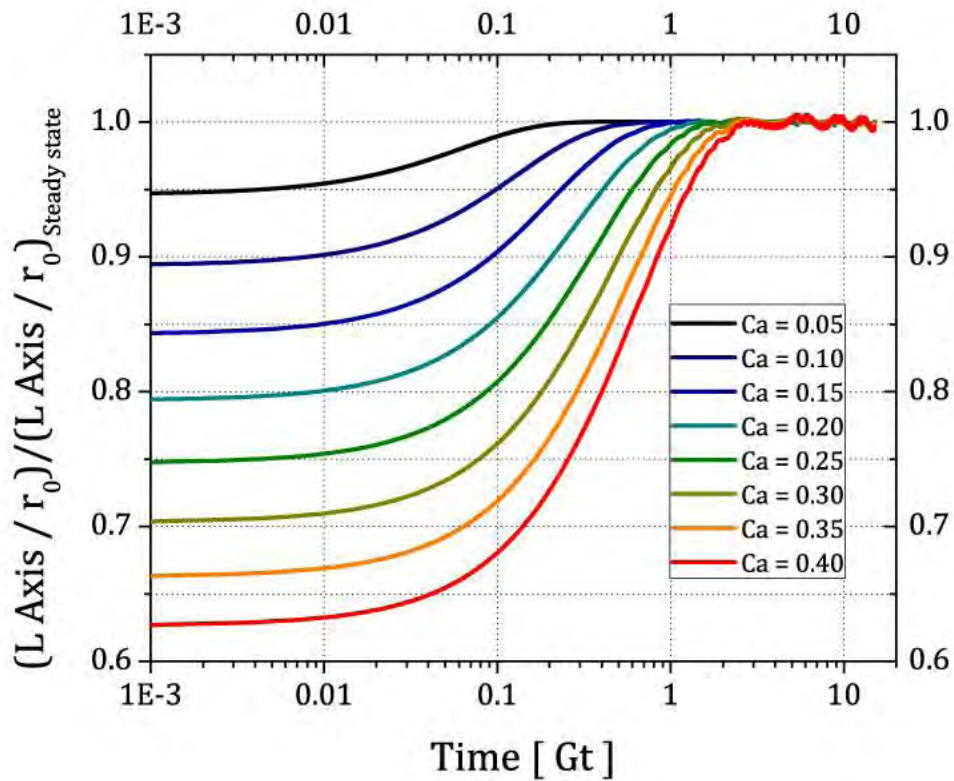


Figure 5.4 Time evolution of principal L axis normalized wrt. stationary value, which  $\alpha = 0.13$  for different values of Ca.



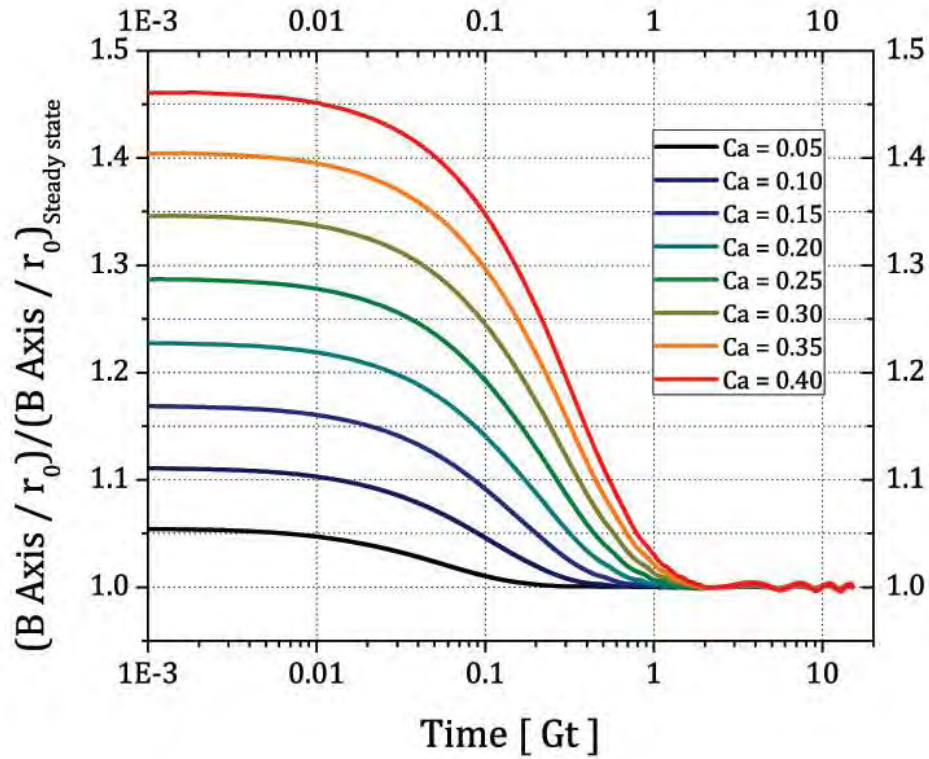


Figure 5.5 Time evolution of principal B axis normalized wrt. stationary value, which  $\alpha = 0.13$  for different values of Ca.

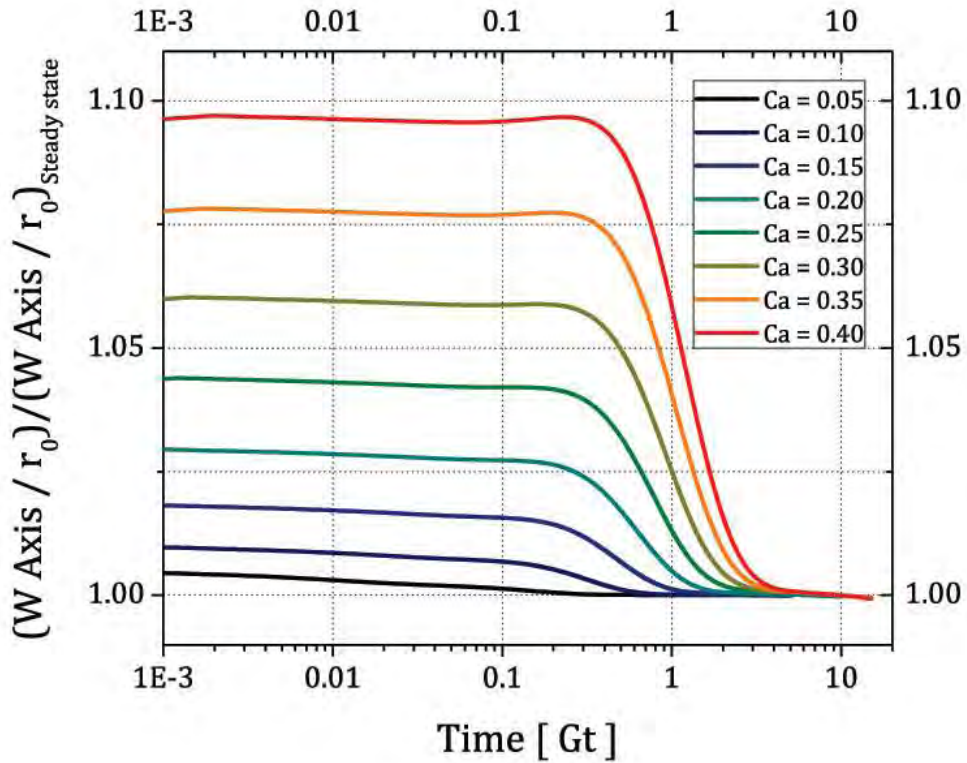
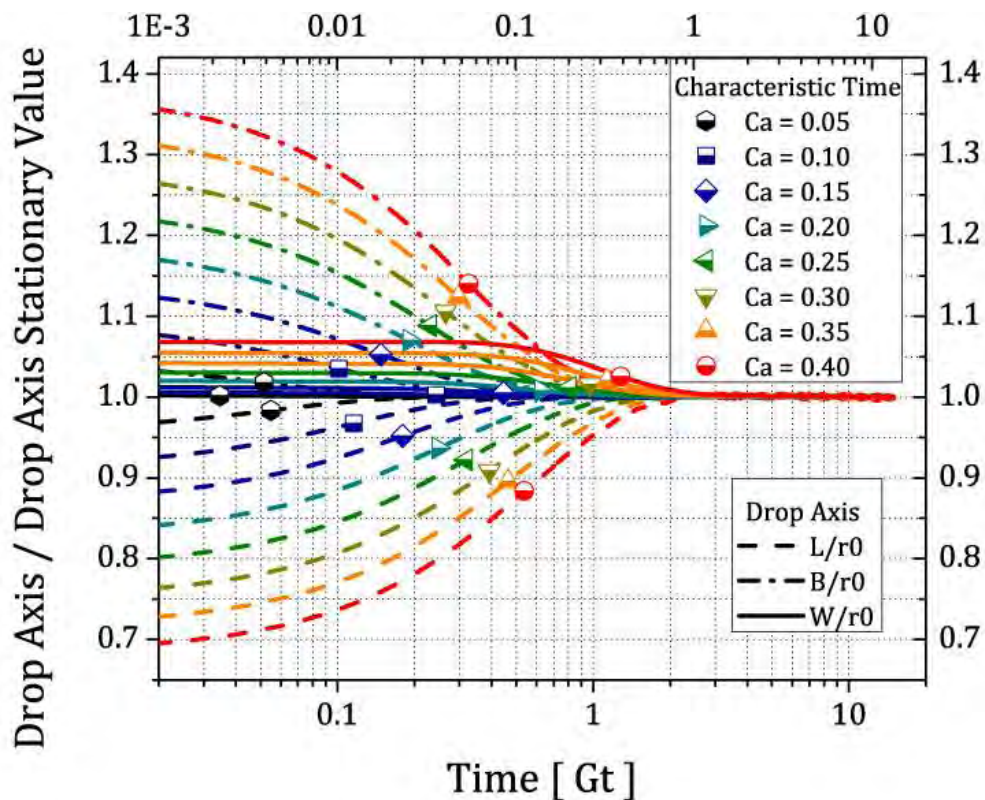


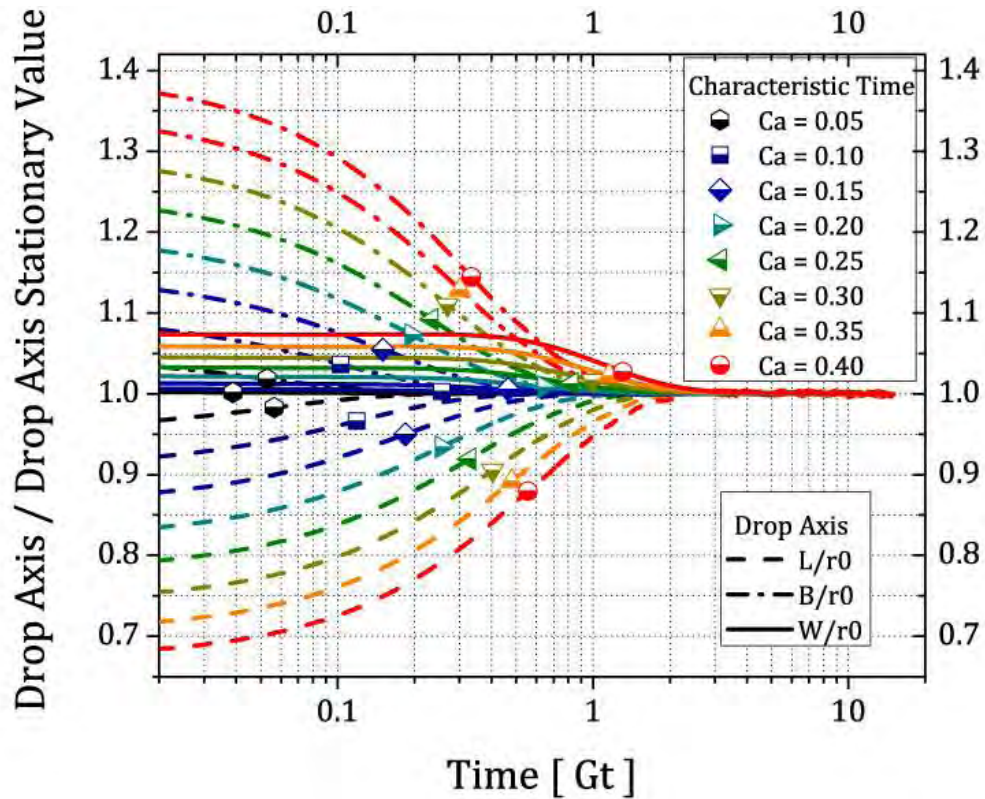
Figure 5.6 Time evolution of principal W axis normalized wrt. stationary value, which  $\alpha = 0.13$  for different values of Ca.

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  $\alpha=0.03$  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.

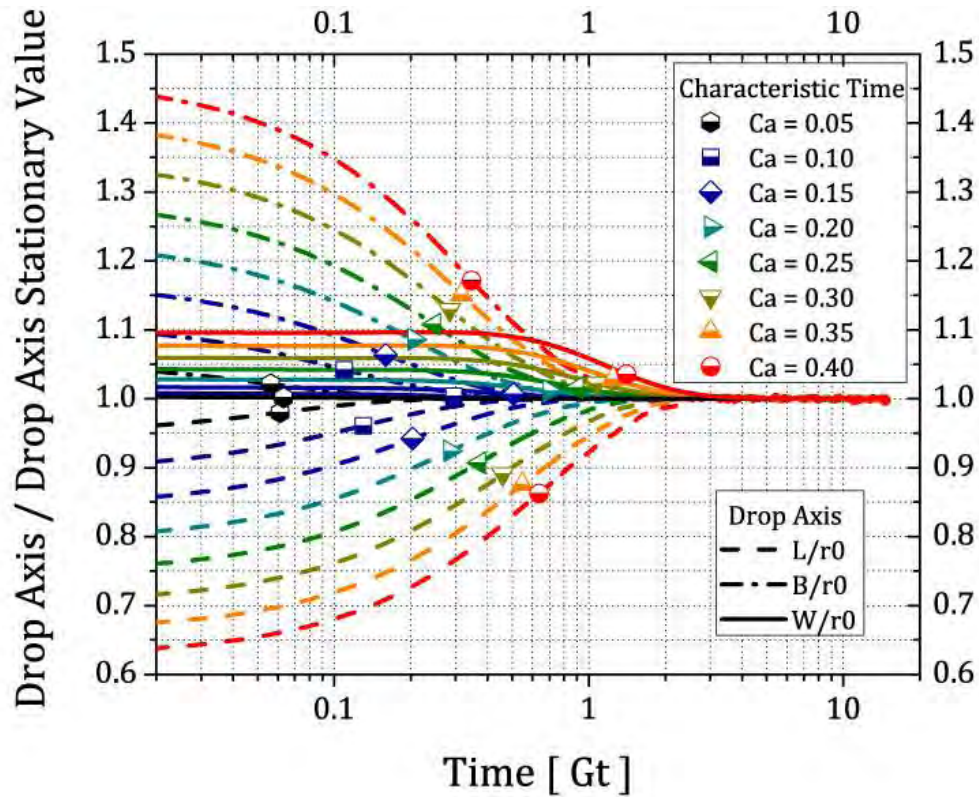


**Figure 5.8** Time evolution of principal axes normalized wrt. stationary value, which  $\alpha=0.05$  for different values of  $Ca$ . Characteristic times in symbols.

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 [Gt] 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.





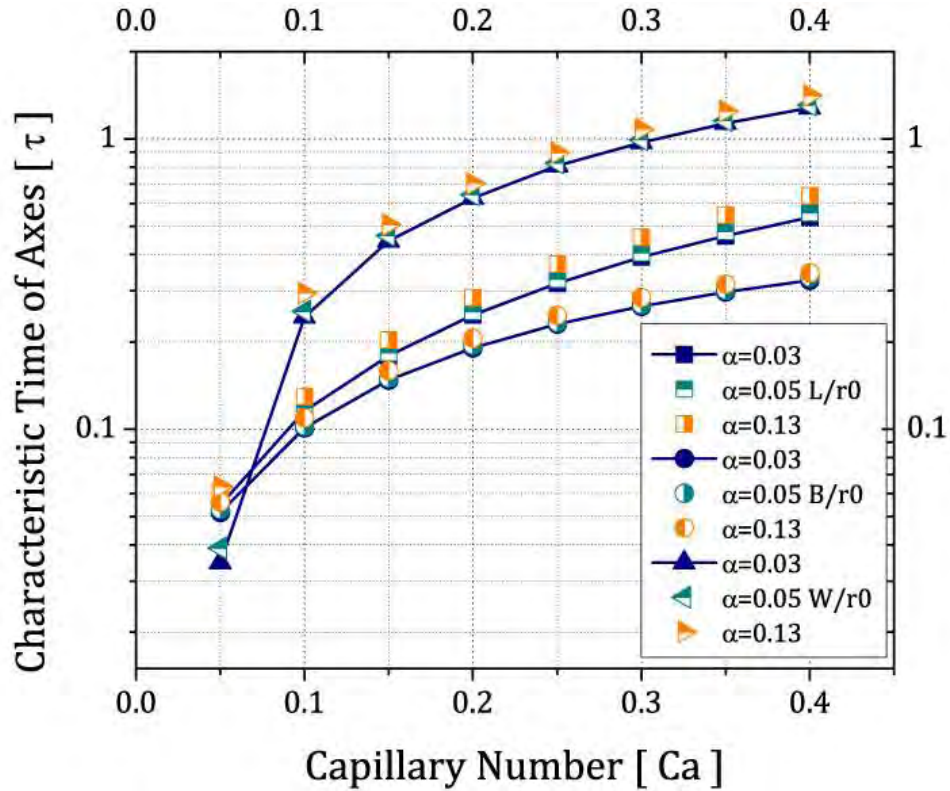
**Figure 5.9** Time evolution of principal axes normalized wrt. stationary value, which  $\alpha=0.13$  for different values of  $Ca$ . Characteristic times in symbols.

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



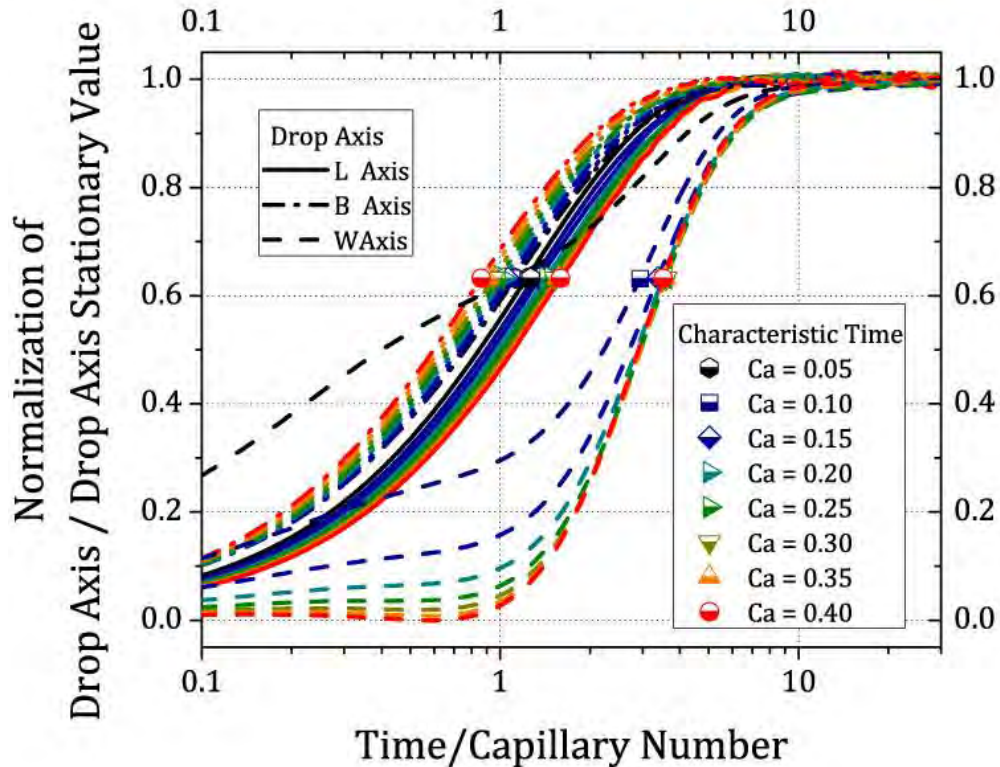
should consider as well as the characteristic time of the  $W$ -axis, because the steady state ought to be determined by the slowest axis. It is only then that we can be sure that the stationary state was obtained.



**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 a 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 than 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,  $\gamma$ . This method has the advantage of measuring  $\gamma$  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:

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

The constants  $\mu_1$  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  $\mu_2$ , 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),

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

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

$$\begin{aligned}\varepsilon \frac{D\mathbf{A}(t)}{Dt} &\equiv \varepsilon \frac{\partial \mathbf{A}(t)}{\partial t} + \mathbf{u} \cdot \nabla \mathbf{A}(t) = \varepsilon \frac{\partial \mathbf{A}(t)}{\partial t} - \varepsilon Ca \boldsymbol{\omega} \cdot \mathbf{A}(t) + \varepsilon Ca \mathbf{A}(t) \cdot \boldsymbol{\omega} \\ &= \frac{5Ca \mathbf{e}}{2\lambda_\mu + 3} - \frac{40(\lambda_\mu + 1) \varepsilon \mathbf{A}(t)}{(2\lambda_\mu + 3)(19\lambda_\mu + 16)} + \mathcal{O}(\varepsilon Ca, \varepsilon^2).\end{aligned}\quad (5.2)$$

Though complicated in appearance, the simple physical interpretation of Equation (5.2) is that the rate of change of the distortion  $\mathbf{A}(t)$  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  $\mathbf{e}$ , and secondly to the retraction effect of the surface tension (proportional to  $\mathbf{A}(t)$ ). 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  $\mathbf{A}(t)$  at steady state, obtaining

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

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

$$D_T = \frac{19\lambda_\mu + 16}{16\lambda_\mu + 16} Ca, \quad (5.3)$$

for the deformation parameter in *simple shear flow*.

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

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

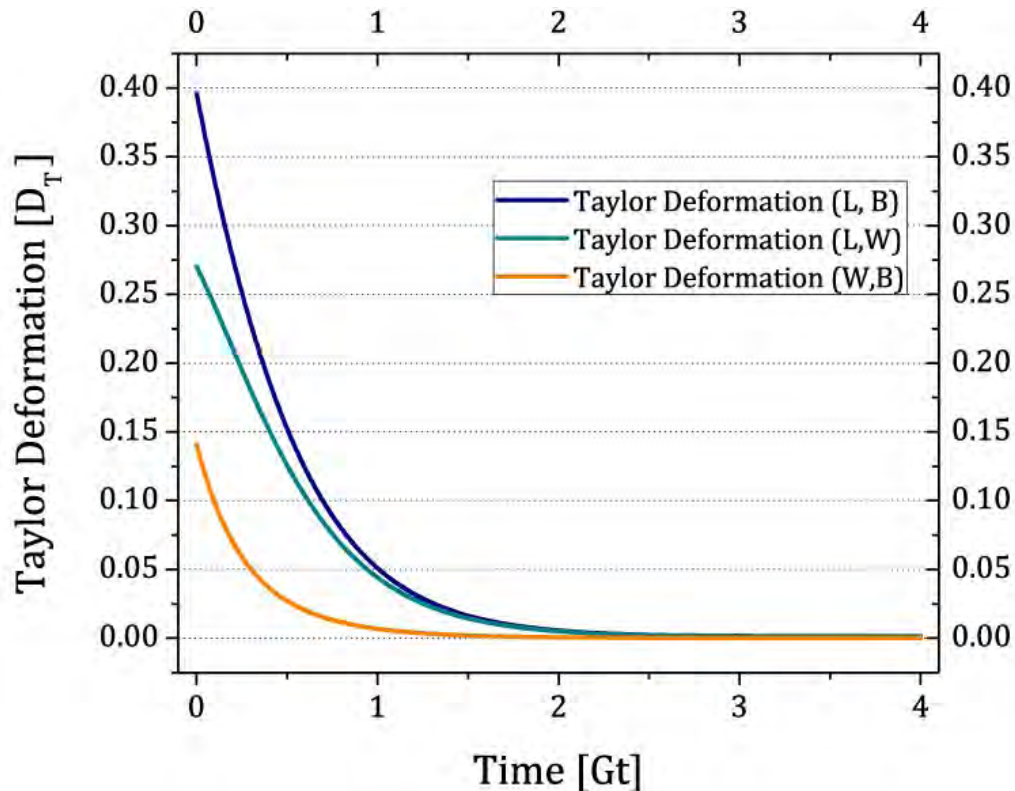
Finally, using Eq. (5.4), Eq. (5.1) is derived.

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,

$$D_{Mo} = L^2 - B^2 \quad (5.5)$$

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.

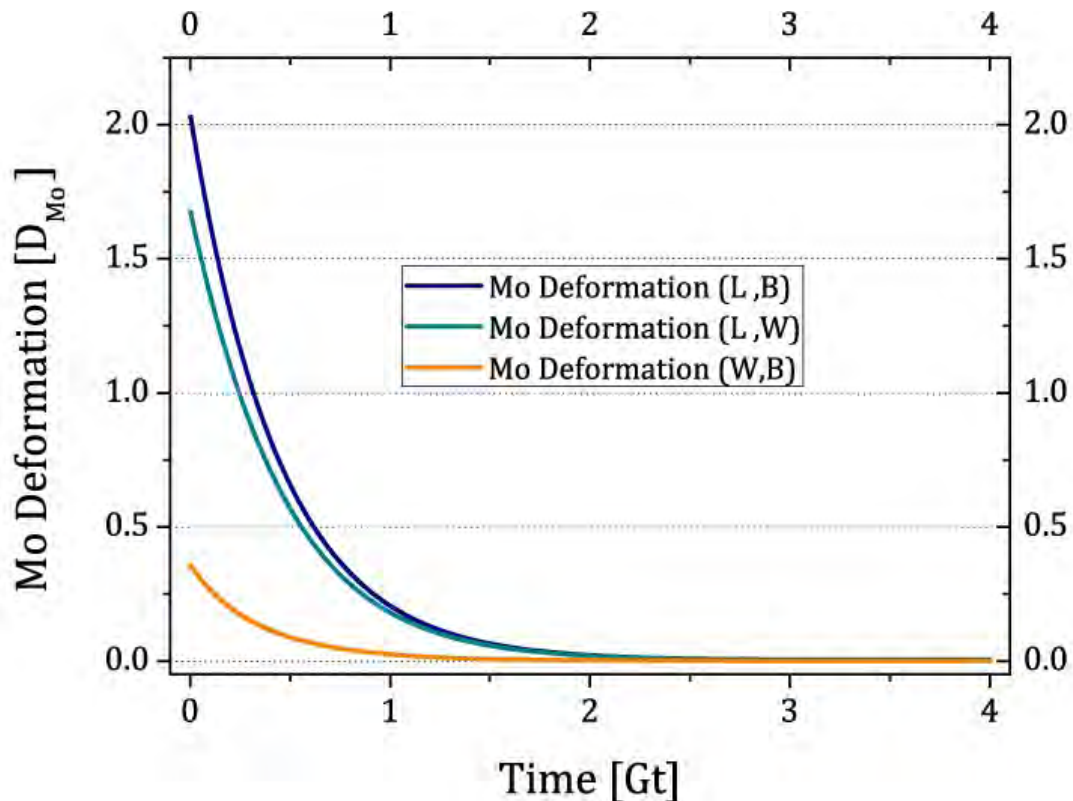


**Figure 5.12** Taylor Deformation of the drop vs time.  $\alpha = 0.13$ ,  $Ca = 0.40$ .

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.

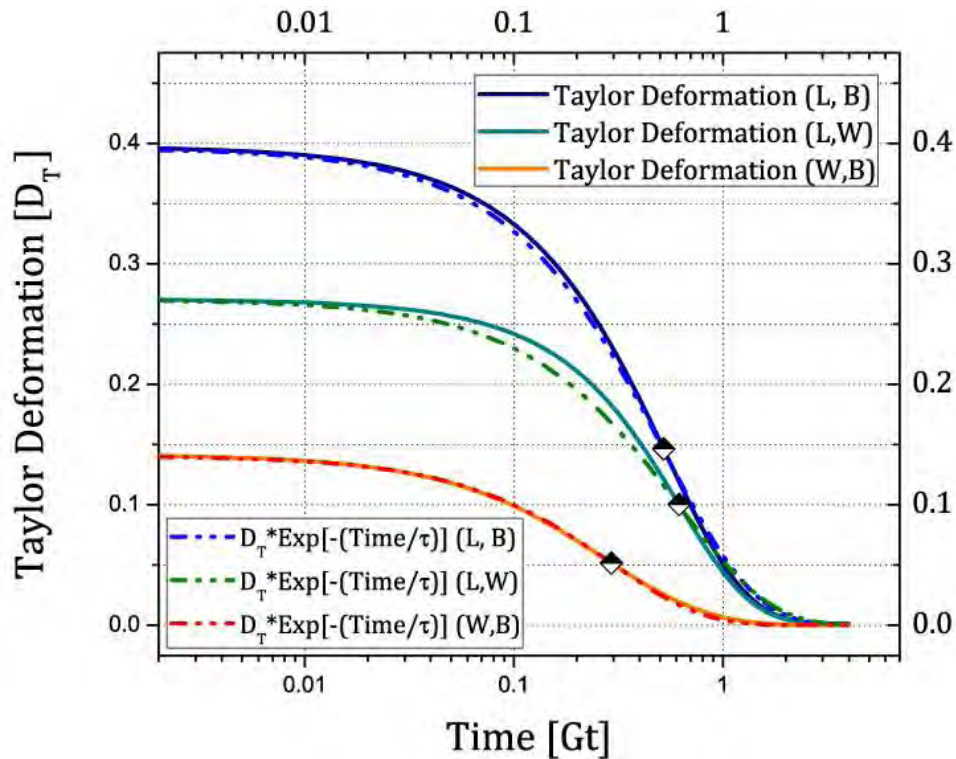


**Figure 5.13** Mo Shape Parameter vs Time.  $\alpha = 0.13$ ,  $Ca = 0.40$ .

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.

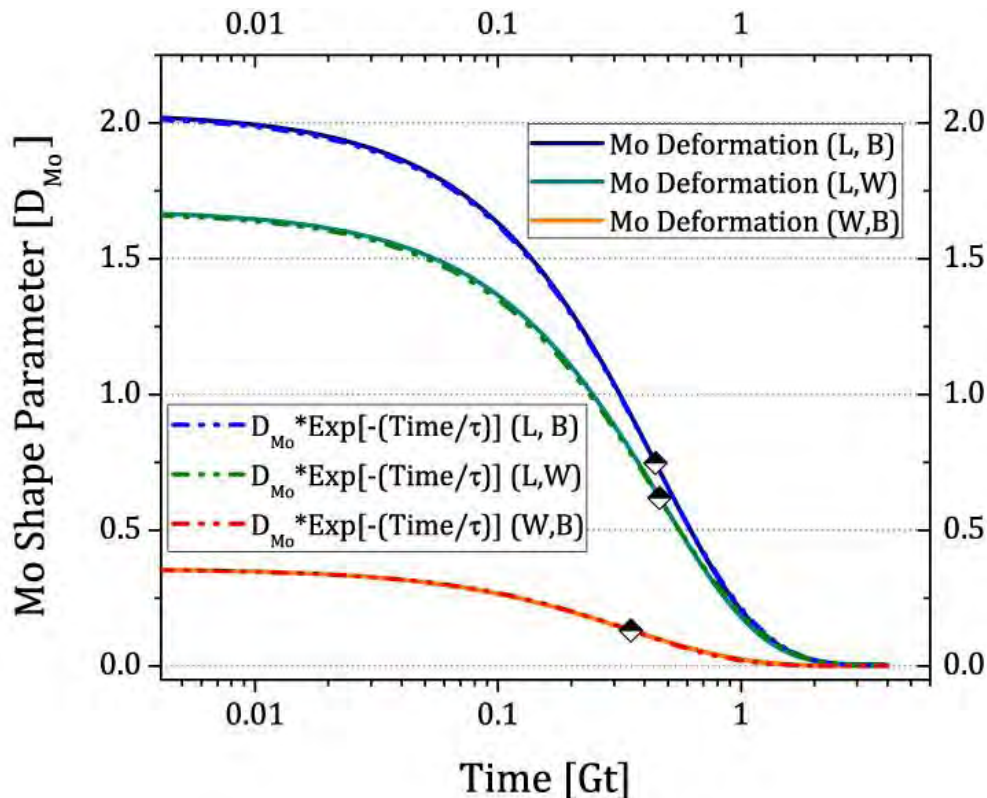
The exponential decay uses the characteristic time  $\tau$  in a model of Taylor Deformation and of the  $Mo$  shape parameter. Fig. 5.14 shows that the retraction evolution model as an exponential decay, is a good approximation for cases when the Taylor deformation involves the  $L$ - and  $B$ -axes ( $xy$ -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  $\tau$  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  $\tau$  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  $\tau$  (diamonds).  $\alpha = 0.13$ ,  $Ca = 0.40$ .



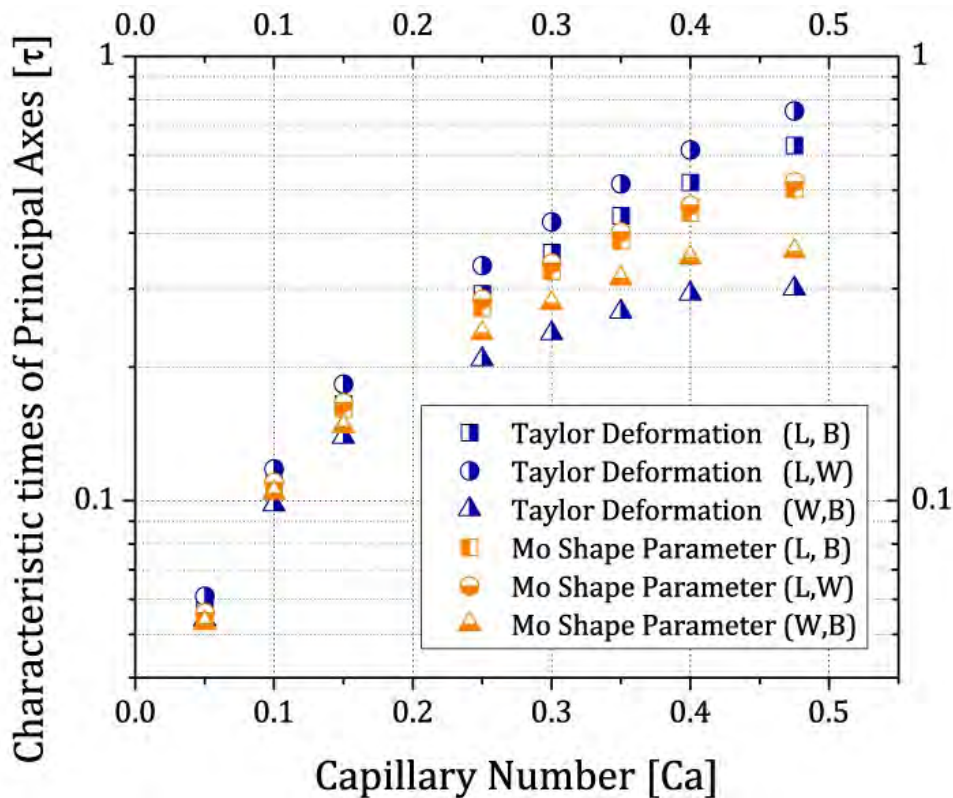
The analysis for the  $Mo$  shape parameter was made as the case before, using the principal axes of the drop to estimate the deformations. The result of the characteristic time for a drop with  $Ca = 0.40$  and a parameter of flow  $\alpha = 0.13$ , is shown in Fig 5.15. The match of the evolution of 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  $\tau$  (diamonds).  $\alpha=0.13$ ,  $Ca=0.40$ .

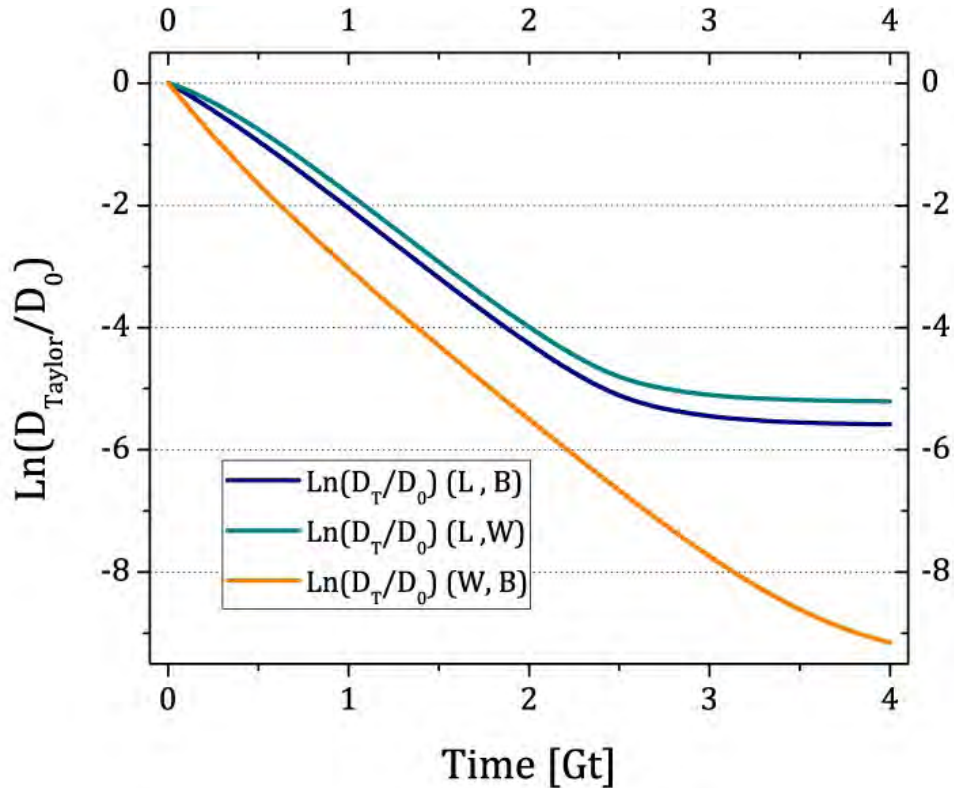
The characteristic time of deformation on the principal plane is plotted in Fig. 5.16. All values of the characteristic time using the  $Mo$ -shape parameter and Taylor Deformation appear to be different in each plane of observation. The characteristic time increases too when the capillary number grows. 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.

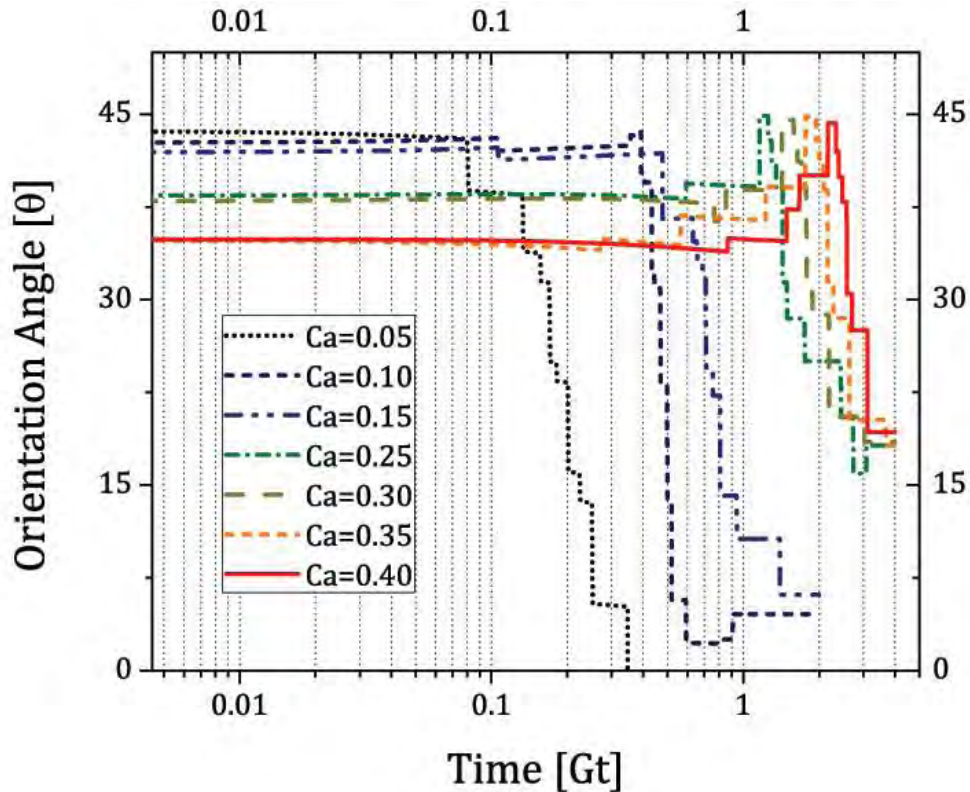


**Figure 5.17**  $\ln(D_T/D_0)$  vs. time. Drop retraction was analyzed with Taylor deformation which  $Ca = 40$  and  $\alpha = 0.13$ .

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.

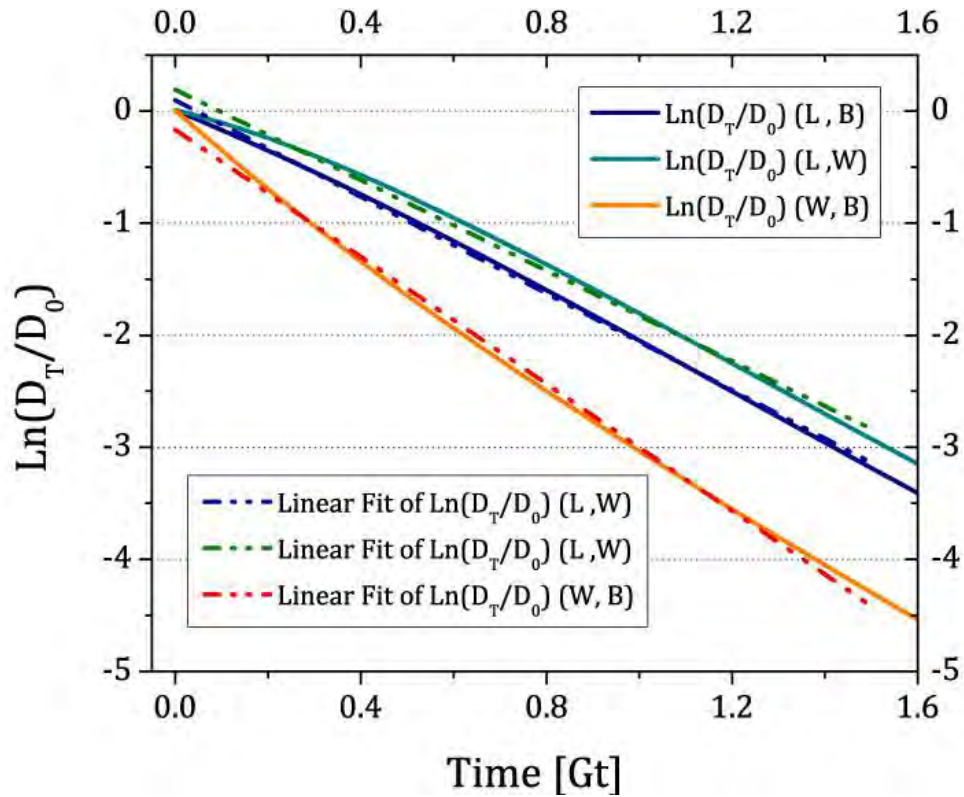
Therefore, the DDR model must assume a clean linear slope due to the logarithmic dependency, with no consideration to residual deformations (either experimental or theoretical). With this fact, the optimum evaluation of the interfacial tension should be based on deformations before the drop attains its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



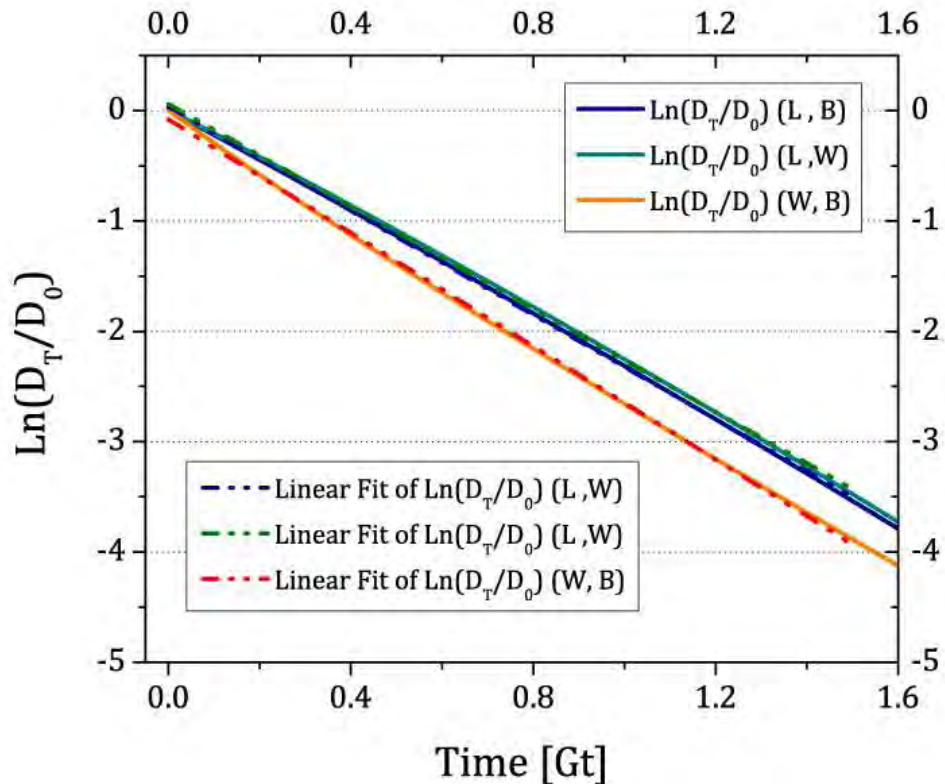
**Figure 5.18** Angle of Orientation vs. time in drop retraction which parameter  $\alpha = 0.13$ .

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  $-\text{Ln}(D_T/D_0)$  for  $5\tau$  of time. Drop retraction was analyzed with Taylor deformation which  $Ca = 40$  and  $\alpha = 0.13$ .



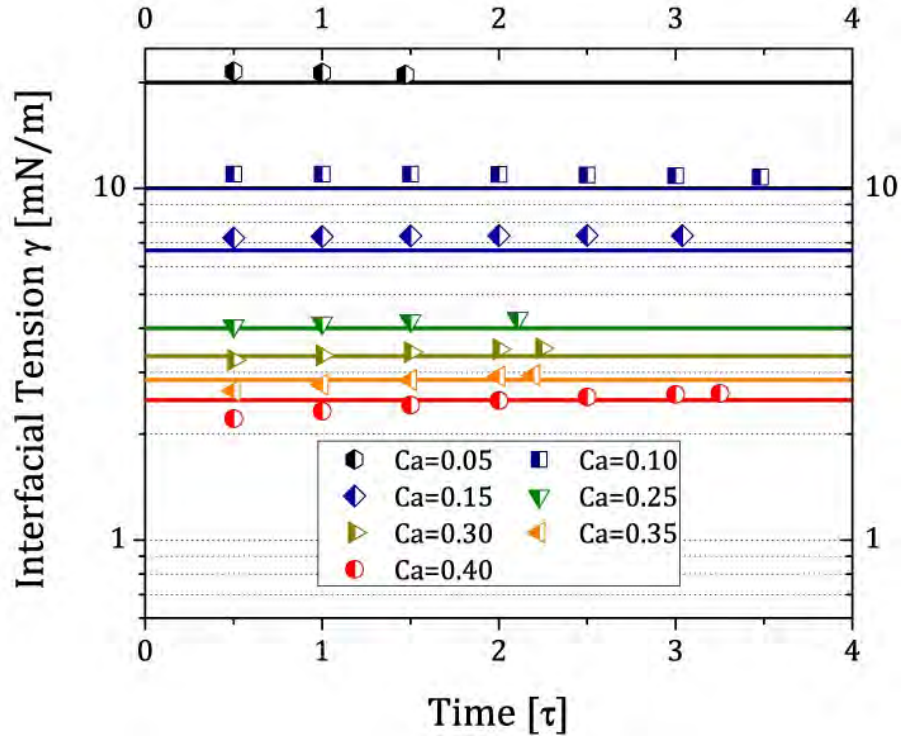
**Figure 5.20** Linear fit of  $-\text{Ln}(D_{Mo}/D_0)$  for  $5\tau$  of time. Drop retraction was analyzed with Mo shape parameter which  $Ca = 40$  and  $\alpha = 0.13$ .

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.

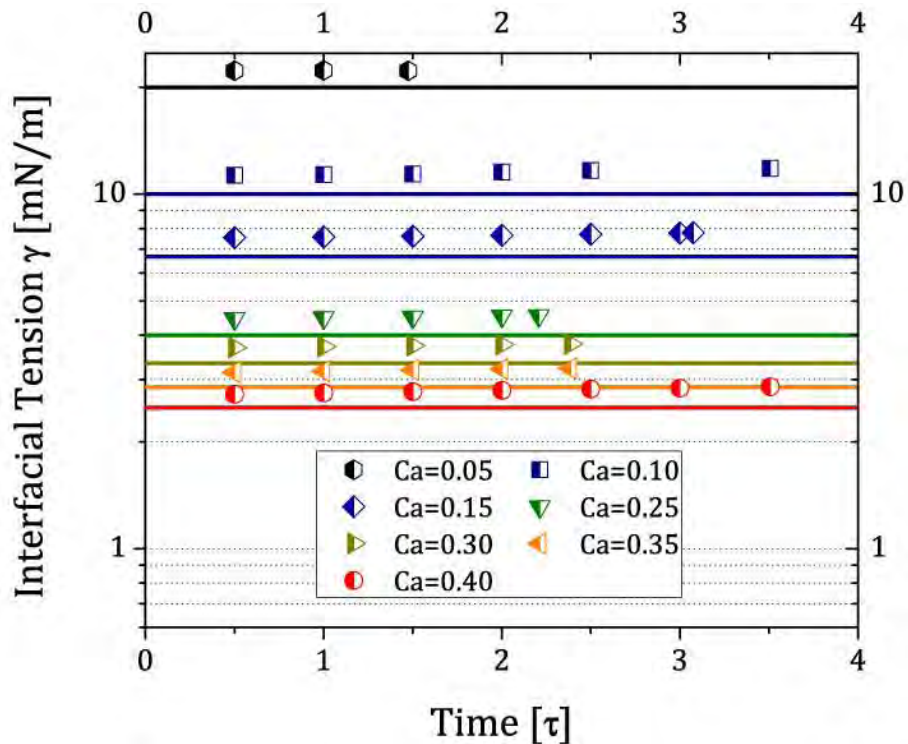
Figures 5.21 and 5.22 indicate with lines the value of the interfacial tension introduced in the numerical scheme. The values obtained with *DDR method* correspond to the symbols. Using multiples of the characteristic time, which was obtained in the same form than in Section 5.1 for the retraction phenomenon 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  $\tau$ . 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  $2\tau$  and  $3\tau$ . Figures 5.23 and 5.24 present the systematic error on the interfacial tension values when using different intervals of the characteristic time  $\tau$ . These errors are presented with respect to the Taylor Deformation and the *Mo-shape* parameter in the *DDR method*.



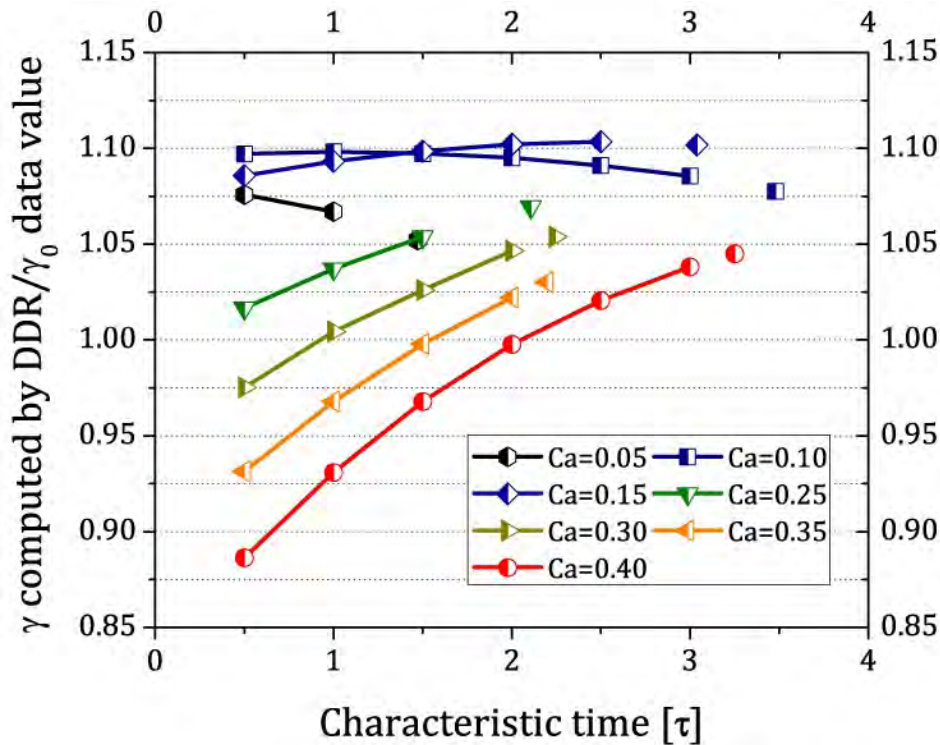
**Figure 5.21** Comparison between the interfacial tension used in numerical simulations (lines) and the interfacial tension obtained with DDR method (symbols). The analysis was made with Taylor deformation, the slopes used were based on multiples of  $\tau$ .



**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  $\tau$ .



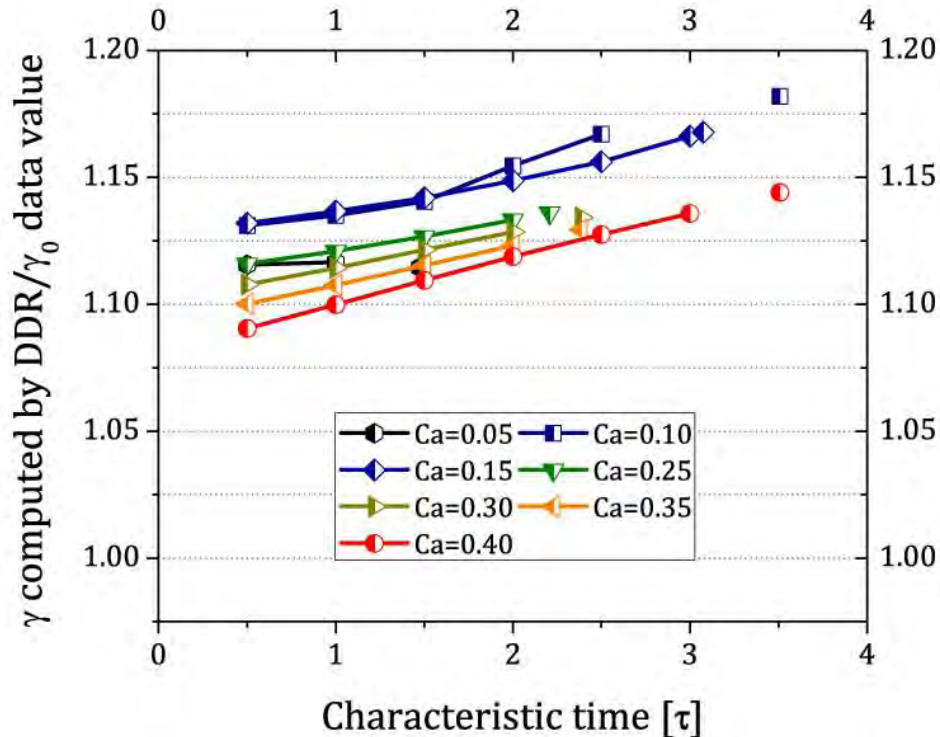
Based on the numerical simulation, the optimal time to obtain the best slope for estimation of the interfacial tension is  $Time \approx 2\tau$ . Plots shown in Figs. 5.23 and 5.25 are now linear and the error is shown to be less than +10%. When values of interfacial tension are large -black color- errors are less than 5% for  $1.5\tau$ . The interfacial tension best value could be obtained at  $2\tau$ , but the orientation angle has changed. If values of  $\gamma$  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  $\gamma$ . A similar accurate prediction is feasible at  $2\tau$ . However, the estimation of the interfacial tension for low capillary numbers —bigger values of  $\gamma$ — have an accuracy of less than 10%; the difference in this case is the over-estimation of  $\gamma$ . 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  $\tau$ .

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  $-\ln(D_{M0}/D_0)$  appears better than that of  $-\ln(D_T/D_0)$ . However, values of  $\gamma$  appear to be less accurate than the case of Taylor Deformation.

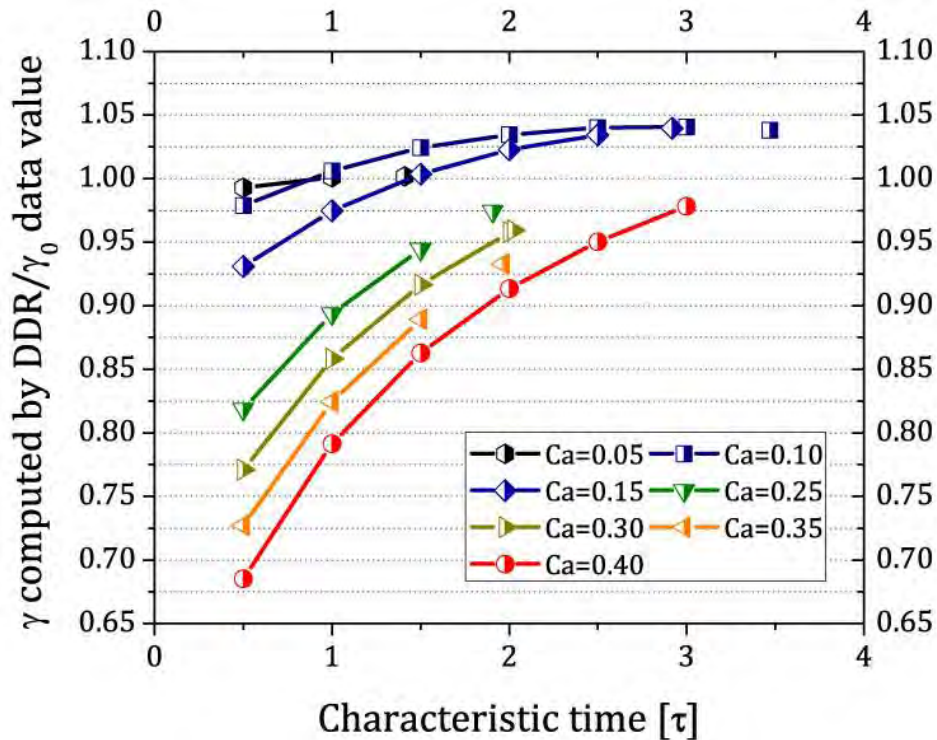
In Figure 5.22, is evident the difference between the interfacial tension used and the interfacial tension estimated. Figure 5.24 indicates that the error is always bigger than 7.5%. In the cases of  $0.15 \leq Ca \leq 0.40$ , the optimal time to obtain interfacial tension is in  $0.5\tau \leq Time \leq 2\tau$ . 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  $\tau$ .

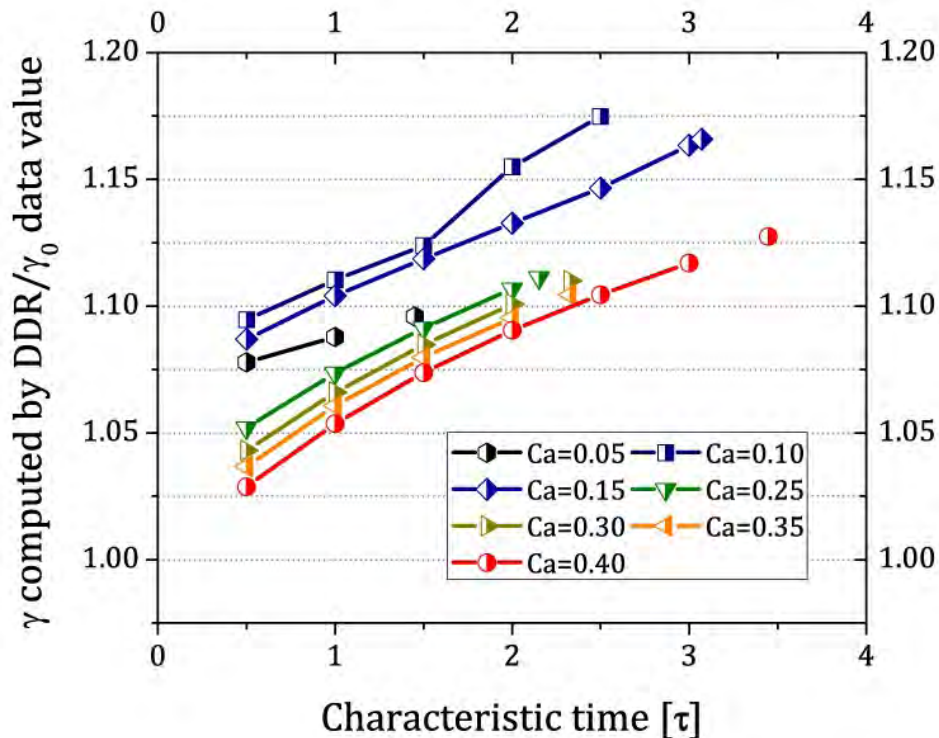
Figures 5.23 and 5.24 indicate that the use of Taylor deformation appear to predict better values of interfacial tension when  $\gamma$  is less than 5 [mN/m]. The method does not seem to be sufficiently accurate for values from 5 to 15 [mN/m].

There are other *DDR methods*, 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  $L' = \mathbf{P} \cdot L$ , on the *xz-plane*;  $\mathbf{P}$  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:  $L'$ . 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  $\tau$ .

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

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  $5\% \leq \text{error} \leq 10\%$  in the case of  $0.5\tau$  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  $1.5\tau$ . 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  $Ca$ ,  $\lambda_\mu$  and the flow parameter  $\alpha$ . 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  $\gamma$  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 an 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*,  $\dot{\epsilon}$ . 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  $\alpha$ , thus here the Figures employed to illustrate the relevant results correspond to only the regime for  $\alpha = 0.13$ .





Universidad Nacional  
Autónoma de México

Dirección General de Bibliotecas de la UNAM

**Biblioteca Central**



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

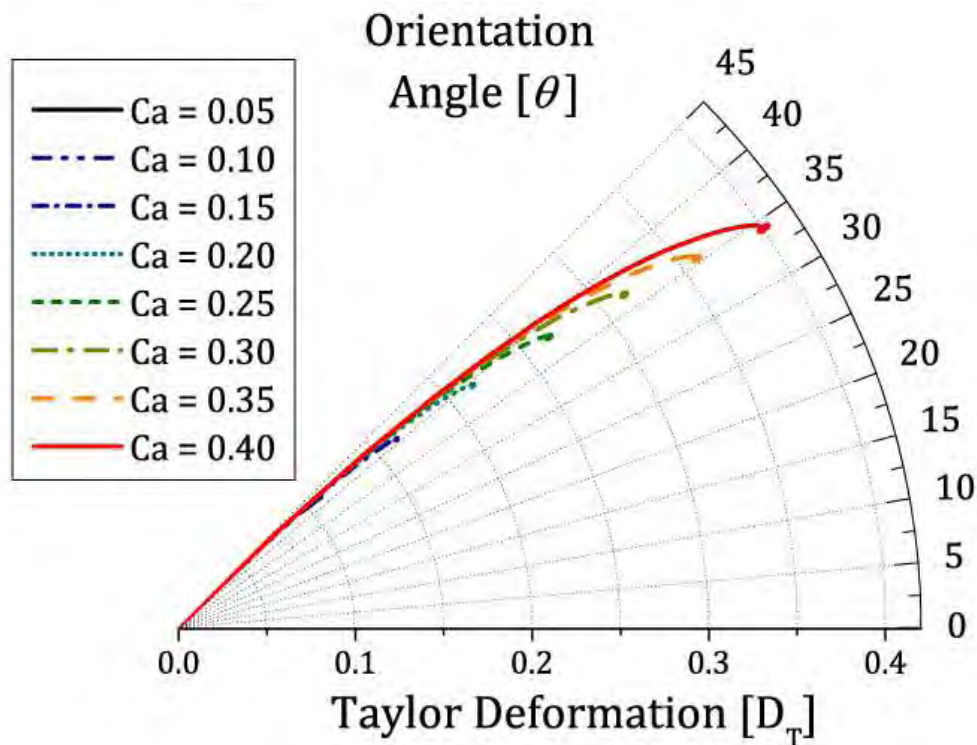
**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

## 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.

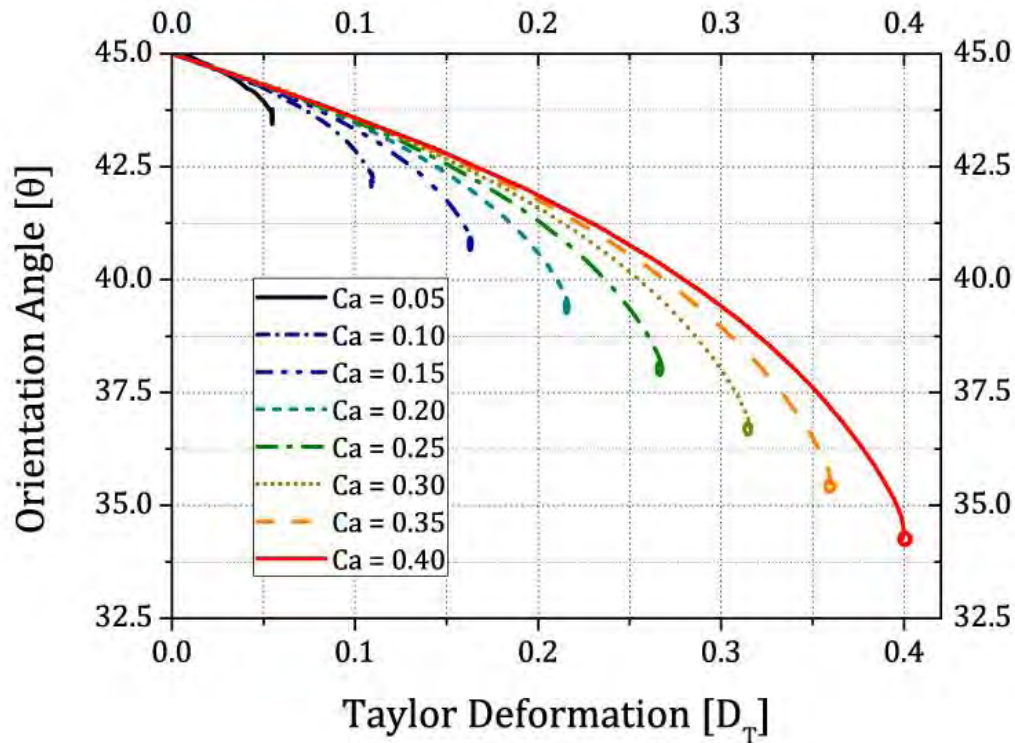


**Figure 6.1** Polar representation of Angle of Orientation vs. Taylor Deformation with  $\alpha = 0.13$  and  $\lambda_\mu = 0.012$ .

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.

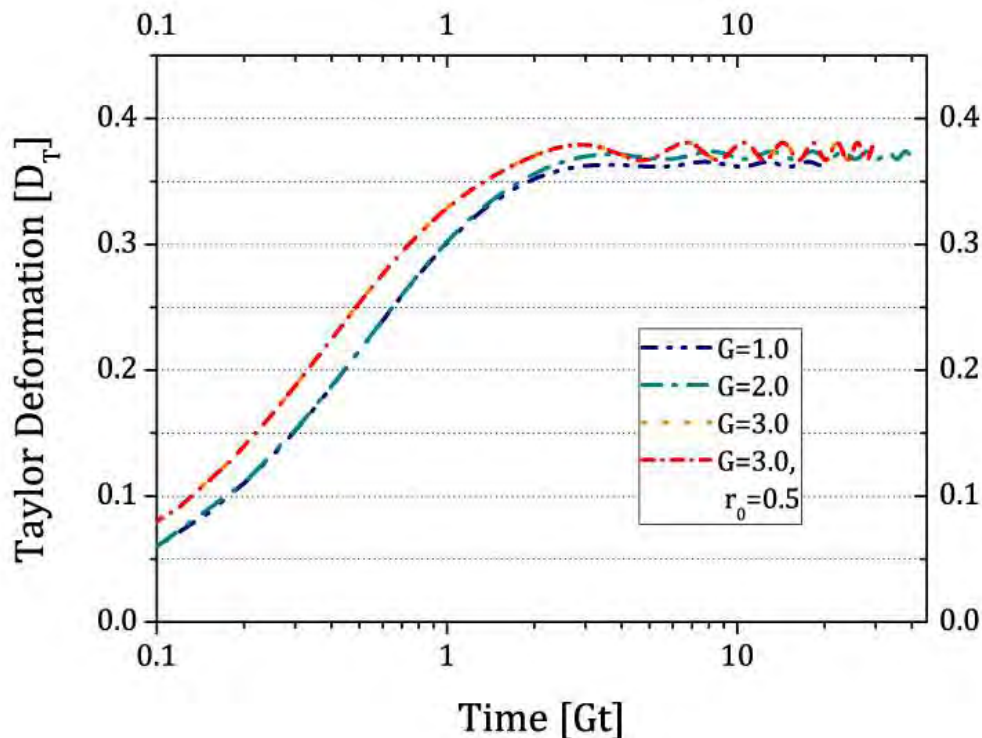


**Figure 6.2** Angle of Orientation vs. Taylor Deformation with  $\alpha = 0.13$  and  $\lambda_\mu = 0.012$ .

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.

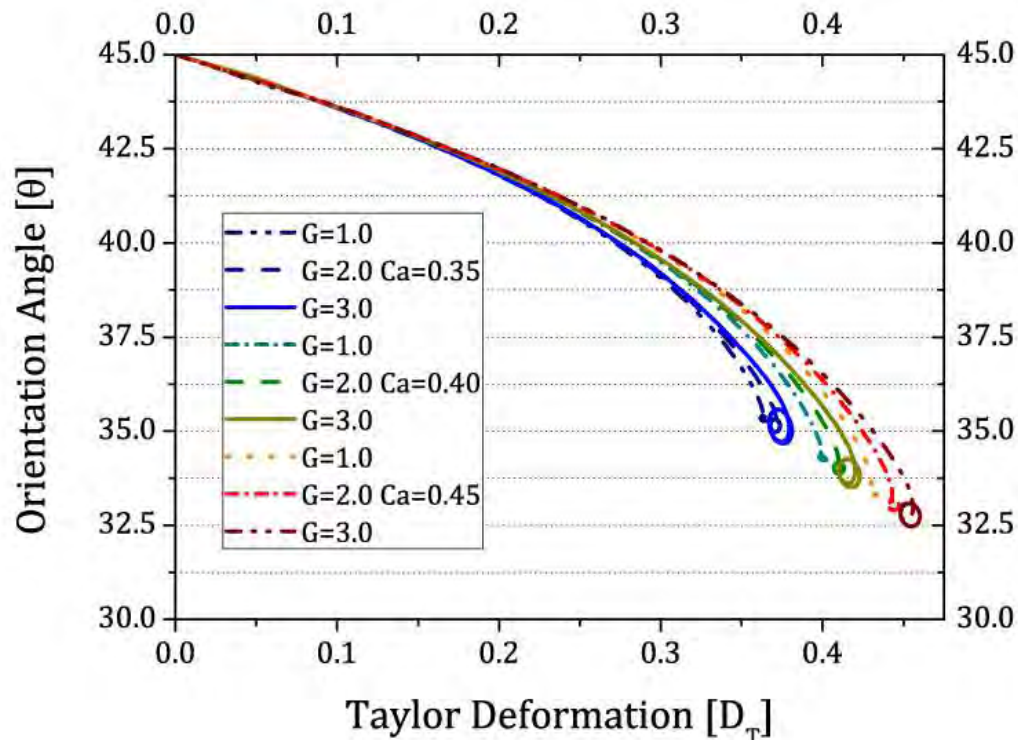
Comparing the polar plots data of Reyes (Reyes, 2005) and Rosas (Rosas, Reyes, Minzoni, & Geffroy, 2014) the *2D-numerical* and the experimental data referred have a similar behavior (trajectories around a point). However, the analysis of these curved trajectories around a point was omitted because the 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.

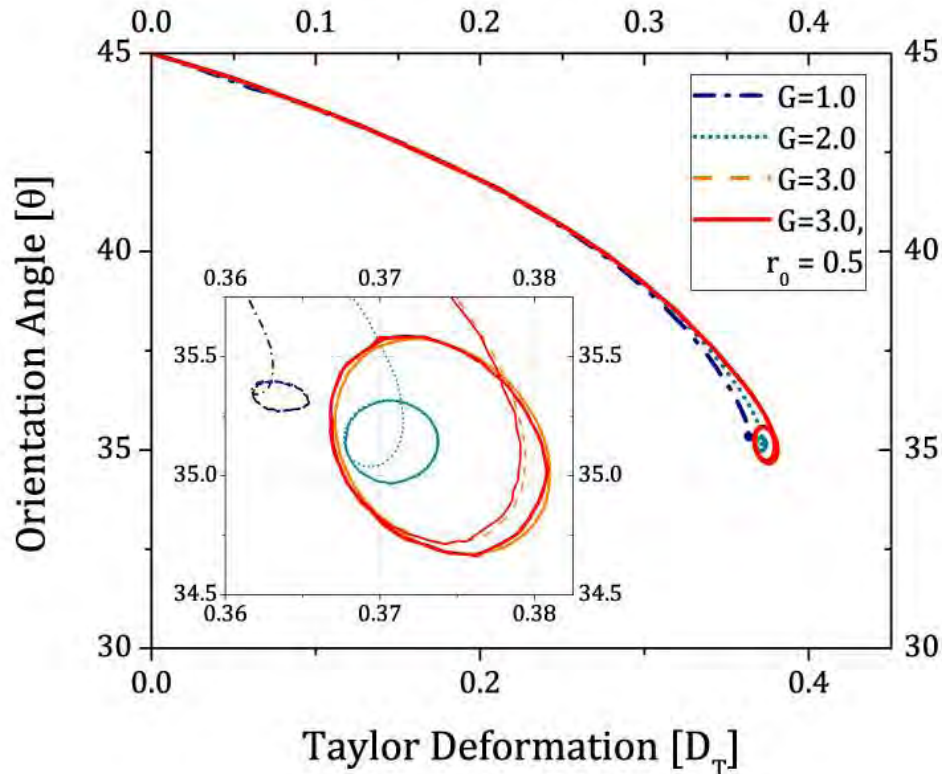


**Figure 6.4** Angle of Orientation vs Taylor Deformation for drops with the same Capillary Number with different values of  $G$

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.*



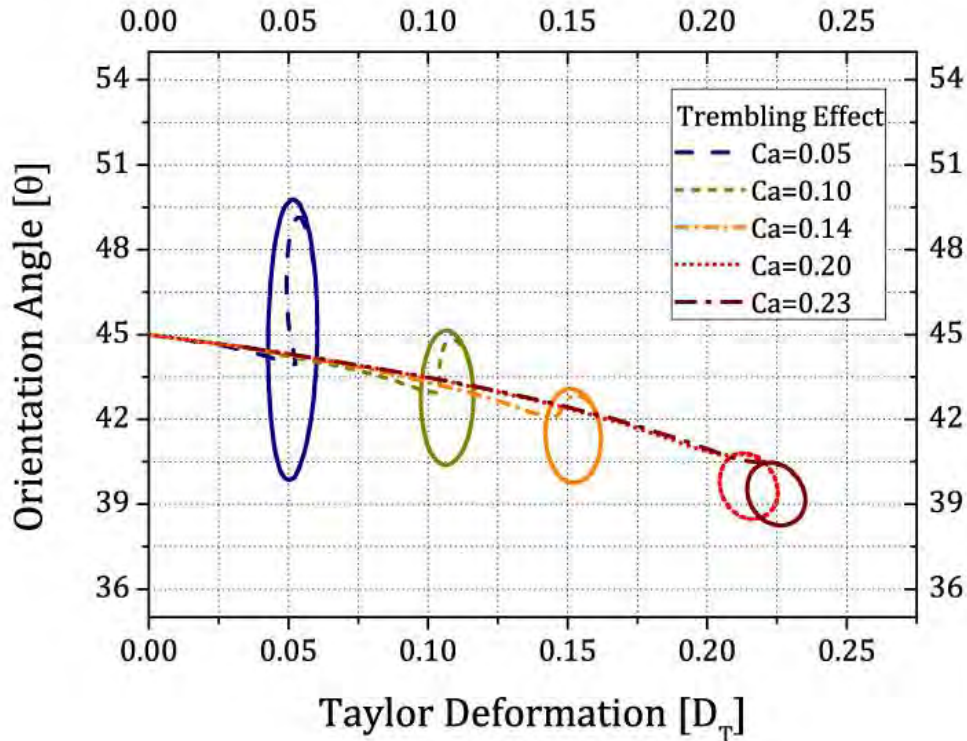
**Figure 6.5** Angle of Orientation vs. Capillary Number for  $Ca = 0.35$  and different values of  $G$ .

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  $G \gg 1$ ,  $\alpha = 0.13$ ,  $\lambda_\mu = 0.01$ .

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

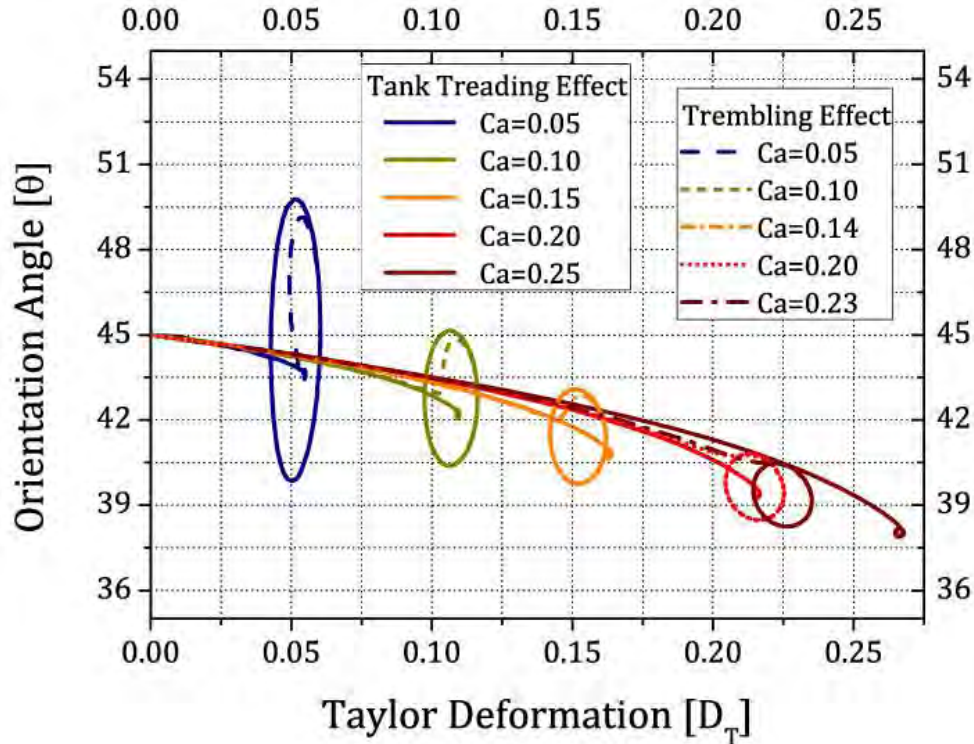
reorientation are observed; *i.e.*, *the deformed vesicle trembles*. The plot in time of angle vs. deformation of this effect in vesicles is like the trajectories observed in Figs. 6.4, when  $G > 1$ . Figure 6.5 shows an analysis in the steady regime when  $Ca = 0.35$ , but different values of  $G$ . 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.

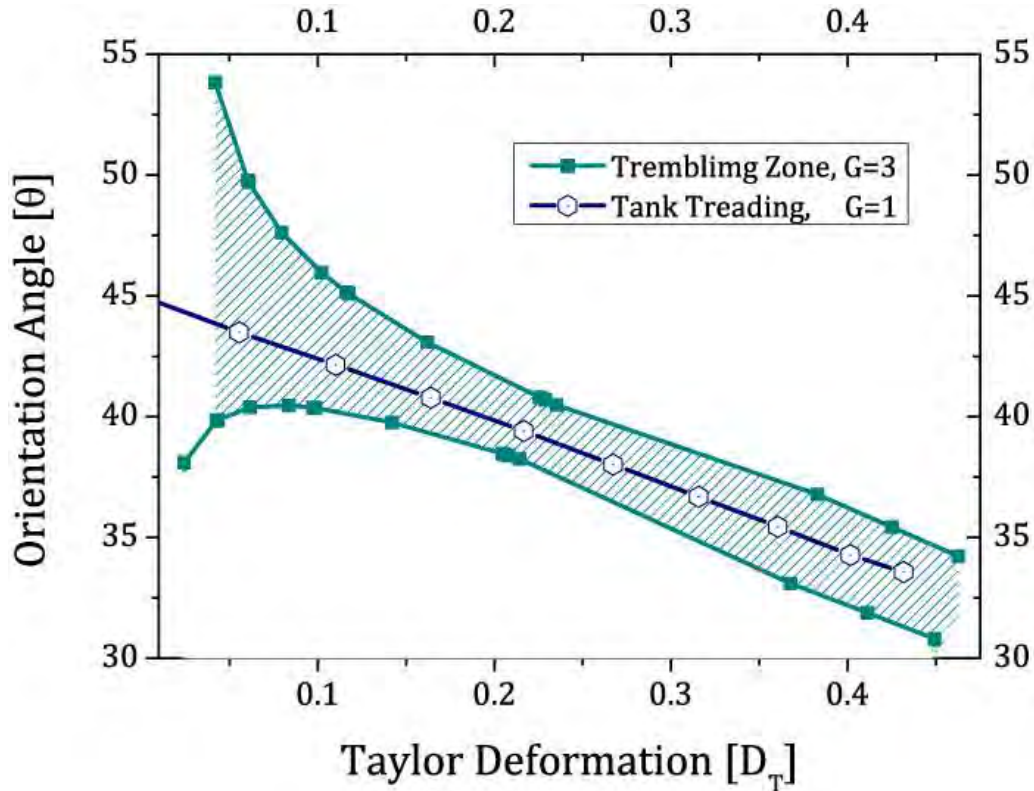
Figure 6.7 shows different drop simulations for cases when values of  $G$  are larger than 1, and the *trembling* solution is observed, in all cases. As the capillary number is reduced, the observed closed trajectories become more elongated. In other words, for a small capillary number and for very large  $G$  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.

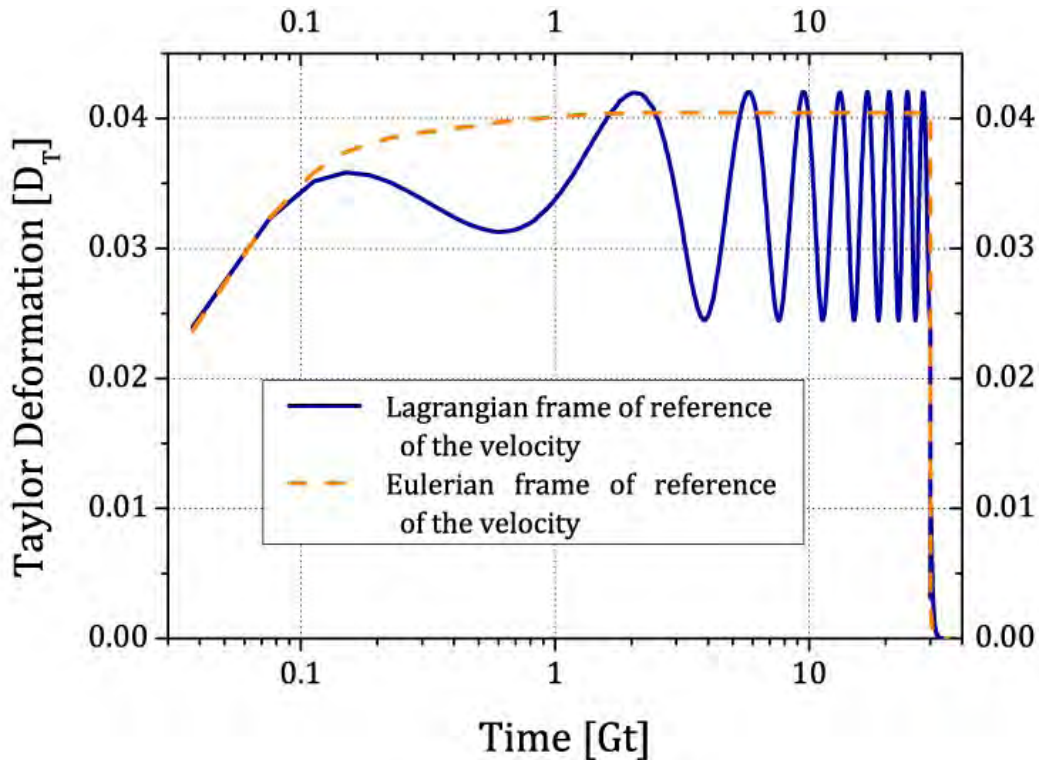


**Figure 6.8** Comparison of the trajectories of steady states for Angle of Orientation vs. Taylor Deformation of different Capillary Number, but different values of  $G$ , with  $\alpha = 0.13$  and  $\lambda_\mu = 0.012$ . The blue symbols are the steady states of drop deformation for capillary numbers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

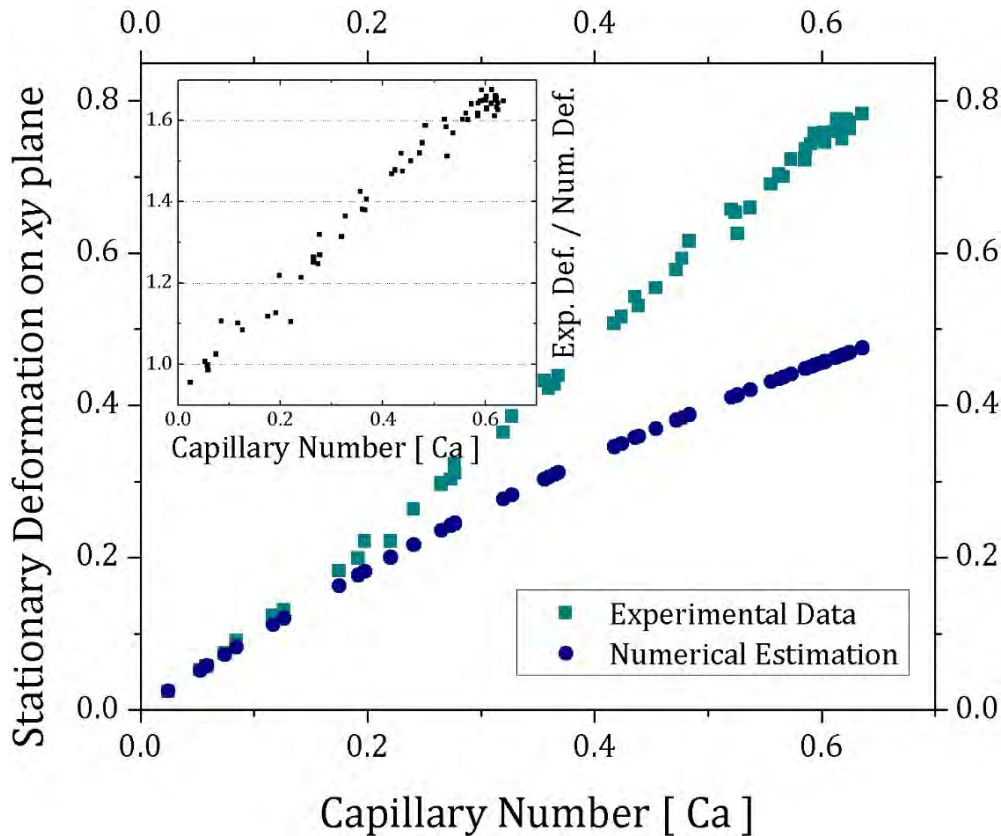
In the previous Section, I emphasize the importance of *the intensity of the flow* for the selection of the solution describing the drop deformation elucidating the *tank-treading*, *trembling* and other possible solutions. That is, the numerical experiments maintained the values of the capillary number 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  $Ca < 0.2$ : 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  $\alpha$  increases, differences decrease from 60% (for  $\alpha = 0.03$ ) to 25% (for  $\alpha = 0.13$ ).

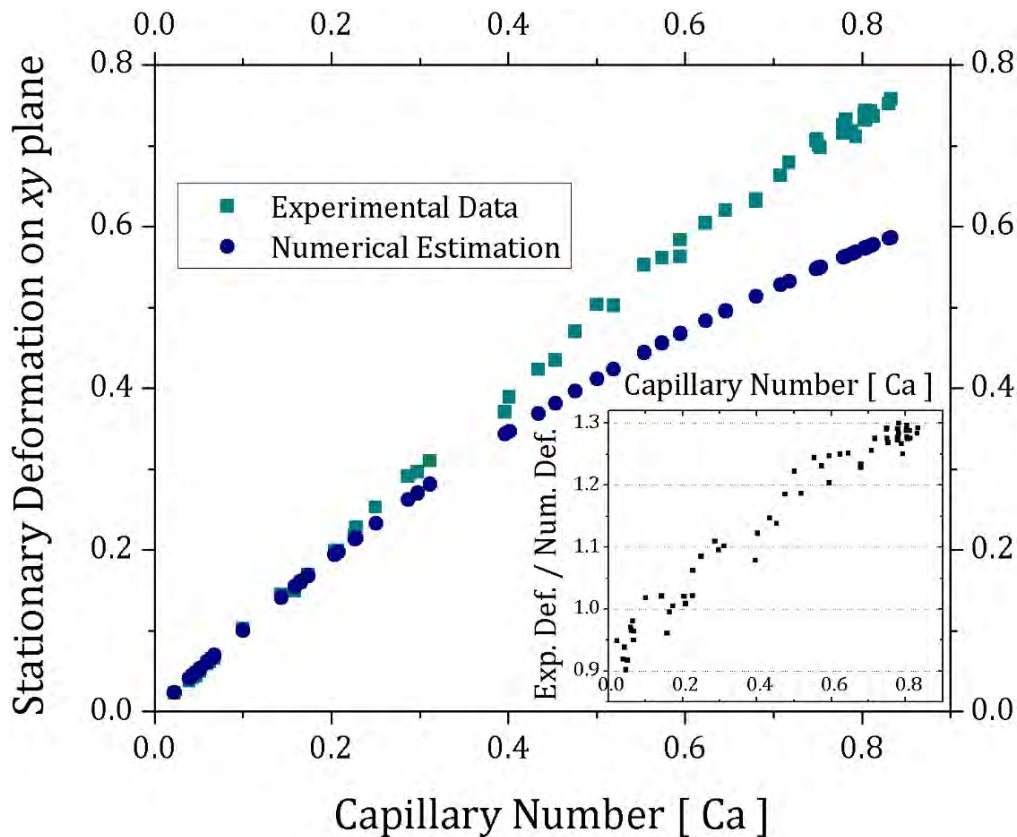


**Figure 6.10** Comparison between experimental and numerical data of steady deformation on  $xy$  plane vs capillary numbers with  $\alpha = 0.03$ ,  $\lambda_\mu = 0.012$ . The insert Figure is the numerical error w.r.t. the experimental data.

The input data for numerical experiments, employed in previous Chapters, were selected based on the best match between numerical vs. experimental data. The

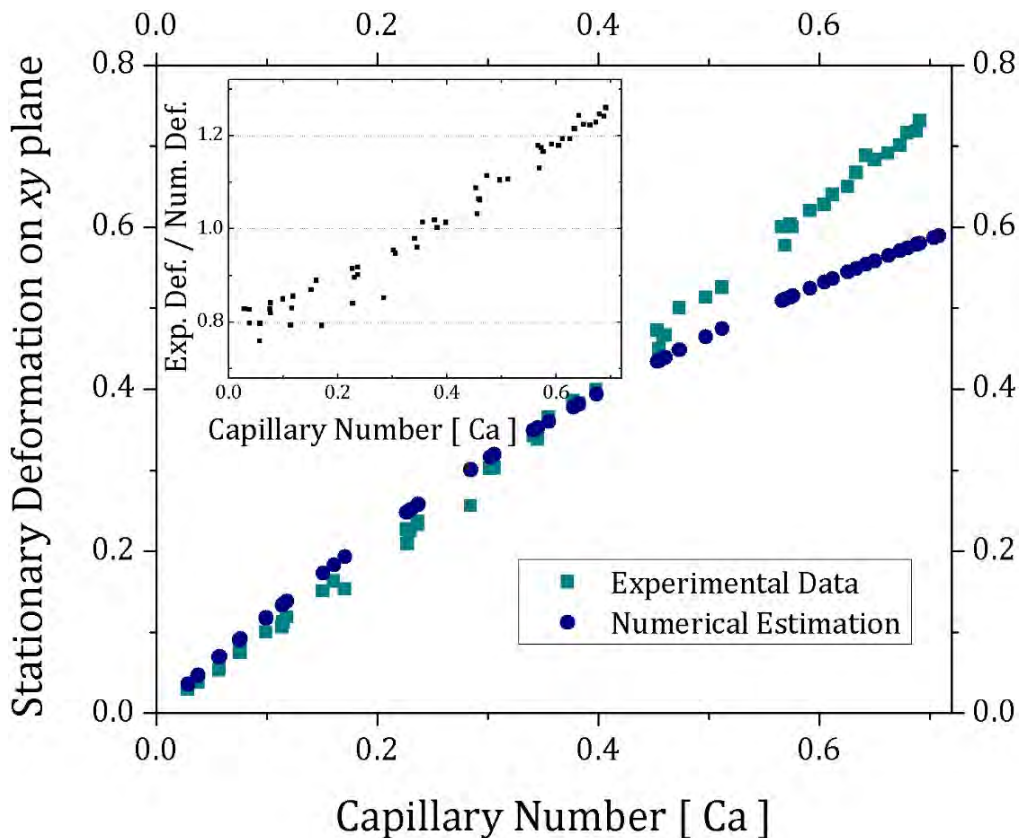
next *question to address* was to study and understand possible reasons for the failure of the numerical predictions of the drop deformation during the steady state phase and for the larger capillary numbers. The main motivation was the possibility of having an accurate numerical method to simulate large 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.



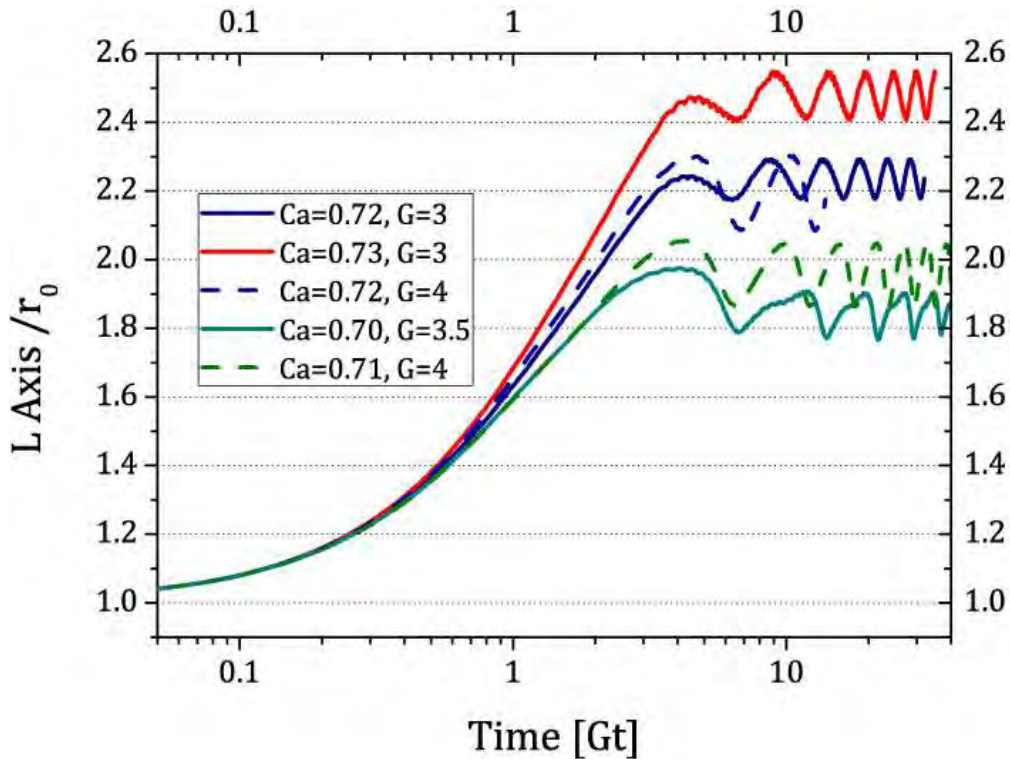
**Figure 6.12** Comparison between experimental and numerical data of steady deformation on  $xy$  plane vs capillary numbers with  $\alpha = 0.13$ ,  $\lambda_\mu = 0.012$ . The inset Figure is the numerical error w.r.t. the experimental data.

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 simula-

tions 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 be 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.



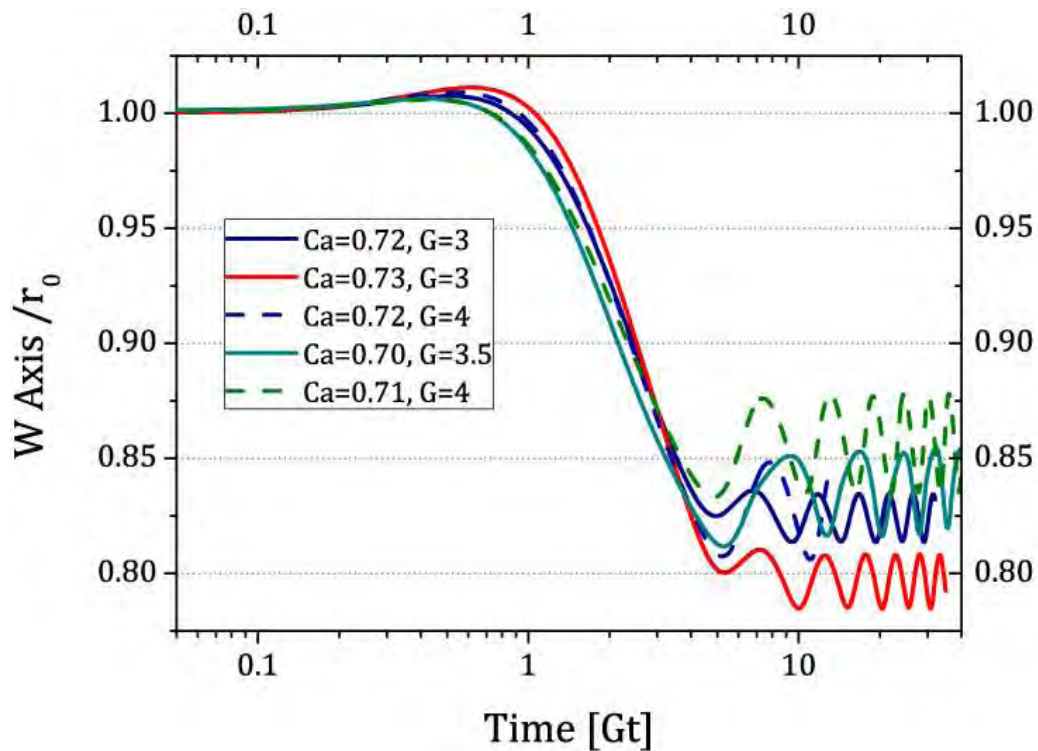


Figure 6.14 Comparison of the  $W$ -axes when capillary number is increased as function of Parameter  $G$ , which  $\lambda_\mu = 0.012$ ,  $\alpha = 0.13$ .

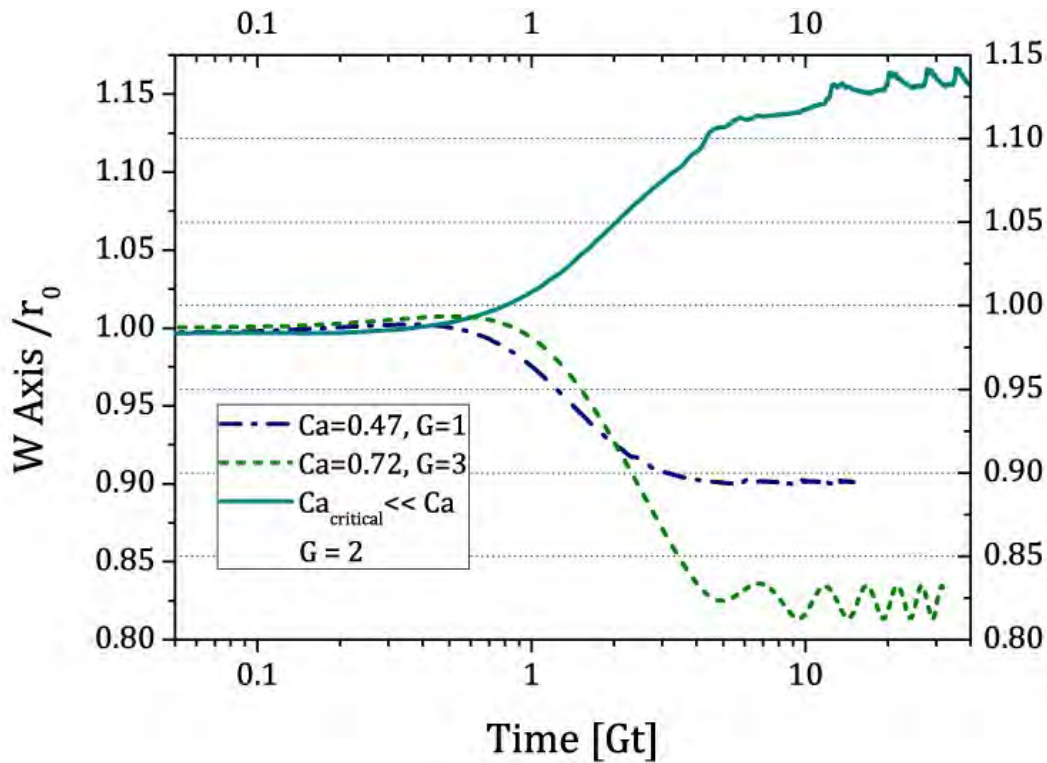


Figure 6.15 Comparison of the  $W$ -axes when  $Ca_{cr} \ll Ca$ , which  $\lambda_\mu = 0.012$ ,  $\alpha = 0.13$ .



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.

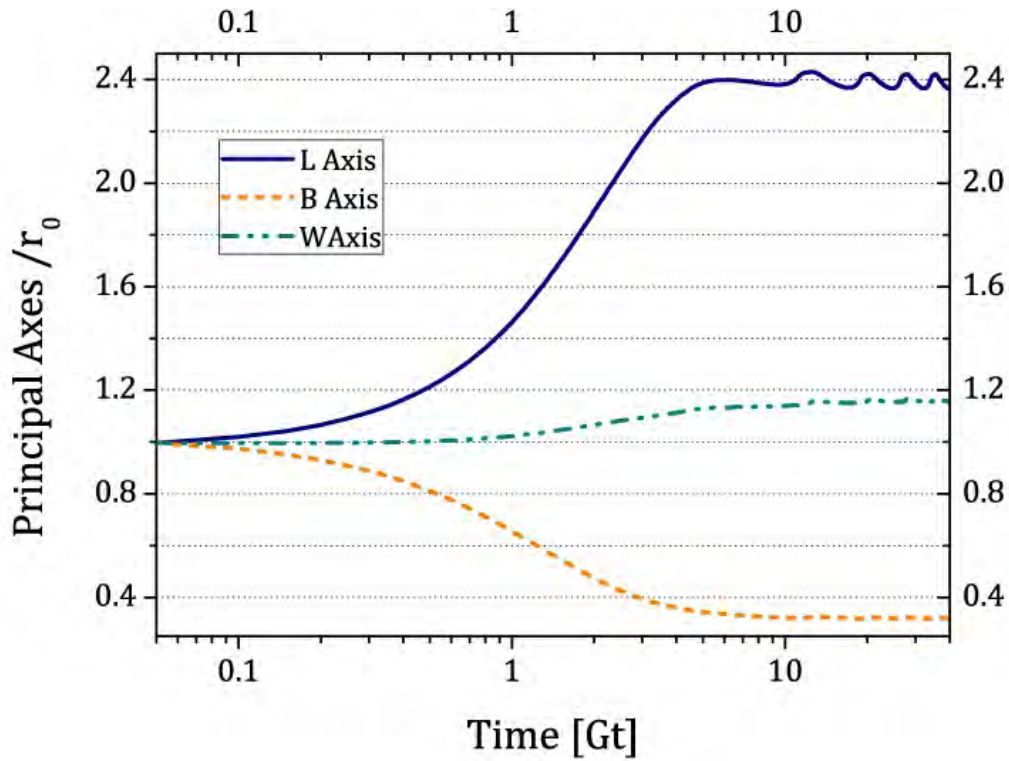


Figure 6.16 Principal Axes evolution in the time for a drop with  $Ca_{cr} \ll Ca$ ,  $\lambda_\mu = 0.012$ ,  $\alpha = 0.13$ .

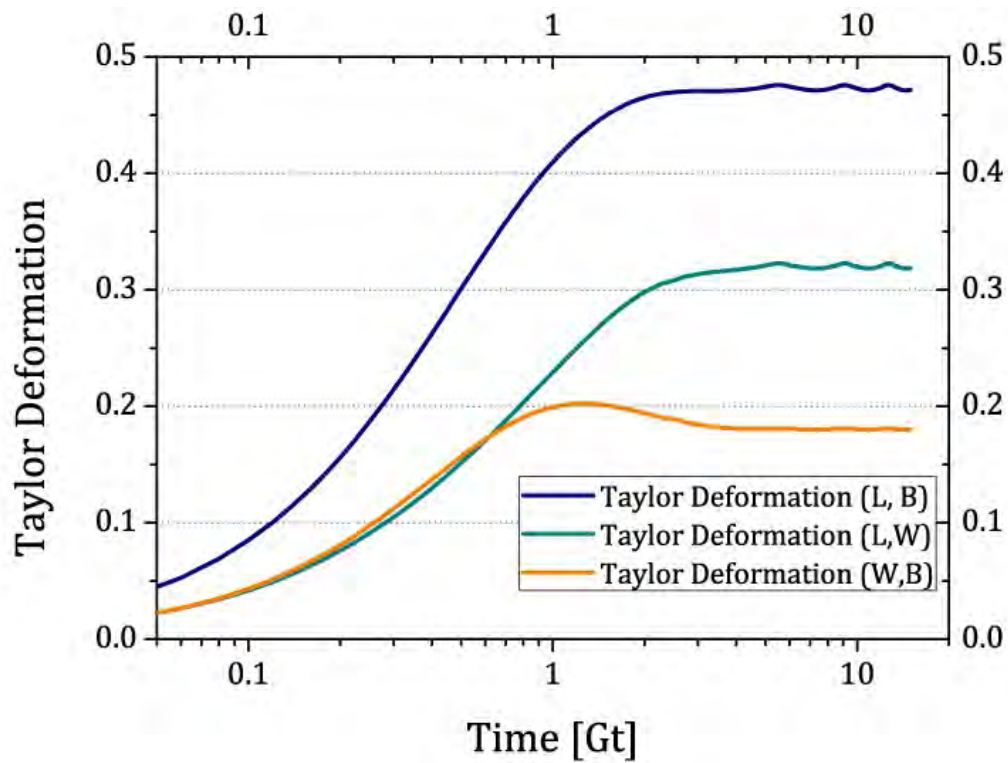
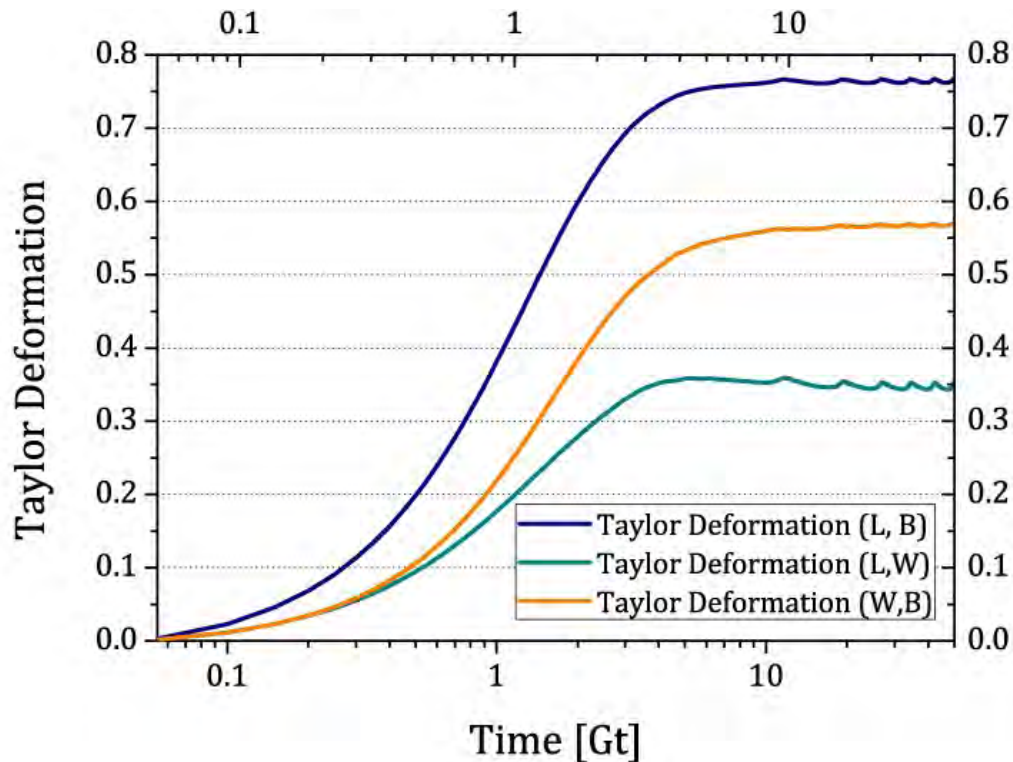


Figure 6.17 Taylor Deformation in different planes:  $Ca = 0.40$ ,  $\lambda_\mu = 0.012$ ,  $\alpha = 0.13$ .

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.

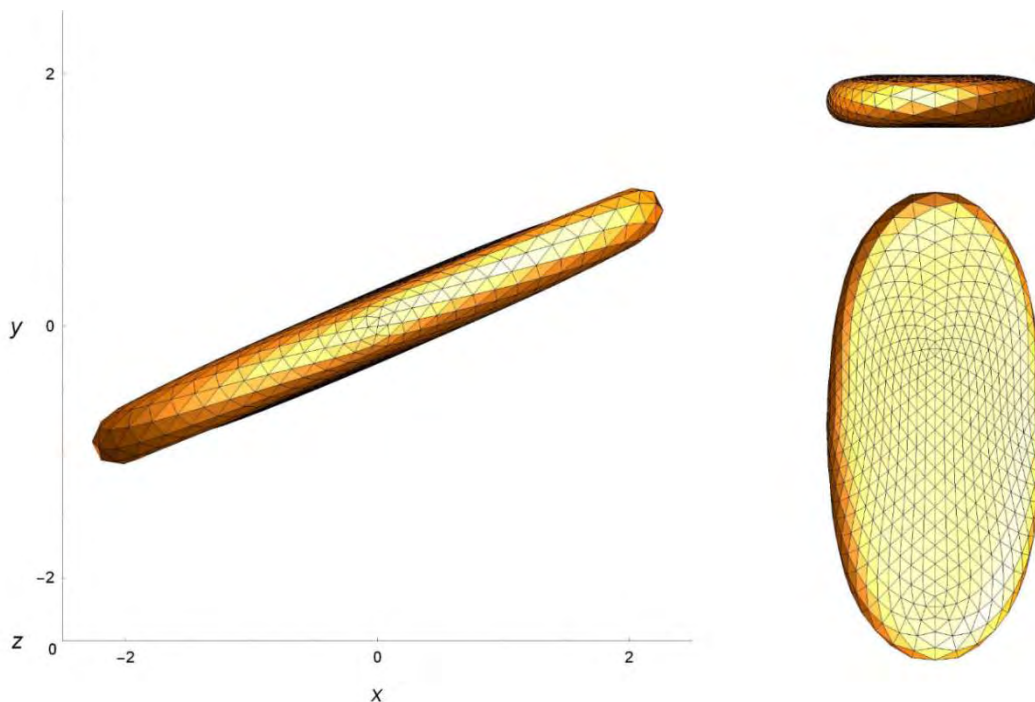


**Figure 6.18** Taylor Deformation with  $Ca_{cr} \ll Ca$ ,  $\lambda_\mu = 0.012$ ,  $\alpha = 0.13$ .

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

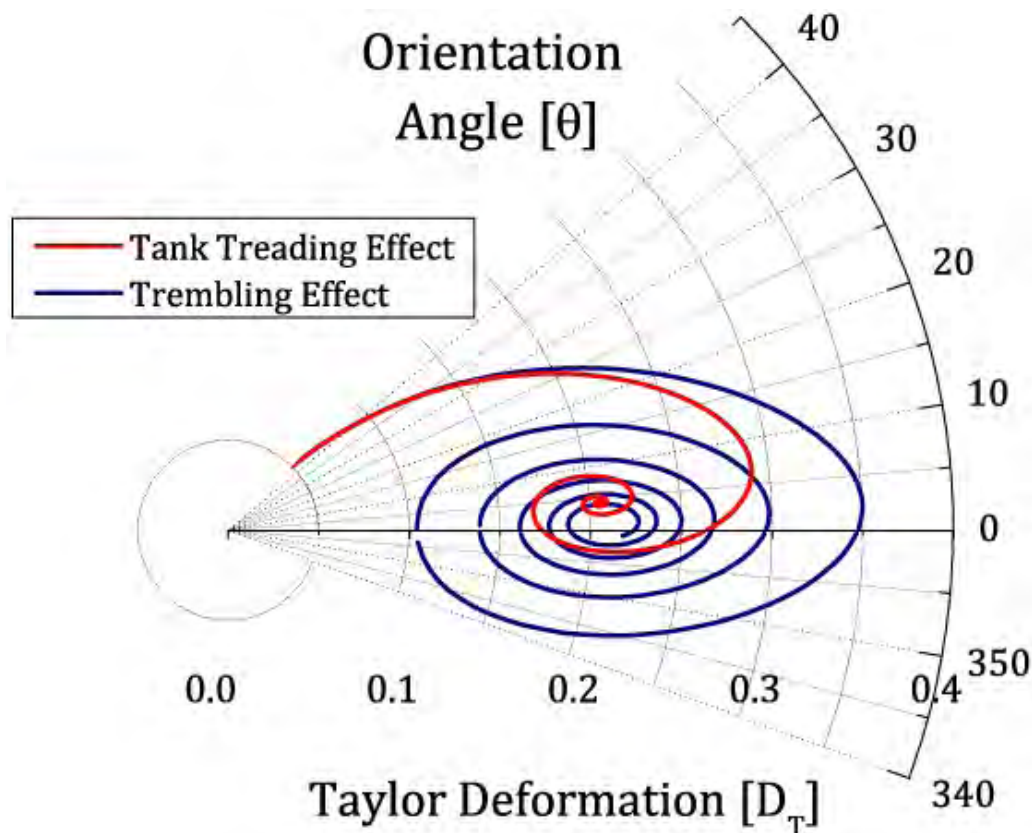
and numerical experiments. Therefore, there is a need to understand in detail the behavior of critical deformation of drops. However, even though, there is a good approximation by using the BEM3D code for capillary numbers below critical values, there is no good answer to address discrepancies, mainly 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  $\theta = 23.85^\circ$ .

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 shapes 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.



Universidad Nacional  
Autónoma de México

Dirección General de Bibliotecas de la UNAM

**Biblioteca Central**



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

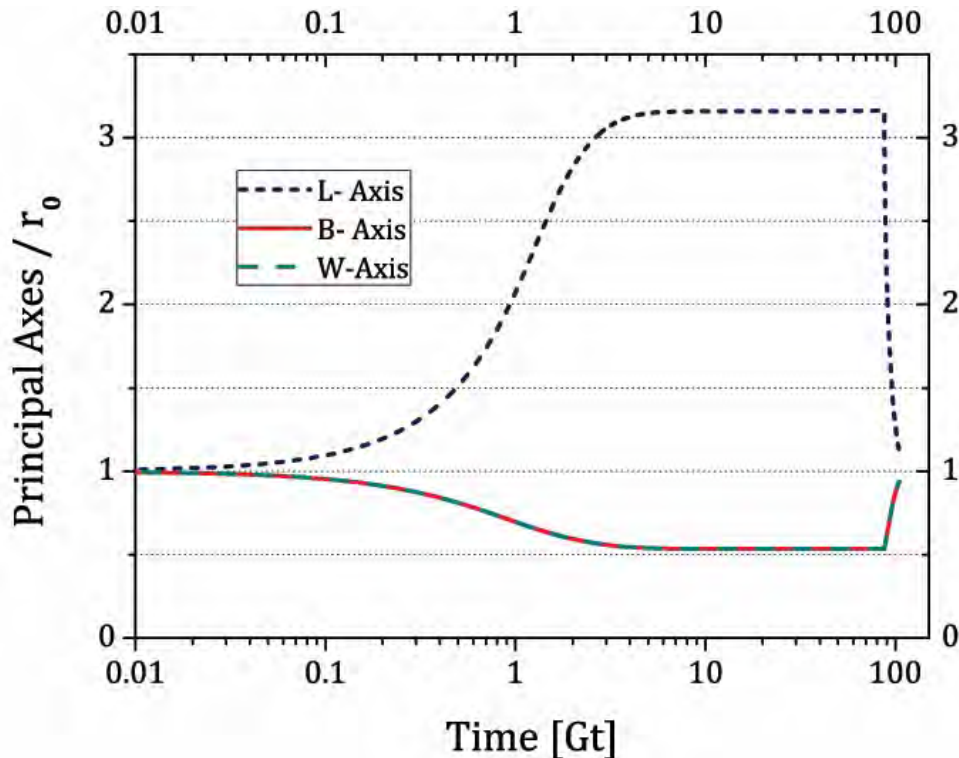
**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

## 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  $\lambda_\mu = 0.01$ .

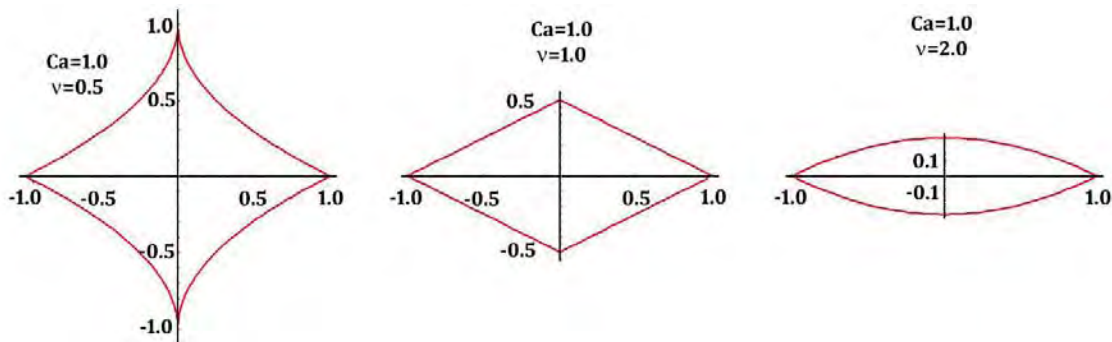


**Figure 7.1** Evolution of principal axes of a drop in uniaxial flow.  $\lambda_\mu = 0.01$ ,  $Ca = 0.31$

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  $\lambda_\mu \ll 1$ ; 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.

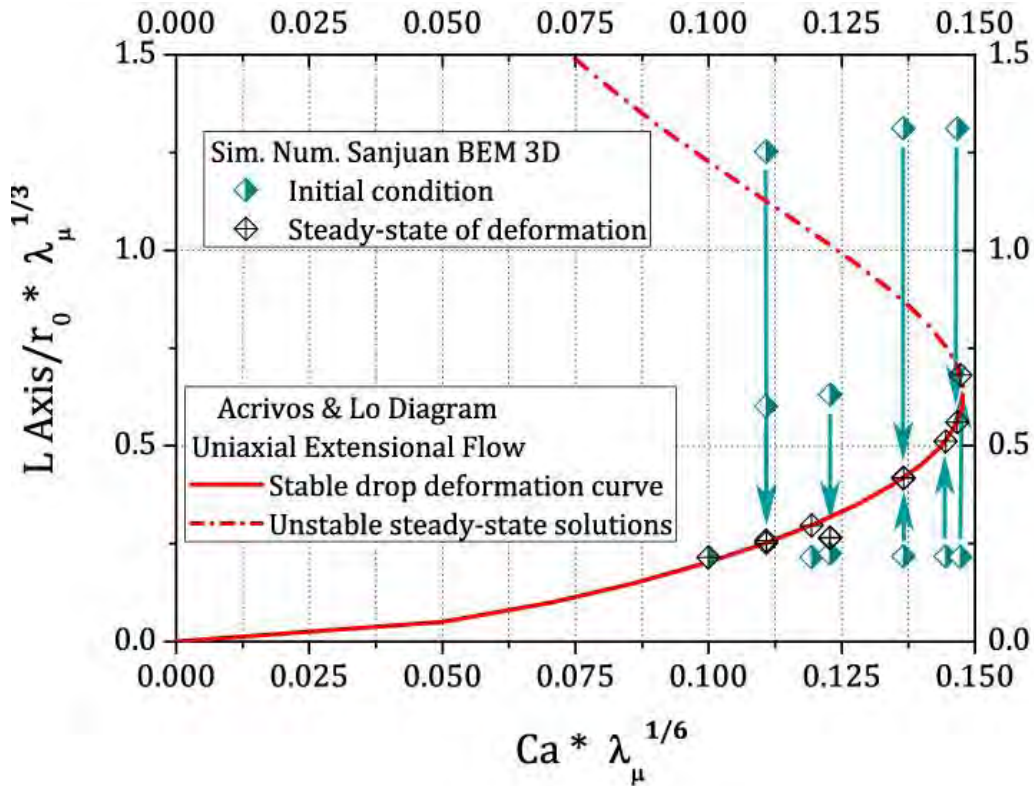
This analytical drop-form solution is plotted in Figure 7.2. The analytical solution is a function of the length of the axis of elongation ( $z$  in this case) and the  $\nu$  parameter (parameter which is a function of pressure of the drop, capillary number and intensity of the flow). The interface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

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

If the value of  $\nu$  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  $\lambda_\mu$  in those diagrams. My numerical method uses values of the ratio of viscosity  $\lambda_\mu = 0.01$ . Here,  $\lambda_\mu$  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  $Ca * \lambda_\mu^{1/6}$  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  $Ca * \lambda_\mu^{1/6} = 0.1$ . 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.,  $0.1 \leq Ca * \lambda_\mu^{1/6} < 0.148$ .

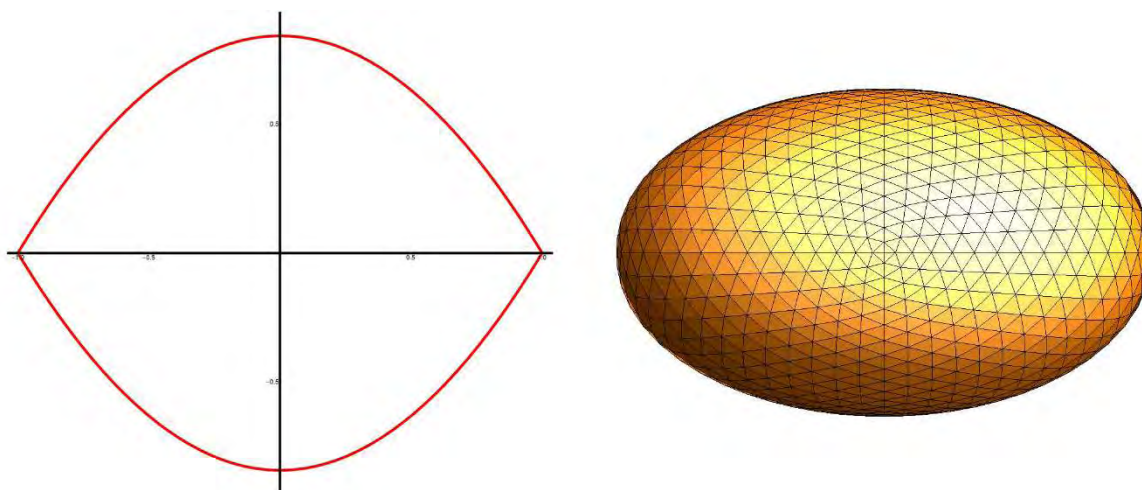
With this information, a series of numerical simulations were carried out. In Figure 7.3, green markers correspond to the initial elongation (arbitrary) conditions of the simulations. Black markers are the long term steady state of deformation that the numerical simulation attained. In the numerical 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 matter the initial value, drops will always go to the steady state condition (black markers, directly below).

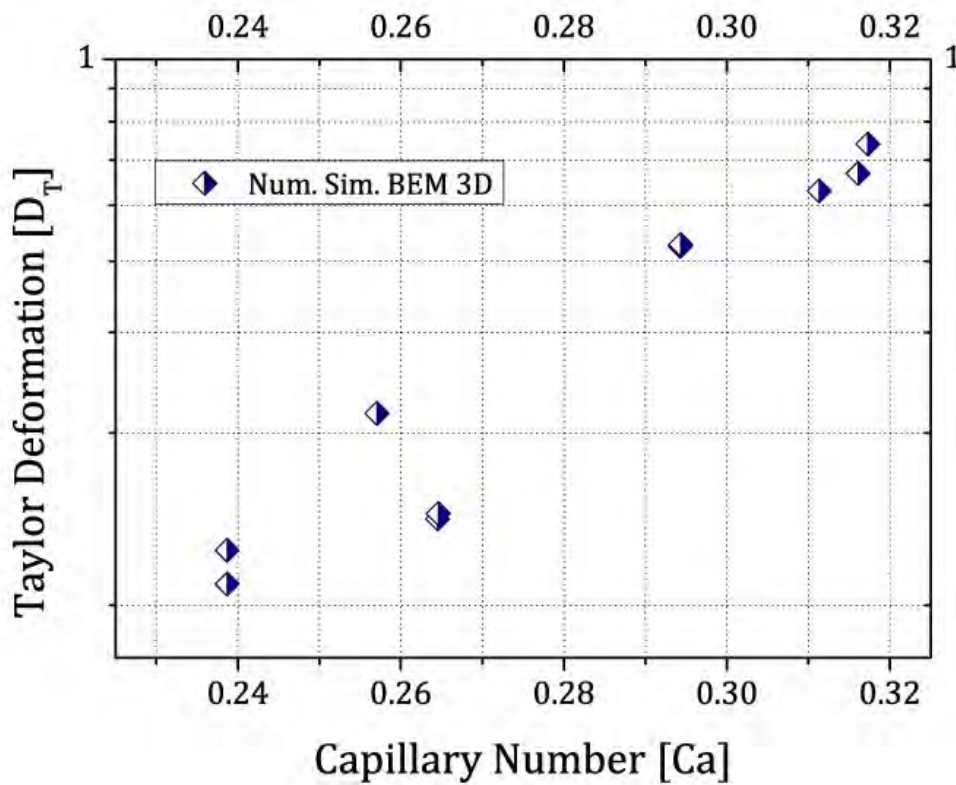
Figure 7.4 is a comparison of the steady shape of a drop in uniaxial flow. The left image is the theoretical results of Acrivos & Lo, employing Eq. 7.1 to obtain the shape of the drop with  $\nu = 2$ ,  $Ca = 0.26$ , and  $L - axis = 1.375$ . The predicted shape by numerical computations is presented 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, right.  $Ca = 0.26$  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  $\lambda_\mu$ , 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  $\nu$  to  $\nu = 2.5$ , the new shape obtained is similar in global form to that of the BEM3D model. However, the 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.



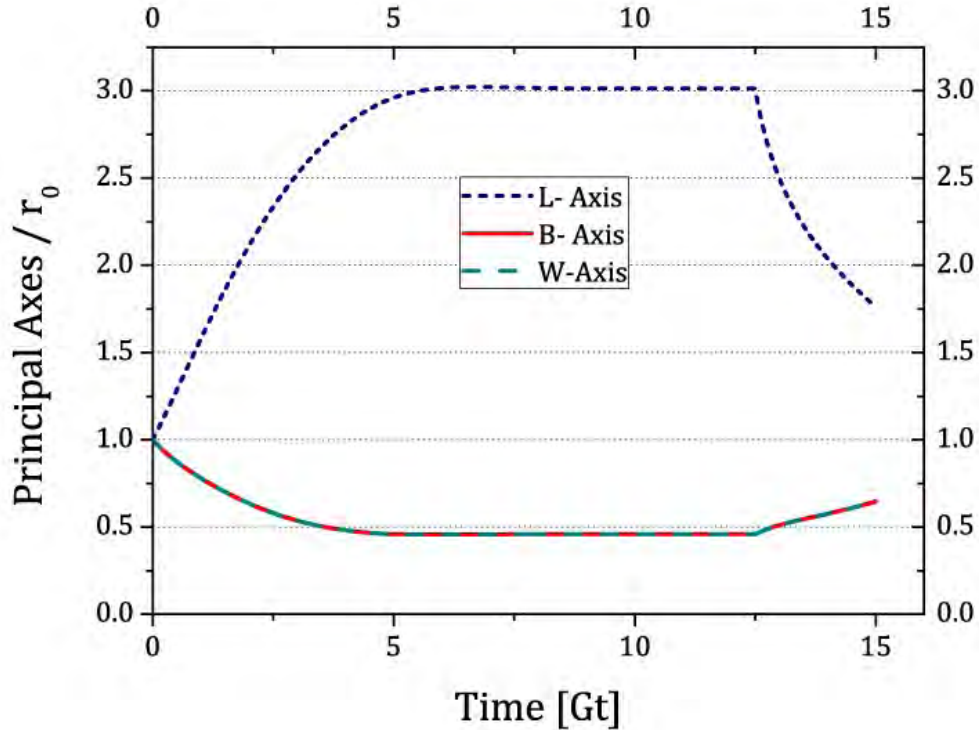
**Figure 7.5** Taylor Deformation vs Capillary number for a drop in extensional flow 3D.

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  $\lambda_\mu$ . However, if  $\lambda_\mu$  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.

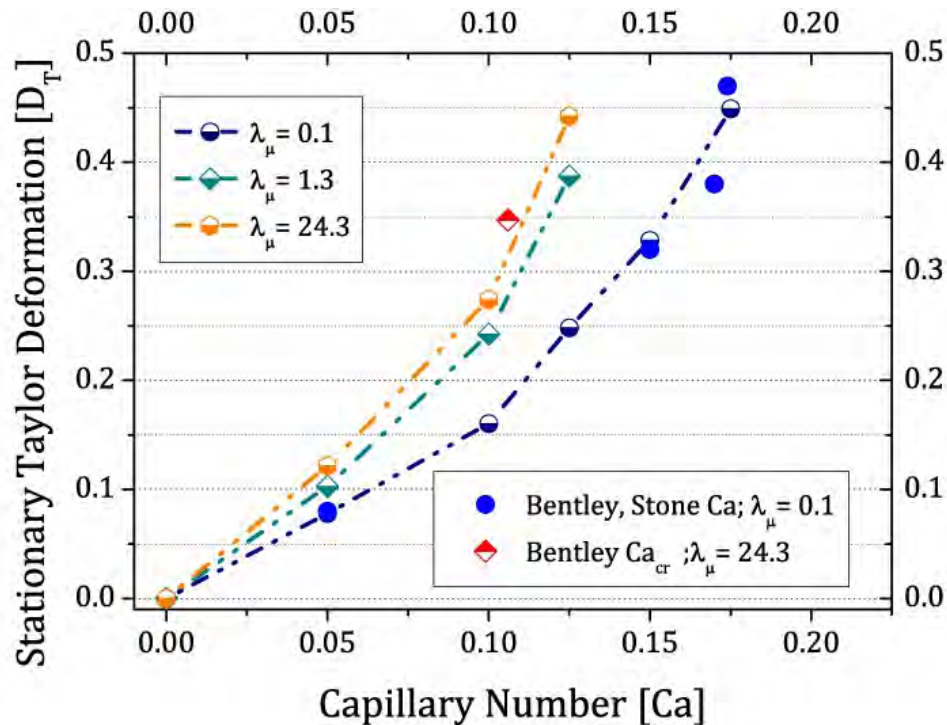


**Figure 7.6** Steady state attain by a drop in extensional 3D-flow with  $\lambda_\mu = 1.0$  and  $Ca = 0.1$ .

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.



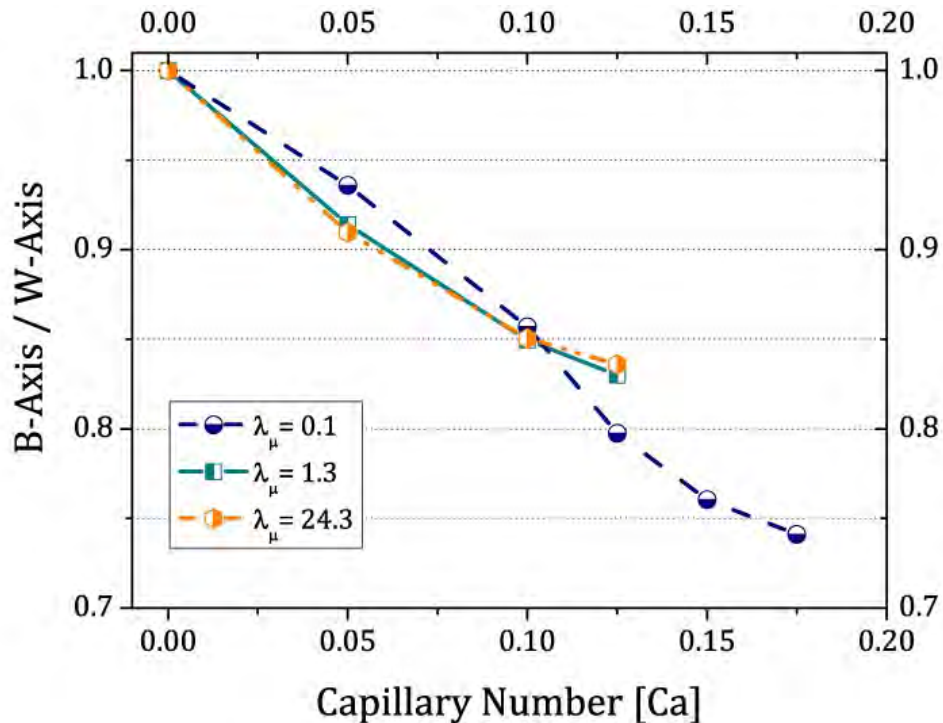
**Figure 7.7** Stationary deformation vs capillary numbers with different values of  $\lambda_\mu$  for extensional 2D-flow. Comparison with exp. data of  $Ca_{cr}$

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  $\alpha$  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  $\lambda_\mu$ , 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.



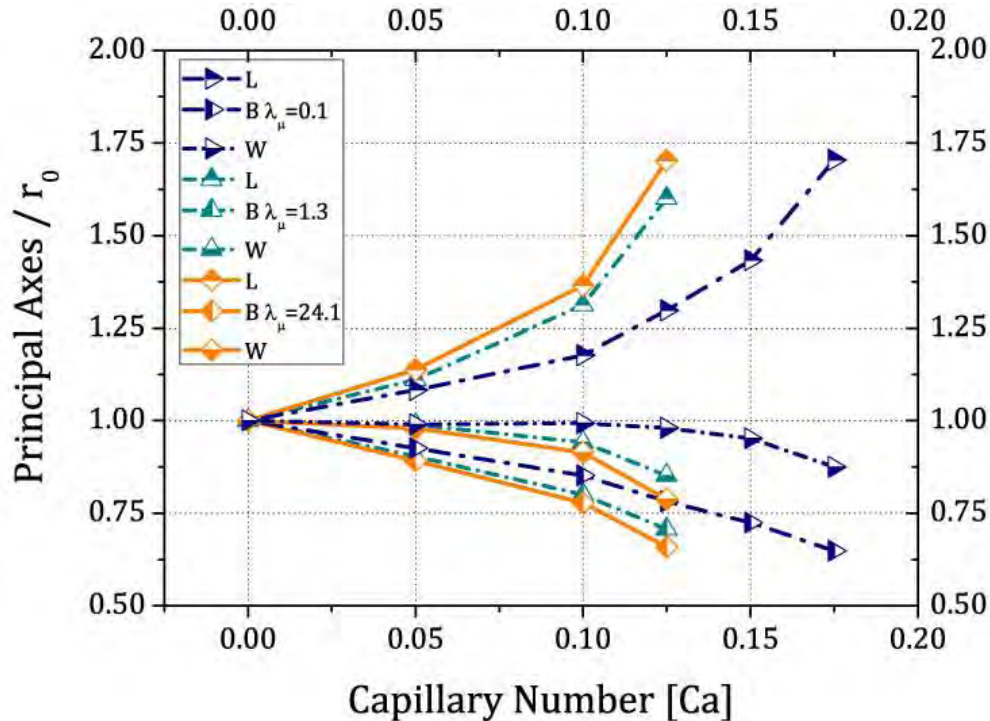
**Figure 7.8** Ratio of *B*- wrt. *W*-axis for different values of  $\lambda_\mu$  in pure extensional 2D-flow axis.

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  $\alpha$  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



Universidad Nacional  
Autónoma de México

Dirección General de Bibliotecas de la UNAM

**Biblioteca Central**



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

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  $\alpha$ , 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: sweeping —varying the planar flow *parameter*  $\alpha$ — 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:  $\beta$ , which augments the compressive character of the flow along the neutral direction). In this way, I expect to study possible drop deformation 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.

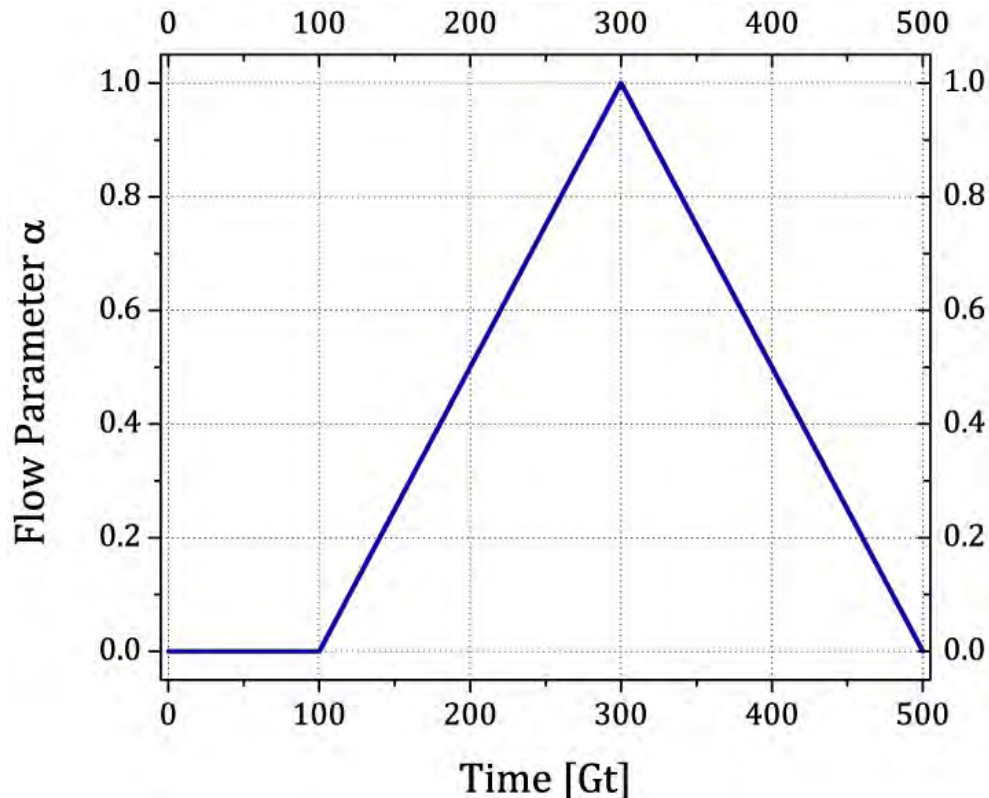
## 8.1 Hysteresis in strong flows

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 to 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  $\alpha$ . 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  $\alpha$ : using Eq. 1.9. So, the parameter  $G$  can only be a function of  $\alpha$ ; *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.

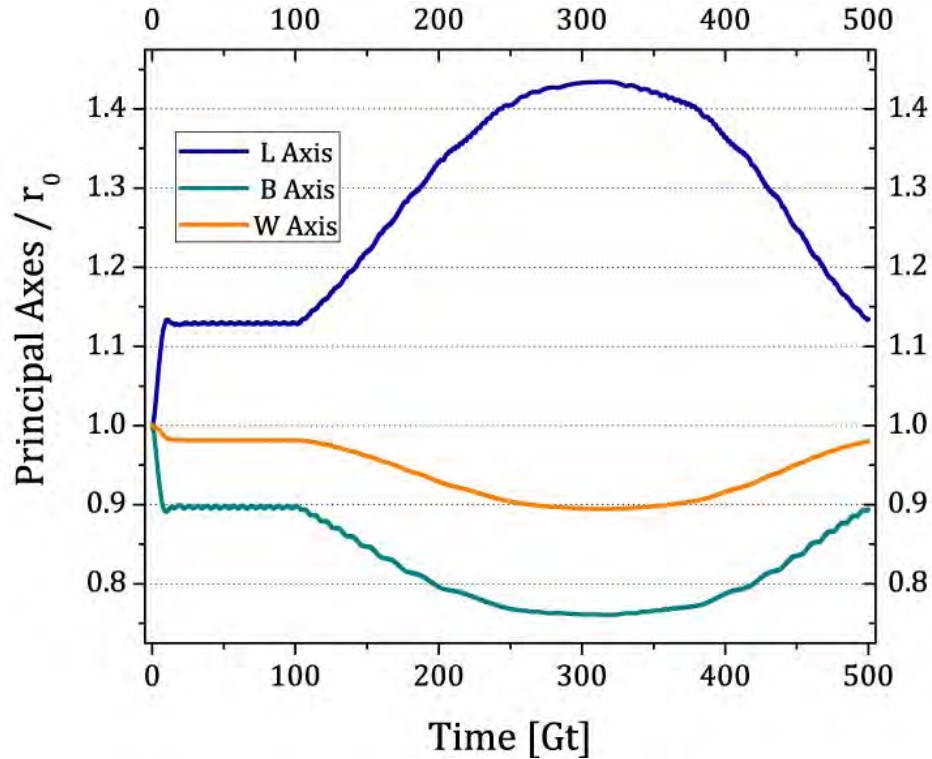
The next numerical experiments were performed using three stages. By a first phase, applying a *simple shear flow* to the continuum phase, and waiting for the drop to attain the steady state of deformation. The next phase, characterized by a continuous sequence of flows, requires changing the parameter  $\alpha$  from zero to one while holding  $Ca$  constant. Subsequently, the third phase, requires  $\alpha$  to be varied from one to zero. The evolution of  $\alpha$  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  $\alpha$  in time for a numerical simulation of a flow induced hysteric loop. The first portion of the flow corresponds to a simple shear flow. At  $Gt = 100$ , the parameter  $\alpha$  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.

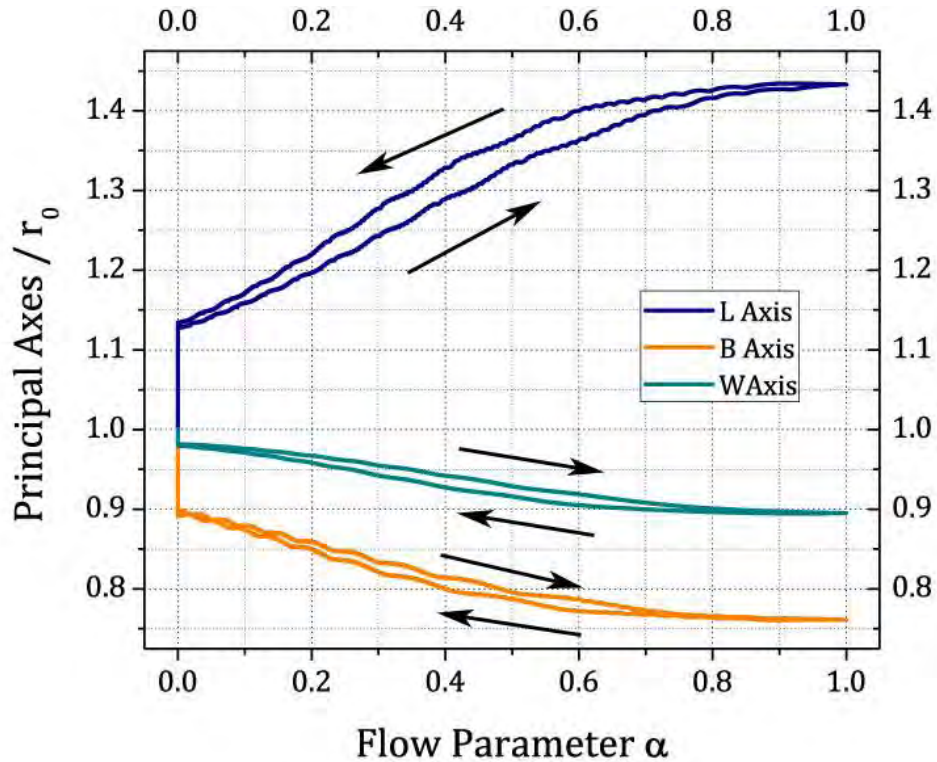


**Figure 8.2** Evolution of principal axes length scales vs time; i.e., vs the parameter  $\alpha$ . Time = 500 [Gt],  $\lambda_\mu = 16$ , from simple shear flow  $\alpha = 0$  to extensional 2D-flow  $\alpha = 1$ . Time:  $100 < Gt < 500$ .

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*  $\alpha$  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  $\alpha$  on drop deformations is analyzed in Fig. 8.3 by plotting the lengths of the principal axes versus the parameter  $\alpha$ . 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  $\alpha$  flow parameter changed. However, the data in Fig. 8.3 indicates a curved more complicated because for each value of  $\alpha$ , 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  $\alpha$ .



**Figure 8.3** Principal axes vs flow parameter alpha  $\alpha$ . 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

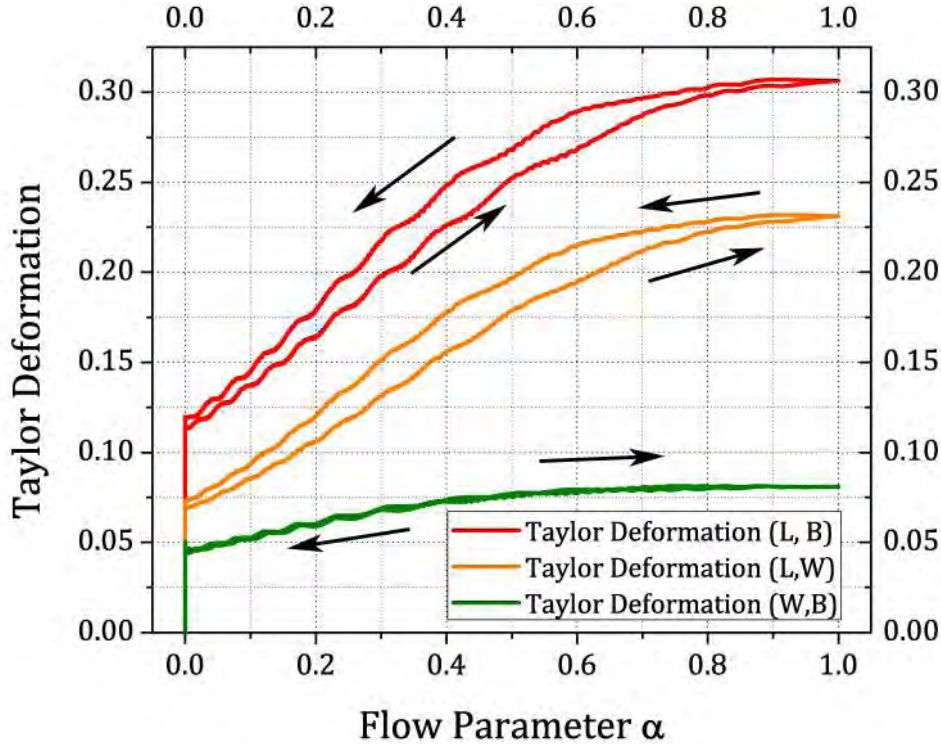
*trembling effect*, which is a consequence of the huge value of the *parameter G*. Then, as the flow changes to a more elongational character ( $\alpha$  increases) the oscillations tend to disappear. When  $\alpha$  is one, —fully *extensional 2D-flow*— there are no oscillations. Finally, when the flow becomes again  $\alpha$  equal to zero, the oscillations reappear.

In Figures 8.2 and 8.3, oscillations of the lengths as the value of the  $\alpha$  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  $\alpha$  (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  $\alpha$  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  $\alpha$  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  $\alpha$  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  $\alpha$ . 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, Blawdziewicz, 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.

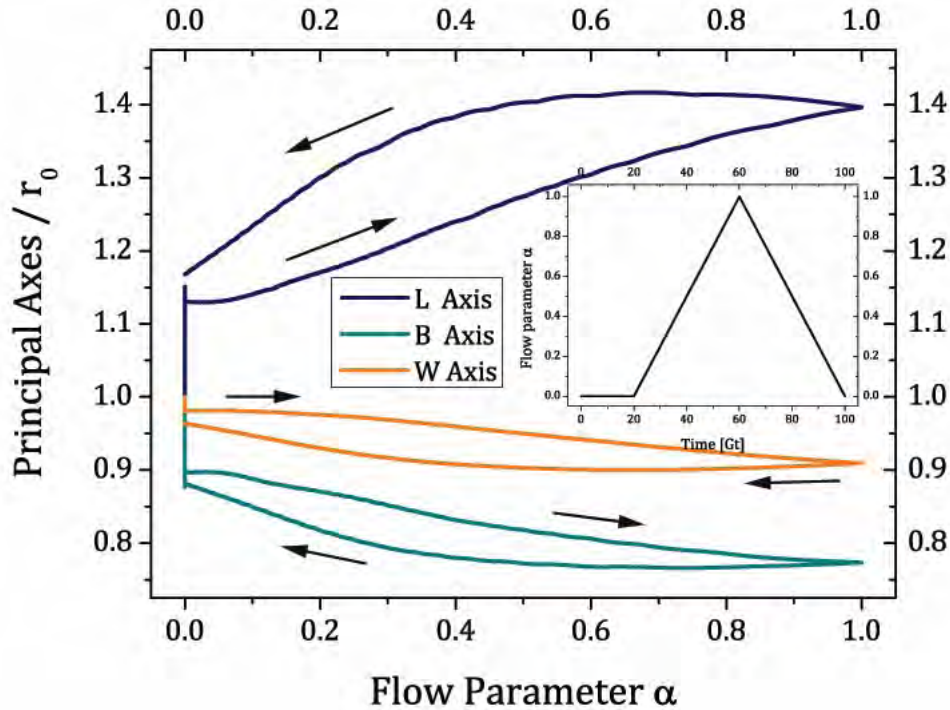
Using the information of Figure 3.2, the stationary state of the *L-axis*, for a capillary number of  $Ca = 0.40$ , is the same as the maximum measure observed by Young *et al.* Reviewing the diagram of Taylor deformation *vs. Ca* in Fig. 3.5, the steady state deformation attained for this capillary 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  $\alpha$ . 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

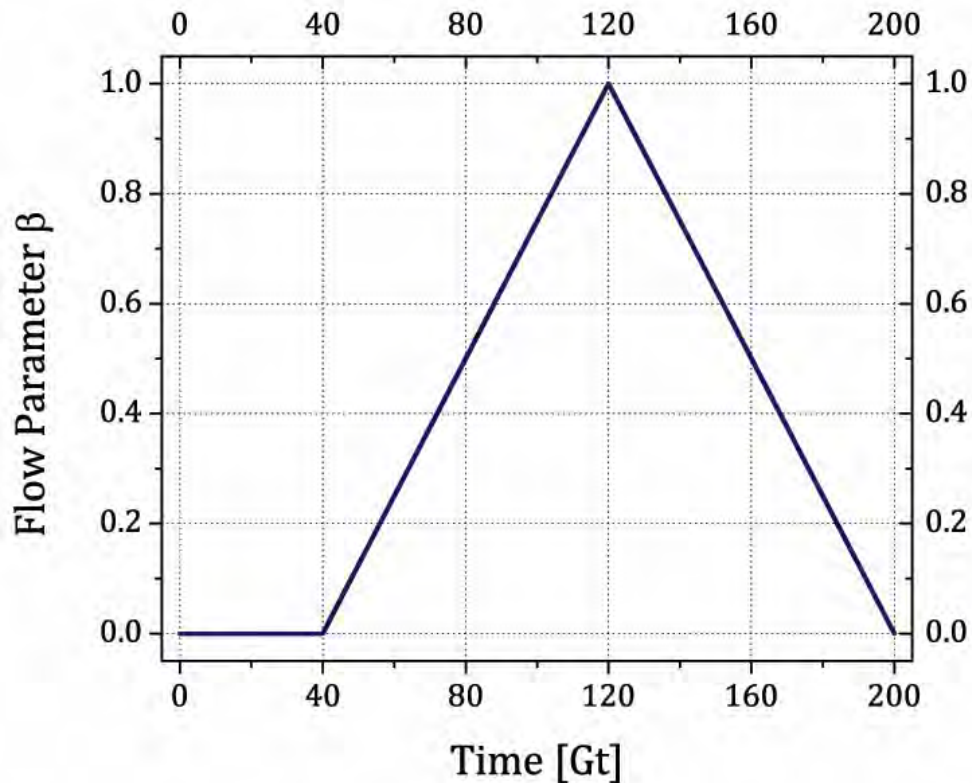
The previous Section addresses the hysteretic behavior of drop deformation as a consequence of the type of *two-dimensional flow* applied in the continuum phase. Also, in Chapter 7 the asymmetric drop deformation is studied when caused by (1) *two-dimensional extensional* and (2) *uniaxial flow*. In 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*  $\beta$ . Thus, by varying  $\beta$ , 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

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

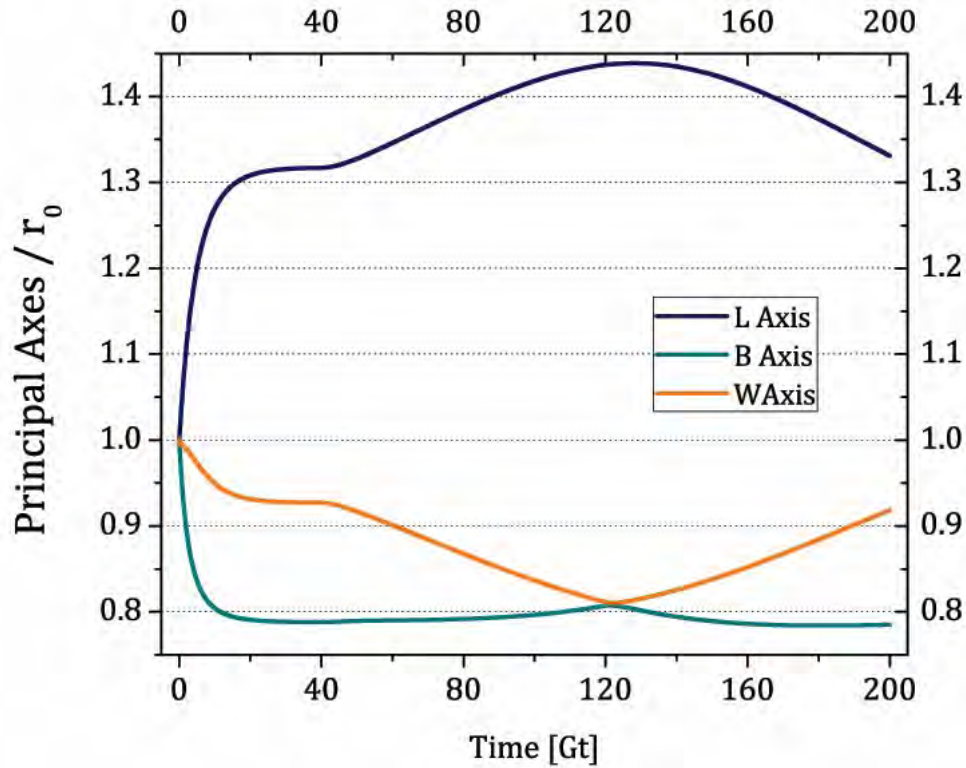
where  $G$  is the intensity of the rate of deformation tensor (Makosco, 1994). Now,  $\beta$  is the parameter characterizing the strength flow along the symmetric (third) axis. Values of  $\beta$  goes from zero (*extensional 2D-flow*), to one (*uniaxial flow*).



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

The *parameter*  $\beta$  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  $0 \leq \beta \leq 1$ .

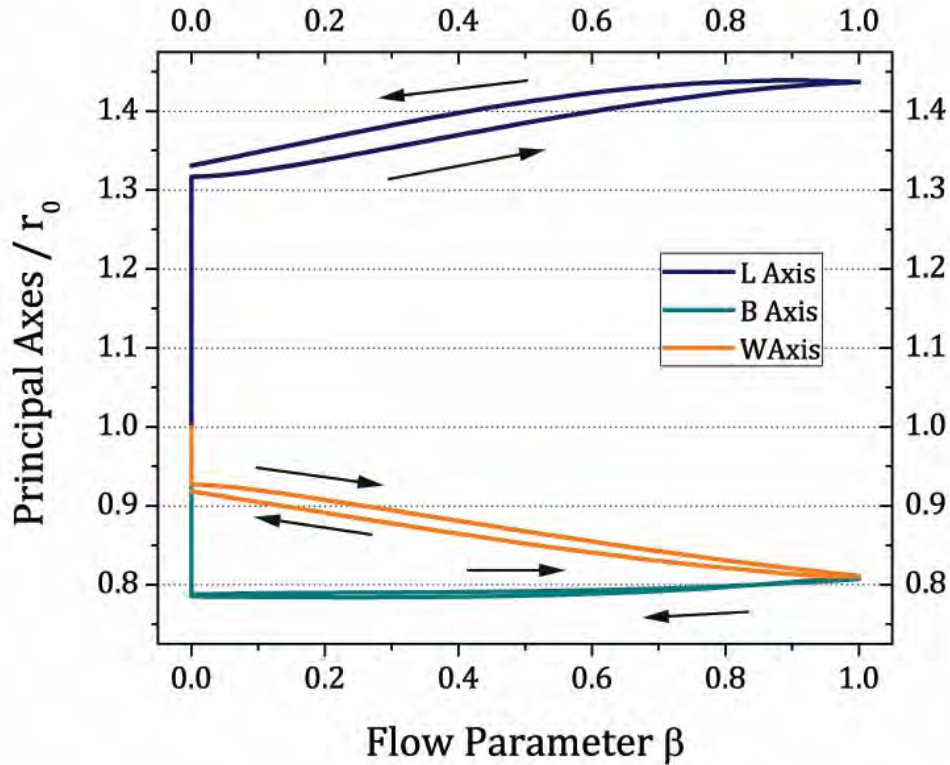


**Figure 8.7** Evolution of principal axes vs time. Time = 200Gt,  $\lambda_\mu = 16$ , from extensional 2D-flow  $\beta = 0$  to uniaxial flow  $\beta = 1$ . Time:  $40 < Gt < 200$

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  $\beta$  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  $\beta$ , 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  $\alpha$ ),

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  $\alpha$  flows.



**Figure 8.8** Principal axes vs flow parameter betta  $\beta$ . With  $\lambda_u = 16$ , 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  $\beta$  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 unsettled 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 involved 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 implications 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



Universidad Nacional  
Autónoma de México



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.



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 significantly 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  $\lambda_\mu$  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  $\alpha$  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,  $G \ll 1$ .

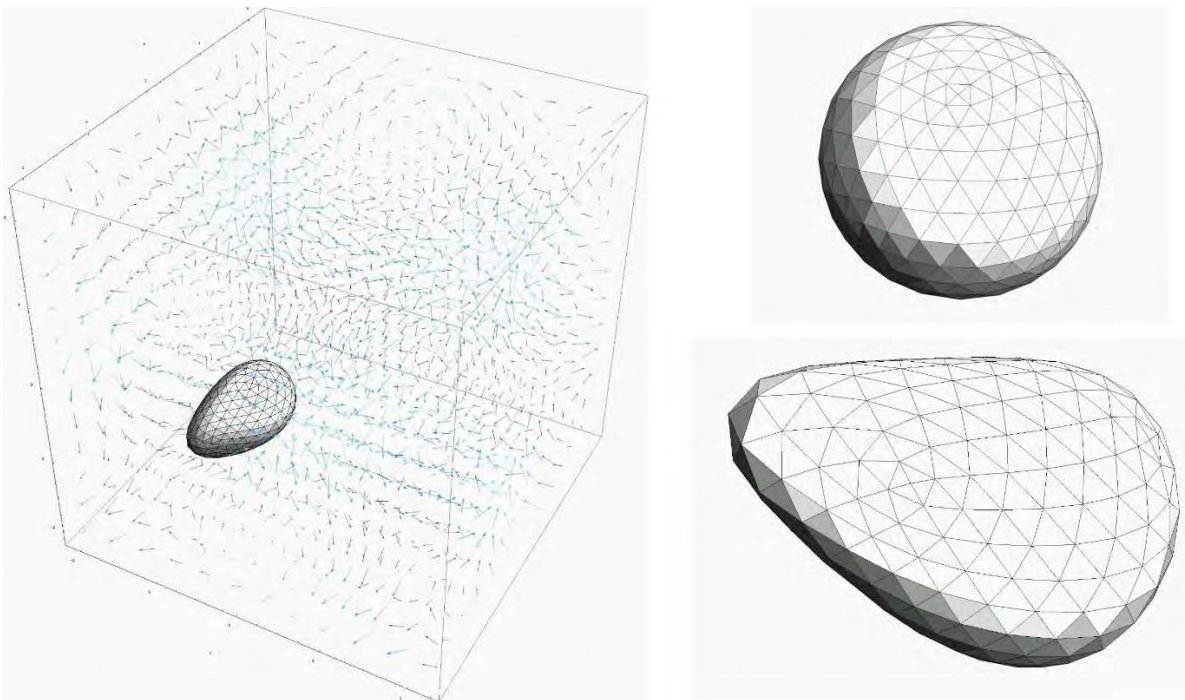
There are a few published papers that establish the existence of Hopf bifurcation solutions for drops deformed by flows. However, there still exists a need in fluid mechanics to understand possible branches of these solutions, mainly because different type of shapes may imply different inner and outer 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  $\lambda_\mu$  attains values different from zero, this theoretical approximation starts to deviate from predictions, because  $\lambda_\mu$  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  $\lambda_\mu$ . 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  $\alpha$  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  $\lambda_\mu$  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,  $\beta$ . As was observed, those states of deformations depend of  $\lambda_\mu$  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 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<sub>2</sub> 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.
- Blawdziewicz, J., Cristini, V., & Loewenberg, M. (2003). Multiple stationary states for deformable drops in linear Stokes flows. *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). Monomial 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., Blawdziewicz, J., & Loewenberg, M. (2000). An Adaptive Mesh Algorithm for Evolving Surfaces: Simulations of Drop Breakup and Coalescence. *Journal of Computational Physics*, 168, 445-463.



Universidad Nacional  
Autónoma de México



**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

- 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 Hydrodynamics*. 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 viscoelastic 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). Adaptive 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). *Hydrodynamics*. 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 Sequential & Parallel. An Unified Approach*. London, Sydney, Toronto, Mexico, New Delhi.: Prentice Hall.
- Mo, H. Z. (2000). A new method to determine interfacial tension from the retraction of ellipsoidal 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 Theoretical and Computational Fluid Dynamics*. New 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 Elements* , 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 constrained 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: Compañía 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 effect of nonlinear control on the behavior of a liquid drop in elongational 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 suspensions 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 from 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 and breakup of immiscible drops in oscillatory shear. *J. Rheol.*, 46(5), 1279-1293.
- Young, Y. N., Blawdziewicz, 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 retraction 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<sup>1</sup>, M A H Reyes<sup>2</sup>, A A Minzoni<sup>3</sup> and E Geffroy<sup>1</sup><sup>1</sup>Instituto de Investigaciones en Materiales, Universidad Nacional Autónoma de México, Mexico City.<sup>2</sup>Departamento de Termofluidos, Facultad de Ingeniería. Universidad Nacional Autónoma de México, Mexico City.<sup>3</sup>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 al [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.



Content from this work may be used under the terms of the Creative Commons Attribution 3.0 licence. Any further distribution of this work must maintain attribution to the author(s) and the title of the work, journal citation and DOI.

Published under licence by IOP Publishing Ltd



Universidad Nacional  
Autónoma de México

Dirección General de Bibliotecas de la UNAM

**Biblioteca Central**



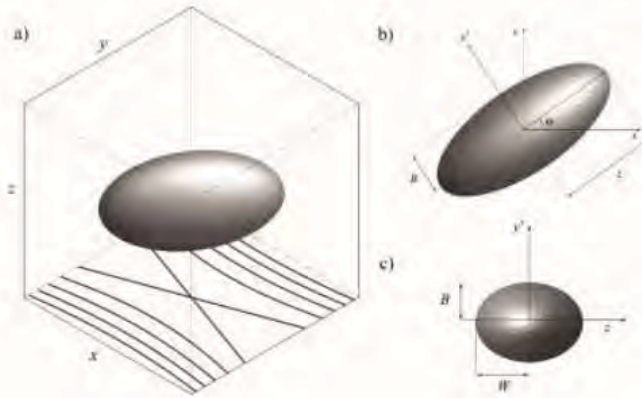
**UNAM – Dirección General de Bibliotecas**  
**Tesis Digitales**  
**Restricciones de uso**

**DERECHOS RESERVADOS ©**  
**PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL**

Todo el material contenido en esta tesis esta protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.





**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  $\mu_0$  and surface tension  $\sigma$ . The continuum phase has a viscosity  $\mu_1$ . 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]

$$\begin{aligned} \mu \nabla^2 \mathbf{u} &= \nabla p, \\ \nabla \cdot \mathbf{u} &= 0. \end{aligned} \quad (1)$$

The velocity  $\mathbf{u}$  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:

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

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

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

The drop is subject to a two dimensional linear incident flow

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

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

$$G = (|\mathbf{II}_{2,D}|)^{1/2}. \quad (5)$$

Equation (4) assumes that the  $\alpha$  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  $\dot{\epsilon}$ .

The dynamic of drop deformation is characterized by three dimensionless numbers. The viscosity ratio  $\lambda_\mu$ , the capillary number  $Ca$  equation (6) and The flow parameter  $\alpha$  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, where  $r_0$  is the non-deformed radius of the drop.

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

Finally, the flow parameter  $\alpha$  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



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:

$$D_T = \frac{L - B}{L + B}, \quad (7)$$

where  $L$  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  $u_0$  is the flow on the surface of the drop and  $f_0$  is the force on the surface of the drop.

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

where  $f = \sigma \cdot \hat{n}$ , 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

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

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  $u^\infty(x_0)$ . 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.



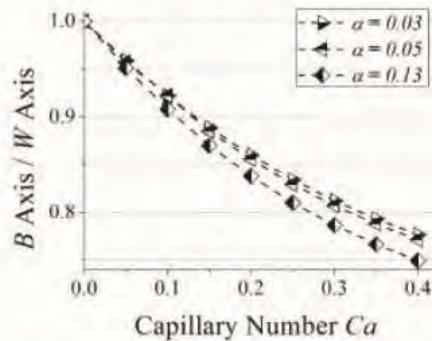


Figure 4. Stationary ratio of  $B$  wrt  $W$  axes in numerical simulations for a flow with  $\alpha = 0.03$ ,  $\alpha = 0.05$  and  $\alpha = 0.13$  for different values of  $Ca$ .

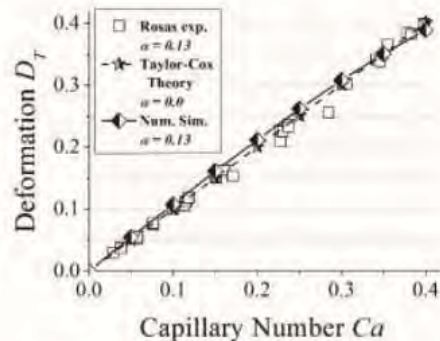


Figure 5. Stationary Taylor Deformation values for numerical simulations and experimental data for  $\alpha = 0.13$  and analytical results of Taylor-Cox at  $\alpha = 0.0$  for different values of  $Ca$ .

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  $\alpha$  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).

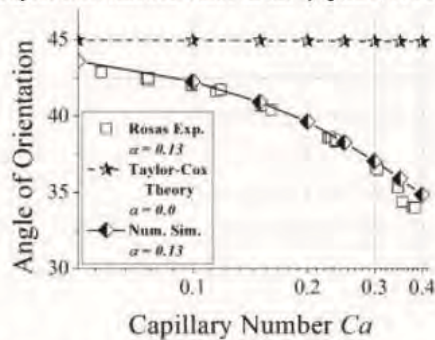


Figure 6. Orientation of the drop for numerical simulations and experimental data for  $\alpha = 0.13$  and analytical results of Taylor-Cox  $\alpha = 0.0$  for different values of  $Ca$ .

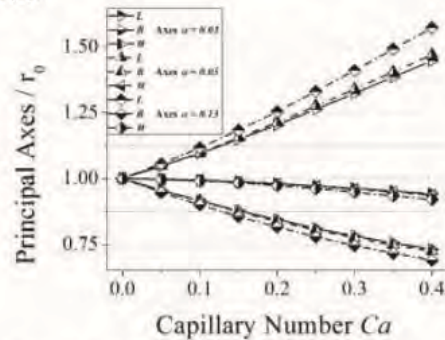


Figure 7. Stationary values of the principal axes of the drop for a flow with  $\alpha = 0.03$ ,  $\alpha = 0.05$  and  $\alpha = 0.13$  for different values of  $Ca$ .

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

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  $\alpha$  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](mailto: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.

D:\Darth Vader\Escritorio\prtcl\_mkl\Prctl\_3DMKL.f90 1

```
1 PROGRAM Prctl_3D
2 =====
3 !
4 ! Version 0.5 C. POZRIKIDIS
5 !-----
6 ! Version 0.7 MARCO ANTONIO REYES HUESCA
7 !-----
8 ! Versión 1.0 ALFREDO SANJUAN SANJUAN
9 !
10 ! **Mexico D.F., in the Year of Our Lord of 2012**
11 !
12 =====
13 !
14 ! This code is a numerical project about a drop deformation in Stokes' Flows. You would see this versio was
15 ! obtained of the wor of Pozrikidis as a begining. However, this code has oher aplicacion because it shows what's
16 ! happen in a drop immerse in another fluid when someone applies on continue phase a strong flow.
17 !
18 ! This program computes the following Stokes flows:
19 !
20 ! 1) Flow due to the motion of a 3-dimensional drop immerse in another immiscible fluid. In this case, the
21 ! particle is an ellipsoid with:
22 !
23 ! x semi-axis equal to a
24 ! y semi-axis equal to b
25 ! z semi-axis equal to c
26 !-----
27 ! SYMBOLS:
28 !-----
29 ! p(i,j) ..... coordinates of the ith node (j=1,2,3)
30 ! ne(k,j) ..... ne(k,1) is the number of elements adjacent to point k
31 ! ne(k,2), ... are the elements numbers, j = 2, ..., 7 for this triangulation
32 ! up to six
33 ! n(k,i) ..... connectivity table: points for element k, i = 1,...,6
34 ! nbe(k,j) ..... the three neighboring elements of element k (j=1,2,3)
35 ! arel(i) ..... surface area of ith element
36 ! npts ..... total number of points
37 ! nelm ..... total number of elements
38 ! x0, y0, z0 ..... coordinates of collocation points
39 ! vnx0, vny0, vnz0 ..... unit normal vector at collocation points
40 ! alpha0, beta0, gamma0 ..... parameters for quadratic xi-eta isoparametric mapping
41 !-----
42 !
43 !The initial thing is to USE all MODULES
44 USE Mod_SharedVars !
45 USE Mod_TRMCoefs !
46 USE Mod_Trgl_Octa !
47 USE Mod_Gauss_Coefs !
48 USE Solucion_Ax_b !
49 USE Mod_SNEDOS !
50 USE Mod_Data_Files !
51 USE Mod_Builder_Matrix_Arrays !
52 USE Mod_Prtcl_3D_Geo !, ONLY: Printel, ahc, elm_geom, elm_geom2 !
53 USE Mod_Correction !
54 USE Mod_Nodal_Interp !
55 USE Mod_Velocity_Menu !
56 USE Mod_VelFieldFW !
57 !-----
58 !
59 ! In this par we call the libraries used in this code
60 !-----
61 ! MCL libraries
62 USE mkl95_lapack
63 USE mkl95_precision
64 USE omp_lib
65 !-----
66 ! IMSL libraries.
67 USE IMSL_LIBRARIES
68 ! USE CS1GD_INT
69 ! USE lin_sol_gen_int
70 ! IMSL Library header file
71 ! INCLUDE 'link_f90_static.h'
72 !-----
```

D:\Darth Vader\Escritorio\prtcl\_mkl\Prctl\_3DMKL.f90 2

```
73 !-----
74 IMPLICIT NONE
75 !-----
76 !
77 ! Initialization of Variables
78 !
79 !In this part of the code the variables has been clasified because there are a lot of them. Other reason is
80 !there's a necessity of modify in future, so is necessary to know the new variables and what are the obsolete or
81 !old variables to delete them.
82 !Inside these classification there are subcategories, fist the character variables, then integer variables and
83 !finally the real variables. This manner of variable initialization comes from CHAPMAN FORTRAN BOOK'S
84 !-----
85 ! Input Data Variables
86 !These variables are the variables obtained from INPUTDATA file.
87 INTEGER :: IFlow !Type of flow
88 INTEGER :: ndiv !Level of triangulation
89 INTEGER :: mint, NGL !Number mint is Gauss_triangle base point number, it's used to
90 !make the regular integral. NGL is used to make singular
91 INTEGER :: stoper !step when stop the flow
92 INTEGER :: reducer !step when reduces the flow to half value
93 INTEGER :: Iprec
94 INTEGER :: Ireg
95 REAL (KIND = DBL) :: boa, coa !these variables are used in subroutine scale_drop_adjust.
96 REAL (KIND = DBL) :: req !
97 REAL (KIND = DBL) :: exp, cyp, czp !
98 REAL (KIND = DBL) :: phi1, phi2, phi3 !
99 REAL (KIND = DBL) :: visc, visc2 !Viscosity of continuous phase and drop
100 REAL (KIND = DBL) :: gamma !Surface tension
101 REAL (KIND = DBL) :: h !stride
102 REAL (KIND = DBL) :: alpha, gi, alphantom ! parameter of flow and intensity of field flow. Code appendix
103 REAL (KIND = DBL) :: omicron !parameter of transition between eulerian and lagrangian advance
104 !-----
105 LOGICAL :: adjc !!Adjust of center
106 !-----
107 !
108 ! Variables of Mod_Trgl_Octa
109 INTEGER :: npts, nelm !number of elements and points
110 !-----
111 ! In this part appears the variables which fix the frame coordenates. This part could be changed to other module
112 !-----
113 ! It was in 21 / August / 2012
114 ! Variables of Mod_Geo
115 ! In this par, there are theold variables, until this module will change
116 REAL (KIND = DBL) :: srf_area, srf_area_n
117 REAL (KIND = DBL) :: prt_vlm, prt_vlm_n
118 REAL (KIND = DBL) :: cx, cy, cz
119 REAL (KIND = DBL) :: Dt, Dtm, Dttm, lmajor, bminor, wminor, Tiempo, angulo
120 REAL (KIND = DBL) :: xi, eta
121 REAL (KIND = DBL) :: Dxdxi, Dydxi, Dzdx1
122 REAL (KIND = DBL) :: Dxdet, Dydet, Dzdet
123 REAL (KIND = DBL) :: hs
124 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: xmom, ymom, zmom
125 !-----
126 ! Variables of Green's Functions **** review the Module: Mod_Builder_Matrix_Arrays
127 REAL (KIND = DBL) :: cdg, cdt, fc !cdg, cdt are coefficients for the dimensionality of the flow
128 ! Variables of matrix array
129 INTEGER, ALLOCATABLE, DIMENSION(:) :: ipiv
130 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: GM, RM, Uoo, Vel0, Vel1
131 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: K1, K2, K3 !, K4
132 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: TH
133 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: rhs
134 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: sln
135 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: Resist
136 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: BL, Sol
137 !-----
138 ! Indices and constants in the code.
139 CHARACTER ( 8 ) :: DATE
140 CHARACTER (12) :: TIME
141 CHARACTER (100) :: msg
142 INTEGER :: null, NFour, NSeven
143 INTEGER :: i, j, k, l1, l2, l3, l4, l5, l6, m
```



```

144 INTEGER :: icohose
145 INTEGER :: inelm, inelm2, jnelm, jnelm2
146 INTEGER :: mdim
147 INTEGER :: stride
148 INTEGER :: isymg, iwlpvt, istop
149 INTEGER :: md, mode, neigh_elm, lelm, index !md I use to control the stability of mesh
150 INTEGER :: ifile = 0, fstatus = 0, astatus = 0, Istp = 0, lerror = 0
151 INTEGER :: sstatus=0 !second index status
152 REAL :: CPUTime
153 REAL (KIND = DBL) :: pih, pi2, pi4, pi6, pi8
154 REAL (KIND = DBL) :: phi, shIFt, rr, rrr
155 REAL (KIND = DBL) :: thet0, thetz, thet
156 REAL (KIND = DBL) :: lmu
157 !-----
158 ! Other variables, in the future, there wont be nothing here. :)
159 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: Vel0x, Vel0y, Vel0z
160 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: Vel1x, Vel1y, Vel1z
161 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: Vel1x1, Vel1y1, Vel1z1
162 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: Vel1x2, Vel1y2, Vel1z2
163 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: Vel1x3, Vel1y3, Vel1z3
164 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: Vel1x4, Vel1y4, Vel1z4
165 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fx_node, fy_node, fz_node
166 ! Influence matrix
167 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: dfl_rm, dfl_rhs
168 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: residual
169 !REAL (KIND = DBL) :: sum1, sum2, sum3, sum4, sum5, &
170 !& sum6, sum7, sum8, sum9
171 !REAL (KIND = DBL) :: xhat, yhat, zhat
172 !REAL (KIND = DBL) :: fnormal
173 !REAL (KIND = DBL) :: frcx, frcy, frcz
174 !REAL (KIND = DBL) :: trqx, trays, trqz
175 !REAL (KIND = DBL) :: trcx, trcy, trcz
176 !REAL (KIND = DBL), DIMENSION(2) :: p0, u0TRM
177 !REAL (KIND = DBL) :: wPrt, Frt1, Frt2
178 !-----
179 ! Name list are here. Today (14 / 08 / 2012), there are only 2 namelist, but only one's worked :(
180 !-----
181 !
182 NAMELIST / InputData / IFlow, &
183 & ndiv, boa, coa, req, cpx, cyp, czp, &
184 & phi1, phi2, phi3, &
185 & mnt, NGL, Ns, NP, &
186 & visc, visc2, gamma, &
187 & alpha, gi, &
188 & h, steps, stoper, reducer, &
189 & adjc, omicron, doomsday, &
190 & eps, Iprec, Ireg
191 ! end of namelist
192 !-----
193 ! Name list are here. It will work :)
194 !-----
195 !NAMELIST / TRMCoefs / R, w, vR, Ory, StPnt, &
196 !& a, de, g, M1, M2, &
197 !& A0, B0, C0, D0, Kc, &
198 !& An, Bn, Cn, Dn, &
199 !& lambda, shrate
200 !-----
201 NAMELIST / TRMCoefs / StPnt, shrate, lambda, &
202 & R, Ory, de, g, a, vR, &
203 & w, M1, Kc, &
204 & A0, B0, C0, D0, M2, &
205 & An, Bn, Cn, Dn
206 !-----
207 ! Initialization of constants
208 ! This part has the values of constants like, pi, null, etc.
209 !-----
210 pih = 0.500*pi
211 pi2 = 2.000*pi
212 pi4 = 4.000*pi
213 pi6 = 6.000*pi
214 pi8 = 8.000*pi
215 !-----

```

```

216 null = 0
217 Nfour = 4
218 Nseven = 7
219 !-----
220 CPUTime = 0
221 !-----
222 !
223 ! Open files statements.
224 ! This part has the new feature in OPEN statement. IMSG is very important.
225 !-----
226 OPEN(UNIT = 10, FILE = 'Prtc1_3D.nml', STATUS = 'OLD', ACTION = 'READ', & !there is IOSMG in all open
227 & DELIM = 'QUOTE', IOSTAT = fstatus, IOMSG = msg) !statements
228 !-----
229 !The first statement is obtain the data and time.
230 CALL DATE_AND_TIME( DATE, TIME)
231 !-----
232 ! In this part, the file Prtc1_3D.nml is opened and red. The code has two warning statements, the first is
233 !about the existence of file, and the second is about the file has the correct values in their variables.
234 !The use of this conditions is necessary to find problems with the input data.
235 IF (fstatus == 0) then
236 READ(UNIT = 10, NML=InputData, IOSTAT = sstatus, IOMSG = msg)
237 !-----
238 !This is the second condition, here there is a warning about the structure of file Prtc1_3D
239 IF (sstatus == 0) then
240 CLOSE(UNIT = 10)
241 ELSE
242 OPEN(UNIT = ULogi, FILE = 'Prtc13D_Logi.log', STATUS = 'REPLACE', ACTION = 'WRITE')
243 WRITE (ULogi,*) 'DATE: ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
244 WRITE (ULogi,*) 'Error, little Padawan, InputData has a mistake!'
245 WRITE (ULogi,*) 'First Error index = ', fstatus
246 WRITE (ULogi,*) 'number of error'
247 WRITE (ULogi,*) 'Second Error index = ', sstatus
248 WRITE (ULogi,*) ' message of error:'
249 WRITE (ULogi,FMT=*) TRIM(msg)
250 WRITE (ULogi,*) ' message of error ended'
251 WRITE (ULogi,*) 'Now, go to find the mistake, Do not see this more time!'
252 WRITE (ULogi,FMT=*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
253 WRITE(ULogi,*) 'The Force will be with you'
254 CLOSE(UNIT = 10)
255 lerror = 1
256 END IF
257 !-----
258 !
259 ELSE
260 OPEN(UNIT = ULogi, FILE = 'Prtc13D_Logi.log', STATUS = 'REPLACE', ACTION = 'WRITE')
261 WRITE (ULogi,*) 'DATE: ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
262 WRITE (ULogi,*) 'Error, little Padawan, InputData has a mistake!'
263 WRITE (ULogi,*) 'number of error'
264 WRITE (ULogi,*) 'First Error index = ', fstatus
265 WRITE (ULogi,*) ' message of error:'
266 WRITE (ULogi,FMT=*) TRIM(msg)
267 WRITE (ULogi,*) ' message of error ended'
268 WRITE (ULogi,*) 'Now, go to find the mistake, Do not see this more time!'
269 WRITE (ULogi,FMT=*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
270 WRITE(ULogi,*) 'The Force will be with you'
271 CLOSE(UNIT = 10)
272 WRITE (ULogi,*) ' DATE: ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
273 lerror = 1
274 END IF
275 !-----
276 ! Open TRMIII files statements.
277 !-----
278 ! This part has the new feature in OPEN statement. IMSG is very important.
279 !-----
280 OPEN(UNIT = 10, FILE = 'TRMCoefs.nml', STATUS = 'OLD', ACTION = 'READ', & !there is IOSMG in all open
281 & DELIM = 'QUOTE', IOSTAT = fstatus, IOMSG = msg) !statements
282 !-----
283 !The first statement is obtain the data and time.
284 CALL DATE_AND_TIME( DATE, TIME)
285 !-----
286 ! In this part, the file Prtc1_3D.nml is opened and red. The code has two warning statements, the first is
287 !about the existence of file, and the second is about the file has the correct values in their variables.
288 !The use of this conditions is necessary to find problems with the input data.

```

```

288 IF (fstatus == 0) then
289 READ(UNIT = 10, NML=TRMCoefs, IOSTAT = sstatus, IOMSG = msg)
290 !-----
291 !This is the second condition, here there is a warning about the structure of file Prctl_3D
292 IF (sstatus == 0) then
293 CLOSE(UNIT = 10)
294 ELSE
295 OPEN(UNIT = ULog1, FILE = 'Prctl3D_Log1.log', STATUS = 'REPLACE', ACTION = 'WRITE')
296 WRITE (ULog1,*) ' DATE: ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
297 WRITE (ULog1,*) 'Error, little Padawan, TRMCoefs has a mistake!'
298 WRITE (ULog1, *) 'First Error Index = ', fstatus
299 WRITE (ULog1,*) 'number of error'
300 WRITE (ULog1, *) 'Second Error Index = ', sstatus
301 WRITE (ULog1,*) ' message of error:'
302 WRITE (ULog1,FMT=*) TRIM(msg)
303 WRITE (ULog1,*) ' message of error ended'
304 WRITE (ULog1,*) 'Now, go to find the mistake, Do not see this more time!'
305 WRITE (ULog1,FMT=*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
306 WRITE(ULog1,*) 'The Force will be with you'
307 CLOSE(UNIT = 10)
308 Ierror = 1
309 END IF
310 !-----
311 ELSE
312 OPEN(UNIT = ULog1, FILE = 'Prctl3D_Log1.log', STATUS = 'REPLACE', ACTION = 'WRITE')
313 WRITE (ULog1,*) ' DATE: ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
314 WRITE (ULog1,*) 'Error, little Padawan, TRMCoefs has a mistake!'
315 WRITE (ULog1,*) 'number of error'
316 WRITE (ULog1, *) 'First Error index = ', fstatus
317 WRITE (ULog1,*) ' message of error:'
318 WRITE (ULog1,FMT=*) TRIM(msg)
319 WRITE (ULog1,*) ' message of error ended'
320 WRITE (ULog1,*) 'Now, go to find the mistake, Do not see this more time!'
321 WRITE (ULog1,FMT=*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
322 WRITE(ULog1,*) 'The Force will be with you'
323 CLOSE(UNIT = 10)
324 WRITE (ULog1,*) ' DATE: ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
325 Ierror = 1
326 END IF
327 !-----
328 ! The next step is to have an IF structure to decide and to continue the code. The name was init, it was
329 ! changed by bolt, because is a part of a door and is a statement for coming inside the code.
330 Bolt: IF (sstatus == 0) THEN
331 !-----
332 ! The initial code had the condition on fstatus, because that code don't have IOMSG option. This new code has
333 ! the condition around second error index called sstatus. This statement establish a second look if ther is a
334 ! mistake in the data input file.
335 !-----
336 ! The nex step is to open the file Prctl3D_Dat which has the values of drop's geometry.
337 OPEN(UNIT = Udat, FILE = 'Prctl3D_Dat.dat', &
338 & STATUS = 'REPLACE', ACTION = 'WRITE')
339 WRITE (Udat,*) ' Prctl3D_Dat.dat '
340 WRITE (Udat,*) ' DATE: ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
341 WRITE (Udat,*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
342 WRITE (Udat,*) ' '
343 !-----
344 ! prepare the file Prctl3D_log.
345 OPEN(UNIT = Ulog, FILE = 'Prctl3D_log.log', &
346 & STATUS = 'REPLACE', ACTION = 'WRITE')
347
348 WRITE (Ulog,*) ' DATE: ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
349 WRITE (Ulog,*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
350 WRITE (Ulog,*) ' '
351 WRITE (Ulog,*) ndiv
352 !-----
353 ! prepare the file Prctl3D_Geo for visualization in Mathematica.
354 OPEN (UNIT = Ugeo, FILE = 'Prctl3D_Geo.dat', &
355 & STATUS = 'REPLACE', ACTION = 'WRITE')
356 !-----
357 ! Read TRM Coefficients. Today (16 / 08 /2012) there is not used.
358 ! IF (iflow == 3) THEN
359 ! OPEN(UNIT = 10, FILE = 'TRMCoefs.nml', STATUS = 'OLD', &

```

```

360 ! & ACTION = 'READ', DELIM = 'QUOTE', IOSTAT = astatus)
361 ! IF (astatus == 0) THEN
362 ! | 'Procede to read data from file'
363 ! READ (UNIT = 10, NML=TRMCoefs, IOSTAT = fstatus)
364 ! CLOSE (UNIT = 10)
365 ! END IF
366 ! END IF
367 !-----
368 !-----
369 ! Triangulate the unit sphere
370 ! In this par there is a method todiscretize shape and generate the connectivity table
371 !-----
372 CALL trgl_octa(ndiv, npts, nelm)
373 ! Allocation of p, n, ne and nbe occurs in this subroutine.
374 !-----
375 !-----
376 ! Triangulate the unit sphere
377 ! In this par there is a file with the geometry, the values of
378 !-----
379 ! Call the subroutine trgl_octa
380 ! CALL trgl_octa_read(ndiv, npts, nelm)
381 ! Allocation of p, n, ne and nbe occurs in this subroutine.
382 !-----
383 !-----
384 ! The next part computes the expansion to specified shape and equivalent radius. The idea is introduce this
385 ! statements in a new subroutine.
386 CALL drop_scale_adjust(p, npts, nelm, &
387 & sboa, coa, req, &
388 & eps, czp, &
389 & scxp, cyp, czp, &
390 & sph1, ph12, ph13 )
391 !-----
392 !-----
393 ! The follow action is to updated the quadratures
394 !-----
395 ! Read the quadratures. First there are an allocate statement for arrays, then the values of coefficients are
396 ! read and finally there is the compute of quadratures over each element.
397 ALLOCATE (alphaQ(nelm), betaQ(nelm), gammaQ(nelm))
398 alphaQ = 0.000
399 betaQ = 0.000
400 gammaQ = 0.000
401 CALL Gauss_Legendre(NGL) ! Allocation of WW and ZZ
402 CALL Gauss_Trgl(mint) ! Allocation of xiq, etq, wq
403 !-----
404 ! Here we develop the advance in one step of time
405 !-----
406 ! Compute the coefficients alphaQ, betaQ, gammaQ, for the quadratic xi-eta mapping of each element
407 !$omp parallel
408 !$omp do
409 DO k = 1, nelm
410 i1 = n(k,1)
411 i2 = n(k,2)
412 i3 = n(k,3)
413 i4 = n(k,4)
414 i5 = n(k,5)
415 i6 = n(k,6)
416 CALL abc(p(i1,1), p(i1,2), p(i1,3), &
417 & p(i2,1), p(i2,2), p(i2,3), &
418 & p(i3,1), p(i3,2), p(i3,3), &
419 & p(i4,1), p(i4,2), p(i4,3), &
420 & p(i5,1), p(i5,2), p(i5,3), &
421 & p(i6,1), p(i6,2), p(i6,3), &
422 & alphaQ(k), betaQ(k), gammaQ(k))
423 END DO
424 !$omp end do
425 !$omp end parallel
426 !-----
427 !-----
428 ! Collocation points will be placed at the element centroids
429 ! Compute:
430 ! 1) coordinates of collocation points
431 ! 2) normal vector at collocation points

```

```

432 !-----
433 ! There are the ALLOCATE statement and the initialization of values in the arrays used to have the
434 ! features of collocation points.
435 ALLOCATE ( & x0(nelm), y0(nelm), z0(nelm), &
436 & vx0(nelm), vny0(nelm), vnz0(nelm), &
437 & vna(6*nelm,4))
438 x0 = 0.000
439 y0 = 0.000
440 z0 = 0.000
441 vx0 = 0.000
442 vny0 = 0.000
443 vnz0 = 0.000
444 vna1 = 0.000
445 x1 = 1.000/3.000
446 eta = 1.000/3.000
447 !$omp parallel
448 !$omp do
449 !-----
450 DO i = 1, nelm
451 ! will be interpolate the normal vector and features of collocation points.
452 i1 = n(i,1)
453 i2 = n(i,2)
454 i3 = n(i,3)
455 i4 = n(i,4)
456 i5 = n(i,5)
457 i6 = n(i,6)
458 !-----
459 CALL interp_p(p(i1,1), p(i1,2), p(i1,3), &
460 & p(i2,1), p(i2,2), p(i2,3), &
461 & p(i3,1), p(i3,2), p(i3,3), &
462 & p(i4,1), p(i4,2), p(i4,3), &
463 & p(i5,1), p(i5,2), p(i5,3), &
464 & p(i6,1), p(i6,2), p(i6,3), &
465 & alpha0(i), beta0(i), gamma0(i), &
466 & xi, eta, &
467 & x0(i), y0(i), z0(i), &
468 & DxDx1, DyDy1, DxDz1, &
469 & DxDet, DyDet, DzDet, &
470 & vx0(i), vny0(i), vnz0(i), &
471 & hs)
472 !-----
473 END DO
474 !$omp end do
475 !$omp end parallel
476 !-----
477 ! compute:
478 ! The surface area of the individual elements x, y, and z moments over each element
479 ! The particle surface area and volume
480 ! The mean curvature of each element: crvml
481 ! The averaged element normal vector
482 !-----
483 ALLOCATE (arel(nelm), crvml(nelm), farel(nelm), &
484 & xmom(nelm), ymom(nelm), zmom(nelm))
485 stride = 0
486 !$omp parallel
487 CALL elm_geom(nelm, npts, mint, &
488 & xmom, ymom, zmom, &
489 & srf_area, prt_vlm, &
490 & cx, cy, cz, stride)
491 !$omp end parallel
492 CALL interp_curv(nelm, ne, nbe, &
493 & crvml)
494 !-----
495 !CALL elm_geom4(nelm, npts, mint, &
496 !& xmom, ymom, zmom, &
497 !& srf_area, prt_vlm, &
498 !& cx, cy, cz)
499 !-----
500 ! normalize surface area and volume
501 srf_area_n = srf_area/(pi4*req*req)
502 prt_vlm_n = prt_vlm/(pi4*req*req/3.000)
503 !-----

```

```

504 ! The obtained values are writing on UDATA file.
505 WRITE (Udat,*)
506 WRITE (Udat,*) stride
507 WRITE (Udat,110) srf_area_n
508 WRITE (Udat,111) prt_vlm_n
509 WRITE (Udat,112) cx,cy,cz
510 WRITE (Udat,*)
511 thet0 = alpha
512 !-----
513 !$omp parallel
514 CALL D_taylor(p, npts, DT, DTm, DTmm, lmajor, bminor, wminor, angulo, cx, cy, cz, req)
515 Tiempo = DBLE(stride)*h*gi
516 !$omp end parallel
517 WRITE (Ugeo,103) Tiempo, lmajor, bminor, wminor, DT, DTm, DTmm, angulo, bminor/wminor, 0.000
518 !-----
519 CALL write_datan(nelm, stride)
520 CALL write_datac(nelm, stride)
521 CALL write_datak(nelm, stride)
522 ! CALL write_datar(nelm, stride)
523 md = IFlow
524 alphantom = alpha
525 !-----
526 CALL interp_evm(npts, ne, nbe, &
527 & vx0, vny0, vnz0, &
528 & vna)
529 !-----
530 ! Three rows at a time,
531 ! corresponding to the x, y, z
532 ! components of the integral equation
533 !-----
534 ! System size
535 !The system size is obtained in the next sentences, the sistem is tridimensional, therefore, the size is
536 !threefold element size. That because we have coordinates in x, y and z.
537 !-----
538 Mdim = 3*nelm
539 ALLOCATE(GM(Mdim), TM(Mdim,Mdim), Uoo(Mdim), Vel0(Mdim), Vel1(Mdim))!, rhs(Mdim,3), sln(Mdim,3), &
540 ! & ipiv(Mdim))
541 ! initialize
542 GM = 0.000
543 Uoo = 0.000
544 Vel0 = 0.000
545 CALL dissociate(nelm, Vel0, &
546 & Vel0x, Vel0y, Vel0z)
547 Vel1 = 0.000
548 TM = 0.000
549 CALL write_datavl(nelm, stride, Vel0x, Vel0y, Vel0z)
550 !-----
551 ! The coefficient lamda mu is obtained here.
552 cdg = 1.000/(pi8*visc)
553 lmu = visc2/visc
554 cdt = (1.000-lmu)/(pi8)
555 !-----
556 ! Running over the matrix GM and TM
557 CALL Builder_GM(GM, cdg, nelm, gamma, mint, NGL)
558 CALL Builder_TM(TM, cdt, lmu, nelm, mint, NGL)
559 ! CALL write_datak6(nelm, stride, GM)
560 !-----
561 CALL Infinity_flow(IFlow, alpha, gi, Uoo, stride, steps, nelm, h)
562 !-----
563 Vel0 = GM - Uoo
564 !-----
565 CALL answer_axb1(Mdim, TM, Vel0, Vel1)
566 !-----
567 CALL dissociate(nelm, Vel1, &
568 & Vel1x, Vel1y, Vel1z)
569 !-----
570 ALLOCATE ( xel(nelm), yel(nelm), zel(nelm), &
571 & pel(npts, 3))
572 ! ALLOCATE(K1(npts,3), K2(npts,3), K3(npts,3))
573 ALLOCATE(K1(nelm), K2(nelm), K3(nelm))
574 xel = 0.000
575 yel = 0.000

```

```

576      zel = 0.000
577      pel = 0.000
578      K1 = 0.000
579      K2 = 0.000
580      K3 = 0.000
581 !=====
582 !      CALL Euler_Method3(x0, y0, z0,&
583 !                      & Velix, Velyi, Veliz, &
584 !                      &h, nelm, omicron)
585 !=====
586 !      CALL interp_e( npts, ne, nbe,&
587 !                  & x0, y0, z0,&
588 !                  & p
589 !=====
590 !      CALL interp_ven( npts, ne, nbe,&
591 !                    & Velix, Velyi, Veliz, Vel)
592 !=====
593 ! The RK3 starts
594 !=====
595      xel=x0
596      yel=y0
597      zel=z0
598      K1 = Veli
599      pel= p
600 !=====
601 !      CALL write_datan(nelm, stride)
602 !      CALL write_dataac(nelm, stride)
603 !      CALL write_dataak(nelm, stride)
604      istop = doomsday
605      doomsday = 1
606 !=====
607 ! This is the master loop to advance the time in dynamic drop deformation.
608 ! The next lines are the same over here. However, the comands will be without comments unless will be necessary
609 ! The Force will be with you, The year of our Lord 2013.
610 !=====
611      DO m = 1, steps / 2
612 !=====
613 !      CALL Euler_Met5(p, 0.500*h, K1, npts, omicron)
614 !=====
615      CALL RK31(x0, y0, z0, 0.500*h, K1, nelm, omicron)
616 !=====
617      CALL interp_e( npts, ne, nbe,&
618 !                  & x0, y0, z0,&
619 !                  & p
620 !=====
621      select case (md)
622      case (1:11)
623          if ( m > stoper) then
624              iflow = 1
625          else
626              iflow=md
627          end if
628      case (12)
629          if ( m > reducer) then
630              iflow = 13
631          else
632              iflow = 12
633          end if
634      case (16)
635          if ( m > reducer) then
636              iflow = 17
637          else
638              iflow = 16
639          end if
640      case (15)
641          if ( m > reducer) then
642              iflow = 15
643          else
644              iflow = 10
645          end if
646      case (19)
647          if ( m > stoper) then

```

```

648          iflow = 13
649          alpha = alphantom
650          thet0 = ((2.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
651          !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
652          elseif ( m > reducer) then
653              iflow = 12
654              alpha = alphantom
655              thet0 = ((-0.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
656              !thet0 = (((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride))))
657          else
658              iflow = 10
659              alpha = 0.000
660              thet0 = alpha
661          end if
662      case (20)
663          if ( m > stoper) then
664              iflow = 17
665              alpha = alphantom
666              thet0 = ((2.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
667              !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
668          elseif ( m > reducer) then
669              iflow = 16
670              alpha = alphantom
671              thet0 = ((-0.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
672              !thet0 = (((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride))))
673          else
674              iflow = 10
675              alpha = 0.000
676              thet0 = alpha
677          end if
678      case (21)
679          if ( m > stoper) then
680              iflow = 16
681              alpha = alphantom
682              thet0 = ((-1.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
683              !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
684          elseif ( m > reducer) then
685              iflow = 17
686              alpha = alphantom
687              thet0 = ((1.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
688              !thet0 = (((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride))))
689          else
690              iflow = 10
691              alpha = 1.000
692              thet0 = alpha
693          end if
694      case (22)
695          if ( m > stoper) then
696              iflow = 72
697              alpha = alphantom
698              thet0 = ((2.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
699              !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
700          elseif ( m > reducer) then
701              iflow = 71
702              alpha = alphantom
703              thet0 = ((-0.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
704              !thet0 = (((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride))))
705          else
706              iflow = 7
707              alpha = 0.000
708              thet0 = alpha
709          end if
710      case (23)
711          if ( m > stoper) then
712              iflow = 71
713              alpha = alphantom
714              thet0 = ((-1.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
715              !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
716          elseif ( m > reducer) then
717              iflow = 72
718              alpha = alphantom
719              thet0 = ((1.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))

```

```

720      !thet0 = ((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
721      else
722          iflow = 7
723          alpha = 1.000
724          thet0 = alpha
725      end if
726      case default
727          iflow = 10
728      end select
729 !=====
730 CALL Connector_vol(p, npts, prt_vlm_n, cx, cy, cz)
731 !=====
732 !$omp parallel
733 !$omp do
734     DO k = 1, nelm
735         i1 = n(k,1)
736         i2 = n(k,2)
737         i3 = n(k,3)
738         i4 = n(k,4)
739         i5 = n(k,5)
740         i6 = n(k,6)
741         CALL abc(p(i1,1), p(i1,2), p(i1,3), &
742                & p(i2,1), p(i2,2), p(i2,3), &
743                & p(i3,1), p(i3,2), p(i3,3), &
744                & p(i4,1), p(i4,2), p(i4,3), &
745                & p(i5,1), p(i5,2), p(i5,3), &
746                & p(i6,1), p(i6,2), p(i6,3), &
747                & alphaQ(k), betaQ(k), gammaQ(k))
748     END DO
749 !$omp end do
750 !$omp end parallel
751 !-----
752     x0 = 0.000
753     y0 = 0.000
754     z0 = 0.000
755     vx0 = 0.000
756     vny0 = 0.000
757     vnz0 = 0.000
758     xi = 1.000/3.000
759     eta = 1.000/3.000
760 !-----
761 !$omp parallel
762 !$omp do
763     DO i = 1, nelm
764         ! will be interpolate the normal vector and features of collocation points.
765         i1 = n(i,1)
766         i2 = n(i,2)
767         i3 = n(i,3)
768         i4 = n(i,4)
769         i5 = n(i,5)
770         i6 = n(i,6)
771 !-----
772         CALL interp_p(p(i1,1), p(i1,2), p(i1,3), &
773                    & p(i2,1), p(i2,2), p(i2,3), &
774                    & p(i3,1), p(i3,2), p(i3,3), &
775                    & p(i4,1), p(i4,2), p(i4,3), &
776                    & p(i5,1), p(i5,2), p(i5,3), &
777                    & p(i6,1), p(i6,2), p(i6,3), &
778                    & alphaQ(i), betaQ(i), gammaQ(i), &
779                    & xi, eta, &
780                    & x0(i), y0(i), z0(i), &
781                    & DxDx1, DyDx1, DzDx1, &
782                    & DxDet, DyDet, DzDet, &
783                    & vx0(i), vny0(i), vnz0(i), &
784                    & hs)
785 !-----
786     END DO
787 !$omp end do
788 !$omp end parallel
789 !-----
790 CALL elm_geom(nelm, npts, mint, &
791              & xmom, ymom, zmom, &

```

```

792     & srf_area, prt_vlm, &
793     & cx, cy, cz, stride)
794 !call interp_curv( nelm, ne, nbe,&
795                 & crvmel)
796 !-----
797 ! normalize surface area and volume
798 srf_area_n = srf_area / (pi4*req*req)
799 prt_vlm_n = prt_vlm / (pi4*req*req*req/3.000)
800 !-----
801 CALL interp_evn(npts, ne, nbe,&
802               & vx0, vny0, vnz0,&
803               & vna)
804 !-----
805 GM = 0.000
806 Uoo = 0.000
807 Vel0 = 0.000
808 Vell = 0.000
809 TM = 0.000
810 !-----
811 CALL Builder_GM(GM, cdg, nelm, gamma, mint, NGL)
812 CALL Builder_TM(TM, cdt, lmu, nelm, mint, NGL)
813 !-----
814 CALL Infinity_Flow(iflow, thet0, gl, Uoo, m, steps, nelm, h)
815 Vel0 = GM - Uoo
816 !-----
817 CALL answer_xbi(Mdim, TM, Vel0, Vell)
818 !-----
819 CALL dissociate(nelm, Vell, &
820               & Velix, Velly, Vellz)
821 !-----
822 !CALL interp_ven( npts, ne, nbe,&
823                & Vellx, Velly, Vellz, Vel)
824 !-----
825 K2 = Vell
826 p = pel
827 x0 = xel
828 y0 = yel
829 z0 = zel
830 !-----
831 ! CALL Euler_Met6(p, h, K1, K2, npts, omicron)
832 !-----
833 CALL RK32(x0, y0, z0, h, K1, K2, nelm, omicron)
834 !-----
835 CALL interp_e( npts, ne, nbe,&
836               & x0, y0, z0,&
837               & p )
838 !-----
839 select case (md)
840 case (1:11)
841     if ( m > stoper) then
842         iflow = 1
843     else
844         iflow=md
845     end if
846 case (12)
847     if ( m > reducer) then
848         iflow = 13
849     else
850         iflow = 12
851     end if
852 case (16)
853     if ( m > reducer) then
854         iflow = 17
855     else
856         iflow = 16
857     end if
858 case (15)
859     IF ( m > reducer) then
860         iflow = 15
861     else
862         iflow = 10
863     end if

```

```

864 case (19)
865   if (m > stoper) then
866     iflow = 13
867     alpha = alphantom
868     thet0 = ((2.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
869     !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
870   elseif (m > reducer) then
871     iflow = 12
872     alpha = alphantom
873     thet0 = ((-0.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
874     !thet0 = ((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
875   else
876     iflow = 10
877     alpha = 0.000
878     thet0 = alpha
879   end if
880 case (20)
881   if (m > stoper) then
882     iflow = 17
883     alpha = alphantom
884     thet0 = ((2.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
885     !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
886   elseif (m > reducer) then
887     iflow = 16
888     alpha = alphantom
889     thet0 = ((-0.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
890     !thet0 = ((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
891   else
892     iflow = 10
893     alpha = 0.000
894     thet0 = alpha
895   end if
896 case (21)
897   if (m > stoper) then
898     iflow = 16
899     alpha = alphantom
900     thet0 = ((-1.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
901     !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
902   elseif (m > reducer) then
903     iflow = 17
904     alpha = alphantom
905     thet0 = ((1.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
906     !thet0 = ((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
907   else
908     iflow = 10
909     alpha = 1.000
910     thet0 = alpha
911   end if
912 case (22)
913   if (m > stoper) then
914     iflow = 72
915     alpha = alphantom
916     thet0 = ((2.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
917     !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
918   elseif (m > reducer) then
919     iflow = 71
920     alpha = alphantom
921     thet0 = ((-0.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
922     !thet0 = ((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
923   else
924     iflow = 7
925     alpha = 0.000
926     thet0 = alpha
927   end if
928 case (23)
929   if (m > stoper) then
930     iflow = 71
931     alpha = alphantom
932     thet0 = ((-1.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
933     !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
934   elseif (m > reducer) then
935     iflow = 72

```

```

936     alpha = alphantom
937     thet0 = ((1.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
938     !thet0 = ((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
939   else
940     iflow = 7
941     alpha = 1.000
942     thet0 = alpha
943   end if
944 case default
945   iflow = 10
946   end select
947 !-----
948 CALL Corrector_vol(p, npts, prt_vlm_n, cx, cy, cz)
949 !-----
950 !$omp parallel
951 !$omp do
952   DO k = 1, nelm
953     i1 = n(k,1)
954     i2 = n(k,2)
955     i3 = n(k,3)
956     i4 = n(k,4)
957     i5 = n(k,5)
958     i6 = n(k,6)
959     CALL abc(p(i1,1), p(i1,2), p(i1,3), &
960            & p(i2,1), p(i2,2), p(i2,3), &
961            & p(i3,1), p(i3,2), p(i3,3), &
962            & p(i4,1), p(i4,2), p(i4,3), &
963            & p(i5,1), p(i5,2), p(i5,3), &
964            & p(i6,1), p(i6,2), p(i6,3), &
965            & alphas(k), betas(k), gammas(k))
966   END DO
967 !$omp end do
968 !$omp end parallel
969 !-----
970     x0 = 0.000
971     y0 = 0.000
972     z0 = 0.000
973     vx0 = 0.000
974     vy0 = 0.000
975     vz0 = 0.000
976     xi = 1.000/3.000
977     eta = 1.000/3.000
978 !-----
979 !$omp parallel
980 !$omp do
981   DO i = 1, nelm
982     ! will be interpolate the normal vector and features of collocation points.
983     i1 = n(i,1)
984     i2 = n(i,2)
985     i3 = n(i,3)
986     i4 = n(i,4)
987     i5 = n(i,5)
988     i6 = n(i,6)
989 !-----
990     CALL interp_p(p(i1,1), p(i1,2), p(i1,3), &
991            & p(i2,1), p(i2,2), p(i2,3), &
992            & p(i3,1), p(i3,2), p(i3,3), &
993            & p(i4,1), p(i4,2), p(i4,3), &
994            & p(i5,1), p(i5,2), p(i5,3), &
995            & p(i6,1), p(i6,2), p(i6,3), &
996            & alphas(i), betas(i), gammas(i), &
997            & xi, eta, &
998            & x0(i), y0(i), z0(i), &
999            & dx0xi, dy0xi, dz0xi, &
1000            & dx0et, dy0et, dz0et, &
1001            & vx0(i), vy0(i), vz0(i), &
1002            & hs)
1003 !-----
1004   END DO
1005 !$omp end do
1006 !$omp end parallel
1007 !-----

```



```

1008 CALL elm_geom(nelm, npts, mint, &
1009 & xcom, ycom, zcom, &
1010 & srf_area, prt_vlm, &
1011 & cx, cy, cz, stride)
1012 Call interp_curv(nelm, ne, nbe,&
1013 & crvmel)
1014 !=====
1015 ! normalize surface area and volume
1016 srf_area_n = srf_area / (pi*req*req)
1017 prt_vlm_n = prt_vlm / (pi*req*req*req/3.000)
1018 !=====
1019 CALL interp_evn(npts, ne, nbe,&
1020 & vnx0, vny0, vnz0,&
1021 & vna)
1022 !=====
1023 GM = 0.000
1024 Uoo = 0.000
1025 Vel0 = 0.000
1026 Vell = 0.000
1027 TM = 0.000
1028 !=====
1029 CALL Builder_GM(GM, cdg, nelm, gamma, mint, NGL)
1030 CALL Builder_TM(TM, cdt, lmu, nelm, mint, NGL)
1031 !=====
1032 CALL Infinity_flow(Iflow, thet0, gi, Uoo, m, steps, nelm, h)
1033 Vel0 = GM - Uoo
1034 !=====
1035 CALL answer_pdb(dia, TM, Vel0, Vell)
1036 !=====
1037 CALL dissociate(nelm, Vell, &
1038 & Vellx, Velly, Vellz)
1039 !=====
1040 !CALL interp_vnf(npts, ne, nbe,&
1041 ! & Vellx, Velly, Vellz, Vel)
1042 !=====
1043 K3 = Vell
1044 p = pel
1045 x0 = xel
1046 y0 = yel
1047 z0 = zel
1048 !=====
1049 ! CALL RK3(p, K1, K2, K3, npts, h)
1050 CALL RK3(x0, y0, z0, &
1051 & K1, K2, K3, nelm, h)
1052 !=====
1053 CALL interp_e(npts, ne, nbe,&
1054 & x0, y0, z0,&
1055 & p )
1056 pel = p
1057 !=====
1058 select case (md)
1059 case (1:11)
1060 if (m > stoper) then
1061 Iflow = 1
1062 else
1063 Iflow=md
1064 end if
1065 case (12)
1066 if (m > reducer) then
1067 Iflow = 13
1068 else
1069 Iflow = 12
1070 end if
1071 case (16)
1072 if (m > reducer) then
1073 Iflow = 17
1074 else
1075 Iflow = 16
1076 end if
1077 case (15)
1078 if (m > reducer) then
1079 Iflow = 15

```

```

1080 else
1081 Iflow = 10
1082 end if
1083 case (18)
1084 if (m > stoper) then
1085 Iflow = 1
1086 else
1087 Iflow = 18
1088 end if
1089 case (19)
1090 if (m > stoper) then
1091 Iflow = 13
1092 alpha = alphantom
1093 thet0 = ((2.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
1094 !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
1095 elseif (m > reducer) then
1096 Iflow = 12
1097 alpha = alphantom
1098 thet0 = ((-0.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
1099 !thet0 = ((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
1100 else
1101 Iflow = 18
1102 alpha = 0.000
1103 thet0 = alpha
1104 end if
1105 case (20)
1106 if (m > stoper) then
1107 Iflow = 17
1108 alpha = alphantom
1109 thet0 = ((2.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
1110 !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
1111 elseif (m > reducer) then
1112 Iflow = 16
1113 alpha = alphantom
1114 thet0 = ((-0.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
1115 !thet0 = ((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
1116 else
1117 Iflow = 10
1118 alpha = 0.000
1119 thet0 = alpha
1120 end if
1121 case (21)
1122 if (m > stoper) then
1123 Iflow = 16
1124 alpha = alphantom
1125 thet0 = ((-1.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
1126 !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
1127 elseif (m > reducer) then
1128 Iflow = 17
1129 alpha = alphantom
1130 thet0 = ((1.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
1131 !thet0 = ((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
1132 else
1133 Iflow = 10
1134 alpha = 1.000
1135 thet0 = alpha
1136 end if
1137 case (22)
1138 if (m > stoper) then
1139 Iflow = 72
1140 alpha = alphantom
1141 thet0 = ((2.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
1142 !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
1143 elseif (m > reducer) then
1144 Iflow = 71
1145 alpha = alphantom
1146 thet0 = ((-0.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
1147 !thet0 = ((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
1148 else
1149 Iflow = 7
1150 alpha = 0.000
1151 thet0 = alpha

```

```

1152     end if
1153     case (23)
1154     if (m > stoper) then
1155         iflow = 71
1156         alpha = alphantom
1157         thet0 = ((-1.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
1158         !thet0 = ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
1159     elseif (m > reducer) then
1160         iflow = 72
1161         alpha = alphantom
1162         thet0 = ((1.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))
1163         !thet0 = ((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride)))
1164     else
1165         iflow = 7
1166         alpha = 1.000
1167         thet0 = alpha
1168     end if
1169     case default
1170     iflow = 10
1171     end select
1172 !-----
1173     CALL Corrector_vol(p, npts, prt_vlm_n, cx, cy, cz)
1174 !-----
1175 !$omp parallel
1176 !$omp do
1177     DO k = 1, nelm
1178         11 = n(k,1)
1179         12 = n(k,2)
1180         13 = n(k,3)
1181         14 = n(k,4)
1182         15 = n(k,5)
1183         16 = n(k,6)
1184         CALL abc(p(i1,1), p(i1,2), p(i1,3), &
1185             & p(i2,1), p(i2,2), p(i2,3), &
1186             & p(i3,1), p(i3,2), p(i3,3), &
1187             & p(i4,1), p(i4,2), p(i4,3), &
1188             & p(i5,1), p(i5,2), p(i5,3), &
1189             & p(i6,1), p(i6,2), p(i6,3), &
1190             & alphaQ(k), betaQ(k), gammaQ(k))
1191     END DO
1192 !$omp end do
1193 !$omp end parallel
1194 !-----
1195     x0 = 0.000
1196     y0 = 0.000
1197     z0 = 0.000
1198     vx0 = 0.000
1199     vny0 = 0.000
1200     vnz0 = 0.000
1201     xi = 1.000/3.000
1202     eta = 1.000/3.000
1203 !-----
1204 !$omp parallel
1205 !$omp do
1206     DO i = 1, nelm
1207     ! will be interpolate the normal vector and features of collocation points.
1208         11 = n(i,1)
1209         12 = n(i,2)
1210         13 = n(i,3)
1211         14 = n(i,4)
1212         15 = n(i,5)
1213         16 = n(i,6)
1214 !-----
1215         CALL interp_p(p(i1,1), p(i1,2), p(i1,3), &
1216             & p(i2,1), p(i2,2), p(i2,3), &
1217             & p(i3,1), p(i3,2), p(i3,3), &
1218             & p(i4,1), p(i4,2), p(i4,3), &
1219             & p(i5,1), p(i5,2), p(i5,3), &
1220             & p(i6,1), p(i6,2), p(i6,3), &
1221             & alphaQ(i), betaQ(i), gammaQ(i), &
1222             & xi, eta, &
1223             & x0(i), y0(i), z0(i), &

```

```

1224         & DxDx1, DyDx1, DzDx1, &
1225         & DxDet, DyDet, DzDet, &
1226         & vnx0(i), vny0(i), vnz0(i), &
1227         & h)
1228 !-----
1229     END DO
1230 !$omp end do
1231 !$omp end parallel
1232 !-----
1233     CALL elm_geom(nelm, npts, mint, &
1234         & xmom, ymom, zmom, &
1235         & srf_area, prt_vlm, &
1236         & cx, cy, cz, stride)
1237     !call interp_curv(nelm, ne, nbe,&
1238         ! & crvml)
1239 !-----
1240 ! normalize surface area and volume
1241     srf_area_n = srf_area / (pi*req*req)
1242     prt_vlm_n = prt_vlm / (pi*req*req*req/3.000)
1243 !-----
1244 ! The obtained values are writing on UData file.
1245     WRITE (Udat,*)
1246     WRITE (Udat,i07) m
1247     WRITE (Udat,i10) srf_area_n
1248     WRITE (Udat,i11) prt_vlm_n
1249     WRITE (Udat,i12) cx,cy,cz
1250     WRITE (Udat,*) md, iflow
1251     WRITE (Udat,*)
1252 !-----
1253     CALL D_taylor(p, npts, DT, DTm, DTmm, lmajor, bminor, wminor, angulo, cx, cy, cz, req)
1254     Tiempo = DBLE(m)*h*gi
1255     WRITE (UGeo,i03) Tiempo, lmajor, bminor, wminor, DT, DTm, DTmm, angulo, bminor/wminor, thet0
1256 !-----
1257     CALL interp_evn(npts, ne, nbe,&
1258         & vnx0, vny0, vnz0,&
1259         & vna)
1260 !-----
1261     GM = 0.000
1262     Uoo = 0.000
1263     Vel0 = 0.000
1264     Vel1 = 0.000
1265     TW = 0.000
1266 !-----
1267     CALL Builder_GM(GM, cdg, nelm, gamma, mint, NGL)
1268     CALL Builder_TM(TM, cdt, lmu, nelm, mint, NGL)
1269 !-----
1270     CALL Infinity_Flow(iflow, thet0, gl, Uoo, m, steps, nelm, h)
1271     Vel0 = GM - Uoo
1272 !-----
1273     CALL answer_axb1(Mdim,TM,Vel0,Vel1)
1274 !-----
1275     CALL dissociate(nelm, Vell, &
1276         & Vellx, Velly, Vellz)
1277 !-----
1278     !CALL interp_ven(npts, ne, nbe,&
1279         ! & Vellx, Velly, Vellz, Vel)
1280 !-----
1281     K1 = Vel1
1282     xel= x0
1283     yel= y0
1284     zel= z0
1285 !-----
1286     stride = m
1287     if (doomsday == istop) then
1288         CALL write_datan(nelm, stride)
1289         CALL write_datak(nelm, stride)
1290         CALL write_dataak(nelm, stride)
1291         CALL write_datav1(nelm, stride, Vellx, Velly, Vellz)
1292         doomsday=1
1293     else
1294         doomsday = doomsday + 1
1295     end if

```

```

1296 !=====
1297 END DO
1298 !=====
1299 CALL CPU_TIME(CPUTime)
1300 WRITE (Udat,*) ' CPU Time: ', CPUTime, 'seconds'
1301 WRITE (Ulog,*) ' CPU Time: ', CPUTime, 'seconds'
1302 i1 = FLOOR(CPUTime/3600)
1303 CPUTime = CPUTime - i1*3600
1304 i2 = FLOOR(CPUTime/60)
1305 CPUTime = CPUTime - i2*60
1306 WRITE (Udat,i20) i1, i2, NINT(CPUTime)
1307 WRITE (Ulog,i20) i1, i2, NINT(CPUTime)
1308 CALL DATE_AND_TIME(DATE, TIME)
1309 WRITE (Udat,*) ' DATE: ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
1310 WRITE (Udat,*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
1311 WRITE (Udat,*) ' '
1312 WRITE (Ulog,*) ' DATE: ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
1313 WRITE (Ulog,*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
1314 WRITE (Udat,*) ' '
1315 WRITE (Udat,*) ' stime = ', 0
1316 WRITE (Udat,*) ' '
1317 WRITE (Udat,*) ' Status = ', 0
1318 WRITE (Udat,*) ' '
1319 CLOSE (Ugeo)
1320 CLOSE (Udat)
1321 !=====
1322 ! There is the option if there is wrong whit the input data file.
1323 ELSE Bolt
1324 !=====
1325 OPEN(UNIT = Ulog, FILE = 'Prctl3D_Log.log', &
1326 & STATUS = 'REPLACE', ACTION = 'WRITE')
1327 WRITE (Ulog,*) 'Warning, InputData File unavailable!'
1328 WRITE (Ulog,*) 'Warning, InputData File unavailable!'
1329 WRITE (Ulog,*) ' or another Input File is missing!'
1330 WRITE (Ulog,*) ' Error1 = ', lerror
1331 WRITE (Ulog,*) ' Error = ', fstatus
1332 WRITE (Ulog,*) ' message of error', TRIM(msg)
1333 OPEN(UNIT = Ulog1, FILE = 'Prctl3D_Log1.log',STATUS = 'REPLACE', ACTION = 'WRITE')
1334 WRITE (Ulog1,*) ' DATE: ', DATE(1:4), '/', DATE(5:6), '/', DATE(7:8)
1335 WRITE (Ulog1,*) 'Error, little Padawan, InputData has a mistake!'
1336 WRITE (Ulog1,*) 'First Error index = ', fstatus
1337 WRITE (Ulog1,*) 'number of error'
1338 WRITE (Ulog1,*) 'Second Error index = ', sstatus
1339 WRITE (Ulog1,*) ' message of error:'
1340 WRITE (Ulog1,FMT=*) TRIM(msg)
1341 WRITE (Ulog1,*) ' message of error ended'
1342 WRITE (Ulog1,*) 'Now, go to find the mistake, do not see this more time!'
1343 WRITE (Ulog1,FMT=*) ' TIME = ', TIME(1:2), ':', TIME(3:4), ':', TIME(5:6), TIME(7:12)
1344 WRITE(Ulog1,*) 'The Force will be with you'
1345 CLOSE(UNIT = l0)
1346 lerror = 1
1347 END IF Bolt
1348 !=====
1349 ! The code finished here
1350 !=====
1351 ! Arguments to different kind to save datas. There are formats to write.
1352 100 FORMAT(1x,14,10(1x,ES24.16))
1353 101 FORMAT(10(1x,ES24.16))
1354 102 FORMAT(10(1x,ES24.16))
1355 103 FORMAT(10(1x,ES24.16))
1356 104 FORMAT(10(1x,ES24.16))
1357 105 FORMAT(10(1x,ES24.16))
1358 106 FORMAT(1x,ES24.16)
1359 107 FORMAT(8(1x,i10))
1360 108 FORMAT(4(1x,i10))
1361 110 FORMAT(" Surface Area :",ES24.16)
1362 111 FORMAT(" Volume :",ES24.16)
1363 112 FORMAT(" Centroid :",3(ES24.16))
1364 130 FORMAT(" Determinant =",ES24.16)
1365 200 FORMAT(100(1x,ES24.16))
1366 201 FORMAT(10(1x,ES24.16))
1367 205 FORMAT(10(1x,ES24.16))

```

```

1368 ! 210 FORMAT(1x,14,10(1x,ES24.16))
1369 120 FORMAT (1X, ' CPU Time: ', I4, ' ', I2.2, ' ', I2.2)
1370 !=====
1371 END PROGRAM Prctl_3D

```

```

1 MODULE Mod_SharedVars
2 =====
3 ! : Version: 0.7 created on -- / III /2010
4 !
5 !
6 ! : Version: 1.0 created on 09 / 08 / 2012
7 !
8 !
9 ! This module was made to keep the variables which are common in this code. In this module, you would see the
10 ! features of DBL which is a value of real kind. The variables were established in order similar in main code.
11 =====
12 IMPLICIT NONE
13 =====
14 !
15 ! As was mentioned above, in this place was kept the variable DBL. DBL is a variable which permit the code to
16 ! have the same value in all real variables. This decision is very important because it gives versatility at the
17 ! code, in other words, it's possible to use this code in diferents PC's and it doesn't matters the kind of that
18 ! PC, the value will be the same in the space assigned to real variables.
19 !
20 ! The kind number is explained in Chapman books, the value 8 refers to a system of 64x
21 ! INTEGER, PARAMETER :: DBL = 8 !SELECTED_REAL_KIND(p=15, r = 300), alternative value
22 !
23 ! This places will be the value of pi number.
24 ! REAL (KIND = DBL), PARAMETER :: PI = 3.141592653589793238462643383279500
25 !
26 ! This place ther is the value of epsilon number. This number helps to discriminate small values and assigned
27 ! zero to them
28 ! REAL (KIND = DBL) :: epsilon, eps
29 !
30 !
31 ! Common blocks have differents values or as this case similar variables. Below this explanation you would
32 ! find differents common variables used in more than one modules. The variables appear in order of comes in the
33 ! code.
34 !
35 ! Input Data
36 ! INTEGER :: Ns, Np, doomsday
37 !
38 ! Variables of Two-Roll Mill
39 ! INTEGER, PARAMETER :: Ncs = 100 !Number of coefficients to made the sumatories
40 ! REAL (KIND = DBL), DIMENSION(2) :: R, w, vr !Radii of Mills, w is the radial velocity?
41 ! REAL (KIND = DBL), DIMENSION(2) :: OrY !Distance from origin to mill centers (direction y)
42 ! REAL (KIND = DBL), DIMENSION(2) :: sfpnt !Stagnation point coordinates (x,y)
43 ! REAL (KIND = DBL) :: a, de, g, M1, M2 !Geometry Data: a is the distance between lines of
44 ! !foci, de is the mill ceter distance, g is the gap
45 ! REAL (KIND = DBL) :: A0, B0, C0, D0, Kc !First term coefficients and parameter K
46 ! REAL (KIND = DBL), DIMENSION(4) :: An, Bn, Cn, Dn !Coefficients
47 ! REAL (KIND = DBL) :: lambda, shrate ! Flow Type parameter and shear rate.
48 !
49 ! Module trgl_octa
50 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: p, pel
51 ! INTEGER, ALLOCATABLE, DIMENSION(:,:) :: ne
52 ! INTEGER, ALLOCATABLE, DIMENSION(:,:) :: n, nbe, nbel
53 !
54 ! Module Gauss_Coefs
55 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: zz, ww !quadrature weights of Gauss-Legendre quadrature
56 ! REAL (KIND = DBL) :: wall
57 !
58 ! Module Prtl1_3D_Geo
59 ! Quadratic xi-eta isoparametric mapping
60 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: alphaQ, betaQ, gammaQ !quadrature weights of quadrature
61 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: xiQ, etaQ, wq !parameters xi-eta mapping
62 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: arel, farel !area
63 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: crvml !curvature media
64 !
65 ! Module Prtl1_3D_sf, sfs, slp and dlp
66 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x0, y0, z0 !collocation point coordinates
67 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: xel, yel, zel !collocation point coordinates
68 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: vnx0, vny0, vnz0 !normal vector point coordinates
69 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: vnx02, vny02, vnz02 !normal vector point coordinates
70 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: vna, vel1, vna1 !normal vector point coordinates
71 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: vna1 !normal vector point coordinates
72 !

```

```

73 ! Text types
74 ! INTEGER :: Udat = 80
75 ! INTEGER :: UGeo = 85
76 ! INTEGER :: UDef = 86
77 ! INTEGER :: Ulog = 89
78 ! INTEGER :: Ulog1 = 90
79 ! INTEGER :: Uvel = 91
80 ! INTEGER :: Uctr = 95
81 ! INTEGER :: Delf = 92
82 ! INTEGER :: Utmp = 93
83 ! INTEGER :: UDF2 = 87
84 !
85 END MODULE Mod_SharedVars

```

D:\Darth Vader\Escritorio\prtcl\_mkl\Mod\_trgl\_octa.f90 1

```
1 MODULE Mod_Trgl_Octa
2 =====
3 | Version: 0.5 created on 26 / IX / 2007
4 |
5 |-----
6 | Version: 0.7 created on -- / III / 2010
7 |
8 |-----
9 | Version: 1.0 created on 20 / 08 / 2012
10 |
11 |-----
12 | This module makes the geometry of the drop. At the beginning, the drop started as an octahedron, which has
13 | eight curved faces; then it is divided in many elements as you can see in the subroutine Trgl_Octa.
14 |-----
15 | CONTAINS
16 |-----
17 | SUBROUTINE trgl_octa(ndiv, npts, nelm)
18 |
19 |-----
20 | Triangulation of the unit sphere by subdividing a regular octahedron into six-node quadratic triangles.
21 |-----
22 | Variables
23 |-----
24 | nelm ..... number of surface elements
25 | npts ..... number of nodal points
26 | x(i,j), y(i,j), z(i,j) ..... Cartesian coords of point j on element i
27 | p(i,j) ..... (x,y,z) coords. of surface node labeled i (j=1,2,3)
28 |
29 | x = p(i,1)
30 | y = p(i,2)
31 | z = p(i,3)
32 |
33 | n,j(i) ..... node number of point j on element i, where j=1,...,6
34 | ne(i,j) ..... ne(i,1) is the number of elements touching node i,
35 | ne(i,2:ne(i,3)) are the corresponding element labels
36 |
37 | nbe(i,j) ..... label of element sharing side j of element i where j = 1, 2, 3
38 | ndiv .... level of discretization of starting octahedron (0-ndiv)
39 |-----
40 | Modules used in this subroutine.
41 | USE Mod_SharedVars, ONLY: DBL, Ulog, p, ne, n, nbe, nbel
42 |-----
43 | IMPLICIT NONE
44 |-----
45 | Variables IN/OUT
46 |-----
47 | INTEGER, INTENT(IN) :: ndiv
48 | INTEGER, INTENT(OUT) :: npts, nelm
49 |-----
50 | Variables used inside the subroutine
51 |-----
52 | INTEGER :: istat = 0, iflag
53 | INTEGER :: fstatus
54 | INTEGER :: nsa
55 | INTEGER :: i, j, k, l, jcount, jcount, kcount, num
56 | REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x, y, z
57 | REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: xn, yn, zn
58 | REAL (KIND = DBL), PARAMETER :: eps=0.0000001
59 | REAL (KIND = DBL) :: r
60 | REAL (KIND = DBL), DIMENSION(3) :: ptm
61 | TYPE :: Ps
62 | REAL (KIND = DBL), DIMENSION(3) :: px
63 | TYPE (Ps), POINTER :: pp
64 | END TYPE Ps
65 | TYPE (Ps), POINTER :: head
66 | TYPE (Ps), POINTER :: ptr
67 | TYPE (Ps), POINTER :: ptmp
68 |-----
69 | Initial octahedron (0'th level discretization (8 elements)). In this part, nodes are set manually.
70 | nelm = 8
71 | IF (ndiv > 0) THEN
72 |   nsa = nelm*(4**ndiv)
73 | ELSE
```

D:\Darth Vader\Escritorio\prtcl\_mkl\Mod\_trgl\_octa.f90 2

```
73 |   nsa = nelm
74 | END IF
75 |-----
76 | Allocation of all arrays with the information of drop's geometry
77 | ALLOCATE (x(nsa, 6), y(nsa, 6), z(nsa, 6), &
78 | & xn(nsa, 6), yn(nsa, 6), zn(nsa, 6), STAT = istat)
79 |-----
80 | Conditional to have the errors, if there is any problem in allocation statement
81 | IF (istat /= 0) THEN
82 |   x = 0.000
83 |   y = 0.000
84 |   z = 0.000
85 |   xn = 0.000
86 |   yn = 0.000
87 |   zn = 0.000
88 |   p = 0.000
89 |   n = 0
90 |   ne = 0
91 |   nbe = 0
92 |   fstatus = 0
93 | ELSE
94 |   Corner points
95 |-----
96 | Upper half of xz plane
97 | x(1,1)= 0.0
98 | y(1,1)= 0.0
99 | z(1,1)= 1.0
100 |-----
101 | x(1,2)= 1.0
102 | y(1,2)= 0.0
103 | z(1,2)= 0.0
104 |-----
105 | x(1,3)= 0.0
106 | y(1,3)= 1.0
107 | z(1,3)= 0.0
108 |-----
109 | x(5,1)= 1.0
110 | y(5,1)= 0.0
111 | z(5,1)= 0.0
112 |-----
113 | x(5,2)= 0.0
114 | y(5,2)= 0.0
115 | z(5,2)= -1.0
116 |-----
117 | x(5,3)= 0.0
118 | y(5,3)= 1.0
119 | z(5,3)= 0.0
120 |-----
121 | x(6,1)= 0.0
122 | y(6,1)= 0.0
123 | z(6,1)= -1.0
124 |-----
125 | x(6,2)= -1.0
126 | y(6,2)= 0.0
127 | z(6,2)= 0.0
128 |-----
129 | x(6,3)= 0.0
130 | y(6,3)= 1.0
131 | z(6,3)= 0.0
132 |-----
133 | x(2,1)= -1.0
134 | y(2,1)= 0.0
135 | z(2,1)= 0.0
136 |-----
137 | x(2,2)= 0.0
138 | y(2,2)= 0.0
139 | z(2,2)= 1.0
140 |-----
141 | x(2,3)= 0.0
142 | y(2,3)= 1.0
143 | z(2,3)= 0.0
144 |-----
```

```

145 ! Corner points lower half xz plane
146   x(4,1)= 0.0
147   y(4,1)= 0.0
148   z(4,1)= 1.0
149 -----
150   x(4,2)= 0.0
151   y(4,2)=-1.0
152   z(4,2)= 0.0
153 -----
154   x(4,3)= 1.0
155   y(4,3)= 0.0
156   z(4,3)= 0.0
157 -----
158   x(8,1)= 1.0
159   y(8,1)= 0.0
160   z(8,1)= 0.0
161 -----
162   x(8,2)= 0.0
163   y(8,2)=-1.0
164   z(8,2)= 0.0
165 -----
166   x(8,3)= 0.0
167   y(8,3)= 0.0
168   z(8,3)=-1.0
169 -----
170   x(7,1)= 0.0
171   y(7,1)= 0.0
172   z(7,1)=-1.0
173 -----
174   x(7,2)= 0.0
175   y(7,2)=-1.0
176   z(7,2)= 0.0
177 -----
178   x(7,3)=-1.0
179   y(7,3)= 0.0
180   z(7,3)= 0.0
181 -----
182   x(3,1)=-1.0
183   y(3,1)= 0.0
184   z(3,1)= 0.0
185 -----
186   x(3,2)= 0.0
187   y(3,2)=-1.0
188   z(3,2)= 0.0
189 -----
190   x(3,3)= 0.0
191   y(3,3)= 0.0
192   z(3,3)= 1.0
193 -----
194 ! compute the mid-points of the sides numbered 4, 5, 6
195 ! FORALL (i = 1:nelm)
196 -----
197   x(i,4)= 0.5*(x(i,1)+x(i,2))
198   y(i,4)= 0.5*(y(i,1)+y(i,2))
199   z(i,4)= 0.5*(z(i,1)+z(i,2))
200 -----
201   x(i,5)= 0.5*(x(i,2)+x(i,3))
202   y(i,5)= 0.5*(y(i,2)+y(i,3))
203   z(i,5)= 0.5*(z(i,2)+z(i,3))
204 -----
205   x(i,6)= 0.5*(x(i,3)+x(i,1))
206   y(i,6)= 0.5*(y(i,3)+y(i,1))
207   z(i,6)= 0.5*(z(i,3)+z(i,1))
208 -----
209 ! END FORALL
210 -----
211 ! Compute node coordinates on each element for discretization levels 1 through ndiv
212 -----
213 ! Condition to divide the initial octahedron from 8 elements to (8**ndiv) elements
214 ! IF (ndiv > 0) THEN
215 !   DO i = 1, ndiv
216 !     num = 1

```

```

217 ! -----
218 ! Loop over old elements to divide every element in four new elements
219 !   DO j = 1, nelm
220 ! -----
221 ! Assign corner points to sub-elements
222 ! -----
223 ! First sub-element
224   xn(num,1)= x(j,1)
225   yn(num,1)= y(j,1)
226   zn(num,1)= z(j,1)
227 -----
228   xn(num,2)= x(j,4)
229   yn(num,2)= y(j,4)
230   zn(num,2)= z(j,4)
231 -----
232   xn(num,3)= x(j,6)
233   yn(num,3)= y(j,6)
234   zn(num,3)= z(j,6)
235 -----
236   xn(num,4)= 0.5*(xn(num,1)+xn(num,2))
237   yn(num,4)= 0.5*(yn(num,1)+yn(num,2))
238   zn(num,4)= 0.5*(zn(num,1)+zn(num,2))
239 -----
240   xn(num,5)= 0.5*(xn(num,2)+xn(num,3))
241   yn(num,5)= 0.5*(yn(num,2)+yn(num,3))
242   zn(num,5)= 0.5*(zn(num,2)+zn(num,3))
243 -----
244   xn(num,6)= 0.5*(xn(num,3)+xn(num,1))
245   yn(num,6)= 0.5*(yn(num,3)+yn(num,1))
246   zn(num,6)= 0.5*(zn(num,3)+zn(num,1))
247 -----
248 ! Second sub-element
249   xn(num+1,1)= x(j,4)
250   yn(num+1,1)= y(j,4)
251   zn(num+1,1)= z(j,4)
252 -----
253   xn(num+1,2)= x(j,2)
254   yn(num+1,2)= y(j,2)
255   zn(num+1,2)= z(j,2)
256 -----
257   xn(num+1,3)= x(j,5)
258   yn(num+1,3)= y(j,5)
259   zn(num+1,3)= z(j,5)
260 -----
261   xn(num+1,4)= 0.5*(xn(num+1,1)+xn(num+1,2))
262   yn(num+1,4)= 0.5*(yn(num+1,1)+yn(num+1,2))
263   zn(num+1,4)= 0.5*(zn(num+1,1)+zn(num+1,2))
264 -----
265   xn(num+1,5)= 0.5*(xn(num+1,2)+xn(num+1,3))
266   yn(num+1,5)= 0.5*(yn(num+1,2)+yn(num+1,3))
267   zn(num+1,5)= 0.5*(zn(num+1,2)+zn(num+1,3))
268 -----
269   xn(num+1,6)= 0.5*(xn(num+1,3)+xn(num+1,1))
270   yn(num+1,6)= 0.5*(yn(num+1,3)+yn(num+1,1))
271   zn(num+1,6)= 0.5*(zn(num+1,3)+zn(num+1,1))
272 -----
273 ! Third sub-element
274   xn(num+2,1)= x(j,6)
275   yn(num+2,1)= y(j,6)
276   zn(num+2,1)= z(j,6)
277 -----
278   xn(num+2,2)= x(j,5)
279   yn(num+2,2)= y(j,5)
280   zn(num+2,2)= z(j,5)
281 -----
282   xn(num+2,3)= x(j,3)
283   yn(num+2,3)= y(j,3)
284   zn(num+2,3)= z(j,3)
285 -----
286   xn(num+2,4)= 0.5*(xn(num+2,1)+xn(num+2,2))
287   yn(num+2,4)= 0.5*(yn(num+2,1)+yn(num+2,2))
288   zn(num+2,4)= 0.5*(zn(num+2,1)+zn(num+2,2))

```



```

289 -----
290      xn(num+2,5)= 0.5*(xn(num+2,2)+xn(num+2,3))
291      yn(num+2,5)= 0.5*(yn(num+2,2)+yn(num+2,3))
292      zn(num+2,5)= 0.5*(zn(num+2,2)+zn(num+2,3))
293 -----
294      xn(num+2,6)= 0.5*(xn(num+2,3)+xn(num+2,1))
295      yn(num+2,6)= 0.5*(yn(num+2,3)+yn(num+2,1))
296      zn(num+2,6)= 0.5*(zn(num+2,3)+zn(num+2,1))
297 -----
298 ! Fourth sub-element
299      xn(num+3,1)= x(j,4)
300      yn(num+3,1)= y(j,4)
301      zn(num+3,1)= z(j,4)
302 -----
303      xn(num+3,2)= x(j,5)
304      yn(num+3,2)= y(j,5)
305      zn(num+3,2)= z(j,5)
306 -----
307      xn(num+3,3)= x(j,6)
308      yn(num+3,3)= y(j,6)
309      zn(num+3,3)= z(j,6)
310 -----
311      xn(num+3,4)= 0.5*(xn(num+3,1)+xn(num+3,2))
312      yn(num+3,4)= 0.5*(yn(num+3,1)+yn(num+3,2))
313      zn(num+3,4)= 0.5*(zn(num+3,1)+zn(num+3,2))
314 -----
315      xn(num+3,5)= 0.5*(xn(num+3,2)+xn(num+3,3))
316      yn(num+3,5)= 0.5*(yn(num+3,2)+yn(num+3,3))
317      zn(num+3,5)= 0.5*(zn(num+3,2)+zn(num+3,3))
318 -----
319      xn(num+3,6)= 0.5*(xn(num+3,3)+xn(num+3,1))
320      yn(num+3,6)= 0.5*(yn(num+3,3)+yn(num+3,1))
321      zn(num+3,6)= 0.5*(zn(num+3,3)+zn(num+3,1))
322 -----
323 ! Four new elements were generated
324      num = num+4
325 -----
326 ! END of old element loop
327      END DO
328      nelm=nelm*4
329 -----
330 ! Rename the new points and place them in the master list
331      DO k=1,nelm
332         DO l=1,6
333            x(k,l) = xn(k,l)
334            y(k,l) = yn(k,l)
335            z(k,l) = zn(k,l)
336 -----
337 ! Zero just in case
338            xn(k,l) = 0.0
339            yn(k,l) = 0.0
340            zn(k,l) = 0.0
341 -----
342         END DO
343 -----
344 ! END of level loop of 1
345      END DO
346 -----
347      END IF
348 ! Create a list of surface nodes by looping over all elements and adding nodes not already found in the list.
349 ! Fill the connectivity table n(i,j) node numbers of element points 1-6
350      ALLOCATE(n(nelm, 6), nbe(nelm, 3), nbel(nelm, 3), STAT = IStat)
351 -----
352 ! First element is set manually
353 ! Initialization of the linked list
354      ALLOCATE(head, STAT=istatp) ! Allocate new value
355      tail => head ! Tail pts to new value
356      NULLIFY(tail%pp) ! Nullify pp in new value
357      k = 1
358      tail%px = (/ x(1,k), y(1,k), z(1,k) /) ! Store point
359 -----
360 ! Loop over nodes in first element

```

```

361      DO k = 2, 6
362         ALLOCATE(tail%pp, STAT=istatp) ! Allocate new value
363         tail => tail%pp ! Tail pts to new value
364         NULLIFY(tail%pp) ! Nullify pp in new value
365         tail%px = (/ x(1,k), y(1,k), z(1,k) /) ! Store point
366         ! tail%px(1) = x(1,k) ! Store point
367         ! tail%px(2) = y(1,k) ! Store point
368         ! tail%px(3) = z(1,k) ! Store point
369 -----
370      END DO
371 -----
372      n(1,1) = 1
373      n(1,2) = 2
374      n(1,3) = 3
375      n(1,4) = 4
376      n(1,5) = 5
377      n(1,6) = 6
378 -----
379      npts = 6
380 ! Loop over elements, to know if the node had been associated before.
381      DO i = 2, nelm
382 -----
383 ! Loop over element nodes
384         DO j = 1, 6
385            IFlag = 0
386            ptr => head
387 -----
388 ! Cycle DO WHILE
389            DO k = 1, npts
390               IF (.NOT. ASSOCIATED(ptr) ) EXIT ! Pointer valid?
391               ptr = ptr%pp ! Yes: Write value
392               ptr => ptr%pp ! Get next pointer
393 -----
394 ! Condition which checks if a node had been associated before.
395               IF ((ABS(x(i,j)-ptm(1)) <= eps) .AND. &
396                  & (ABS(y(i,j)-ptm(2)) <= eps) .AND. &
397                  & (ABS(z(i,j)-ptm(3)) <= eps)) THEN
398                  ! the node has been previously recorded
399                  IFlag = 1
400                  ! the jth local node of element i
401                  n(i,j) = k
402                  ! is the kth global node
403 -----
404            END IF
405 -----
406 ! Record the node
407            IF (IFlag == 0) THEN
408               npts = npts + 1
409               ALLOCATE(tail%pp, STAT=istatp) ! Allocate new value
410               tail => tail%pp ! Tail pts to new value
411               NULLIFY(tail%pp) ! Nullify pp in new value
412               tail%px = (/ x(i,j), y(i,j), z(i,j) /) ! Store point
413               ! tail%px(1) = x(i,j) ! Store point
414               ! tail%px(2) = y(i,j) ! Store point
415               ! tail%px(3) = z(i,j) ! Store point
416 -----
417            n(i,j) = npts
418            END IF
419 -----
420            END DO
421 -----
422 ! END of loop over elements
423      END DO
424 -----
425      ALLOCATE(p(npts, 3), ne(npts, 7), STAT = IStat)
426      k = 1
427      ptr => head
428 -----
429      output2: DO
430         IF (.NOT. ASSOCIATED(ptr) ) EXIT ! Pointer valid?
431         p(k,:) = ptr%px(:) ! Yes: Write value
432         ptr => ptr%pp ! Get next pointer
433         k = k + 1

```

```

433     END DO output2
434 -----
435     NULLIFY (head, ptr, tail)
436 -----
437 ! Generate connectivity table ne(i,j) for elements touching node i
438     ne = 0
439 ! Loop over global nodes
440     DO i=1,npts
441         ! ne(i,1) = 0
442         ! icount = 1
443 -----
444 ! Loop over local nodes
445         DO j = 1, nelm
446 -----
447 ! Loop over nodes
448             DO k = 1, 6
449 -----
450 ! Condition to see the connectivity
451                 IF((ABS(p(1,1))-x(j,k)) <= eps) .AND. &
452                     & (ABS(p(1,2))-y(j,k)) <= eps) .AND. &
453                     & (ABS(p(1,3))-z(j,k)) <= eps) THEN
454                     icount=icount+1
455                     ne(i,1)=ne(i,1)+1
456                     ne(i,icount)=j
457                 END IF
458             END DO
459 -----
460     END DO
461 -----
462 ! END of loop over surface points
463     END DO
464 -----
465 ! Create connectivity table nbe(i,j) for neighboring elements j of element i. Testing is Done with respect to
466 ! the mid-points (for boundary elements with only 2 neighbors, the array entry will be zero)
467 -----
468     nbel = 0
469 -----
470 ! Loop over elements
471     DO i=1,nelm
472         jcount=1
473 -----
474 ! Loop over mid-points
475         DO j=4,6
476 -----
477 ! Test element
478             DO k=1,nelm
479 -----
480 ! Loop over mid-points
481                 IF(k == i) THEN
482 -----
483                     DO l=4,6
484                         IF((ABS(x(1,j))-x(k,l)) <= eps) .AND. &
485                             & (ABS(y(1,j))-y(k,l)) <= eps) .AND. &
486                             & (ABS(z(1,j))-z(k,l)) <= eps) &
487                             & nbel(i, jcount) = k
488                     END DO
489 -----
490                 END IF
491                 ! END of test element
492             END DO
493 -----
494         IF(nbel(i,jcount) /= 0) jcount=jcount+1
495 -----
496     END DO
497 -----
498 ! END of loop over elements
499     END DO
500 -----
501 -----
502 ! Create connectivity table nbe(i,j) for neighboring elements j of element i. Testing is Done with respect to
503 ! the mid-points (for boundary elements with only 2 neighbors, the array entry will be zero)
504 -----

```

```

505     nbe = 0
506     DO k=1,nelm
507         kcount = 1
508         DO j=1,nelm
509             IF (k/=j) THEN
510                 DO l=4,6
511                     IF((ABS(n(k,4))-n(j,l)) < 1) THEN
512                         nbe(k, kcount) = j
513                         kcount = kcount+1
514                     ELSE IF((ABS(n(k,5))-n(j,l)) < 1) THEN
515                         nbe(k, kcount) = j
516                         kcount = kcount+1
517                     ELSE
518                         IF((ABS(n(k,6))-n(j,l)) < 1) THEN
519                             nbe(k, kcount) = j
520                             kcount = kcount+1
521                         END IF
522                     END IF
523                 END DO
524             END IF
525         END DO
526     END DO
527 -----
528 ! Project points p(i,j) onto the unit sphere
529 -----
530     DO i=1,npts
531         r=DSQRT(p(i,1)**2+p(i,2)**2+p(i,3)**2)
532         p(1,1) = p(1,1)/r
533         p(1,2) = p(1,2)/r
534         p(1,3) = p(1,3)/r
535     END DO
536 -----
537 !
538     OPEN(UNIT = 500, FILE = 'n.dat', STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
539     IF (fstatus == 0) THEN
540         WRITE(500,*) npts
541         WRITE(500,*) nelm
542 -----
543         DO i = 1, nelm
544             WRITE (500,107) n(i,1),n(i,2),n(i,3),n(i,4),n(i,5),n(i,6)
545         END DO
546     END IF
547 -----
548     CLOSE(UNIT = 500)
549 -----
550     OPEN(UNIT = 500, FILE = 'nbe.dat', STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
551     IF (fstatus == 0) THEN
552 -----
553         DO i = 1, nelm
554             WRITE (500,108) nbe(i,1),nbe(i,2),nbe(i,3)
555         END DO
556     END IF
557 -----
558     CLOSE(UNIT = 500)
559 -----
560     OPEN(UNIT = 500, FILE = 'ne.dat', STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
561     IF (fstatus == 0) THEN
562 -----
563         DO i = 1, npts
564             WRITE (500,107) ne(1,1),ne(1,2),ne(1,3),ne(1,4),ne(1,5),ne(1,6),ne(1,7)
565         END DO
566     END IF
567 -----
568     CLOSE(UNIT = 500)
569 -----
570     107 FORMAT(8(1x,110))
571     108 FORMAT(4(1x,110))
572 -----
573 ! Done.
574 ! La comedia e finita :)
575 -----
576 ELSE

```

```

577 WRITE(Ulog,*) ' Error in allocation'
578 WRITE(Ulog,*) 'Little Padawan, you must review the code again'
579 WRITE(Ulog,*) ' Subroutine trgl_octa'
580 END IF
581 !-----
582 END SUBROUTINE trgl_octa
583 !-----
584 SUBROUTINE drop_scale_adjust(p, npts, nelm,&
585 &boa, coa, req, &
586 &eps, &
587 &cxp, cyp, czp, &
588 &ph11, ph12, ph13 )
589 !-----
590 ! This is a new subroutine to adjust the right size of the drop.
591 !-----
592 ! Variables
593 !-----
594 ! nelm ..... number of surface elements
595 ! scale ..... scale of the drop
596 ! x_axis, y_axis, z_axis ..... rate of drop'size in every coordinate axis
597 ! cs, sn .....
598 ! tmpx, tmpy, tmpz .....
599 ! p(i,j) ..... (x,y,z) coords. of surface node labeled i (j=1,2,3)
600 ! x = p(i,1)
601 ! y = p(i,2)
602 ! z = p(i,3)
603 ! req ..... equivalent radius
604 ! ph11, ph12, ph13 ..... angles of rotation around the axes
605 !-----
606 ! Modules used in this subroutine.
607 USE Mod_ShareVars, ONLY: DBL, Ulog, p, Pi, Udat
608 !-----
609 IMPLICIT NONE
610 !-----
611 ! Variables IN/OUT
612 !-----
613 INTEGER, INTENT(IN) :: npts, nelm
614 REAL (KIND = DBL), INTENT(IN) :: boa, coa, req !These variables are used in subroutine
615 REAL (KIND = DBL), INTENT(IN) :: eps !scale_drop_adjust.
616 REAL (KIND = DBL), INTENT(IN) :: cxp, cyp, czp !
617 REAL (KIND = DBL), INTENT(INOUT) :: ph11, ph12, ph13 !
618 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:, :), INTENT(INOUT) :: p
619 !-----
620 !-----
621 INTEGER :: i, j
622 REAL (KIND = DBL) :: oot
623 REAL (KIND = DBL) :: scale
624 REAL (KIND = DBL) :: x_axis, y_axis, z_axis
625 REAL (KIND = DBL) :: cs, sn
626 REAL (KIND = DBL) :: tmpx, tmpy, tmpz
627 !-----
628 ! The next part computes the expansion to specified shape and equivalent radius. The idea is introduce this
629 ! statements in a new subroutine.
630 oot = 1.000/3.000
631 scale = req/(boa*coa)**oot
632 x_axis = scale
633 y_axis = scale*boa
634 z_axis = scale*coa
635 !-----
636 ! scale is made her
637 FORALL (i=1:npts)
638 p(i,1) = x_axis*p(i,1)
639 p(i,2) = y_axis*p(i,2)
640 p(i,3) = z_axis*p(i,3)
641 END FORALL
642 !-----
643 WRITE (Udat,*) " ellipsoid x semi-axis = ",x_axis
644 WRITE (Udat,*) " ellipsoid y semi-axis = ",y_axis
645 WRITE (Udat,*) " ellipsoid z semi-axis = ",z_axis
646 !-----
647 ! This section is made the rotation by angles ph11,ph12,ph13 around the x,y,z axes
648 IF (ABS(ph11) > eps .OR. &

```

```

649 & ABS(ph12) > eps .OR. &
650 & ABS(ph13) > eps ) THEN
651 ph11 = ph11*Pi ! scale in x axis
652 ph12 = ph12*Pi ! scale in y axis
653 ph13 = ph13*Pi ! scale in z axis
654 !-----
655 cs = DCOS(ph13)
656 sn = DSIN(ph13)
657 !-----
658 ! Rotate about the z axis
659 DO i=1,npts
660 tmpx = cs*p(i,1)+sn*p(i,2)
661 tmpy =-sn*p(i,1)+cs*p(i,2)
662 tmpz = p(i,3)
663 p(i,1) = tmpx
664 p(i,2) = tmpy
665 p(i,3) = tmpz
666 tmpx = 0.000
667 tmpy = 0.000
668 tmpz = 0.000
669 END DO
670 cs = DCOS(ph11)
671 sn = DSIN(ph11)
672 !-----
673 ! Rotate about the x axis
674 DO i = 1, npts
675 tmpx = p(i,1)
676 tmpy = cs*p(i,2)+sn*p(i,3)
677 tmpz =-sn*p(i,2)+cs*p(i,3)
678 p(i,1) = tmpx
679 p(i,2) = tmpy
680 p(i,3) = tmpz
681 tmpx = 0.000
682 tmpy = 0.000
683 tmpz = 0.000
684 END DO
685 cs = DCOS(ph12)
686 sn = DSIN(ph12)
687 !-----
688 ! Rotate about the y axis
689 DO i = 1, npts
690 tmpx = cs*p(i,1)-sn*p(i,3)
691 tmpy = p(i,2)
692 tmpz = sn*p(i,1)+cs*p(i,3)
693 p(i,1) = tmpx
694 p(i,2) = tmpy
695 p(i,3) = tmpz
696 tmpx = 0.000
697 tmpy = 0.000
698 tmpz = 0.000
699 END DO
700 !-----
701 ! Unscale of drop and final statement
702 ph11 = ph11/Pi
703 ph12 = ph12/Pi
704 ph13 = ph13/Pi
705 END IF
706 !-----
707 ! Translate center to specified position
708 p(:,1) = p(:,1) + cxp
709 p(:,2) = p(:,2) + cyp
710 p(:,3) = p(:,3) + czp
711 WRITE (Udat,*) 'Translation was successful'
712 WRITE (Udat,*) " prtc1_3d: number of nodes : ",npts
713 WRITE (Udat,*) " prtc1_3d: number of elements: ",nelm
714 END SUBROUTINE drop_scale_adjust
715 !-----
716 SUBROUTINE trgl_octa_read(ndiv, npts, nelm)
717 !-----
718 ! Triangulation of the unit sphere by subdividing a regular octahedron into six-node quadratic triangles.
719 !-----
720 ! Variables

```

```

721 !-----
722 ! nelm ..... number of surface elements
723 ! npts ..... number of nodal points
724 ! x(i,j), y(i,j), z(i,j) ..... Cartesian coords of point j on element i
725 ! p(i,j) ..... (x,y,z) coords. of surface node labeled i (j=1,2,3)
726 !     x = p(i,1)
727 !     y = p(i,2)
728 !     z = p(i,3)
729 !
730 ! n(i,j) ..... node number of point j on element i, where j=1,...,6
731 ! ne(i,j) ..... ne(i,1) is the number of elements touching node i.
732 !     ne(i,2:ne(i,1)) are the corresponding element labels
733 ! nbe(i,j) ..... label of element sharing side j of element i where j = 1, 2, 3
734 ! ndiv ..... level of discretization of starting octahedron (0=ndiv)
735 !-----
736 ! Modules used in this subroutine.
737 ! USE Mod_SharedVars, ONLY: DBL, ULog1, ULog, p, ne, n, nbe
738 !-----
739 ! IMPLICIT NONE
740 !-----
741 ! Variables IN/OUT
742 !-----
743 ! INTEGER, INTENT(IN) :: ndiv
744 ! INTEGER, INTENT(OUT) :: npts, nelm
745 !-----
746 ! Variables used inside the subroutine
747 ! INTEGER :: fstatus
748 ! INTEGER :: i, j, k, i1, i2, i3, i4, i5, i6
749 ! REAL (KIND = DBL), PARAMETER :: eps=0.0000001
750 !-----
751 ! Create a list of surface nodes by looping over all elements and adding nodes not already found in the list.
752 ! Fill the connectivity table n(i,j) node numbers of element points 1-6
753 nelm = 8*(4**(ndiv))
754 npts = nelm + nelm + 2
755 ALLOCATE(p(npts, 3), ne(npts, 7))
756 ALLOCATE(n(nelm, 6), nbe(nelm, 3))
757 n = 0
758 ne = 0
759 nbe = 0
760 p = 0.000
761 fstatus = 0
762 !-----
763 !-----
764 OPEN(UNIT = 500, FILE = 'n.dat', STATUS = 'OLD', ACTION = 'READ', IOSTAT = fstatus)
765 IF (fstatus == 0) THEN
766 READ(500,*) npts
767 READ(500,*) nelm
768 DO i = 1, nelm
769 READ(500,107) n(i,1),n(i,2),n(i,3),n(i,4),n(i,5),n(i,6)
770 END DO
771 CLOSE(UNIT = 500)
772 !-----
773 ELSE
774 OPEN(UNIT = ULog1, FILE = 'Prtcl3D_Log1.log', STATUS = 'REPLACE', ACTION = 'WRITE')
775 WRITE (ULog1,*) 'Error, little Padawan, InputData has a mistake!'
776 WRITE (ULog1,*) 'First Error index = ', fstatus
777 WRITE (ULog1,*) 'There was error in the n file'
778 WRITE (ULog1,*) 'The Force will be with you'
779 CLOSE(UNIT = 10)
780 END IF
781 !-----
782 !-----
783 OPEN(UNIT = 500, FILE = 'nbe.dat', STATUS = 'OLD', ACTION = 'READ', IOSTAT = fstatus)
784 IF (fstatus == 0) THEN
785 DO i = 1, nelm
786 READ(500,108) nbe(i,1),nbe(i,2),nbe(i,3)
787 END DO
788 CLOSE(UNIT = 500)
789 !-----
790 !-----
791 !-----
792 !-----

```

```

793 ELSE
794 OPEN(UNIT = ULog1, FILE = 'Prtcl3D_Log1.log', STATUS = 'REPLACE', ACTION = 'WRITE')
795 WRITE (ULog1,*) 'Error, little Padawan, InputData has a mistake!'
796 WRITE (ULog1,*) 'First Error index = ', fstatus
797 WRITE (ULog1,*) 'There was error in the nbe file'
798 WRITE(ULog1,*) 'The Force will be with you'
799 CLOSE(UNIT = 10)
800 END IF
801 !-----
802 !-----
803 OPEN(UNIT = 500, FILE = 'ne.dat', STATUS = 'OLD', ACTION = 'READ', IOSTAT = fstatus)
804 IF (fstatus == 0) THEN
805 DO i = 1, npts
806 READ(500,107) ne(i,1),ne(i,2),ne(i,3),ne(i,4),ne(i,5),ne(i,6),ne(i,7)
807 END DO
808 CLOSE(UNIT = 500)
809 !-----
810 !-----
811 ELSE
812 OPEN(UNIT = ULog1, FILE = 'Prtcl3D_Log1.log', STATUS = 'REPLACE', ACTION = 'WRITE')
813 WRITE (ULog1,*) 'Error, little Padawan, InputData has a mistake!'
814 WRITE (ULog1,*) 'First Error index = ', fstatus
815 WRITE (ULog1,*) 'There was error in the nbe file'
816 WRITE(ULog1,*) 'The Force will be with you'
817 CLOSE(UNIT = 10)
818 END IF
819 !-----
820 !-----
821 OPEN(UNIT = 500, FILE = 'Prtcl3D_geometry.dat', STATUS = 'OLD', ACTION = 'READ', IOSTAT = fstatus)
822 IF (fstatus == 0) THEN
823 DO i = 1, npts
824 READ(500,103) p(i,1),p(i,2),p(i,3)
825 !END DO
826 !CLOSE(UNIT = 500)
827 !-----
828 !-----
829 DO i = 1, nelm
830 ! will be interpolate the normal vector and features of collocation points.
831 i1 = n(i,1)
832 i2 = n(i,2)
833 i3 = n(i,3)
834 i4 = n(i,4)
835 i5 = n(i,5)
836 i6 = n(i,6)
837 !-----
838 READ(500,103) p(i1,1), p(i1,2), p(i1,3)
839 READ(500,103) p(i2,1), p(i2,2), p(i2,3)
840 READ(500,103) p(i3,1), p(i3,2), p(i3,3)
841 READ(500,103) p(i4,1), p(i4,2), p(i4,3)
842 READ(500,103) p(i5,1), p(i5,2), p(i5,3)
843 READ(500,103) p(i6,1), p(i6,2), p(i6,3)
844 !-----
845 END DO
846 !-----
847 !-----
848 ELSE
849 OPEN(UNIT = ULog1, FILE = 'Prtcl3D_Log1.log', STATUS = 'REPLACE', ACTION = 'WRITE')
850 WRITE (ULog1,*) 'Error, little Padawan, InputData has a mistake!'
851 WRITE (ULog1,*) 'First Error index = ', fstatus
852 WRITE (ULog1,*) 'There was error in the geometry file'
853 WRITE(ULog1,*) 'The Force will be with you'
854 CLOSE(UNIT = 10)
855 END IF
856 !-----
857 ! Done.
858 !-----
859 !-----
860 103 FORMAT(10(1x,E24.16))
861 107 FORMAT(8(1x,I10))
862 108 FORMAT(4(1x,I10))
863 !-----
864 END SUBROUTINE trgl_octa_read

```

```
865 !=====
866 END MODULE Mod_Trgl_Octa
867
```

```

1 MODULE Mod_Prtcl_3D_Geo
2 =====
3 ! Version: 0.5 created on 26 / IX / 2007
4 !
5 !-----
6 ! Version: 0.7 created on -- / III / 2010
7 !
8 !-----
9 ! Version: 0.9 created on 23 / 08 / 2012
10 ! Version: 1.0 created on 14 / 11 / 2012
11 !
12 !-----
13 CONTAINS
14 =====
15 SUBROUTINE pritel(k, Index, c)
16 !-----
17 ! This subroutine prints drop's geometry. It has two options.
18 ! Print successive nodes of element k in file unit 1
19 ! Index = 1: print the whole element
20 ! Index = 2: print the 4 subelements
21 !-----
22 ! Variables
23 !-----
24 ! k ..... number of element
25 ! index ..... type of print
26 ! nfour ..... index to print only the element
27 ! nseven ..... index to print the subelements
28 ! c ..... firstly, it was the value of shear strain, now, it will change. ;)
29 !-----
30 USE Mod_SharedVars, ONLY: DBL, ULog, UGeo, p, ne, n, nbe
31 !-----
32 IMPLICIT NONE
33 !-----
34 INTEGER, INTENT(IN) :: k, index
35 !-----
36 INTEGER :: nfour, nseven, i
37 REAL (KIND = DBL), DIMENSION(:), INTENT(IN) :: c
38 !-----
39 ! constants
40 nfour = 4
41 nseven = 1
42 !-----
43 ! There is a CASE instruction to print the element.
44 !-----
45 SELECT CASE (index)
46 CASE (1)
47 WRITE (UGeo,100) nseven
48 i = n(k,1)
49 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
50 i = n(k,4)
51 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
52 i = n(k,2)
53 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
54 i = n(k,5)
55 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
56 i = n(k,3)
57 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
58 i = n(k,6)
59 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
60 i = n(k,1)
61 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
62 CASE (2)
63 !--- first
64 WRITE (UGeo,100) nfour
65 i = n(k,1)
66 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
67 i = n(k,4)
68 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
69 i = n(k,6)
70 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
71 i = n(k,1)
72 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)

```

```

73 !--- second
74 i = n(k,4)
75 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
76 i = n(k,2)
77 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
78 i = n(k,5)
79 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
80 i = n(k,4)
81 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
82 !--- third
83 i = n(k,4)
84 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
85 i = n(k,5)
86 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
87 i = n(k,6)
88 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
89 i = n(k,4)
90 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
91 !--- fourth
92 i = n(k,6)
93 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
94 i = n(k,5)
95 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
96 i = n(k,3)
97 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
98 i = n(k,6)
99 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
100 CASE DEFAULT
101 WRITE (UGeo,100) nfour
102 i = n(k,1)
103 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
104 i = n(k,4)
105 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
106 i = n(k,6)
107 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
108 i = n(k,1)
109 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
110 i = n(k,4)
111 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
112 i = n(k,2)
113 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
114 i = n(k,5)
115 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
116 i = n(k,4)
117 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
118 i = n(k,4)
119 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
120 i = n(k,5)
121 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
122 i = n(k,6)
123 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
124 i = n(k,4)
125 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
126 i = n(k,6)
127 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
128 i = n(k,5)
129 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
130 i = n(k,3)
131 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
132 i = n(k,6)
133 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
134 WRITE (ULog,*)
135 WRITE (ULog,*) ' Geo_Pritel'
136 WRITE (ULog,*)
137 WRITE (ULog,*) ' Chosen index is not available'
138 WRITE (ULog,*) ' It was taken index= 2'
139 END SELECT
140 100 FORMAT(1x,14,10(1x,ES24.16))
141 101 FORMAT(10(1x,ES24.16))
142 END SUBROUTINE pritel
143 !-----
144 !

```



D:\Darth Vader\Escritorio\prtcl mkl\Mod Prtcl 3D\_Geo.f90 3

```
145 SUBROUTINE abc(x1, y1, z1, &
146 & x2, y2, z2, &
147 & x3, y3, z3, &
148 & x4, y4, z4, &
149 & x5, y5, z5, &
150 & x6, y6, z6, &
151 & al, be, ga)
152 =====
153 ! This subroutine compute the parametric representation constants alpha, beta, gamma
154 ! =====
155 USE Mod_SharedVars, ONLY: DBL
156 ! =====
157 IMPLICIT NONE
158 ! =====
159 ! Variables
160 ! =====
161 REAL (KIND = DBL), INTENT(IN) :: x1, y1, z1, & !coordinates of each point in the element
162 & x2, y2, z2, &
163 & x3, y3, z3, &
164 & x4, y4, z4, &
165 & x5, y5, z5, &
166 & x6, y6, z6
167 REAL (KIND = DBL), INTENT(OUT) :: al, be, ga !constants alpha, beta and gamma (weights)
168 ! =====
169 ! Variables inside the subroutine
170 ! =====
171 REAL (KIND = DBL) :: d42, d41 !distances of the element on the segment 1 4 2
172 REAL (KIND = DBL) :: d63, d61 !distances of the element on the segment 3 6 1
173 REAL (KIND = DBL) :: d52, d53 !distances of the element on the segment 2 5 3
174 ! =====
175 ! Initialize
176 d42 = (x4 - x2)**2 + (y4 - y2)**2 + (z4 - z2)**2
177 d41 = (x4 - x1)**2 + (y4 - y1)**2 + (z4 - z1)**2
178 d63 = (x6 - x3)**2 + (y6 - y3)**2 + (z6 - z3)**2
179 d61 = (x6 - x1)**2 + (y6 - y1)**2 + (z6 - z1)**2
180 d52 = (x5 - x2)**2 + (y5 - y2)**2 + (z5 - z2)**2
181 d53 = (x5 - x3)**2 + (y5 - y3)**2 + (z5 - z3)**2
182 ! =====
183 d42 = DSQRT(d42)
184 d41 = DSQRT(d41)
185 d63 = DSQRT(d63)
186 d61 = DSQRT(d61)
187 d52 = DSQRT(d52)
188 d53 = DSQRT(d53)
189 ! =====
190 al = 1.000/(1.000 + d42/d41)
191 be = 1.000/(1.000 + d63/d61)
192 ga = 1.000/(1.000 + d52/d53)
193 ! =====
194 END SUBROUTINE abc
195 ! =====
196 SUBROUTINE elm_geom(nelm, npts, mint, &
197 & xmom, ymom, zmom, &
198 & area, vlm, &
199 & cx, cy, cz, stride)
200 ! =====
201 ! This subroutine is a new version of Elm_Geo Subroutine.
202 ! Compute:
203 ! *The surface area of the individual elements x, y, and z moments over each element
204 ! *the total particle surface area and volume
205 ! Now, (25 / August / 2012) this subroutine was cut.
206 ! =====
207 USE Mod_Nodal_Interp
208 USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, Ulog,&
209 & alpha0, beta0, gamma0, &
210 & arel, crvmel, &
211 & vnx0, vny0, vnz0, &
212 & zz, ww, &
213 & xiq, etq, wq
214 ! =====
215 IMPLICIT NONE
```

D:\Darth Vader\Escritorio\prtcl mkl\Mod Prtcl 3D\_Geo.f90 4

```
217 ! =====
218 ! Variables
219 ! =====
220 INTEGER, INTENT(IN) :: nelm !number of elements
221 INTEGER, INTENT(IN) :: npts !number of points on the surface
222 INTEGER, INTENT(IN) :: mint !order of triangle quadrature
223 INTEGER, INTENT(IN) :: stride !order of triangle and Gauss-Legendre quadratures
224 REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom !coordinates of the moments of the drop
225 REAL (KIND = DBL), INTENT(OUT) :: area, vlm !area and volume of each element
226 REAL (KIND = DBL), INTENT(OUT) :: cx, cy, cz !drop's centroid coordinates
227 ! =====
228 ! Variables inside the subroutine
229 ! =====
230 INTEGER :: i, k !counters
231 INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
232 ! =====
233 REAL (KIND = DBL) :: xi, eta !variables of weigh to integrate over a triangle
234 REAL (KIND = DBL) :: x, y, z !coordinates of the f(x,y,z)=f(xi,eta)
235 REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi !coordinates of the tangential vector over the xi axis
236 REAL (KIND = DBL) :: DxDeta, DyDeta, DzDeta !coordinates of the tangential vector over the eta axis
237 REAL (KIND = DBL) :: vnx, vny, vnz !normal vector coordinates of the element
238 REAL (KIND = DBL) :: hs !surface metric on a triangle
239 REAL (KIND = DBL) :: al, be, ga, alc, bec, gac !integration weigh coefficients
240 REAL (KIND = DBL) :: cf, fil !integration weigh coefficients
241 REAL (KIND = DBL), DIMENSION(6) :: xxi, eet !variables of weigh over in triangle (xi,eta)
242 REAL (KIND = DBL), DIMENSION(6) :: DxDx, DyDx, DzDx !tangential vector over xi axis in triangle (xi,eta)
243 REAL (KIND = DBL), DIMENSION(6) :: DxDeta, DyDeta, DzDeta !tangential vector over eta axis in triangle (xi,eta)
244 REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz !normal vector in triangle (xi,eta)
245 REAL (KIND = DBL) :: bx1, by1, bv21, & !binormal vectors around in triangle (xi,eta)
246 & bx2, by2, bv22, &
247 & bx3, by3, bv23
248 REAL (KIND = DBL) :: crvx, crvy, crvz, curv !curvature
249 ! =====
250 ! initialize
251 ! =====
252 area = 0.000
253 vlm = 0.000
254 cx = 0.000
255 cy = 0.000
256 cz = 0.000
257 arel = 0.000
258 xmom = 0.000
259 ymom = 0.000
260 zmom = 0.000
261 crvmel = 0.000
262 fil = 0.000
263 ! =====
264 Outer: DO k = 1, nelm
265 ! =====
266 i1 = n(k,1)
267 i2 = n(k,2)
268 i3 = n(k,3)
269 i4 = n(k,4)
270 i5 = n(k,5)
271 i6 = n(k,6)
272 al = alpha0(k)
273 be = beta0(k)
274 ga = gamma0(k)
275 alc = 1.000-al
276 bec = 1.000-be
277 gac = 1.000-ga
278 ! =====
279 ! Compute surface area and volume of the individual elements x, y, and z moments over each element total
280 !particle surface area and volume
281 ! =====
282 DO i = 1, mint
283 xi = xiq(i)
284 eta = etq(i)
285 CALL interp_p(p(i1,1), p(i1,2), p(i1,3), &
286 & p(i2,1), p(i2,2), p(i2,3), &
287 & p(i3,1), p(i3,2), p(i3,3), &
288 & p(i4,1), p(i4,2), p(i4,3), &
```

D:\Darth Vader\Escritorio\prtcl mk1\Mod Prtcl 3D Geo.f90 5

```
289      &      p(15,1), p(15,2), p(15,3), &
290      &      p(16,1), p(16,2), p(16,3), &
291      &      al, be, ga, &
292      &      xi, eta, &
293      &      x, y, z, &
294      &      Dx0x1, Dy0x1, Dz0x1, &
295      &      Dx0et, Dy0et, Dz0et, &
296      &      vnx, vny, vnz, &
297      &      hs)
298      cf = hs*wq(1)
299      arel(k) = arel(k) + cf
300      xmom(k) = xmom(k) + cf*x
301      ymom(k) = ymom(k) + cf*y
302      zmom(k) = zmom(k) + cf*z
303      vlm = vlm + (x*vnx+y*vny+z*vnz)*cf
304      END DO
305      arel(k) = 0.500*arel(k)
306      xmom(k) = 0.500*xmom(k)
307      ymom(k) = 0.500*ymom(k)
308      zmom(k) = 0.500*zmom(k)
309      area = area + arel(k)
310      cx = cx + xmom(k)
311      cy = cy + ymom(k)
312      cz = cz + zmom(k)
313      !-----
314      ! compute the average value of the normal vector the mean curvature as a contour integral using the nifty
315      ! formula (4.2.10) of Pozrikidis (1997)
316      !-----
317      xxi(1) = 0.000
318      eet(1) = 0.000
319      xxi(2) = 1.000
320      eet(2) = 0.000
321      xxi(3) = 0.000
322      eet(3) = 1.000
323      xxi(4) = al
324      eet(4) = 0.000
325      xxi(5) = ga
326      eet(5) = gac
327      xxi(6) = 0.000
328      eet(6) = be
329      xi = 1.000/3.000
330      eta = 1.000/3.000
331      CALL interp_p4(p(11,1), p(11,2), p(11,3), &
332      &      p(12,1), p(12,2), p(12,3), &
333      &      p(13,1), p(13,2), p(13,3), &
334      &      p(14,1), p(14,2), p(14,3), &
335      &      p(15,1), p(15,2), p(15,3), &
336      &      p(16,1), p(16,2), p(16,3), &
337      &      al, be, ga, &
338      &      xi, eta, &
339      &      curv, stride )
340      !-----
341      ! will project curvature vector onto the normal vector at the centroid only needs the colocation vectors
342      ! in this value of curvature I made a division per 8 (1/8.0) when I expected do for 4 (1/4.0)
343      !-----
344      crvmel(k) = curv [(0.2500*(crvx*vnx0(k)+crvy*vny0(k)+crvz*vnz0(k))*fil)/arel(k)]
345      END DO Outer
346      !-----
347      ! final computation of the surface-centroid and volume
348      !-----
349      !-----
350      cx = cx/area
351      cy = cy/area
352      cz = cz/area
353      vlm = vlm/6.000
354      !03 FORMAT(10(1x,ES24.16))
355      !-----
356      END SUBROUTINE elm_geom
357      !-----
358      !-----
359      SUBROUTINE elm_geom4(nelm, npts, mint, &
```

D:\Darth Vader\Escritorio\prtcl mk1\Mod Prtcl 3D Geo.f90 6

```
360      &      xmom, ymom, zmom, &
361      &      area, vlm, &
362      &      cx, cy, cz) &
363      !-----
364      ! This subroutine is a new version of Elm_Geo Subroutine.
365      ! Compute:
366      ! *The surface area of the individual elements x, y, and z moments over each element
367      ! *the total particle surface area and volume
368      ! Now, (25 / August / 2012) this subroutine was cut.
369      !-----
370      USE Mod_Nodal_Interp
371      USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog, &
372      &      alphaQ, betaQ, gammaQ, &
373      &      arel, crvmel, &
374      &      vnx0, vny0, vnz0, &
375      &      ZZ, vny, vnz, &
376      &      xiq, etq, wq
377      !-----
378      IMPLICIT NONE
379      !-----
380      ! Variables
381      !-----
382      INTEGER, INTENT(IN) :: nelm !number of elements
383      INTEGER, INTENT(IN) :: npts !number of points on the surface
384      INTEGER, INTENT(IN) :: mint !order of triangle quadrature
385      REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom !coordinates of the moments of the drop
386      REAL (KIND = DBL), INTENT(OUT) :: area, vlm !area and volume of each element
387      REAL (KIND = DBL), INTENT(OUT) :: cx, cy, cz !drop's centroid coordinates
388      !-----
389      ! Variables inside the subroutine
390      !-----
391      INTEGER :: i, k !counters
392      INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
393      !-----
394      REAL (KIND = DBL) :: xi, eta !variables of weigh to integrate over a triangle
395      REAL (KIND = DBL) :: x, y, z !coordinates of the f(x,y,z)=F(xi,eta)
396      REAL (KIND = DBL) :: Dx0xi, Dy0xi, Dz0xi !coordinates of the tangential vector over the xi axis
397      REAL (KIND = DBL) :: Dx0et, Dy0et, Dz0et !coordinates of the tangential vector over the eta axis
398      REAL (KIND = DBL) :: vnx, vny, vnz !normal vector coordinates of the element
399      REAL (KIND = DBL) :: hs !surface metric on a triangle
400      REAL (KIND = DBL) :: al, be, ga, alc, bec, gac !integration weigh coefficients
401      REAL (KIND = DBL) :: cf, fil !integration weigh coefficients
402      REAL (KIND = DBL), DIMENSION(6) :: xxi, eet !variables of weigh over in triangle (xi,eta)
403      REAL (KIND = DBL), DIMENSION(6) :: Dx0x, Dy0x, Dz0x !tangential vector over xi axis in triangle (xi,eta)
404      REAL (KIND = DBL), DIMENSION(6) :: Dx0e, Dy0e, Dz0e !tangential vector over eta axis in triangle (xi,eta)
405      REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz !normal vector in triangle (xi,eta)
406      REAL (KIND = DBL) :: bvxi, bvyi, bvzi, & !binormal vectors around in triangle (xi,eta)
407      &      bvx2, bvy2, bvz2, &
408      &      bvxi3, bvy3, bvz3
409      REAL (KIND = DBL) :: crvx, crvy, crvz !curvature
410      !-----
411      ! initialize
412      !-----
413      area = 0.000
414      vlm = 0.000
415      cx = 0.000
416      cy = 0.000
417      cz = 0.000
418      arel = 0.000
419      xmom = 0.000
420      ymom = 0.000
421      zmom = 0.000
422      crvmel = 0.000
423      fil = 0.000
424      !-----
425      ! OPEN (9,file="curvmel.out")
426      !-----
427      Outer: DO k = 1, nelm
428      !-----
429      i1 = n(k,1)
430      i2 = n(k,2)
431      i3 = n(k,3)
```

```

432 14 = n(k,4)
433 15 = n(k,5)
434 16 = n(k,6)
435 al = alphaQ(k)
436 be = betaQ(k)
437 ga = gammaQ(k)
438 alc = 1.000-al
439 bec = 1.000-be
440 gac = 1.000-ga
441 !-----
442 ! Compute surface area and volume of the individual elements x, y, and z moments over each element total
443 !particle surface area and volume
444 !-----
445
446 DO i = 1, mint
447   xl = xiq(i)
448   eta = etq(i)
449   CALL interp_p(p(11,1), p(11,2), p(11,3), &
450     & p(12,1), p(12,2), p(12,3), &
451     & p(13,1), p(13,2), p(13,3), &
452     & p(14,1), p(14,2), p(14,3), &
453     & p(15,1), p(15,2), p(15,3), &
454     & p(16,1), p(16,2), p(16,3), &
455     & al, be, ga, &
456     & xl, eta, &
457     & x, y, z, &
458     & DxDx1, DyDx1, DzDx1, &
459     & DxDet, DyDet, DzDet, &
460     & vnx, vny, vnz, &
461     & hs)
462   cf = hs*sq(i)
463   arel(k) = arel(k) + cf
464   xmom(k) = xmom(k) + cf*x
465   ymom(k) = ymom(k) + cf*y
466   zmom(k) = zmom(k) + cf*z
467   vlm = vlm + (x*vnx+y*vny+z*vnz)*cf
468 END DO
469
470
471 arel(k) = 0.500*arel(k)
472 xmom(k) = 0.500*xmom(k)
473 ymom(k) = 0.500*ymom(k)
474 zmom(k) = 0.500*zmom(k)
475 area = area + arel(k)
476 cx = cx + xmom(k)
477 cy = cy + ymom(k)
478 cz = cz + zmom(k)
479
480 !-----
481 ! compute the average value of the normal vector the mean curvature as a contour integral using the nifty
482 ! formula (4.2.10) of Pozrikidis (1997)
483 !-----
484 xxi(1) = 0.000
485 eet(1) = 0.000
486 xxi(2) = 1.000
487 eet(2) = 0.000
488 xxi(3) = 0.000
489 eet(3) = 1.000
490 xxi(4) = al
491 eet(4) = 0.000
492 xxi(5) = ga
493 eet(5) = gac
494 xxi(6) = 0.000
495 eet(6) = be
496 DO i = 1, 6
497   xl = xxi(i)
498   eta = eet(i)
499   CALL interp_p(p(11,1), p(11,2), p(11,3), &
500     & p(12,1), p(12,2), p(12,3), &
501     & p(13,1), p(13,2), p(13,3), &
502     & p(14,1), p(14,2), p(14,3), &
503     & p(15,1), p(15,2), p(15,3), &

```

```

504     & p(16,1), p(16,2), p(16,3), &
505     & al, be, ga, &
506     & xl, eta, z, &
507     & x, y, z, &
508     & DxDx(i), DyDx(i), DzDx(i), &
509     & DxDe(i), DyDe(i), DzDe(i), &
510     & vx(i), vy(i), vz(i), &
511     & hs)
512 END DO
513
514 bvx1 = 0.000
515 bvy1 = 0.000
516 bvz1 = 0.000
517 bvx2 = 0.000
518 bvy2 = 0.000
519 bvz2 = 0.000
520 bvx3 = 0.000
521 bvy3 = 0.000
522 bvz3 = 0.000
523 crvx = 0.000
524 crvy = 0.000
525 crvz = 0.000
526 !-----
527 ! computation of curvature line integral along segment 1-4-2
528 !-----
529 bvx1 = vy(1)*DzDx(1)-vz(1)*DyDx(1)
530 bvy1 = vz(1)*DxDx(1)-vx(1)*DzDx(1)
531 bvz1 = vx(1)*DyDx(1)-vy(1)*DxDx(1)
532 !-----
533 bvx2 = vy(4)*DzDx(4)-vz(4)*DyDx(4)
534 bvy2 = vz(4)*DxDx(4)-vx(4)*DzDx(4)
535 bvz2 = vx(4)*DyDx(4)-vy(4)*DxDx(4)
536 !-----
537 bvx3 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
538 bvy3 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
539 bvz3 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
540 !-----
541 crvx = al*bvx1 + bvx2 + alc*bvx3
542 crvy = al*bvy1 + bvy2 + alc*bvy3
543 crvz = al*bvz1 + bvz2 + alc*bvz3
544 !-----
545 ! computation of curvature line integral along segment 2-5-3
546 !-----
547 bvx1 = 0.000
548 bvy1 = 0.000
549 bvz1 = 0.000
550 bvx2 = 0.000
551 bvy2 = 0.000
552 bvz2 = 0.000
553 bvx3 = 0.000
554 bvy3 = 0.000
555 bvz3 = 0.000
556 !-----
557 bvx1 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
558 bvy1 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
559 bvz1 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
560 !-----
561 bvx2 = vy(5)*DzDx(5)-vz(5)*DyDx(5)
562 bvy2 = vz(5)*DxDx(5)-vx(5)*DzDx(5)
563 bvz2 = vx(5)*DyDx(5)-vy(5)*DxDx(5)
564 !-----
565 bvx3 = vy(3)*DzDx(3)-vz(3)*DyDx(3)
566 bvy3 = vz(3)*DxDx(3)-vx(3)*DzDx(3)
567 bvz3 = vx(3)*DyDx(3)-vy(3)*DxDx(3)
568 !-----
569 crvx = crvx - gac*bvx1 - bvx2 - ga*bvx3
570 crvy = crvy - gac*bvy1 - bvy2 - ga*bvy3
571 crvz = crvz - gac*bvz1 - bvz2 - ga*bvz3
572 !-----
573 bvx1 = vy(2)*DzDe(2)-vz(2)*DyDe(2)
574 bvy1 = vz(2)*DxDe(2)-vx(2)*DzDe(2)
575 bvz1 = vx(2)*DyDe(2)-vy(2)*DxDe(2)

```

```

576 |-----|
577 |      bvx2 = vy(5)*DzDe(5)-vz(5)*DyDe(5)
578 |      bvy2 = vz(5)*DxDe(5)-vx(5)*DzDe(5)
579 |      bvz2 = vx(5)*DyDe(5)-vy(5)*DxDe(5)
580 |-----|
581 |      bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
582 |      bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
583 |      bvz3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
584 |-----|
585 |      crvx = crvx + gac*bvx1 + bvx2 + ga*bvx3
586 |      crvy = crvy + gac*bvy1 + bvy2 + ga*bvy3
587 |      crvz = crvz + gac*bvz1 + bvz2 + ga*bvz3
588 |-----|
589 | computation of curvature line integral along segment 3-6-1
590 |-----|
591 |      bvx1 = 0.000
592 |      bvy1 = 0.000
593 |      bvz1 = 0.000
594 |      bvx2 = 0.000
595 |      bvy2 = 0.000
596 |      bvz2 = 0.000
597 |      bvx3 = 0.000
598 |      bvy3 = 0.000
599 |      bvz3 = 0.000
600 |-----|
601 |      bvx1 = vy(1)*DzDe(1)-vz(1)*DyDe(1)
602 |      bvy1 = vz(1)*DxDe(1)-vx(1)*DzDe(1)
603 |      bvz1 = vx(1)*DyDe(1)-vy(1)*DxDe(1)
604 |-----|
605 |      bvx2 = vy(6)*DzDe(6)-vz(6)*DyDe(6)
606 |      bvy2 = vz(6)*DxDe(6)-vx(6)*DzDe(6)
607 |      bvz2 = vx(6)*DyDe(6)-vy(6)*DxDe(6)
608 |-----|
609 |      bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
610 |      bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
611 |      bvz3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
612 |-----|
613 |      crvx = crvx - be*bvx1 - bvx2 - bec*bvx3
614 |      crvy = crvy - be*bvy1 - bvy2 - bec*bvy3
615 |      crvz = crvz - be*bvz1 - bvz2 - bec*bvz3
616 |-----|
617 |      f11 = DSQRT(crvx**2+crvy**2+crvz**2)
618 |-----|
619 |      crvx = crvx/f11
620 |      crvy = crvy/f11
621 |      crvz = crvz/f11
622 |-----|
623 | will project curvature vector onto the normal vector at the centroid only needs the colocation vectors
624 | In this value of curvature I made a division per 8 (1/8.0) when i expected do for 4 (1/4.0)
625 |-----|
626 |      curmel(k) = (0.12500*(crvx*vnx0(k)+crvy*vny0(k)+crvz*vnz0(k))*f11)/arel(k)
627 |      WRITE (9,*) curmel(k)
628 |-----|
629 |      END DO Outer
630 |-----|
631 |-----|
632 | final computation of the surface-centroid and volume
633 |-----|
634 |      cx = cx/area
635 |      cy = cy/area
636 |      cz = cz/area
637 |      v1m = v1m/6.000
638 |      CLOSE (9)
639 |      FORMAT(10(Ix,ES24.16))
640 |-----|
641 |      END SUBROUTINE elm_geom4
642 |-----|
643 |-----|
644 | SUBROUTINE elm_geom2(nelm, npts, nint, &
645 | & xmom, ymom, zmom, &
646 | & area, v1m, &

```

```

647 | & cx, cy, cz)
648 |-----|
649 | This subroutine is a new version of Elm_Geo Subroutine.
650 | Compute:
651 | *The surface area of the individual elements x, y, and z moments over each element
652 | *the total particle surface area and volume
653 | Now, (25 / August / 2012) this subroutine was cut.
654 |-----|
655 | USE Mod_Nodal_Interp
656 | USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULOG, vna1, &
657 | & alpha0, betaQ, gammaQ, &
658 | & arel, crvmel, &
659 | & vnx0, vny0, vnz0, &
660 | & ZZ, WW, &
661 | & xiq, etq, Wq
662 |-----|
663 | IMPLICIT NONE
664 |-----|
665 | Variables
666 |-----|
667 | INTEGER, INTENT(IN) :: nelm !number of elements
668 | INTEGER, INTENT(IN) :: npts !number of points on the surface
669 | INTEGER, INTENT(IN) :: nint !order of triangle quadrature
670 | REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom !coordinates of the moments of the drop
671 | REAL (KIND = DBL), INTENT(OUT) :: area, v1m !area and volume of each element
672 | REAL (KIND = DBL), INTENT(OUT) :: cx, cy, cz !drop's centroid coordinates
673 |-----|
674 | Variables inside the subroutine
675 |-----|
676 | INTEGER :: i, k !counters
677 | INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
678 |-----|
679 | REAL (KIND = DBL) :: xi, eta !variables of weight to integrate over a triangle
680 | REAL (KIND = DBL) :: x, y, z !coordinates of the f(x,y,z)=F(xi,eta)
681 | REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi !coordinates of the tangential vector over the xi axis
682 | REAL (KIND = DBL) :: DxDet, DyDet, DzDet !coordinates of the tangential vector over the eta axis
683 | REAL (KIND = DBL) :: vnx, vny, vnz !normal vector coordinates of the element
684 | REAL (KIND = DBL) :: hs !surface metric on a triangle
685 | REAL (KIND = DBL) :: al, be, ga, alc, bec, gac !integration weigh coefficients
686 | REAL (KIND = DBL) :: cf !integration weigh coefficients
687 | REAL (KIND = DBL), DIMENSION(6) :: xxi, eet !variables of weight over in triangle (xi,eta)
688 | REAL (KIND = DBL), DIMENSION(6) :: DxDx, DyDx, DzDx !tangential vector over xi axis in triangle (xi,eta)
689 | REAL (KIND = DBL), DIMENSION(6) :: DxDy, DyDe, DzDe !tangential vector over eta axis in triangle (xi,eta)
690 | REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz !normal vector in triangle (xi,eta)
691 | REAL (KIND = DBL) :: bvx1, bvy1, bvz1, & !binormal vectors around in triangle (xi,eta)
692 | & bvx2, bvy2, bvz2, &
693 | & bvx3, bvy3, bvz3
694 | REAL (KIND = DBL) :: crvx, crvy, crvz !curvature
695 |-----|
696 | initialize
697 |-----|
698 |      area = 0.000
699 |      v1m = 0.000
700 |      cx = 0.000
701 |      cy = 0.000
702 |      cz = 0.000
703 |      arel = 0.000
704 |      xmom = 0.000
705 |      ymom = 0.000
706 |      zmom = 0.000
707 |      curmel = 0.000
708 |-----|
709 |      OPEN (9,file="curmel.out")
710 |-----|
711 |      Outer: DO k = 1, nelm
712 |-----|
713 |          i1 = n(k,1)
714 |          i2 = n(k,2)
715 |          i3 = n(k,3)
716 |          i4 = n(k,4)
717 |          i5 = n(k,5)
718 |          i6 = n(k,6)

```

```

719 al = alphaQ(k)
720 be = betaQ(k)
721 ga = gammaQ(k)
722 alc = 1.000e-01
723 bec = 1.000e-01
724 gac = 1.000e-01
-----
726 ! Compute surface area and volume of the individual elements x, y, and z moments over each element total
727 !particle surface area and volume
-----
729
730 DO i = 1, mint
731   xi = xli(i)
732   eta = eti(i)
733
734   CALL interp_p(p(i1,1), p(i1,2), p(i1,3), &
735 & p(i2,1), p(i2,2), p(i2,3), &
736 & p(i3,1), p(i3,2), p(i3,3), &
737 & p(i4,1), p(i4,2), p(i4,3), &
738 & p(i5,1), p(i5,2), p(i5,3), &
739 & p(i6,1), p(i6,2), p(i6,3), &
740 & al, be, ga, &
741 & xi, eta, &
742 & x, y, z, &
743 & DxDxi, DyDxi, DzDxi, &
744 & DxDet, DyDet, DzDet, &
745 & vmx, vny, vnz, &
746 & hs)
747   cf = hs*wg(i)
748   arel(k) = arel(k) + cf
749   xmom(k) = xmom(k) + cf*x
750   ymom(k) = ymom(k) + cf*y
751   zmom(k) = zmom(k) + cf*z
752   vlm = vlm + (x*vmx+y*vny+z*vnz)*cf
753 END DO
754
755 arel(k) = 0.500*arel(k)
756 xmom(k) = 0.500*xmom(k)
757 ymom(k) = 0.500*ymom(k)
758 zmom(k) = 0.500*zmom(k)
759 area = area + arel(k)
760
761 cx = cx + xmom(k)
762 cy = cy + ymom(k)
763 cz = cz + zmom(k)
-----
764 ! compute the average value of the normal vector the mean curvature as a contour integral using the nifty
765 !formula (4.2.18) of Pozrikidis (1997)
766 !
767
768 xxi(1) = 0.000
769 eet(1) = 0.000
770 xxi(2) = 1.000
771 eet(2) = 0.000
772 xxi(3) = 0.000
773 eet(3) = 1.000
774 xxi(4) = al
775 eet(4) = 0.000
776 xxi(5) = ga
777 eet(5) = gac
778 xxi(6) = 0.000
779 eet(6) = be
780 DO i = 1, 6
781   xi = xxi(i)
782   eta = eet(i)
783   CALL interp_p(p(i1,1), p(i1,2), p(i1,3), &
784 & p(i2,1), p(i2,2), p(i2,3), &
785 & p(i3,1), p(i3,2), p(i3,3), &
786 & p(i4,1), p(i4,2), p(i4,3), &
787 & p(i5,1), p(i5,2), p(i5,3), &
788 & p(i6,1), p(i6,2), p(i6,3), &
789 & al, be, ga, &
790 & xi, eta, &

```

```

791 & x, y, z, &
792 & DxDx(i), DyDx(i), DzDx(i), &
793 & DxDe(i), DyDe(i), DzDe(i), &
794 & vx(i), vy(i), vz(i), &
795 & hs)
796 END DO
-----
797
798 bvx1 = 0.000
799 bvy1 = 0.000
800 bvz1 = 0.000
801 bvx2 = 0.000
802 bvy2 = 0.000
803 bvz2 = 0.000
804 bvx3 = 0.000
805 bvy3 = 0.000
806 bvz3 = 0.000
807 crvx = 0.000
808 crvy = 0.000
809 crvz = 0.000
-----
810 !
811 ! computation of curvature line integral along segment 1-4-2
812 !
813 bvx1 = vna1(4*(6*(i-1)),2)*DzDx(1)-vna1(4*(6*(i-1)),3)*DyDx(1)
814 bvy1 = vna1(4*(6*(i-1)),3)*DxDx(1)-vna1(4*(6*(i-1)),2)*DzDx(1)
815 bvz1 = vna1(4*(6*(i-1)),1)*DyDx(1)-vna1(4*(6*(i-1)),2)*DxDx(1)
816 !
817 |vna1(4*(6*(i-1)),1), vna1(4*(6*(i-1)),2), vna1(4*(6*(i-1)),3), &
818 |vna1(2*(6*(i-1)),1), vna1(2*(6*(i-1)),2), vna1(2*(6*(i-1)),3), &
819 |vna1(3*(6*(i-1)),1), vna1(3*(6*(i-1)),2), vna1(3*(6*(i-1)),3), &
820 |vna1(4*(6*(i-1)),1), vna1(4*(6*(i-1)),2), vna1(4*(6*(i-1)),3), &
821 |vna1(5*(6*(i-1)),1), vna1(5*(6*(i-1)),2), vna1(5*(6*(i-1)),3), &
822 |vna1(6*(6*(i-1)),1), vna1(6*(6*(i-1)),2), vna1(6*(6*(i-1)),3), &
823 !
824 bvx2 = vna1(4*(6*(i-1)),2)*DzDx(4)-vna1(4*(6*(i-1)),3)*DyDx(4)
825 bvy2 = vna1(4*(6*(i-1)),3)*DxDx(4)-vna1(4*(6*(i-1)),2)*DzDx(4)
826 bvz2 = vna1(4*(6*(i-1)),1)*DyDx(4)-vna1(4*(6*(i-1)),2)*DxDx(4)
827 !
828 bvx3 = vna1(2*(6*(i-1)),2)*DzDx(2)-vna1(2*(6*(i-1)),3)*DyDx(2)
829 bvy3 = vna1(2*(6*(i-1)),3)*DxDx(2)-vna1(2*(6*(i-1)),2)*DzDx(2)
830 bvz3 = vna1(2*(6*(i-1)),1)*DyDx(2)-vna1(2*(6*(i-1)),2)*DxDx(2)
831 !
832 crvx = al*bvy1 + bvx2 + alc*bvx3
833 crvy = al*bvz1 + bvy2 + alc*bvy3
834 crvz = al*bvz1 + bvz2 + alc*bvz3
835 !
836 ! computation of curvature line integral along segment 2-5-3
837 !
838 bvx1 = 0.000
839 bvy1 = 0.000
840 bvz1 = 0.000
841 bvx2 = 0.000
842 bvy2 = 0.000
843 bvz2 = 0.000
844 bvx3 = 0.000
845 bvy3 = 0.000
846 bvz3 = 0.000
847 !
848 bvx1 = vna1(2*(6*(i-1)),2)*DzDx(2)-vna1(2*(6*(i-1)),3)*DyDx(2)
849 bvy1 = vna1(2*(6*(i-1)),3)*DxDx(2)-vna1(2*(6*(i-1)),2)*DzDx(2)
850 bvz1 = vna1(2*(6*(i-1)),1)*DyDx(2)-vna1(2*(6*(i-1)),2)*DxDx(2)
851 !
852 bvx2 = vna1(5*(6*(i-1)),2)*DzDx(5)-vna1(5*(6*(i-1)),3)*DyDx(5)
853 bvy2 = vna1(5*(6*(i-1)),3)*DxDx(5)-vna1(5*(6*(i-1)),2)*DzDx(5)
854 bvz2 = vna1(5*(6*(i-1)),1)*DyDx(5)-vna1(5*(6*(i-1)),2)*DxDx(5)
855 !
856 bvx3 = vna1(3*(6*(i-1)),2)*DzDx(3)-vna1(3*(6*(i-1)),3)*DyDx(3)
857 bvy3 = vna1(3*(6*(i-1)),3)*DxDx(3)-vna1(3*(6*(i-1)),2)*DzDx(3)
858 bvz3 = vna1(3*(6*(i-1)),1)*DyDx(3)-vna1(3*(6*(i-1)),2)*DxDx(3)
859 !
860 crvx = crvx + gac*bvx1 - bvx2 - ga*bvx3
861 crvy = crvy + gac*bvy1 - bvy2 - ga*bvy3
862 crvz = crvz + gac*bvz1 - bvz2 - ga*bvz3

```

```

863 |-----
864 |   bx1 = vna1(2+(6*(1-1)),2)*DzDe(2)-vna1(2+(6*(1-1)),3)*DyDe(2)
865 |   by1 = vna1(2+(6*(1-1)),3)*DxDe(2)-vna1(2+(6*(1-1)),1)*DzDe(2)
866 |   bv1 = vna1(2+(6*(1-1)),1)*DyDe(2)-vna1(2+(6*(1-1)),2)*DxDe(2)
867 |-----
868 |   bx2 = vna1(5+(6*(1-1)),2)*DzDe(5)-vna1(5+(6*(1-1)),3)*DyDe(5)
869 |   by2 = vna1(5+(6*(1-1)),3)*DxDe(5)-vna1(5+(6*(1-1)),1)*DzDe(5)
870 |   bv2 = vna1(5+(6*(1-1)),1)*DyDe(5)-vna1(5+(6*(1-1)),2)*DxDe(5)
871 |-----
872 |   bx3 = vna1(3+(6*(1-1)),2)*DzDe(3)-vna1(3+(6*(1-1)),3)*DyDe(3)
873 |   by3 = vna1(3+(6*(1-1)),3)*DxDe(3)-vna1(3+(6*(1-1)),1)*DzDe(3)
874 |   bv3 = vna1(3+(6*(1-1)),1)*DyDe(3)-vna1(3+(6*(1-1)),2)*DxDe(3)
875 |-----
876 |   crvx = crvx + gac*bx1 + bx2 + ga*bx3
877 |   crvy = crvy + gac*by1 + by2 + ga*by3
878 |   crvz = crvz + gac*bv1 + bv2 + ga*bv3
879 |-----
880 | computation of curvature line integral along segment 3-6-1
881 |-----
882 |   bx1 = 0.000
883 |   by1 = 0.000
884 |   bv1 = 0.000
885 |   bx2 = 0.000
886 |   by2 = 0.000
887 |   bv2 = 0.000
888 |   bx3 = 0.000
889 |   bv3 = 0.000
890 |-----
891 |-----
892 |   bx1 = vna1(1+(6*(1-1)),2)*DzDe(1)-vna1(1+(6*(1-1)),3)*DyDe(1)
893 |   by1 = vna1(1+(6*(1-1)),3)*DxDe(1)-vna1(1+(6*(1-1)),1)*DzDe(1)
894 |   bv1 = vna1(1+(6*(1-1)),1)*DyDe(1)-vna1(1+(6*(1-1)),2)*DxDe(1)
895 |-----
896 |   bx2 = vna1(6+(6*(1-1)),2)*DzDe(6)-vna1(6+(6*(1-1)),3)*DyDe(6)
897 |   by2 = vna1(6+(6*(1-1)),3)*DxDe(6)-vna1(6+(6*(1-1)),1)*DzDe(6)
898 |   bv2 = vna1(6+(6*(1-1)),1)*DyDe(6)-vna1(6+(6*(1-1)),2)*DxDe(6)
899 |-----
900 |   bx3 = vna1(3+(6*(1-1)),2)*DzDe(3)-vna1(3+(6*(1-1)),3)*DyDe(3)
901 |   by3 = vna1(3+(6*(1-1)),3)*DxDe(3)-vna1(3+(6*(1-1)),1)*DzDe(3)
902 |   bv3 = vna1(3+(6*(1-1)),1)*DyDe(3)-vna1(3+(6*(1-1)),2)*DxDe(3)
903 |-----
904 |   crvx = crvx + be*bx1 - bv2 - bec*bx3
905 |   crvy = crvy + be*by1 - by2 - bec*by3
906 |   crvz = crvz + be*bv1 - bv2 - bec*bv3
907 |-----
908 | will project curvature vector onto the normal vector at the centroid only needs the colocation vectors
909 | In this value of curvature I made a division per 8 (1/8.0) when I expected do for 4 (1/4.0)
910 |-----
911 |   crvml(k) = 0.2500*(crvx*vmx0(k)+crvy*vny0(k)+crvz*vnz0(k)) /arel(k)
912 |   WRITE (9,*)   crvml(k)
913 |-----
914 | END DO Outer
915 |-----
916 | final computation of the surface-centroid and volume
917 |-----
918 |-----
919 |   cx = cx/area
920 |   cy = cy/area
921 |   cz = cz/area
922 |   vlm = vlm/6.000
923 | CLOSE (9)
924 103 FORMAT(10(1X,ES24.16))
925 |-----
926 | END SUBROUTINE elm_geom2
927 |-----
928 | SUBROUTINE elm_geom3(nelm, npts, mint, &
929 | & xmom, ymom, zmom, &
930 | & area, vlm, &
931 | & cx, cy, cz)
932 |-----
933 | This subroutine is a new version of Elm_Geo Subroutine.

```

```

934 |Compute:
935 | *The surface area of the individual elements x, y, and z moments over each element
936 | *the total particle surface area and volume
937 | Now, (25 / August / 2012) this subroutine was cut.
938 |-----
939 | USE Mod_Nodal_Interp
940 | USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog, &
941 | & alphaQ, betaQ, gammaQ, &
942 | & arel, crvml, &
943 | & vnx0, vny0, vnz0, &
944 | & ZZ, WM, &
945 | & xiq, etq, wq
946 |-----
947 | IMPLICIT NONE
948 |-----
949 | Variables
950 |-----
951 | INTEGER, INTENT(IN) :: nelm !number of elements
952 | INTEGER, INTENT(IN) :: npts !number of points on the surface
953 | INTEGER, INTENT(IN) :: mint !order of triangle quadrature
954 | REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom !coordinates of the moments of the drop
955 | REAL (KIND = DBL), INTENT(OUT) :: area, vlm !area and volume of each element
956 | REAL (KIND = DBL), INTENT(OUT) :: cx, cy, cz !drop's centroid coordinates
957 |-----
958 | Variables inside the subroutine
959 |-----
960 | INTEGER :: i, k !counters
961 | INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
962 |-----
963 | REAL (KIND = DBL) :: xi, eta !variables of weigh to integrate over a triangle
964 | REAL (KIND = DBL) :: x, y, z !coordinates of the f(x,y,z)= F(xi,eta)
965 | REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi !coordinates of the tangential vector over the xi axis
966 | REAL (KIND = DBL) :: DxDe, DyDe, DzDe !coordinates of the tangential vector over the eta axis
967 | REAL (KIND = DBL) :: vnx, vny, vnz !normal vector coordinates of the element
968 | REAL (KIND = DBL) :: hs, xs, es !surface metric on a triangle
969 | REAL (KIND = DBL) :: al, be, ga, alc, bec, gac !integration weigh coefficients
970 | REAL (KIND = DBL) :: cf, fill !integration weigh coefficients
971 | REAL (KIND = DBL), DIMENSION(6) :: xxi, eet !variables of weigh over in triangle (xi,eta)
972 | REAL (KIND = DBL), DIMENSION(6) :: DxDx, DyDx, DzDx !tangential vector over xi axis in triangle (xi,eta)
973 | REAL (KIND = DBL), DIMENSION(6) :: DxDy, DyDe, DzDe !tangential vector over eta axis in triangle (xi,eta)
974 | REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz !normal vector in triangle (xi,eta)
975 | REAL (KIND = DBL) :: bv1, bv2, bv3, & !binormal vectors around in triangle (xi,eta)
976 | & bx2, by2, bv2, &
977 | & bx3, by3, bv3
978 | REAL (KIND = DBL) :: crvx, crvy, crvz !curvature
979 |-----
980 | Initialize
981 |-----
982 | area = 0.000
983 | vlm = 0.000
984 | cx = 0.000
985 | cy = 0.000
986 | cz = 0.000
987 | arel = 0.000
988 | xmom = 0.000
989 | ymom = 0.000
990 | zmom = 0.000
991 | crvml = 0.000
992 | fill = 0.000
993 |-----
994 | OPEN (9,file="curvml.out")
995 |-----
996 | Outer: DO k = 1, nelm
997 |-----
998 |   i1 = n(k,1)
999 |   i2 = n(k,2)
1000 |   i3 = n(k,3)
1001 |   i4 = n(k,4)
1002 |   i5 = n(k,5)
1003 |   i6 = n(k,6)
1004 |   al = alphaQ(k)
1005 |   be = betaQ(k)

```



```

1006 ga = gammaQ(k)
1007 alc = 1.000-alc
1008 bec = 1.000-be
1009 gac = 1.000-ga
1010
1011 ! Compute surface area and volume of the individual elements x, y, and z moments over each element total
1012 !particle surface area and volume
1013
1014
1015
1016 DO i = 1, mint
1017   xi = xiq(i)
1018   eta = etq(i)
1019   CALL interp_p(p(i1,1), p(i1,2), p(i1,3), &
1020 & p(i2,1), p(i2,2), p(i2,3), &
1021 & p(i3,1), p(i3,2), p(i3,3), &
1022 & p(i4,1), p(i4,2), p(i4,3), &
1023 & p(i5,1), p(i5,2), p(i5,3), &
1024 & p(i6,1), p(i6,2), p(i6,3), &
1025 & al, be, ga, &
1026 & xi, eta, z, &
1027 & x, y, z, &
1028 & DxDxi, DyDxi, DzDxi, &
1029 & DxDet, DyDet, DzDet, &
1030 & vnx, vny, vnz, &
1031 & hs)
1032   cf = hs*wq(i)
1033   arel(k) = arel(k) + cf
1034   xmom(k) = xmom(k) + cf*x
1035   ymom(k) = ymom(k) + cf*y
1036   zmom(k) = zmom(k) + cf*z
1037   vlm = vlm + (x*vnx+y*vny+z*vnz)*cf
1038   cf=0.000
1039 END DO
1040
1041
1042 arel(k) = 0.500*arel(k)
1043 xmom(k) = 0.500*xmom(k)
1044 ymom(k) = 0.500*ymom(k)
1045 zmom(k) = 0.500*zmom(k)
1046 area = area + arel(k)
1047 cx = cx + xmom(k)
1048 cy = cy + ymom(k)
1049 cz = cz + zmom(k)
1050
1051 ! compute the average value of the normal vector the mean curvature as a contour integral using the nftty
1052 ! formula (4.2.18) of Pozrikidis (1997)
1053
1054 xxi(1) = 0.000
1055 eet(1) = 0.000
1056 xxi(2) = 1.000
1057 eet(2) = 0.000
1058 xxi(3) = 0.000
1059 eet(3) = 1.000
1060 xxi(4) = al
1061 eet(4) = 0.000
1062 xxi(5) = ga
1063 eet(5) = gac
1064 xxi(6) = 0.000
1065 eet(6) = be
1066 DO i = 1, 6
1067   xi = xxi(i)
1068   eta = eet(i)
1069   CALL interp_p3(p(i1,1), p(i1,2), p(i1,3), &
1070 & p(i2,1), p(i2,2), p(i2,3), &
1071 & p(i3,1), p(i3,2), p(i3,3), &
1072 & p(i4,1), p(i4,2), p(i4,3), &
1073 & p(i5,1), p(i5,2), p(i5,3), &
1074 & p(i6,1), p(i6,2), p(i6,3), &
1075 & al, be, ga, &
1076 & xi, eta, z, &
1077 & x, y, z, &

```

```

1078 & DxD(1), DyD(1), DzD(1), &
1079 & DxDe(1), DyDe(1), DzDe(1), &
1080 & vx(1), vy(1), vz(1), &
1081 & hs)
1082 END DO
1083
1084 bvx1 = 0.000
1085 bvy1 = 0.000
1086 bvz1 = 0.000
1087 bvx2 = 0.000
1088 bvy2 = 0.000
1089 bvz2 = 0.000
1090 bvx3 = 0.000
1091 bvy3 = 0.000
1092 bvz3 = 0.000
1093 crvx = 0.000
1094 crvy = 0.000
1095 crvz = 0.000
1096
1097 ! computation of curvature line integral along segment 1-4-2
1098
1099 bvx1 = vy(1)*DzDx(1)-vz(1)*DyDx(1)
1100 bvy1 = vz(1)*DxDx(1)-vx(1)*DzDx(1)
1101 bvz1 = vx(1)*DyDx(1)-vy(1)*DxDx(1)
1102
1103 bvx2 = vy(4)*DzDx(4)-vz(4)*DyDx(4)
1104 bvy2 = vz(4)*DxDx(4)-vx(4)*DzDx(4)
1105 bvz2 = vx(4)*DyDx(4)-vy(4)*DxDx(4)
1106
1107 bvx3 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
1108 bvy3 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
1109 bvz3 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
1110
1111 crvx = al*bvx1 + bvx2 + alc*bvx3
1112 crvy = al*bvy1 + bvy2 + alc*bvy3
1113 crvz = al*bvz1 + bvz2 + alc*bvz3
1114
1115 ! computation of curvature line integral along segment 2-5-3
1116
1117 bvx1 = 0.000
1118 bvy1 = 0.000
1119 bvz1 = 0.000
1120 bvx2 = 0.000
1121 bvy2 = 0.000
1122 bvz2 = 0.000
1123 bvx3 = 0.000
1124 bvy3 = 0.000
1125 bvz3 = 0.000
1126
1127 bvx1 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
1128 bvy1 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
1129 bvz1 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
1130
1131 bvx2 = vy(5)*DzDx(5)-vz(5)*DyDx(5)
1132 bvy2 = vz(5)*DxDx(5)-vx(5)*DzDx(5)
1133 bvz2 = vx(5)*DyDx(5)-vy(5)*DxDx(5)
1134
1135 bvx3 = vy(3)*DzDx(3)-vz(3)*DyDx(3)
1136 bvy3 = vz(3)*DxDx(3)-vx(3)*DzDx(3)
1137 bvz3 = vx(3)*DyDx(3)-vy(3)*DxDx(3)
1138
1139 crvx = crvx - gac*bvx1 - bvx2 - ga*bvx3
1140 crvy = crvy - gac*bvy1 - bvy2 - ga*bvy3
1141 crvz = crvz - gac*bvz1 - bvz2 - ga*bvz3
1142
1143 bvx1 = vy(2)*DzDe(2)-vz(2)*DyDe(2)
1144 bvy1 = vz(2)*DxDe(2)-vx(2)*DzDe(2)
1145 bvz1 = vx(2)*DyDe(2)-vy(2)*DxDe(2)
1146
1147 bvx2 = vy(5)*DzDe(5)-vz(5)*DyDe(5)
1148 bvy2 = vz(5)*DxDe(5)-vx(5)*DzDe(5)
1149 bvz2 = vx(5)*DyDe(5)-vy(5)*DxDe(5)

```

```
1150 |-----|
1151 | bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
1152 | bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
1153 | bvx3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
1154 |-----|
1155 | crvx = crvx + gac*bvx1 + bvx2 + ga*bvx3
1156 | crvy = crvy + gac*bvy1 + bvy2 + ga*bvy3
1157 | crvz = crvz + gac*bvz1 + bvz2 + ga*bvz3
1158 |-----|
1159 | computation of curvature line integral along segment 3-6-1
1160 |-----|
1161 | bvx1 = 0.000
1162 | bvy1 = 0.000
1163 | bvz1 = 0.000
1164 | bvx2 = 0.000
1165 | bvy2 = 0.000
1166 | bvz2 = 0.000
1167 | bvx3 = 0.000
1168 | bvy3 = 0.000
1169 | bvz3 = 0.000
1170 |-----|
1171 | bvx1 = vy(1)*DzDe(1)-vz(1)*DyDe(1)
1172 | bvy1 = vz(1)*DxDe(1)-vx(1)*DzDe(1)
1173 | bvz1 = vx(1)*DyDe(1)-vy(1)*DxDe(1)
1174 |-----|
1175 | bvx2 = vy(6)*DzDe(6)-vz(6)*DyDe(6)
1176 | bvy2 = vz(6)*DxDe(6)-vx(6)*DzDe(6)
1177 | bvz2 = vx(6)*DyDe(6)-vy(6)*DxDe(6)
1178 |-----|
1179 | bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
1180 | bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
1181 | bvz3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
1182 |-----|
1183 | crvx = crvx - be*bvx1 - bvx2 - bec*bvx3
1184 | crvy = crvy - be*bvy1 - bvy2 - bec*bvy3
1185 | crvz = crvz - be*bvz1 - bvz2 - bec*bvz3
1186 |-----|
1187 | fil = DSQR((crrx**2+crry**2+crrz**2)
1188 |!!!-----|
1189 | crvx = crvx/fil
1190 | crvy = crvy/fil
1191 | crvz = crvz/fil
1192 |-----|
1193 | will project curvature vector onto the normal vector at the centroid only needs the colocation vectors
1194 | In this value of curvature I made a division per 8 (1/8.0) when I expected do for 4 (1/4.0)
1195 |-----|
1196 | crvmel(k) = 0.2500*(crrx*vnz0(k)+crry*vny0(k)+crrz*vnz0(k))/arel(k)
1197 | WRITE (9,*) crvmel(k)
1198 |
1199 | END DO Outer
1200 |-----|
1201 | final computation of the surface-centroid and volume
1202 |-----|
1203 |-----|
1204 | cx = cx/area
1205 | cy = cy/area
1206 | cz = cz/area
1207 | vlm = vlm/6.000
1208 | CLOSE (9)
1209 |103 FORMAT(10(1x,ES24.16))
1210 |-----|
1211 | END SUBROUTINE elm_geom3
1212 |-----|
1213 | END MODULE Mod_Prtcl_3D_Geo
1214 |-----|
```

```

1 MODULE Mod_Prtcl_3D_Geo
2 =====
3 ! Version: 0.5 created on 26 / IX / 2007
4 !
5 !-----
6 ! Version: 0.7 created on -- / III / 2010
7 !
8 !-----
9 ! Version: 0.9 created on 23 / 08 / 2012
10 ! Version: 1.0 created on 14 / 11 / 2012
11 !
12 !-----
13 CONTAINS
14 =====
15 SUBROUTINE pritel(k, Index, c)
16 !-----
17 ! This subroutine prints drop's geometry. It has two options.
18 ! Print successive nodes of element k in file unit 1
19 ! Index = 1: print the whole element
20 ! Index = 2: print the 4 subelements
21 !-----
22 ! Variables
23 !-----
24 ! k ..... number of element
25 ! index ..... type of print
26 ! nfour ..... index to print only the element
27 ! nseven ..... index to print the subelements
28 ! c ..... firstly, it was the value of shear strain, now, it will change. ;)
29 !-----
30 USE Mod_SharedVars, ONLY: DBL, ULog, UGeo, p, ne, n, nbe
31 =====
32 IMPLICIT NONE
33 !-----
34 INTEGER, INTENT(IN) :: k, index
35 !-----
36 INTEGER :: nfour, nseven, i
37 REAL (KIND = DBL), DIMENSION(:), INTENT(IN) :: c
38 !-----
39 ! constants
40 nfour = 4
41 nseven = 1
42 !-----
43 ! There is a CASE instruction to print the element.
44 !-----
45 SELECT CASE (index)
46 CASE (1)
47 WRITE (UGeo,100) nseven
48 i = n(k,1)
49 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
50 i = n(k,4)
51 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
52 i = n(k,2)
53 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
54 i = n(k,5)
55 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
56 i = n(k,3)
57 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
58 i = n(k,6)
59 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
60 i = n(k,1)
61 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
62 CASE (2)
63 !--- first
64 WRITE (UGeo,100) nfour
65 i = n(k,1)
66 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
67 i = n(k,4)
68 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
69 i = n(k,6)
70 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
71 i = n(k,1)
72 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)

```

```

73 !--- second
74 i = n(k,4)
75 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
76 i = n(k,2)
77 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
78 i = n(k,5)
79 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
80 i = n(k,4)
81 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
82 !--- third
83 i = n(k,4)
84 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
85 i = n(k,5)
86 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
87 i = n(k,6)
88 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
89 i = n(k,4)
90 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
91 !--- fourth
92 i = n(k,6)
93 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
94 i = n(k,5)
95 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
96 i = n(k,3)
97 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
98 i = n(k,6)
99 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
100 CASE DEFAULT
101 WRITE (UGeo,100) nfour
102 i = n(k,1)
103 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
104 i = n(k,4)
105 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
106 i = n(k,6)
107 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
108 i = n(k,1)
109 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
110 i = n(k,4)
111 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
112 i = n(k,2)
113 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
114 i = n(k,5)
115 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
116 i = n(k,4)
117 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
118 i = n(k,4)
119 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
120 i = n(k,5)
121 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
122 i = n(k,6)
123 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
124 i = n(k,4)
125 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
126 i = n(k,6)
127 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
128 i = n(k,5)
129 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
130 i = n(k,3)
131 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
132 i = n(k,6)
133 WRITE (UGeo,101) p(i,1),p(i,2),p(i,3),c(i)
134 WRITE (ULog,*)
135 WRITE (ULog,*) ' Geo_Pritel'
136 WRITE (ULog,*)
137 WRITE (ULog,*) ' Chosen index is not available'
138 WRITE (ULog,*) ' It was taken index= 2'
139 END SELECT
140 100 FORMAT(1x,14,10(1x,ES24.16))
141 101 FORMAT(10(1x,ES24.16))
142 END SUBROUTINE pritel
143 !-----
144 !

```

D:\Darth Vader\Escritorio\prtcl mkl\Mod Prtcl 3D\_Geo.f90 3

```
145 SUBROUTINE abc(x1, y1, z1, &
146 & x2, y2, z2, &
147 & x3, y3, z3, &
148 & x4, y4, z4, &
149 & x5, y5, z5, &
150 & x6, y6, z6, &
151 & al, be, ga)
152 =====
153 ! This subroutine compute the parametric representation constants alpha, beta, gamma
154 ! =====
155 USE Mod_SharedVars, ONLY: DBL
156 ! =====
157 IMPLICIT NONE
158 ! =====
159 ! Variables
160 ! =====
161 REAL (KIND = DBL), INTENT(IN) :: x1, y1, z1, & !coordinates of each point in the element
162 & x2, y2, z2, &
163 & x3, y3, z3, &
164 & x4, y4, z4, &
165 & x5, y5, z5, &
166 & x6, y6, z6
167 REAL (KIND = DBL), INTENT(OUT) :: al, be, ga !constants alpha, beta and gamma (weights)
168 ! =====
169 ! Variables inside the subroutine
170 ! =====
171 REAL (KIND = DBL) :: d42, d41 !distances of the element on the segment 1 4 2
172 REAL (KIND = DBL) :: d63, d61 !distances of the element on the segment 3 6 1
173 REAL (KIND = DBL) :: d52, d53 !distances of the element on the segment 2 5 3
174 ! =====
175 ! Initialize
176 d42 = (x4 - x2)**2 + (y4 - y2)**2 + (z4 - z2)**2
177 d41 = (x4 - x1)**2 + (y4 - y1)**2 + (z4 - z1)**2
178 d63 = (x6 - x3)**2 + (y6 - y3)**2 + (z6 - z3)**2
179 d61 = (x6 - x1)**2 + (y6 - y1)**2 + (z6 - z1)**2
180 d52 = (x5 - x2)**2 + (y5 - y2)**2 + (z5 - z2)**2
181 d53 = (x5 - x3)**2 + (y5 - y3)**2 + (z5 - z3)**2
182 ! =====
183 d42 = DSQRT(d42)
184 d41 = DSQRT(d41)
185 d63 = DSQRT(d63)
186 d61 = DSQRT(d61)
187 d52 = DSQRT(d52)
188 d53 = DSQRT(d53)
189 ! =====
190 al = 1.000/(1.000 + d42/d41)
191 be = 1.000/(1.000 + d63/d61)
192 ga = 1.000/(1.000 + d52/d53)
193 ! =====
194 END SUBROUTINE abc
195 ! =====
196 SUBROUTINE elm_geom(nelm, npts, mint, &
197 & xmom, ymom, zmom, &
198 & area, vlm, &
199 & cx, cy, cz, stride)
200 ! =====
201 ! This subroutine is a new version of Elm_Geo Subroutine.
202 ! Compute:
203 ! *The surface area of the individual elements x, y, and z moments over each element
204 ! *the total particle surface area and volume
205 ! Now, (25 / August / 2012) this subroutine was cut.
206 ! =====
207 USE Mod_Nodal_Interp
208 USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, Ulog,&
209 & alpha0, beta0, gamma0, &
210 & arel, crvmel, &
211 & vnx0, vny0, vnz0, &
212 & zz, ww, &
213 & xiq, etq, wq
214 ! =====
215 IMPLICIT NONE
```

D:\Darth Vader\Escritorio\prtcl mkl\Mod Prtcl 3D\_Geo.f90 4

```
217 ! =====
218 ! Variables
219 ! =====
220 INTEGER, INTENT(IN) :: nelm !number of elements
221 INTEGER, INTENT(IN) :: npts !number of points on the surface
222 INTEGER, INTENT(IN) :: mint !order of triangle quadrature
223 INTEGER, INTENT(IN) :: stride !order of triangle and Gauss-Legendre quadratures
224 REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom !coordinates of the moments of the drop
225 REAL (KIND = DBL), INTENT(OUT) :: area, vlm !area and volume of each element
226 REAL (KIND = DBL), INTENT(OUT) :: cx, cy, cz !drop's centroid coordinates
227 ! =====
228 ! Variables inside the subroutine
229 ! =====
230 INTEGER :: i, k !counters
231 INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
232 ! =====
233 REAL (KIND = DBL) :: xi, eta !variables of weigh to integrate over a triangle
234 REAL (KIND = DBL) :: x, y, z !coordinates of the f(x,y,z)=f(xi,eta)
235 REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi !coordinates of the tangential vector over the xi axis
236 REAL (KIND = DBL) :: DxDeta, DyDeta, DzDeta !coordinates of the tangential vector over the eta axis
237 REAL (KIND = DBL) :: vnx, vny, vnz !normal vector coordinates of the element
238 REAL (KIND = DBL) :: hs !surface metric on a triangle
239 REAL (KIND = DBL) :: al, be, ga, alc, bec, gac !integration weigh coefficients
240 REAL (KIND = DBL) :: cf, fil !integration weigh coefficients
241 REAL (KIND = DBL), DIMENSION(6) :: xxi, eet !variables of weigh over in triangle (xi,eta)
242 REAL (KIND = DBL), DIMENSION(6) :: DxDx, DyDx, DzDx !tangential vector over xi axis in triangle (xi,eta)
243 REAL (KIND = DBL), DIMENSION(6) :: DxDeta, DyDeta, DzDeta !tangential vector over eta axis in triangle (xi,eta)
244 REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz !normal vector in triangle (xi,eta)
245 REAL (KIND = DBL) :: bx1, by1, bvz1, & !binormal vectors around in triangle (xi,eta)
246 & bx2, by2, bvz2, &
247 & bx3, by3, bvz3
248 REAL (KIND = DBL) :: crvx, crvy, crvz, curv !curvature
249 ! =====
250 ! initialize
251 ! =====
252 area = 0.000
253 vlm = 0.000
254 cx = 0.000
255 cy = 0.000
256 cz = 0.000
257 arel = 0.000
258 xmom = 0.000
259 ymom = 0.000
260 zmom = 0.000
261 crvmel = 0.000
262 fil = 0.000
263 ! =====
264 Outer: DO k = 1, nelm
265 ! =====
266 i1 = n(k,1)
267 i2 = n(k,2)
268 i3 = n(k,3)
269 i4 = n(k,4)
270 i5 = n(k,5)
271 i6 = n(k,6)
272 al = alpha0(k)
273 be = beta0(k)
274 ga = gamma0(k)
275 alc = 1.000-al
276 bec = 1.000-be
277 gac = 1.000-ga
278 ! =====
279 ! Compute surface area and volume of the individual elements x, y, and z moments over each element total
280 !particle surface area and volume
281 ! =====
282 DO i = 1, mint
283 xi = xiq(i)
284 eta = etq(i)
285 CALL interp_p(p(i1,1), p(i1,2), p(i1,3), &
286 & p(i2,1), p(i2,2), p(i2,3), &
287 & p(i3,1), p(i3,2), p(i3,3), &
288 & p(i4,1), p(i4,2), p(i4,3), &
```

```

289      &      p(15,1), p(15,2), p(15,3), &
290      &      p(16,1), p(16,2), p(16,3), &
291      &      al, be, ga, &
292      &      xi, eta, &
293      &      x, y, z, &
294      &      Dx0x1, Dy0x1, Dz0x1, &
295      &      Dx0et, Dy0et, Dz0et, &
296      &      vnx, vny, vnz, &
297      &      hs)
298      cf = hs*wq(1)
299      arel(k) = arel(k) + cf
300      xmom(k) = xmom(k) + cf*x
301      ymom(k) = ymom(k) + cf*y
302      zmom(k) = zmom(k) + cf*z
303      vlm = vlm + (x*vnx+y*vny+z*vnz)*cf
304      END DO
305      arel(k) = 0.500*arel(k)
306      xmom(k) = 0.500*xmom(k)
307      ymom(k) = 0.500*ymom(k)
308      zmom(k) = 0.500*zmom(k)
309      area = area + arel(k)
310      cx = cx + xmom(k)
311      cy = cy + ymom(k)
312      cz = cz + zmom(k)
313      !-----
314      ! compute the average value of the normal vector the mean curvature as a contour integral using the nifty
315      ! formula (4.2.10) of Pozrikidis (1997)
316      !-----
317      xxi(1) = 0.000
318      eet(1) = 0.000
319      xxi(2) = 1.000
320      eet(2) = 0.000
321      xxi(3) = 0.000
322      eet(3) = 1.000
323      xxi(4) = al
324      eet(4) = 0.000
325      xxi(5) = ga
326      eet(5) = gac
327      xxi(6) = 0.000
328      eet(6) = be
329      xi = 1.000/3.000
330      eta = 1.000/3.000
331      CALL interp_p4(p(11,1), p(11,2), p(11,3), &
332      &      p(12,1), p(12,2), p(12,3), &
333      &      p(13,1), p(13,2), p(13,3), &
334      &      p(14,1), p(14,2), p(14,3), &
335      &      p(15,1), p(15,2), p(15,3), &
336      &      p(16,1), p(16,2), p(16,3), &
337      &      al, be, ga, &
338      &      xi, eta, &
339      &      curv, stride )
340      !-----
341      ! will project curvature vector onto the normal vector at the centroid only needs the colocation vectors
342      ! in this value of curvature I made a division per 8 (1/8.0) when I expected do for 4 (1/4.0)
343      !-----
344      crvmel(k) = curv [(0.2500*(crvx*vnx0(k)+crvy*vny0(k)+crvz*vnz0(k))*fil)/arel(k)]
345      END DO Outer
346      !-----
347      ! final computation of the surface-centroid and volume
348      !-----
349      !-----
350      cx = cx/area
351      cy = cy/area
352      cz = cz/area
353      vlm = vlm/6.000
354      !03 FORMAT(10(1x,ES24.16))
355      !-----
356      END SUBROUTINE elm_geom
357      !-----
358      !-----
359      SUBROUTINE elm_geom4(nelm, npts, mint, &

```

```

360      &      xmom, ymom, zmom, &
361      &      area, vlm, &
362      &      cx, cy, cz) &
363      !-----
364      ! This subroutine is a new version of Elm_Geo Subroutine.
365      ! Compute:
366      ! *The surface area of the individual elements x, y, and z moments over each element
367      ! *the total particle surface area and volume
368      ! Now, (25 / August / 2012) this subroutine was cut.
369      !-----
370      USE Mod_Nodal_Interp
371      USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog, &
372      &      alphaQ, betaQ, gammaQ, &
373      &      arel, crvmel, &
374      &      vnx0, vny0, vnz0, &
375      &      ZZ, vny, vnz, &
376      &      xiq, etq, wq
377      !-----
378      IMPLICIT NONE
379      !-----
380      ! Variables
381      !-----
382      INTEGER, INTENT(IN) :: nelm          !number of elements
383      INTEGER, INTENT(IN) :: npts         !number of points on the surface
384      INTEGER, INTENT(IN) :: mint         !order of triangle quadrature
385      REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom !coordinates of the moments of the drop
386      REAL (KIND = DBL), INTENT(OUT) :: area, vlm !area and volume of each element
387      REAL (KIND = DBL), INTENT(OUT) :: cx, cy, cz !drop's centroid coordinates
388      !-----
389      ! Variables inside the subroutine
390      !-----
391      INTEGER :: i, k
392      INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
393      !-----
394      REAL (KIND = DBL) :: xi, eta !variables of weigh to integrate over a triangle
395      REAL (KIND = DBL) :: x, y, z !coordinates of the f(x,y,z)= F(xi,eta)
396      REAL (KIND = DBL) :: Dx0xi, Dy0xi, Dz0xi !coordinates of the tangential vector over the xi axis
397      REAL (KIND = DBL) :: Dx0et, Dy0et, Dz0et !coordinates of the tangential vector over the eta axis
398      REAL (KIND = DBL) :: vnx, vny, vnz !normal vector coordinates of the element
399      REAL (KIND = DBL) :: hs !surface metric on a triangle
400      REAL (KIND = DBL) :: al, be, ga, alc, bec, gac !integration weigh coefficients
401      REAL (KIND = DBL) :: cf, fil !integration weigh coefficients
402      REAL (KIND = DBL), DIMENSION(6) :: xxi, eet !variables of weigh over in triangle (xi,eta)
403      REAL (KIND = DBL), DIMENSION(6) :: Dx0x, Dy0x, Dz0x !tangential vector over xi axis in triangle (xi,eta)
404      REAL (KIND = DBL), DIMENSION(6) :: Dx0e, Dy0e, Dz0e !tangential vector over eta axis in triangle (xi,eta)
405      REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz !normal vector in triangle (xi,eta)
406      REAL (KIND = DBL) :: bvxi, bvyi, bvzi, & !binormal vectors around in triangle (xi,eta)
407      &      bvx2, bvy2, bvz2, &
408      &      bvx3, bvy3, bvz3
409      REAL (KIND = DBL) :: crvx, crvy, crvz !curvature
410      !-----
411      ! initialize
412      !-----
413      area = 0.000
414      vlm = 0.000
415      cx = 0.000
416      cy = 0.000
417      cz = 0.000
418      arel = 0.000
419      xmom = 0.000
420      ymom = 0.000
421      zmom = 0.000
422      crvmel = 0.000
423      fil = 0.000
424      !-----
425      ! OPEN (9,files="curvmel.out")
426      !-----
427      Outer: DO k = 1, nelm
428      !-----
429      i1 = n(k,1)
430      i2 = n(k,2)
431      i3 = n(k,3)

```

```

432 14 = n(k,4)
433 15 = n(k,5)
434 16 = n(k,6)
435 al = alphaQ(k)
436 be = betaQ(k)
437 ga = gammaQ(k)
438 alc = 1.000-al
439 bec = 1.000-be
440 gac = 1.000-ga
441 !-----
442 ! Compute surface area and volume of the individual elements x, y, and z moments over each element total
443 !particle surface area and volume
444 !-----
445
446 DO i = 1, mint
447   xl = xiq(i)
448   eta = etq(i)
449   CALL interp_p(p(11,1), p(11,2), p(11,3), &
450     & p(12,1), p(12,2), p(12,3), &
451     & p(13,1), p(13,2), p(13,3), &
452     & p(14,1), p(14,2), p(14,3), &
453     & p(15,1), p(15,2), p(15,3), &
454     & p(16,1), p(16,2), p(16,3), &
455     & al, be, ga, &
456     & xl, eta, &
457     & x, y, z, &
458     & DxDx1, DyDx1, DzDx1, &
459     & DxDet, DyDet, DzDet, &
460     & vnx, vny, vnz, &
461     & hs)
462   cf = hs*sq(i)
463   arel(k) = arel(k) + cf
464   xmom(k) = xmom(k) + cf*x
465   ymom(k) = ymom(k) + cf*y
466   zmom(k) = zmom(k) + cf*z
467   vlm = vlm + (x*vnx+y*vny+z*vnz)*cf
468 END DO
469
470
471 arel(k) = 0.500*arel(k)
472 xmom(k) = 0.500*xmom(k)
473 ymom(k) = 0.500*ymom(k)
474 zmom(k) = 0.500*zmom(k)
475 area = area + arel(k)
476 cx = cx + xmom(k)
477 cy = cy + ymom(k)
478 cz = cz + zmom(k)
479 !-----
480 ! compute the average value of the normal vector the mean curvature as a contour integral using the nifty
481 ! formula (4.2.10) of Pozrikidis (1997)
482 !-----
483
484 xxi(1) = 0.000
485 eet(1) = 0.000
486 xxi(2) = 1.000
487 eet(2) = 0.000
488 xxi(3) = 0.000
489 eet(3) = 1.000
490 xxi(4) = al
491 eet(4) = 0.000
492 xxi(5) = ga
493 eet(5) = gac
494 xxi(6) = 0.000
495 eet(6) = be
496 DO i = 1, 6
497   xl = xxi(i)
498   eta = eet(i)
499   CALL interp_p(p(11,1), p(11,2), p(11,3), &
500     & p(12,1), p(12,2), p(12,3), &
501     & p(13,1), p(13,2), p(13,3), &
502     & p(14,1), p(14,2), p(14,3), &
503     & p(15,1), p(15,2), p(15,3), &

```

```

504     & p(16,1), p(16,2), p(16,3), &
505     & al, be, ga, &
506     & xl, eta, z, &
507     & x, y, z, &
508     & DxDx(i), DyDx(i), DzDx(i), &
509     & DxDe(i), DyDe(i), DzDe(i), &
510     & vx(i), vy(i), vz(i), &
511     & hs)
512 END DO
513 !-----
514 bvx1 = 0.000
515 bvy1 = 0.000
516 bvz1 = 0.000
517 bvx2 = 0.000
518 bvy2 = 0.000
519 bvz2 = 0.000
520 bvx3 = 0.000
521 bvy3 = 0.000
522 bvz3 = 0.000
523 crvx = 0.000
524 crvy = 0.000
525 crvz = 0.000
526 !-----
527 ! computation of curvature line integral along segment 1-4-2
528 !-----
529 bvx1 = vy(1)*DzDx(1)-vz(1)*DyDx(1)
530 bvy1 = vz(1)*DxDx(1)-vx(1)*DzDx(1)
531 bvz1 = vx(1)*DyDx(1)-vy(1)*DxDx(1)
532 !-----
533 bvx2 = vy(4)*DzDx(4)-vz(4)*DyDx(4)
534 bvy2 = vz(4)*DxDx(4)-vx(4)*DzDx(4)
535 bvz2 = vx(4)*DyDx(4)-vy(4)*DxDx(4)
536 !-----
537 bvx3 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
538 bvy3 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
539 bvz3 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
540 !-----
541 crvx = al*bvx1 + bvx2 + alc*bvx3
542 crvy = al*bvy1 + bvy2 + alc*bvy3
543 crvz = al*bvz1 + bvz2 + alc*bvz3
544 !-----
545 ! computation of curvature line integral along segment 2-5-3
546 !-----
547 bvx1 = 0.000
548 bvy1 = 0.000
549 bvz1 = 0.000
550 bvx2 = 0.000
551 bvy2 = 0.000
552 bvz2 = 0.000
553 bvx3 = 0.000
554 bvy3 = 0.000
555 bvz3 = 0.000
556 !-----
557 bvx1 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
558 bvy1 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
559 bvz1 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
560 !-----
561 bvx2 = vy(5)*DzDx(5)-vz(5)*DyDx(5)
562 bvy2 = vz(5)*DxDx(5)-vx(5)*DzDx(5)
563 bvz2 = vx(5)*DyDx(5)-vy(5)*DxDx(5)
564 !-----
565 bvx3 = vy(3)*DzDx(3)-vz(3)*DyDx(3)
566 bvy3 = vz(3)*DxDx(3)-vx(3)*DzDx(3)
567 bvz3 = vx(3)*DyDx(3)-vy(3)*DxDx(3)
568 !-----
569 crvx = crvx - gac*bvx1 - bvx2 - ga*bvx3
570 crvy = crvy - gac*bvy1 - bvy2 - ga*bvy3
571 crvz = crvz - gac*bvz1 - bvz2 - ga*bvz3
572 !-----
573 bvx1 = vy(2)*DzDe(2)-vz(2)*DyDe(2)
574 bvy1 = vz(2)*DxDe(2)-vx(2)*DzDe(2)
575 bvz1 = vx(2)*DyDe(2)-vy(2)*DxDe(2)

```



```

576 |-----|
577 |   bvx2 = vy(5)*DzDe(5)-vz(5)*DyDe(5)
578 |   bvy2 = vz(5)*DxDe(5)-vx(5)*DzDe(5)
579 |   bvx2 = vx(5)*DyDe(5)-vy(5)*DxDe(5)
580 |-----|
581 |   bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
582 |   bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
583 |   bvx3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
584 |-----|
585 |   crvx = crvx + gac*bvx1 + bvx2 + ga*bvx3
586 |   crvy = crvy + gac*bvy1 + bvy2 + ga*bvy3
587 |   crvz = crvz + gac*bvz1 + bvz2 + ga*bvz3
588 |-----|
589 | computation of curvature line integral along segment 3-6-1
590 |-----|
591 |   bvx1 = 0.000
592 |   bvy1 = 0.000
593 |   bvz1 = 0.000
594 |   bvx2 = 0.000
595 |   bvy2 = 0.000
596 |   bvz2 = 0.000
597 |   bvx3 = 0.000
598 |   bvy3 = 0.000
599 |   bvz3 = 0.000
600 |-----|
601 |   bvx1 = vy(1)*DzDe(1)-vz(1)*DyDe(1)
602 |   bvy1 = vz(1)*DxDe(1)-vx(1)*DzDe(1)
603 |   bvx1 = vx(1)*DyDe(1)-vy(1)*DxDe(1)
604 |-----|
605 |   bvx2 = vy(6)*DzDe(6)-vz(6)*DyDe(6)
606 |   bvy2 = vz(6)*DxDe(6)-vx(6)*DzDe(6)
607 |   bvx2 = vx(6)*DyDe(6)-vy(6)*DxDe(6)
608 |-----|
609 |   bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
610 |   bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
611 |   bvx3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
612 |-----|
613 |   crvx = crvx - be*bvx1 - bvx2 - bec*bvx3
614 |   crvy = crvy - be*bvy1 - bvy2 - bec*bvy3
615 |   crvz = crvz - be*bvz1 - bvz2 - bec*bvz3
616 |-----|
617 |   f11 = DSQRT(crvx**2+crvy**2+crvz**2)
618 |-----|
619 |   crvx = crvx/f11
620 |   crvy = crvy/f11
621 |   crvz = crvz/f11
622 |-----|
623 | will project curvature vector onto the normal vector at the centroid only needs the colocation vectors
624 | In this value of curvature I made a division per 8 (1/8.0) when i expected do for 4 (1/4.0)
625 |-----|
626 |   curmel(k) = (0.12500*(crvx*vx0(k)+crvy*vy0(k)+crvz*vz0(k))*f11)/arel(k)
627 |   WRITE (9,*)   curmel(k)
628 |-----|
629 |   END DO Outer
630 |-----|
631 |-----|
632 | final computation of the surface-centroid and volume
633 |-----|
634 |   cx = cx/area
635 |   cy = cy/area
636 |   cz = cz/area
637 |   v1m = v1m/6.000
638 |   CLOSE (9)
639 |   FORMAT(10(1X,ES24.16))
640 |-----|
641 |   END SUBROUTINE elm_geom4
642 |-----|
643 |-----|
644 | SUBROUTINE elm_geom2(nelm, npts, nint, &
645 | & xmom, ymom, zmom, &
646 | & area, v1m, &

```

```

647 | & cx, cy, cz)
648 |-----|
649 | This subroutine is a new version of Elm_Geo Subroutine.
650 | Compute:
651 | *The surface area of the individual elements x, y, and z moments over each element
652 | *the total particle surface area and volume
653 | Now, (25 / August / 2012) this subroutine was cut.
654 |-----|
655 | USE Mod_Nodal_Interp
656 | USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULOG, vna1, &
657 | & alpha0, betaQ, gammaQ, &
658 | & arel, crvmel, &
659 | & vx0, vny0, vz0, &
660 | & ZZ, WW, &
661 | & xiq, etq, Wq
662 |-----|
663 | IMPLICIT NONE
664 |-----|
665 | Variables
666 |-----|
667 | INTEGER, INTENT(IN) :: nelm !number of elements
668 | INTEGER, INTENT(IN) :: npts !number of points on the surface
669 | INTEGER, INTENT(IN) :: nint !order of triangle quadrature
670 | REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom !coordinates of the moments of the drop
671 | REAL (KIND = DBL), INTENT(OUT) :: area, v1m !area and volume of each element
672 | REAL (KIND = DBL), INTENT(OUT) :: cx, cy, cz !drop's centroid coordinates
673 |-----|
674 | Variables inside the subroutine
675 |-----|
676 | INTEGER :: i, k !counters
677 | INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
678 |-----|
679 | REAL (KIND = DBL) :: xi, eta !variables of weight to integrate over a triangle
680 | REAL (KIND = DBL) :: x, y, z !coordinates of the f(x,y,z)= F(xi,eta)
681 | REAL (KIND = DBL) :: Dx0xi, Dy0xi, Dz0xi !coordinates of the tangential vector over the xi axis
682 | REAL (KIND = DBL) :: Dx0et, Dy0et, Dz0et !coordinates of the tangential vector over the eta axis
683 | REAL (KIND = DBL) :: vnx, vny, vnz !normal vector coordinates of the element
684 | REAL (KIND = DBL) :: hs !surface metric on a triangle
685 | REAL (KIND = DBL) :: al, be, ga, alc, bec, gac !integration weigh coefficients
686 | REAL (KIND = DBL) :: cf !integration weigh coefficients
687 | REAL (KIND = DBL), DIMENSION(6) :: xxi, eet !variables of weight over in triangle (xi,eta)
688 | REAL (KIND = DBL), DIMENSION(6) :: D0Dx, Dy0x, Dz0x !tangential vector over xi axis in triangle (xi,eta)
689 | REAL (KIND = DBL), DIMENSION(6) :: D0De, Dy0e, Dz0e !tangential vector over eta axis in triangle (xi,eta)
690 | REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz !normal vector in triangle (xi,eta)
691 | REAL (KIND = DBL) :: bvx1, bvy1, bvz1, & !binormal vectors around in triangle (xi,eta)
692 | & bvx2, bvy2, bvz2, &
693 | & bvx3, bvy3, bvz3
694 | REAL (KIND = DBL) :: crvx, crvy, crvz !curvature
695 |-----|
696 | initialize
697 |-----|
698 |   area = 0.000
699 |   v1m = 0.000
700 |   cx = 0.000
701 |   cy = 0.000
702 |   cz = 0.000
703 |   arel = 0.000
704 |   xmom = 0.000
705 |   ymom = 0.000
706 |   zmom = 0.000
707 |   curmel = 0.000
708 |-----|
709 | OPEN (9,file="curmel.out")
710 |-----|
711 | Outer: DO k = 1, nelm
712 |-----|
713 |   i1 = n(k,1)
714 |   i2 = n(k,2)
715 |   i3 = n(k,3)
716 |   i4 = n(k,4)
717 |   i5 = n(k,5)
718 |   i6 = n(k,6)

```

```

719 al = alphaQ(k)
720 be = betaQ(k)
721 ga = gammaQ(k)
722 alc = 1.000e-01
723 bec = 1.000e-01
724 gac = 1.000e-01
725
726 ! Compute surface area and volume of the individual elements x, y, and z moments over each element total
727 !particle surface area and volume
728
729
730
731 DO i = 1, mint
732   xi = xli(i)
733   eta = etq(i)
734   CALL interp_p(p(11,1), p(11,2), p(11,3), &
735 & p(12,1), p(12,2), p(12,3), &
736 & p(13,1), p(13,2), p(13,3), &
737 & p(14,1), p(14,2), p(14,3), &
738 & p(15,1), p(15,2), p(15,3), &
739 & p(16,1), p(16,2), p(16,3), &
740 & al, be, ga, &
741 & xi, eta, &
742 & x, y, z, &
743 & DxDxi, DyDxi, DzDxi, &
744 & DxDet, DyDet, DzDet, &
745 & vmx, vny, vnz, &
746 & hs)
747   cf = hs*wg(i)
748   arel(k) = arel(k) + cf
749   xmom(k) = xmom(k) + cf*x
750   ymom(k) = ymom(k) + cf*y
751   zmom(k) = zmom(k) + cf*z
752   vlm = vlm + (x*vmx+y*vny+z*vnz)*cf
753 END DO
754
755 arel(k) = 0.500*arel(k)
756 xmom(k) = 0.500*xmom(k)
757 ymom(k) = 0.500*ymom(k)
758 zmom(k) = 0.500*zmom(k)
759 area = area + arel(k)
760 cx = cx + xmom(k)
761 cy = cy + ymom(k)
762 cz = cz + zmom(k)
763
764 ! compute the average value of the normal vector the mean curvature as a contour integral using the nifty
765 !formula (4.2.18) of Pozrikidis (1997)
766
767
768 xxi(1) = 0.000
769 eet(1) = 0.000
770 xxi(2) = 1.000
771 eet(2) = 0.000
772 xxi(3) = 0.000
773 eet(3) = 1.000
774 xxi(4) = al
775 eet(4) = 0.000
776 xxi(5) = ga
777 eet(5) = gac
778 xxi(6) = 0.000
779 eet(6) = be
780 DO i = 1, 6
781   xi = xxi(i)
782   eta = eet(i)
783   CALL interp_p(p(11,1), p(11,2), p(11,3), &
784 & p(12,1), p(12,2), p(12,3), &
785 & p(13,1), p(13,2), p(13,3), &
786 & p(14,1), p(14,2), p(14,3), &
787 & p(15,1), p(15,2), p(15,3), &
788 & p(16,1), p(16,2), p(16,3), &
789 & al, be, ga, &
790 & xi, eta, &

```

```

791 & x, y, z, &
792 & DxDx(i), DyDx(i), DzDx(i), &
793 & DxDy(i), DyDy(i), DzDy(i), &
794 & vx(i), vy(i), vz(i), &
795 & hs)
796 END DO
797
798 bvx1 = 0.000
799 bvy1 = 0.000
800 bvz1 = 0.000
801 bvx2 = 0.000
802 bvy2 = 0.000
803 bvz2 = 0.000
804 bvx3 = 0.000
805 bvy3 = 0.000
806 bvz3 = 0.000
807 crvx = 0.000
808 crvy = 0.000
809 crvz = 0.000
810
811 ! computation of curvature line integral along segment 1-4-2
812
813 bvx1 = vna1(4+(6*(i-1)),2)*DzDx(1)-vna1(4+(6*(i-1)),3)*DyDx(1)
814 bvy1 = vna1(1+(6*(i-1)),3)*DxDx(1)-vna1(1+(6*(i-1)),2)*DzDx(1)
815 bvz1 = vna1(1+(6*(i-1)),1)*DyDx(1)-vna1(1+(6*(i-1)),2)*DxDx(1)
816
817 |vna1(1+(6*(i-1)),1), vna1(1+(6*(i-1)),2), vna1(1+(6*(i-1)),3), &
818 |vna1(2+(6*(i-1)),1), vna1(2+(6*(i-1)),2), vna1(2+(6*(i-1)),3), &
819 |vna1(3+(6*(i-1)),1), vna1(3+(6*(i-1)),2), vna1(3+(6*(i-1)),3), &
820 |vna1(4+(6*(i-1)),1), vna1(4+(6*(i-1)),2), vna1(4+(6*(i-1)),3), &
821 |vna1(5+(6*(i-1)),1), vna1(5+(6*(i-1)),2), vna1(5+(6*(i-1)),3), &
822 |vna1(6+(6*(i-1)),1), vna1(6+(6*(i-1)),2), vna1(6+(6*(i-1)),3), &
823
824 bvx2 = vna1(4+(6*(i-1)),2)*DzDx(4)-vna1(4+(6*(i-1)),3)*DyDx(4)
825 bvy2 = vna1(4+(6*(i-1)),3)*DxDx(4)-vna1(4+(6*(i-1)),2)*DzDx(4)
826 bvz2 = vna1(4+(6*(i-1)),1)*DyDx(4)-vna1(4+(6*(i-1)),2)*DxDx(4)
827
828 bvx3 = vna1(2+(6*(i-1)),2)*DzDx(2)-vna1(2+(6*(i-1)),3)*DyDx(2)
829 bvy3 = vna1(2+(6*(i-1)),3)*DxDx(2)-vna1(2+(6*(i-1)),2)*DzDx(2)
830 bvz3 = vna1(2+(6*(i-1)),1)*DyDx(2)-vna1(2+(6*(i-1)),2)*DxDx(2)
831
832 crvx = al*bvy1 + bvx2 + alc*bvx3
833 crvy = al*bvz1 + bvy2 + alc*bvy3
834 crvz = al*bvz1 + bvz2 + alc*bvz3
835
836 ! computation of curvature line integral along segment 2-5-3
837
838 bvx1 = 0.000
839 bvy1 = 0.000
840 bvz1 = 0.000
841 bvx2 = 0.000
842 bvy2 = 0.000
843 bvz2 = 0.000
844 bvx3 = 0.000
845 bvy3 = 0.000
846 bvz3 = 0.000
847
848 bvx1 = vna1(2+(6*(i-1)),2)*DzDx(2)-vna1(2+(6*(i-1)),3)*DyDx(2)
849 bvy1 = vna1(2+(6*(i-1)),3)*DxDx(2)-vna1(2+(6*(i-1)),2)*DzDx(2)
850 bvz1 = vna1(2+(6*(i-1)),1)*DyDx(2)-vna1(2+(6*(i-1)),2)*DxDx(2)
851
852 bvx2 = vna1(5+(6*(i-1)),2)*DzDx(5)-vna1(5+(6*(i-1)),3)*DyDx(5)
853 bvy2 = vna1(5+(6*(i-1)),3)*DxDx(5)-vna1(5+(6*(i-1)),2)*DzDx(5)
854 bvz2 = vna1(5+(6*(i-1)),1)*DyDx(5)-vna1(5+(6*(i-1)),2)*DxDx(5)
855
856 bvx3 = vna1(3+(6*(i-1)),2)*DzDx(3)-vna1(3+(6*(i-1)),3)*DyDx(3)
857 bvy3 = vna1(3+(6*(i-1)),3)*DxDx(3)-vna1(3+(6*(i-1)),2)*DzDx(3)
858 bvz3 = vna1(3+(6*(i-1)),1)*DyDx(3)-vna1(3+(6*(i-1)),2)*DxDx(3)
859
860 crvx = crvx + gac*bvx1 - bvx2 - ga*bvx3
861 crvy = crvy + gac*bvy1 - bvy2 - ga*bvy3
862 crvz = crvz + gac*bvz1 - bvz2 - ga*bvz3

```

```

863 |-----
864 |   bx1 = vna1(2+(6*(1-1)),2)*DzDe(2)-vna1(2+(6*(1-1)),3)*DyDe(2)
865 |   by1 = vna1(2+(6*(1-1)),3)*DxDe(2)-vna1(2+(6*(1-1)),1)*DzDe(2)
866 |   bv1 = vna1(2+(6*(1-1)),1)*DyDe(2)-vna1(2+(6*(1-1)),2)*DxDe(2)
867 |-----
868 |   bx2 = vna1(5+(6*(1-1)),2)*DzDe(5)-vna1(5+(6*(1-1)),3)*DyDe(5)
869 |   by2 = vna1(5+(6*(1-1)),3)*DxDe(5)-vna1(5+(6*(1-1)),1)*DzDe(5)
870 |   bv2 = vna1(5+(6*(1-1)),1)*DyDe(5)-vna1(5+(6*(1-1)),2)*DxDe(5)
871 |-----
872 |   bx3 = vna1(3+(6*(1-1)),2)*DzDe(3)-vna1(3+(6*(1-1)),3)*DyDe(3)
873 |   by3 = vna1(3+(6*(1-1)),3)*DxDe(3)-vna1(3+(6*(1-1)),1)*DzDe(3)
874 |   bv3 = vna1(3+(6*(1-1)),1)*DyDe(3)-vna1(3+(6*(1-1)),2)*DxDe(3)
875 |-----
876 |   crvx = crvx + gac*bx1 + bx2 + ga*bx3
877 |   crvy = crvy + gac*by1 + by2 + ga*by3
878 |   crvz = crvz + gac*bv1 + bv2 + ga*bv3
879 |-----
880 | computation of curvature line integral along segment 3-6-1
881 |-----
882 |   bx1 = 0.000
883 |   by1 = 0.000
884 |   bv1 = 0.000
885 |   bx2 = 0.000
886 |   by2 = 0.000
887 |   bv2 = 0.000
888 |   bx3 = 0.000
889 |   bv3 = 0.000
890 |-----
891 |-----
892 |   bx1 = vna1(1+(6*(1-1)),2)*DzDe(1)-vna1(1+(6*(1-1)),3)*DyDe(1)
893 |   by1 = vna1(1+(6*(1-1)),3)*DxDe(1)-vna1(1+(6*(1-1)),1)*DzDe(1)
894 |   bv1 = vna1(1+(6*(1-1)),1)*DyDe(1)-vna1(1+(6*(1-1)),2)*DxDe(1)
895 |-----
896 |   bx2 = vna1(6+(6*(1-1)),2)*DzDe(6)-vna1(6+(6*(1-1)),3)*DyDe(6)
897 |   by2 = vna1(6+(6*(1-1)),3)*DxDe(6)-vna1(6+(6*(1-1)),1)*DzDe(6)
898 |   bv2 = vna1(6+(6*(1-1)),1)*DyDe(6)-vna1(6+(6*(1-1)),2)*DxDe(6)
899 |-----
900 |   bx3 = vna1(3+(6*(1-1)),2)*DzDe(3)-vna1(3+(6*(1-1)),3)*DyDe(3)
901 |   by3 = vna1(3+(6*(1-1)),3)*DxDe(3)-vna1(3+(6*(1-1)),1)*DzDe(3)
902 |   bv3 = vna1(3+(6*(1-1)),1)*DyDe(3)-vna1(3+(6*(1-1)),2)*DxDe(3)
903 |-----
904 |   crvx = crvx + be*bx1 - bv2 - bec*bx3
905 |   crvy = crvy + be*by1 - by2 - bec*by3
906 |   crvz = crvz + be*bv1 - bv2 - bec*bv3
907 |-----
908 | will project curvature vector onto the normal vector at the centroid only needs the colocation vectors
909 | In this value of curvature I made a division per 8 (1/8.0) when I expected do for 4 (1/4.0)
910 |-----
911 |   crvml(k) = 0.2500*(crvx*vx0(k)+crvy*vy0(k)+crvz*vz0(k)) /arel(k)
912 |   WRITE (9,*)   crvml(k)
913 |-----
914 | END DO Outer
915 |-----
916 | final computation of the surface-centroid and volume
917 |-----
918 |-----
919 |   cx = cx/area
920 |   cy = cy/area
921 |   cz = cz/area
922 |   vlm = vlm/6.000
923 | CLOSE (9)
924 103 FORMAT(10(1X,ES24.16))
925 |-----
926 | END SUBROUTINE elm_geom2
927 |-----
928 | SUBROUTINE elm_geom3(nelm, npts, mint, &
929 | & xmom, ymom, zmom, &
930 | & area, vlm, &
931 | & cx, cy, cz)
932 |-----
933 | This subroutine is a new version of Elm_Geo Subroutine.

```

```

934 |Compute:
935 | *The surface area of the individual elements x, y, and z moments over each element
936 | *the total particle surface area and volume
937 | Now, (25 / August / 2012) this subroutine was cut.
938 |-----
939 | USE Mod_Nodal_Interp
940 | USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog, &
941 | & alphaQ, betaQ, gammaQ, &
942 | & arel, crvml, &
943 | & vx0, vy0, vz0, &
944 | & ZZ, WW, &
945 | & xiq, etq, wq
946 |-----
947 | IMPLICIT NONE
948 |-----
949 | Variables
950 |-----
951 | INTEGER, INTENT(IN) :: nelm !number of elements
952 | INTEGER, INTENT(IN) :: npts !number of points on the surface
953 | INTEGER, INTENT(IN) :: mint !order of triangle quadrature
954 | REAL (KIND = DBL), DIMENSION(:), INTENT(OUT) :: xmom, ymom, zmom !coordinates of the moments of the drop
955 | REAL (KIND = DBL), INTENT(OUT) :: area, vlm !area and volume of each element
956 | REAL (KIND = DBL), INTENT(OUT) :: cx, cy, cz !drop's centroid coordinates
957 |-----
958 | Variables inside the subroutine
959 |-----
960 | INTEGER :: i, k !counters
961 | INTEGER :: i1, i2, i3, i4, i5, i6 !Indices to obtain node numbers from each element
962 |-----
963 | REAL (KIND = DBL) :: xi, eta !variables of weigh to integrate over a triangle
964 | REAL (KIND = DBL) :: x, y, z !coordinates of the f(x,y,z)= F(xi,eta)
965 | REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi !coordinates of the tangential vector over the xi axis
966 | REAL (KIND = DBL) :: DxDeta, DyDeta, DzDeta !coordinates of the tangential vector over the eta axis
967 | REAL (KIND = DBL) :: vx, vy, vz !normal vector coordinates of the element
968 | REAL (KIND = DBL) :: hs, xs, es !surface metric on a triangle
969 | REAL (KIND = DBL) :: al, be, ga, alc, bec, gac !integration weigh coefficients
970 | REAL (KIND = DBL) :: cf, fill !integration weigh coefficients
971 | REAL (KIND = DBL), DIMENSION(6) :: xxi, eet !variables of weigh over in triangle (xi,eta)
972 | REAL (KIND = DBL), DIMENSION(6) :: DxDx, DyDx, DzDx !tangential vector over xi axis in triangle (xi,eta)
973 | REAL (KIND = DBL), DIMENSION(6) :: DxDeta, DyDeta, DzDeta !tangential vector over eta axis in triangle (xi,eta)
974 | REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz !normal vector in triangle (xi,eta)
975 | REAL (KIND = DBL) :: bv1, bv2, bv3, & !binormal vectors around in triangle (xi,eta)
976 | & bx2, by2, bv2, &
977 | & bx3, by3, bv3
978 | REAL (KIND = DBL) :: crvx, crvy, crvz !curvature
979 |-----
980 | Initialize
981 |-----
982 | area = 0.000
983 | vlm = 0.000
984 | cx = 0.000
985 | cy = 0.000
986 | cz = 0.000
987 | arel = 0.000
988 | xmom = 0.000
989 | ymom = 0.000
990 | zmom = 0.000
991 | crvml = 0.000
992 | fill = 0.000
993 |-----
994 | OPEN (9,file="curvml.out")
995 |-----
996 | Outer: DO k = 1, nelm
997 |-----
998 |   i1 = n(k,1)
999 |   i2 = n(k,2)
1000 |   i3 = n(k,3)
1001 |   i4 = n(k,4)
1002 |   i5 = n(k,5)
1003 |   i6 = n(k,6)
1004 |   al = alphaQ(k)
1005 |   be = betaQ(k)

```

```

1006 ga = gammaQ(k)
1007 alc = 1.000-alc
1008 bec = 1.000-be
1009 gac = 1.000-ga
1010
1011 ! Compute surface area and volume of the individual elements x, y, and z moments over each element total
1012 !particle surface area and volume
1013
1014
1015
1016 DO i = 1, mint
1017   xi = xiq(i)
1018   eta = etq(i)
1019   CALL interp_p(p(i1,1), p(i1,2), p(i1,3), &
1020     & p(i2,1), p(i2,2), p(i2,3), &
1021     & p(i3,1), p(i3,2), p(i3,3), &
1022     & p(i4,1), p(i4,2), p(i4,3), &
1023     & p(i5,1), p(i5,2), p(i5,3), &
1024     & p(i6,1), p(i6,2), p(i6,3), &
1025     & al, be, ga, &
1026     & xi, eta, z, &
1027     & x, y, z, &
1028     & DxDxi, DyDxi, DzDxi, &
1029     & DxDet, DyDet, DzDet, &
1030     & vnx, vny, vnz, &
1031     & hs)
1032   cf = hs*wq(i)
1033   arel(k) = arel(k) + cf
1034   xmom(k) = xmom(k) + cf*x
1035   ymom(k) = ymom(k) + cf*y
1036   zmom(k) = zmom(k) + cf*z
1037   vlm = vlm + (x*vnx+y*vny+z*vnz)*cf
1038   cf=0.000
1039 END DO
1040
1041
1042 arel(k) = 0.500*arel(k)
1043 xmom(k) = 0.500*xmom(k)
1044 ymom(k) = 0.500*ymom(k)
1045 zmom(k) = 0.500*zmom(k)
1046 area = area + arel(k)
1047 cx = cx + xmom(k)
1048 cy = cy + ymom(k)
1049 cz = cz + zmom(k)
1050
1051 ! compute the average value of the normal vector the mean curvature as a contour integral using the nftty
1052 ! formula (4.2.18) of Pozrikidis (1997)
1053
1054 xxi(1) = 0.000
1055 eet(1) = 0.000
1056 xxi(2) = 1.000
1057 eet(2) = 0.000
1058 xxi(3) = 0.000
1059 eet(3) = 1.000
1060 xxi(4) = al
1061 eet(4) = 0.000
1062 xxi(5) = ga
1063 eet(5) = gac
1064 xxi(6) = 0.000
1065 eet(6) = be
1066 DO i = 1, 6
1067   xi = xxi(i)
1068   eta = eet(i)
1069   CALL interp_p3(p(i1,1), p(i1,2), p(i1,3), &
1070     & p(i2,1), p(i2,2), p(i2,3), &
1071     & p(i3,1), p(i3,2), p(i3,3), &
1072     & p(i4,1), p(i4,2), p(i4,3), &
1073     & p(i5,1), p(i5,2), p(i5,3), &
1074     & p(i6,1), p(i6,2), p(i6,3), &
1075     & al, be, ga, &
1076     & xi, eta, z, &
1077     & x, y, z, &

```

```

1078     & DxDx(1), DyDx(1), DzDx(1), &
1079     & DxDe(1), DyDe(1), DzDe(1), &
1080     & vx(1), vy(1), vz(1), &
1081     & hs)
1082 END DO
1083
1084 bvx1 = 0.000
1085 bvy1 = 0.000
1086 bvz1 = 0.000
1087 bvx2 = 0.000
1088 bvy2 = 0.000
1089 bvz2 = 0.000
1090 bvx3 = 0.000
1091 bvy3 = 0.000
1092 bvz3 = 0.000
1093 crvx = 0.000
1094 crvy = 0.000
1095 crvz = 0.000
1096
1097 ! computation of curvature line integral along segment 1-4-2
1098
1099 bvx1 = vy(1)*DzDx(1)-vz(1)*DyDx(1)
1100 bvy1 = vz(1)*DxDx(1)-vx(1)*DzDx(1)
1101 bvz1 = vx(1)*DyDx(1)-vy(1)*DxDx(1)
1102
1103 bvx2 = vy(4)*DzDx(4)-vz(4)*DyDx(4)
1104 bvy2 = vz(4)*DxDx(4)-vx(4)*DzDx(4)
1105 bvz2 = vx(4)*DyDx(4)-vy(4)*DxDx(4)
1106
1107 bvx3 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
1108 bvy3 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
1109 bvz3 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
1110
1111 crvx = al*bvx1 + bvx2 + alc*bvx3
1112 crvy = al*bvy1 + bvy2 + alc*bvy3
1113 crvz = al*bvz1 + bvz2 + alc*bvz3
1114
1115 ! computation of curvature line integral along segment 2-5-3
1116
1117 bvx1 = 0.000
1118 bvy1 = 0.000
1119 bvz1 = 0.000
1120 bvx2 = 0.000
1121 bvy2 = 0.000
1122 bvz2 = 0.000
1123 bvx3 = 0.000
1124 bvy3 = 0.000
1125 bvz3 = 0.000
1126
1127 bvx1 = vy(2)*DzDx(2)-vz(2)*DyDx(2)
1128 bvy1 = vz(2)*DxDx(2)-vx(2)*DzDx(2)
1129 bvz1 = vx(2)*DyDx(2)-vy(2)*DxDx(2)
1130
1131 bvx2 = vy(5)*DzDx(5)-vz(5)*DyDx(5)
1132 bvy2 = vz(5)*DxDx(5)-vx(5)*DzDx(5)
1133 bvz2 = vx(5)*DyDx(5)-vy(5)*DxDx(5)
1134
1135 bvx3 = vy(3)*DzDx(3)-vz(3)*DyDx(3)
1136 bvy3 = vz(3)*DxDx(3)-vx(3)*DzDx(3)
1137 bvz3 = vx(3)*DyDx(3)-vy(3)*DxDx(3)
1138
1139 crvx = crvx - gac*bvx1 - bvx2 - ga*bvx3
1140 crvy = crvy - gac*bvy1 - bvy2 - ga*bvy3
1141 crvz = crvz - gac*bvz1 - bvz2 - ga*bvz3
1142
1143 bvx1 = vy(2)*DzDe(2)-vz(2)*DyDe(2)
1144 bvy1 = vz(2)*DxDx(2)-vx(2)*DzDe(2)
1145 bvz1 = vx(2)*DyDe(2)-vy(2)*DxDx(2)
1146
1147 bvx2 = vy(5)*DzDe(5)-vz(5)*DyDe(5)
1148 bvy2 = vz(5)*DxDx(5)-vx(5)*DzDe(5)
1149 bvz2 = vx(5)*DyDe(5)-vy(5)*DxDx(5)

```

```
1150 |-----|
1151 | bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
1152 | bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
1153 | bvx3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
1154 |-----|
1155 | crvx = crvx + gac*bvx1 + bvx2 + ga*bvx3
1156 | crvy = crvy + gac*bvy1 + bvy2 + ga*bvy3
1157 | crvz = crvz + gac*bvz1 + bvz2 + ga*bvz3
1158 |-----|
1159 | computation of curvature line integral along segment 3-6-1
1160 |-----|
1161 | bvx1 = 0.000
1162 | bvy1 = 0.000
1163 | bvz1 = 0.000
1164 | bvx2 = 0.000
1165 | bvy2 = 0.000
1166 | bvz2 = 0.000
1167 | bvx3 = 0.000
1168 | bvy3 = 0.000
1169 | bvz3 = 0.000
1170 |-----|
1171 | bvx1 = vy(1)*DzDe(1)-vz(1)*DyDe(1)
1172 | bvy1 = vz(1)*DxDe(1)-vx(1)*DzDe(1)
1173 | bvz1 = vx(1)*DyDe(1)-vy(1)*DxDe(1)
1174 |-----|
1175 | bvx2 = vy(6)*DzDe(6)-vz(6)*DyDe(6)
1176 | bvy2 = vz(6)*DxDe(6)-vx(6)*DzDe(6)
1177 | bvz2 = vx(6)*DyDe(6)-vy(6)*DxDe(6)
1178 |-----|
1179 | bvx3 = vy(3)*DzDe(3)-vz(3)*DyDe(3)
1180 | bvy3 = vz(3)*DxDe(3)-vx(3)*DzDe(3)
1181 | bvz3 = vx(3)*DyDe(3)-vy(3)*DxDe(3)
1182 |-----|
1183 | crvx = crvx - be*bvx1 - bvx2 - bec*bvx3
1184 | crvy = crvy - be*bvy1 - bvy2 - bec*bvy3
1185 | crvz = crvz - be*bvz1 - bvz2 - bec*bvz3
1186 |-----|
1187 | fil = DSQR((crrx**2+crry**2+crrz**2)
1188 |!!!-----|
1189 | crvx = crvx/fil
1190 | crvy = crvy/fil
1191 | crvz = crvz/fil
1192 |-----|
1193 | will project curvature vector onto the normal vector at the centroid only needs the colocation vectors
1194 | In this value of curvature I made a division per 8 (1/8.0) when I expected do for 4 (1/4.0)
1195 |-----|
1196 | crvmel(k) = 0.2500*(crrx*vnz0(k)+crry*vny0(k)+crrz*vnz0(k))/arel(k)
1197 | WRITE (9,*) crvmel(k)
1198 |
1199 | END DO Outer
1200 |-----|
1201 | final computation of the surface-centroid and volume
1202 |-----|
1203 |-----|
1204 | cx = cx/area
1205 | cy = cy/area
1206 | cz = cz/area
1207 | vlm = vlm/6.000
1208 | CLOSE (9)
1209 |103 FORMAT(10(1x,ES24.16))
1210 |-----|
1211 | END SUBROUTINE elm_geom3
1212 |-----|
1213 | END MODULE Mod_Prtcl_3D_Geo
1214 |-----|
```

D:\Darth Vader\Escritorio\prtcl mkl\Mod\_Nodal\_Interp.f90 1

```
1 MODULE Mod_Nodal_Interp
2 =====
3 |Version 0.9      10 / August / 2012
4 |Version 1.0     14 / 11 / 2012
5 |
6 |----- Alfredo Sanjuan Sanjuan, in other words, me :)|
7 CONTAINS
8 =====
9 SUBROUTINE interp_p( x1, y1, z1, &
10 & x2, y2, z2, &
11 & x3, y3, z3, &
12 & x4, y4, z4, &
13 & x5, y5, z5, &
14 & x6, y6, z6, &
15 & al, be, ga, &
16 & xi, eta, &
17 & x, y, z, &
18 & DxDet, DyDet, DzDet, &
19 & DxDet, DyDet, DzDet, &
20 & vnx, vny, vnz, &
21 & hs )
22 =====
23 | This subroutine interpolate over an element to compute geometrical variables including the following:
24 | 1) Position vector
25 | 2) Tangential vectors in the xi and eta directions
26 | 3) Unit normal vector
27 | 4) Area and volume of each element
28 =====
29 USE Mod_SharedVars, ONLY: DBL
30 =====
31 IMPLICIT NONE
32 =====
33 | Variables
34 REAL (KIND = DBL), INTENT(IN) :: x1, y1, z1, & !coordinates of first node of the element
35 & x2, y2, z2, & !coordinates of second node of the element
36 & x3, y3, z3, & !coordinates of third node of the element
37 & x4, y4, z4, & !coordinates of fourth node of the element
38 & x5, y5, z5, & !coordinates of fifth node of the element
39 & x6, y6, z6, & !coordinates of sixth node of the element
40 REAL (KIND = DBL), INTENT(IN) :: al, be, ga !constants alpha, beta and gamma
41 REAL (KIND = DBL), INTENT(IN) :: xi, eta !variables to integrate over a triangle
42 REAL (KIND = DBL), INTENT(OUT) :: x, y, z !coordinates of the f(x,y,z)=F(xi,eta)
43 REAL (KIND = DBL), INTENT(OUT) :: DxDet, DyDet, DzDet, & !Derivates of the tangential vectors over the
44 & DxDet, DyDet, DzDet !element
45 REAL (KIND = DBL), INTENT(OUT) :: vnx, vny, vnz !normal vector of the element
46 REAL (KIND = DBL), INTENT(OUT) :: hs !surface metric
47 =====
48 | Variables inside the subroutine
49 =====
50 REAL (KIND = DBL) :: alc, bec, gac, xs, es, hhs !complements of alpha, beta and gamma
51 REAL (KIND = DBL) :: alalc, bebec, gagac !other constants of alpha, beta and gamma
52 REAL (KIND = DBL) :: phi1, phi2, phi3, phi4, phi5, phi6 !components phi(xi,eta)
53 REAL (KIND = DBL) :: dphi1, dphi2, dphi3, dphi4, dphi5, dphi6 !derivate of phi(xi,eta) wrt xi
54 REAL (KIND = DBL) :: pphi1, pphi2, pphi3, pphi4, pphi5, pphi6 !derivate of phi(xi,eta) wrt eta
55 =====
56 | Prepare the constants of this subroutine.
57 |-----
58 alc = 1.000-al
59 bec = 1.000-be
60 gac = 1.000-ga
61 alalc = al*alc
62 bebec = be*bec
63 gagac = ga*gac
64 |-----
65 | In this part, the subroutine evaluates basis functions, it obtains the tangential and normal vectors.
66 |This is based on Numerical Computation in Science and Engineering, C.Pozrikidis, 1998, pp.305-312
67 |-----
68 phi2 = xi*(xi - al + eta*(al - ga)/gac)/alc
69 phi3 = eta*(eta - be + xi*(be + ga - 1.000)/ga)/bec
70 phi4 = xi*(1.000 - xi - eta)/alalc
71 phi5 = xi*eta/gagac
```

D:\Darth Vader\Escritorio\prtcl mkl\Mod\_Nodal\_Interp.f90 2

```
72 phi6 = eta*(1.000 - xi - eta)/bebec
73 phi1 = 1.000-phi2-phi3-phi4-phi5-phi6
74 |-----
75 | Interpolate the position vector (x, y, z)
76 |-----
77 x = xi*phi1 + x2*phi2 + x3*phi3 + x4*phi4 + x5*phi5 + x6*phi6
78 y = y1*phi1 + y2*phi2 + y3*phi3 + y4*phi4 + y5*phi5 + y6*phi6
79 z = z1*phi1 + z2*phi2 + z3*phi3 + z4*phi4 + z5*phi5 + z6*phi6
80 |-----
81 | In this part suroutine evaluates xi derivatives of basis functions
82 |-----
83 dphi2 = (2.000*xi - al + eta*(al - ga)/gac)/alc
84 dphi3 = eta*(be + ga - 1.000)/(ga*bec)
85 dphi4 = (1.000 - 2.000*xi - eta)/alalc
86 dphi5 = eta/gagac
87 dphi6 = -eta/bebec
88 dph1 = -dphi2 - dphi3 - dphi4 - dphi5 - dphi6
89 |-----
90 | Compute dx/dxi from xi derivatives of phi
91 |-----
92 DxDet = x1*dphi1 + x2*dphi2 + x3*dphi3 + x4*dphi4 + x5*dphi5 + x6*dphi6
93 DyDet = y1*dphi1 + y2*dphi2 + y3*dphi3 + y4*dphi4 + y5*dphi5 + y6*dphi6
94 DzDet = z1*dphi1 + z2*dphi2 + z3*dphi3 + z4*dphi4 + z5*dphi5 + z6*dphi6
95 es = DSQRT(DxDet*DxDet + DyDet*DyDet + DzDet*DzDet)
96 |-----
97 | Normalization of normal vector
98 |-----
99 DxDet = DxDet/es
100 DyDet = DyDet/es
101 DzDet = DzDet/es
102 |-----
103 | In this part suroutine evaluates eta derivatives of basis functions
104 |-----
105 pphi2 = xi*(al - ga)/(alc*gac)
106 pphi3 = (2.000*eta - be + xi*(be + ga - 1.000)/ga)/bec
107 pphi4 = -xi/alalc
108 pphi5 = xi/gagac
109 pphi6 = (1.000 - xi - 2.000*eta)/bebec
110 pph1 = -pphi2 - pphi3 - pphi4 - pphi5 - pphi6
111 |-----
112 | Compute dx/deta from eta derivatives of phi
113 |-----
114 DxDet = x1*pphi1 + x2*pphi2 + x3*pphi3 + x4*pphi4 + x5*pphi5 + x6*pphi6
115 DyDet = y1*pphi1 + y2*pphi2 + y3*pphi3 + y4*pphi4 + y5*pphi5 + y6*pphi6
116 DzDet = z1*pphi1 + z2*pphi2 + z3*pphi3 + z4*pphi4 + z5*pphi5 + z6*pphi6
117 es = DSQRT(DxDet*DxDet + DyDet*DyDet + DzDet*DzDet)
118 |-----
119 | Normalization of normal vector
120 |-----
121 DxDet = DxDet/es
122 DyDet = DyDet/es
123 DzDet = DzDet/es
124 |-----
125 |-----
126 | Normal vector vn = (DxDet)*(DxDet)
127 | Surface metric hs = norm(vn)
128 |-----
129 vnx = DyDet * DzDet - DyDet * DzDet
130 vny = DzDet * DxDet - DzDet * DxDet
131 vnz = DxDet * DyDet - DxDet * DyDet
132 hhs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
133 |-----
134 | Normalization of normal vector
135 |-----
136 vnx = vnx/hhs
137 vny = vny/hhs
138 vnz = vnz/hhs
139 hs = es*xs/hhs
140 |-----
141 END SUBROUTINE interp_p
```



```

142 !=====
143 ! Subroutine INTERP_E is a new method to obtain the normal vectors on every node on dorp's surface. The
144 ! name has the end _E which is an abbreviation of element. Firstly, this decision aids to differentiate this
145 ! method of subroutine INTERP_P. Second idea was to continue with this notation on manner to name subroutines to
146 ! have similar environment names. Finally INTERP_E is very important because this subroutine is fundamental to
147 ! advanced the dynamic of dorp deformation in the time. Therefore, this subroutine is in this new MODULE
148 !=====
149 SUBROUTINE interp_e( npts, ne, nbe,&
150 & vx0, vny0, vnz0,&
151 & vna)
152 !=====
153 ! This subroutine computes the normal vector of every node on the surface
154 !=====
155 USE Mod_SharedVars, ONLY: DBL, p, x0, y0, z0
156 !=====
157 IMPLICIT NONE
158 !=====
159 ! Variables
160 !=====
161 INTEGER, INTENT(IN) :: npts !number of nodes
162 INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: ne !table of connectivities per node
163 INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: nbe !table of connectivities per element
164 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: vx0, vny0, vnz0 !arrays of the components of
165 !normal vectors over collocation points
166 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(OUT) :: vna !array whit the values of unit normal
167 !vector of every node over the surface
168 !=====
169 ! Variables inside the subroutine
170 !=====
171 INTEGER :: i, j !counters
172 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: norma
173 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: dist1,dist2, absdist
174 !=====
175 ! Initialize
176 ALLOCATE(vna(npts,3), norma(npts), dist1(npts),dist2(npts),absdist(npts))
177 vna = 0.000
178 norma = 0.000
179 dist1 = 0.000
180 dist2 = 0.000
181 absdist = 0.000
182 !=====
183 !=====
184 WHERE(ne(:,1) == 4)
185 vna(:,1) = (vx0(ne(:,2)) + vx0(ne(:,3)) + vx0(ne(:,4)) + vx0(ne(:,5)))/4.000
186 vna(:,2) = (vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(ne(:,4)) + vny0(ne(:,5)))/4.000
187 vna(:,3) = (vnz0(ne(:,2)) + vnz0(ne(:,3)) + vnz0(ne(:,4)) + vnz0(ne(:,5)))/4.000
188 END WHERE
189 !=====
190 !=====
191 !=====
192 !=====
193 !=====
194 !=====
195 !=====
196 !=====
197 WHERE(ne(:,1) == 6)
198 vna(:,1) = (vx0(ne(:,2)) + vx0(ne(:,3)) + vx0(ne(:,4)) + vx0(ne(:,5)) +
199 & vx0(ne(:,6)) + vx0(ne(:,7)))/6.000
200 vna(:,2) = (vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(ne(:,4)) + vny0(ne(:,5)) +
201 & vny0(ne(:,6)) + vny0(ne(:,7)))/6.000
202 vna(:,3) = (vnz0(ne(:,2)) + vnz0(ne(:,3)) + vnz0(ne(:,4)) + vnz0(ne(:,5)) +
203 & vnz0(ne(:,6)) + vnz0(ne(:,7)))/6.000
204 END WHERE
205 !=====
206 !=====
207 !=====
208 !=====
209 !=====
210 !=====
211 !=====
212 !=====
213 WHERE(ne(:,1) == 2)

```

```

214 !=====
215 !=====
216 !=====
217 !=====
218 !=====
219 !=====
220 !=====
221 !=====
222 !=====
223 !=====
224 !=====
225 !=====
226 !=====
227 !=====
228 !=====
229 !=====
230 !=====
231 !=====
232 !=====
233 !=====
234 !=====
235 !=====
236 !=====
237 !=====
238 !=====
239 !=====
240 !=====
241 !=====
242 !=====
243 !=====
244 !=====
245 !! dist1(:) = DSQRT((p(:,1)-x0(ne(:,2)))**2 + (p(:,2)-y0(ne(:,3)))**2 + (p(:,3)-z0(ne(:,2)))**2)
246 !! dist2(:) = DSQRT((p(:,1)-x0(ne(:,3)))**2 + (p(:,2)-y0(ne(:,3)))**2 + (p(:,3)-z0(ne(:,3)))**2)
247 !!
248 !! absdist(:) = DABS(1.000/dist1(:)) + (1.000/dist2(:))
249 !=====
250 !=====
251 !=====
252 !=====
253 !=====
254 END WHERE
255 !=====
256 !=====
257 !=====
258 !=====
259 !=====
260 !=====
261 !=====
262 ! This subroutine computes the normal vector of every node on the surface
263 !=====
264 USE Mod_SharedVars, ONLY: DBL, p, x0, y0, z0
265 !=====
266 IMPLICIT NONE
267 !=====
268 ! Variables
269 !=====
270 INTEGER, INTENT(IN) :: npts !number of nodes
271 INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: ne !table of connectivities per node
272 INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: nbe !table of connectivities per element
273 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: vx0, vny0, vnz0 !arrays of the components of
274 !normal vectors over collocation points
275 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(OUT) :: vna !array whit the values of unit normal
276 !vector of every node over the surface

```

```

277 !-----
278 ! Variables inside the subroutine
279 !-----
280 INTEGER :: i, j !Counters
281 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: norma
282 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: dist1,dist2, absdist
283 !-----
284 ! Initialize
285 ALLOCATE(vna(npts,3), norma(npts), dist1(npts),dist2(npts),absdist(npts))
286 vna = 0.000
287 norma = 0.000
288 dist1 = 0.000
289 dist2 = 0.000
290 absdist=0.000
291 !-----
292 WHERE(ne(:,1)==4)
293 vna(:,1)=(vx0(ne(:,2)) + vx0(ne(:,3)) + vx0(ne(:,4)) + vx0(ne(:,5)))/4.000
294 vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(ne(:,4)) + vny0(ne(:,5)))/4.000
295 vna(:,3)=(vz0(ne(:,2)) + vz0(ne(:,3)) + vz0(ne(:,4)) + vz0(ne(:,5)))/4.000
296 !-----
297 !vna(:,1)=(vx0(ne(:,2)) + vx0(ne(:,3)) + vx0(ne(:,4)) + vx0(ne(:,5)))&
298 !& + vx0(nbe(ne(:,2),1)) + vx0(nbe(ne(:,2),2)) + vx0(nbe(ne(:,2),3)) &
299 !& + vx0(nbe(ne(:,3),1)) + vx0(nbe(ne(:,3),2)) + vx0(nbe(ne(:,3),3)) &
300 !& + vx0(nbe(ne(:,4),1)) + vx0(nbe(ne(:,4),2)) + vx0(nbe(ne(:,4),3)) &
301 !& + vx0(nbe(ne(:,5),1)) + vx0(nbe(ne(:,5),2)) + vx0(nbe(ne(:,5),3)))/16.000
302 !vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(ne(:,4)) + vny0(ne(:,5)))&
303 !& + vny0(nbe(ne(:,2),1)) + vny0(nbe(ne(:,2),2)) + vny0(nbe(ne(:,2),3)) &
304 !& + vny0(nbe(ne(:,3),1)) + vny0(nbe(ne(:,3),2)) + vny0(nbe(ne(:,3),3)) &
305 !& + vny0(nbe(ne(:,4),1)) + vny0(nbe(ne(:,4),2)) + vny0(nbe(ne(:,4),3)) &
306 !& + vny0(nbe(ne(:,5),1)) + vny0(nbe(ne(:,5),2)) + vny0(nbe(ne(:,5),3)))/16.000
307 !vna(:,3)=(vz0(ne(:,2)) + vz0(ne(:,3)) + vz0(ne(:,4)) + vz0(ne(:,5)))&
308 !& + vz0(nbe(ne(:,2),1)) + vz0(nbe(ne(:,2),2)) + vz0(nbe(ne(:,2),3)) &
309 !& + vz0(nbe(ne(:,3),1)) + vz0(nbe(ne(:,3),2)) + vz0(nbe(ne(:,3),3)) &
310 !& + vz0(nbe(ne(:,4),1)) + vz0(nbe(ne(:,4),2)) + vz0(nbe(ne(:,4),3)) &
311 !& + vz0(nbe(ne(:,5),1)) + vz0(nbe(ne(:,5),2)) + vz0(nbe(ne(:,5),3)))/16.000
312 !-----
313 !vna(:,1)=(vx0(ne(:,2)) + vx0(ne(:,3)) + vx0(ne(:,4)) + vx0(ne(:,5)))&
314 !& + 3.000*(vx0(nbe(ne(:,2),1)) + vx0(nbe(ne(:,2),2)) + vx0(nbe(ne(:,2),3))) &
315 !& + 3.000*(vx0(nbe(ne(:,3),1)) + vx0(nbe(ne(:,3),2)) + vx0(nbe(ne(:,3),3))) &
316 !& + 3.000*(vx0(nbe(ne(:,4),1)) + vx0(nbe(ne(:,4),2)) + vx0(nbe(ne(:,4),3))) &
317 !& + 3.000*(vx0(nbe(ne(:,5),1)) + vx0(nbe(ne(:,5),2)) + vx0(nbe(ne(:,5),3)))/40.000
318 !vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(ne(:,4)) + vny0(ne(:,5)))&
319 !& + 3.000*(vny0(nbe(ne(:,2),1)) + vny0(nbe(ne(:,2),2)) + vny0(nbe(ne(:,2),3))) &
320 !& + 3.000*(vny0(nbe(ne(:,3),1)) + vny0(nbe(ne(:,3),2)) + vny0(nbe(ne(:,3),3))) &
321 !& + 3.000*(vny0(nbe(ne(:,4),1)) + vny0(nbe(ne(:,4),2)) + vny0(nbe(ne(:,4),3))) &
322 !& + 3.000*(vny0(nbe(ne(:,5),1)) + vny0(nbe(ne(:,5),2)) + vny0(nbe(ne(:,5),3)))/40.000
323 !vna(:,3)=(vz0(ne(:,2)) + vz0(ne(:,3)) + vz0(ne(:,4)) + vz0(ne(:,5)))&
324 !& + 3.000*(vz0(nbe(ne(:,2),1)) + vz0(nbe(ne(:,2),2)) + vz0(nbe(ne(:,2),3))) &
325 !& + 3.000*(vz0(nbe(ne(:,3),1)) + vz0(nbe(ne(:,3),2)) + vz0(nbe(ne(:,3),3))) &
326 !& + 3.000*(vz0(nbe(ne(:,4),1)) + vz0(nbe(ne(:,4),2)) + vz0(nbe(ne(:,4),3))) &
327 !& + 3.000*(vz0(nbe(ne(:,5),1)) + vz0(nbe(ne(:,5),2)) + vz0(nbe(ne(:,5),3)))/40.000
328 !-----
329 ! END WHERE
330 !-----
331 WHERE(ne(:,1) == 6)
332 vna(:,1)=(vx0(ne(:,2)) + vx0(ne(:,3)) + vx0(ne(:,4)) + vx0(ne(:,5)) &
333 & + vx0(ne(:,6)) + vx0(ne(:,7)))/6.000
334 vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(ne(:,4)) + vny0(ne(:,5)) &
335 & + vny0(ne(:,6)) + vny0(ne(:,7)))/6.000
336 vna(:,3)=(vz0(ne(:,2)) + vz0(ne(:,3)) + vz0(ne(:,4)) + vz0(ne(:,5)) &
337 & + vz0(ne(:,6)) + vz0(ne(:,7)))/6.000
338 !-----
339 !vna(:,1)=(vx0(ne(:,2)) + vx0(ne(:,3)) + vx0(ne(:,4)) + vx0(ne(:,5)) &
340 !& + vx0(ne(:,6)) + vx0(ne(:,7))) &
341 !& + vx0(nbe(ne(:,2),1)) + vx0(nbe(ne(:,2),2)) + vx0(nbe(ne(:,2),3)) &
342 !& + vx0(nbe(ne(:,3),1)) + vx0(nbe(ne(:,3),2)) + vx0(nbe(ne(:,3),3)) &
343 !& + vx0(nbe(ne(:,4),1)) + vx0(nbe(ne(:,4),2)) + vx0(nbe(ne(:,4),3)) &
344 !& + vx0(nbe(ne(:,5),1)) + vx0(nbe(ne(:,5),2)) + vx0(nbe(ne(:,5),3)) &
345 !& + vx0(nbe(ne(:,6),1)) + vx0(nbe(ne(:,6),2)) + vx0(nbe(ne(:,6),3)) &
346 !& + vx0(nbe(ne(:,7),1)) + vx0(nbe(ne(:,7),2)) + vx0(nbe(ne(:,7),3)))/24.000
347 !
348 !vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(ne(:,4)) + vny0(ne(:,5)) &
349 !& + vny0(ne(:,6)) + vny0(ne(:,7))) &

```

```

349 !& + vny0(nbe(ne(:,2),1)) + vny0(nbe(ne(:,2),2)) + vny0(nbe(ne(:,2),3)) &
350 !& + vny0(nbe(ne(:,3),1)) + vny0(nbe(ne(:,3),2)) + vny0(nbe(ne(:,3),3)) &
351 !& + vny0(nbe(ne(:,4),1)) + vny0(nbe(ne(:,4),2)) + vny0(nbe(ne(:,4),3)) &
352 !& + vny0(nbe(ne(:,5),1)) + vny0(nbe(ne(:,5),2)) + vny0(nbe(ne(:,5),3)) &
353 !& + vny0(nbe(ne(:,6),1)) + vny0(nbe(ne(:,6),2)) + vny0(nbe(ne(:,6),3)) &
354 !& + vny0(nbe(ne(:,7),1)) + vny0(nbe(ne(:,7),2)) + vny0(nbe(ne(:,7),3)))/24.000
355 !
356 !vna(:,3)=(vz0(ne(:,2)) + vz0(ne(:,3)) + vz0(ne(:,4)) + vz0(ne(:,5)) &
357 !& + vz0(ne(:,6)) + vz0(ne(:,7))) &
358 !& + vz0(nbe(ne(:,2),1)) + vz0(nbe(ne(:,2),2)) + vz0(nbe(ne(:,2),3)) &
359 !& + vz0(nbe(ne(:,3),1)) + vz0(nbe(ne(:,3),2)) + vz0(nbe(ne(:,3),3)) &
360 !& + vz0(nbe(ne(:,4),1)) + vz0(nbe(ne(:,4),2)) + vz0(nbe(ne(:,4),3)) &
361 !& + vz0(nbe(ne(:,5),1)) + vz0(nbe(ne(:,5),2)) + vz0(nbe(ne(:,5),3)) &
362 !& + vz0(nbe(ne(:,6),1)) + vz0(nbe(ne(:,6),2)) + vz0(nbe(ne(:,6),3)) &
363 !& + vz0(nbe(ne(:,7),1)) + vz0(nbe(ne(:,7),2)) + vz0(nbe(ne(:,7),3)))/24.000
364 ! END WHERE
365 !-----
366 ! WHERE (ne(:,1)==2)
367 !vna(:,1)=(vx0(ne(:,2)) + vx0(ne(:,3)) + vx0(nbe(ne(:,2),1)) + vx0(nbe(ne(:,2),2)) + vx0(nbe(ne(:,2),3)) &
368 !& + vx0(nbe(ne(:,3),1)) + vx0(nbe(ne(:,3),2)) + vx0(nbe(ne(:,3),3)))/8.000
369 !vna(:,2)=(vny0(ne(:,2)) + vny0(ne(:,3)) + vny0(nbe(ne(:,2),1)) + vny0(nbe(ne(:,2),2)) + vny0(nbe(ne(:,2),3)) &
370 !& + vny0(nbe(ne(:,3),1)) + vny0(nbe(ne(:,3),2)) + vny0(nbe(ne(:,3),3)))/8.000
371 !vna(:,3)=(vz0(ne(:,2)) + vz0(ne(:,3)) + vz0(nbe(ne(:,2),1)) + vz0(nbe(ne(:,2),2)) + vz0(nbe(ne(:,2),3)) &
372 !& + vz0(nbe(ne(:,3),1)) + vz0(nbe(ne(:,3),2)) + vz0(nbe(ne(:,3),3)))/8.000
373 !-----
374 !vna(:,1)=(2.000*vx0(ne(:,2)) + 2.000*vx0(ne(:,3)) + vx0(nbe(ne(:,2),1)) + vx0(nbe(ne(:,2),2)) + vx0(nbe(ne(:,2),3)) &
375 !& + vx0(nbe(ne(:,3),1)) + vx0(nbe(ne(:,3),2)) + vx0(nbe(ne(:,3),3)))/10.000
376 !vna(:,2)=(2.000*vny0(ne(:,2)) + 2.000*vny0(ne(:,3)) + vny0(nbe(ne(:,2),1)) + vny0(nbe(ne(:,2),2)) + vny0(nbe(ne(:,2),3)) &
377 !& + vny0(nbe(ne(:,3),1)) + vny0(nbe(ne(:,3),2)) + vny0(nbe(ne(:,3),3)))/10.000
378 !vna(:,3)=(2.000*vz0(ne(:,2)) + 2.000*vz0(ne(:,3)) + vz0(nbe(ne(:,2),1)) + vz0(nbe(ne(:,2),2)) + vz0(nbe(ne(:,2),3)) &
379 !& + vz0(nbe(ne(:,3),1)) + vz0(nbe(ne(:,3),2)) + vz0(nbe(ne(:,3),3)))/10.000
380 !-----
381 !vna(:,1)=(3.000*vx0(nbe(ne(:,2),1)) + 3.000*vx0(nbe(ne(:,2),2)) + 3.000*vx0(nbe(ne(:,2),3)) + 3.000*
382 !& + 3.000*vx0(nbe(ne(:,3),2)) + 3.000*vx0(nbe(ne(:,3),3)) + vx0(ne(:,2)) + vx0(ne(:,3)))/20.000
383 !vna(:,2)=(3.000*vny0(nbe(ne(:,2),1)) + 3.000*vny0(nbe(ne(:,2),2)) + 3.000*vny0(nbe(ne(:,2),3)) + 3.000*
384 !& + 3.000*vny0(nbe(ne(:,3),1)) &
385 !& + 3.000*vny0(nbe(ne(:,3),2)) + 3.000*vny0(nbe(ne(:,3),3)) + vny0(ne(:,2)) + vny0(ne(:,3)))/20.000
386 !vna(:,3)=(3.000*vz0(nbe(ne(:,2),1)) + 3.000*vz0(nbe(ne(:,2),2)) + 3.000*vz0(nbe(ne(:,2),3)) + 3.000*
387 !& + 3.000*vz0(nbe(ne(:,3),1)) &
388 !& + 3.000*vz0(nbe(ne(:,3),2)) + 3.000*vz0(nbe(ne(:,3),3)) + vz0(ne(:,2)) + vz0(ne(:,3)))/20.000
389 !-----
390 !vna(:,1)=( vx0(ne(:,2)) + vx0(ne(:,3)))/2.000
391 !vna(:,2)=( vny0(ne(:,2)) + vny0(ne(:,3)))/2.000
392 !vna(:,3)=( vz0(ne(:,2)) + vz0(ne(:,3)))/2.000
393 !-----
394 ! dist1(1)=DSQRT((p(1)-x0(ne(:,2)))**2+(p(1)-y0(ne(:,2)))**2+(p(1)-z0(ne(:,2)))**2)
395 ! dist2(1)=DSQRT((p(1)-x0(ne(:,3)))**2+(p(1)-y0(ne(:,3)))**2+(p(1)-z0(ne(:,3)))**2)
396 ! absdist(1)= DBS(1.000/dist1(1)) + (1.000/dist2(1))
397 !-----
398 !vna(:,1)= ((vx0(ne(:,2))*dist1(1))+vx0(ne(:,3))*dist2(1))/absdist(1)
399 !vna(:,2)= ((vny0(ne(:,2))*dist1(1))+vny0(ne(:,3))*dist2(1))/absdist(1)
400 !vna(:,3)= ((vz0(ne(:,2))*dist1(1))+vz0(ne(:,3))*dist2(1))/absdist(1)
401 !-----
402 ! END WHERE
403 !-----
404 ! FORALL (i=1:npts)
405 ! vna(i,:)= vna(i,:)*DBLE(REAL(npts))
406 ! END FORALL
407 !-----
408 ! FORALL (i=1:npts)
409 ! norma(i)= DSQRT(SUM(vna(i,:)**2))
410 ! END FORALL

```

```

412 |-----|
413 | FORALL (i=1:npts)
414 |   vna(i,:) = vna(i,:)/norma(i)
415 | END FORALL
416 |-----|
417 | ! WHERE(norma(:)>1.000)
418 |   vna(:,1) = 25.000 !vna(:,1)/norma(:)
419 |   vna(:,2) = 35.000 !vna(:,2)/norma(:)
420 |   vna(:,3) = 45.000 !vna(:,3)/norma(:)
421 | END WHERE
422 |-----|
423 |
424 | END SUBROUTINE interp_evn
425 |-----|
426 |
427 | SUBROUTINE interp_ven( npts, ne, nbe,&
428 |   & Vel1x, Vel1y, Vel1z, Vel)
429 |-----|
430 | ! This subroutine computes the normal vector of every node on the surface
431 |-----|
432 | USE Mod_SharedVars, ONLY: DBL, p
433 |-----|
434 | IMPLICIT NONE
435 |-----|
436 | Variables
437 |-----|
438 | INTEGER, INTENT(IN) :: npts           !number of nodes
439 | INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: ne !table of connectivities per node
440 | INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: nbe !table of connectivities per element
441 |-----|
442 | REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: Vel1x, Vel1y, Vel1z !arrays of the components of
443 | !normal vectors over collocation points
444 |-----|
445 | REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(OUT) :: Vel !array whit the values of unit normal
446 | !vector of every node over the surface
447 |-----|
448 | Variables inside the subroutine
449 |-----|
450 | INTEGER :: i, j                       !Counters
451 | ALLOCATE (Vel(npts,3))
452 | Vel = 0.000
453 |-----|
454 | WHERE(ne(:,1)==4)
455 |   Vel(:,1) = (Vel1x(ne(:,2)) + Vel1x(ne(:,3)) + Vel1x(ne(:,4)) + Vel1x(ne(:,5)))/4.000
456 |   Vel(:,2) = (Vel1y(ne(:,2)) + Vel1y(ne(:,3)) + Vel1y(ne(:,4)) + Vel1y(ne(:,5)))/4.000
457 |   Vel(:,3) = (Vel1z(ne(:,2)) + Vel1z(ne(:,3)) + Vel1z(ne(:,4)) + Vel1z(ne(:,5)))/4.000
458 | END WHERE
459 |-----|
460 | WHERE(ne(:,1) == 6)
461 |   Vel(:,1) = (Vel1x(ne(:,2)) + Vel1x(ne(:,3)) + Vel1x(ne(:,4)) + Vel1x(ne(:,5)) &
462 |     & + Vel1x(ne(:,6)) + Vel1x(ne(:,7)))/6.000
463 |   Vel(:,2) = (Vel1y(ne(:,2)) + Vel1y(ne(:,3)) + Vel1y(ne(:,4)) + Vel1y(ne(:,5)) &
464 |     & + Vel1y(ne(:,6)) + Vel1y(ne(:,7)))/6.000
465 |   Vel(:,3) = (Vel1z(ne(:,2)) + Vel1z(ne(:,3)) + Vel1z(ne(:,4)) + Vel1z(ne(:,5)) &
466 |     & + Vel1z(ne(:,6)) + Vel1z(ne(:,7)))/6.000
467 | END WHERE
468 |-----|
469 | WHERE (ne(:,1)==2)
470 |-----|
471 |   !Vel(:,1) = (Vel1x(ne(:,2)) + Vel1x(ne(:,3)) + Vel1x(nbe(ne(:,2),1)) + Vel1x(nbe(ne(:,2),2)) + Vel1x(nbe(ne(
472 | :,:),3))) &
473 |     & + Vel1x(nbe(ne(:,3),1)) + Vel1x(nbe(ne(:,3),2)) + Vel1x(nbe(ne(:,3),3)))/8.000
474 |   !Vel(:,2) = (Vel1y(ne(:,2)) + Vel1y(ne(:,3)) + Vel1y(nbe(ne(:,2),1)) + Vel1y(nbe(ne(:,2),2)) + Vel1y(nbe(ne(
475 | :,:),3))) &
476 |     & + Vel1y(nbe(ne(:,3),1)) + Vel1y(nbe(ne(:,3),2)) + Vel1y(nbe(ne(:,3),3)))/8.000
477 |   !Vel(:,3) = (Vel1z(ne(:,2)) + Vel1z(ne(:,3)) + Vel1z(nbe(ne(:,2),1)) + Vel1z(nbe(ne(:,2),2)) + Vel1z(nbe(ne(
478 | :,:),3))) &
479 |     & + Vel1z(nbe(ne(:,3),1)) + Vel1z(nbe(ne(:,3),2)) + Vel1z(nbe(ne(:,3),3)))/8.000
480 |-----|
481 |   Vel(:,1) = (Vel1x(ne(:,2)) + Vel1x(ne(:,3)))/2.000
482 |   Vel(:,2) = (Vel1y(ne(:,2)) + Vel1y(ne(:,3)))/2.000

```

```

481 |   Vel(:,3) = (Vel1z(ne(:,2)) + Vel1z(ne(:,3)))/2.000
482 | END WHERE
483 |-----|
484 | ! FORALL (i=1:npts)
485 |   ! vna(i,:) = vna(i,:)*DBLE(REAL(npts))
486 | ! END FORALL
487 |-----|
488 | FORALL (i=1:npts)
489 |   norma(i) = DSQRT(SUM(vna(i,:)**2))
490 | END FORALL
491 |-----|
492 | FORALL (i=1:npts)
493 |   vna(i,:) = vna(i,:)/norma(i)
494 | END FORALL
495 |-----|
496 | ! WHERE(norma(:)>1.000)
497 |   vna(:,1) = 25.000 !vna(:,1)/norma(:)
498 |   vna(:,2) = 35.000 !vna(:,2)/norma(:)
499 |   vna(:,3) = 45.000 !vna(:,3)/norma(:)
500 | END WHERE
501 |-----|
502 |
503 | END SUBROUTINE interp_ven
504 |-----|
505 | SUBROUTINE interp_curv( nelm, ne, nbe,&
506 |   & crvmel)
507 |-----|
508 | ! This subroutine computes the normal vector of every node on the surface
509 |-----|
510 | USE Mod_SharedVars, ONLY: DBL, p
511 |-----|
512 | IMPLICIT NONE
513 |-----|
514 | Variables
515 |-----|
516 | INTEGER, INTENT(IN) :: nelm           !number of elements
517 | INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: ne !table of connectivities per node
518 | INTEGER, ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: nbe !table of connectivities per element
519 | REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: crvmel !array whit the values of kappa
520 |-----|
521 | Variables inside the subroutine
522 |-----|
523 | INTEGER :: i, j                       !Counters
524 | REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: cuvi
525 | ALLOCATE (cuvi(nelm))
526 | cuvi = 0.000
527 |-----|
528 | WHERE(ne(:,1)==4)
529 |   cuvi(:) = (crvmel(ne(:,2)) + crvmel(ne(:,3)) + crvmel(ne(:,4)) + crvmel(ne(:,5)))/4.000
530 | END WHERE
531 |-----|
532 | WHERE(ne(:,1) == 6)
533 |   cuvi(:) = (crvmel(ne(:,2)) + crvmel(ne(:,3)) + crvmel(ne(:,4)) + crvmel(ne(:,5)) &
534 |     & + crvmel(ne(:,6)) + crvmel(ne(:,7)))/6.000
535 | END WHERE
536 |-----|
537 | WHERE (ne(:,1)==2)
538 |-----|
539 |   cuvi(:) = (crvmel(ne(:,2)) + crvmel(ne(:,3)))/2.000
540 | END WHERE
541 |-----|
542 | crvmel = cuvi
543 |-----|
544 |
545 | END SUBROUTINE interp_curv
546 |-----|
547 |
548 | SUBROUTINE do_t_produc(Mdim, nelm, gamma,&
549 |   & G0, RM, crvmel, &
550 |   & vnx0, vny0, vnz0 )
551 |-----|
552 |

```

```

553 ! This subroutine generate th array fo G(x,X0) dot delta f
554 !-----
555 ! Variables
556 !-----
557 ! vnx0, vny0, vnz0 ..... arrays of the components of normal vectors over collocation points
558 ! vna ..... array whit the values of unit normal vector of every node over the surface
559 ! ne ..... table of connectivities per node
560 ! nbe ..... table of connectivities per element
561 ! npts ..... number of nodes
562 !-----
563 USE Mod_SharedVars, ONLY: DBL
564 !-----
565 IMPLICIT NONE
566 !-----
567 ! Variables
568 !-----
569 INTEGER, INTENT(IN) :: Mdim, nelm !
570 REAL (KIND = DBL), INTENT(IN) :: gamma !surface tension constant
571 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: vnx0, vny0, vnz0 !arrays of the components of
572 !normal vectors over collocation points
573 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: crvmel !
574 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: GM !The matrix with the information of
575 !Stokeslets on the surface
576 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(OUT) :: RM !The matrix with the information of
577 !-----
578 ! Variables inside the subroutine
579 !-----
580 INTEGER :: i, j
581 !-----
582 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: MM
583 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:, :) :: MM
584 !-----
585 ! Initialize
586 ALLOCATE(MM(Mdim), RM(Mdim))
587 ! ALLOCATE(MM(Mdim,3), RM(Mdim))
588 MM = 0.000
589 RM = 0.000
590 !-----
591 ! FORALL (i=1:nelm)
592 ! MM(i,1) = SUM(GM(i,1:nelm)*vnx0(i))
593 ! MM(i,2) = SUM(GM(i,1+nelm:nelm+nelm)*vny0(i))
594 ! MM(i,3) = SUM(GM(i,1+nelm+nelm:Mdim)*vnz0(i))
595 ! END FORALL
596 ! FORALL (i=1:nelm)
597 ! MM(i+nelm,1) = SUM(GM(i+nelm,1:nelm)*vnx0(i))
598 ! MM(i+nelm,2) = SUM(GM(i+nelm,1+nelm:nelm+nelm)*vny0(i))
599 ! MM(i+nelm,3) = SUM(GM(i+nelm,1+nelm+nelm:Mdim)*vnz0(i))
600 ! END FORALL
601 ! FORALL (i=1:nelm)
602 ! MM(i+nelm+nelm,1) = SUM(GM(i+nelm+nelm,1:nelm)*vnx0(i))
603 ! MM(i+nelm+nelm,2) = SUM(GM(i+nelm+nelm,1+nelm:nelm+nelm)*vny0(i))
604 ! MM(i+nelm+nelm,3) = SUM(GM(i+nelm+nelm,1+nelm+nelm:Mdim)*vnz0(i))
605 ! END FORALL
606 !-----
607 ! FORALL (i=1:nelm)
608 ! RM(i) = SUM(MM(i,:))
609 ! RM(i+nelm) = SUM(MM(i+nelm,:))
610 ! RM(i+nelm+nelm) = SUM(MM(i+nelm+nelm,:))
611 ! END FORALL
612 !-----
613 ! FORALL (i=1:nelm)
614 ! RM(i) = RM(i) * crvmel(i)*gamma
615 ! RM(i+nelm) = RM(i+nelm) * crvmel(i)*gamma
616 ! RM(i+nelm+nelm) = RM(i+nelm+nelm) * crvmel(i)*gamma
617 ! END FORALL
618 !-----
619 ! FORALL (i=1:nelm)
620 ! MM(i) = vnx0(i) * crvmel(i)*gamma
621 ! MM(i+nelm) = vny0(i) * crvmel(i)*gamma
622 ! MM(i+nelm+nelm) = vnz0(i) * crvmel(i)*gamma
623 !
624 ! END FORALL

```

```

625 !-----
626 ! RM=MATMUL(GM,MM)
627 !-----
628 FORALL (i=1:nelm)
629 RM(i) = GM(i) !* crvmel(i) !* gamma
630 RM(i+nelm) = GM(i+nelm) !* crvmel(i) !* gamma
631 RM(i+nelm+nelm) = GM(i+nelm+nelm) !* crvmel(i) !* gamma
632 END FORALL
633 !-----
634 END SUBROUTINE do_t_produc
635 !-----
636 SUBROUTINE dissociate(nelm, Vell, &
637 & Vellx, Velly, Vellz)
638 !-----
639 USE Mod_SharedVars, ONLY: DBL
640 !-----
641 IMPLICIT NONE
642 !-----
643 ! Variables
644 !-----
645 INTEGER, INTENT(IN) :: nelm !number of elements
646 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: Vell !array whit the values of unit normal
647 !vector of every node over the surface
648 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(OUT) :: Vellx, Velly, Vellz !arrays of the components
649 !of normal vectors over collocation points
650 !-----
651 ! Variables inside the subroutine
652 !-----
653 INTEGER :: i !Counter
654 !-----
655 !
656 Vellx = 0.000
657 Velly = 0.000
658 Vellz = 0.000
659 !-----
660 Vellx = Vell(1:nelm)
661 Velly = Vell(nelm+1:nelm+nelm)
662 Vellz = Vell(nelm+nelm+1:)
663 !-----
664 !
665 ! END SUBROUTINE dissociate
666 !-----
667 !
668 SUBROUTINE Interp_e2( npts,vna1,vna)
669 !-----
670 ! This subroutine computes the normal vector of every node on the surface
671 !-----
672 USE Mod_SharedVars, ONLY: DBL !, p , x0, y0, z0
673 !-----
674 IMPLICIT NONE
675 !-----
676 ! Variables
677 !-----
678 INTEGER, INTENT(IN) :: npts !number of nodes
679 ! INTEGER, ALLOCATABLE, DIMENSION(:, :), INTENT(IN) :: ne !table of connectivities per node
680 ! INTEGER, ALLOCATABLE, DIMENSION(:, :), INTENT(IN) :: nbe !table of connectivities per element
681 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: vnx0, vny0, vnz0 !arrays of the components of
682 !normal vectors over collocation points
683 REAL (KIND = DBL), DIMENSION(:, :), INTENT(IN) :: vna1 !array whit the values of unit normal
684 !vector of every node (local)
685 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:, :), INTENT(OUT) :: vna !array whit the values of unit normal
686 !vector of every node over the surface
687 !-----
688 ! Variables inside the subroutine
689 !-----
690 INTEGER :: i, j !Counters
691 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: norma
692 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: dist1,dist2,dist3, absdist
693 !-----
694 ! Initialize
695 ALLOCATE(vna(npts,3), norma(npts), dist1(npts),dist2(npts),dist3(npts),absdist(npts))
696 vna = 0.000

```

```

697 norma = 0.000
698 dist1 = 0.000
699 dist2 = 0.000
700 dist3 = 0.000
701 ! absdist=0.000
702 !-----
703 FORALL (i=1:npts)
704   dist1(i)= SUM(vna1(:,i), INT(vna1(:,4))=1)
705   dist2(i)= SUM(vna1(:,2), INT(vna1(:,4))=1)
706   dist3(i)= SUM(vna1(:,3), INT(vna1(:,4))=1)
707 END FORALL
708 !-----
709 FORALL (i=1:npts)
710   norma(i)= DSQRT(dist1(i)**2+dist2(i)**2+dist3(i)**2)
711   vna(i,1)= dist1(i)/norma(i)
712   vna(i,2)= dist2(i)/norma(i)
713   vna(i,3)= dist3(i)/norma(i)
714   ! vna(i,:)= (/dist1,dist2,dist3)/norma(i)
715 END FORALL
716 !-----
717 !vna(:,1)= dist1
718 !vna(:,2)= dist2
719 !vna(:,3)= dist3
720 !vna(i,:)= (/dist1,dist2,dist3)/norma(i)
721 !-----
722 END SUBROUTINE interp_e2
723 !-----
724 !-----
725 !-----
726 SUBROUTINE interp_p2( x1, y1, z1, &
727 & x2, y2, z2, &
728 & x3, y3, z3, &
729 & x4, y4, z4, &
730 & x5, y5, z5, &
731 & x6, y6, z6, &
732 & x, y, z, &
733 & nx1, ny1, nz1, &
734 & nx2, ny2, nz2, &
735 & nx3, ny3, nz3, &
736 & nx4, ny4, nz4, &
737 & nx5, ny5, nz5, &
738 & nx6, ny6, nz6, &
739 & vnx, vny, vnz, &
740 & hs )
741 !-----
742 ! This subroutine interpolate over an element to compute geometrical variables including the following:
743 ! 1) Position vector
744 ! 2) Tangential vectors in the xi and eta directions
745 ! 3) Unit normal vector
746 ! 4) Area and volume of each element
747 !-----
748 USE Mod_SharedVars, ONLY: DBL, vna1
749 !-----
750 IMPLICIT NONE
751 !-----
752 ! Variables
753 REAL (KIND = DBL), INTENT(IN) :: x1, y1, z1, & !coordinates of first node of the element
754 & x2, y2, z2, & !coordinates of second node of the element
755 & x3, y3, z3, & !coordinates of third node of the element
756 & x4, y4, z4, & !coordinates of fourth node of the element
757 & x5, y5, z5, & !coordinates of fifth node of the element
758 & x6, y6, z6, & !coordinates of sixth node of the element
759 REAL (KIND = DBL), INTENT(OUT) :: x, y, z !coordinates of the collocation point
760 REAL (KIND = DBL), INTENT(OUT) :: nx1, ny1, nz1, & !normal vector coordinates of first node of the element
761 & nx2, ny2, nz2, & !normal vector coordinates of second node of the element
762 & nx3, ny3, nz3, & !normal vector coordinates of third node of the element
763 & nx4, ny4, nz4, & !normal vector coordinates of fourth node of the element

```

```

764 & element nx5, ny5, nz5, & !normal vector coordinates of fifth node of the element
765 & element nx6, ny6, nz6 !normal vector coordinates of sixth node of the element
766 REAL (KIND = DBL), INTENT(OUT) :: vnx, vny, vnz !normal vector of the element
767 REAL (KIND = DBL), INTENT(OUT) :: hs !surface metric
768 !-----
769 ! Variables inside the subroutine
770 !-----
771 REAL (KIND = DBL) :: w12, w13, w21, w31, w23, w32 !components weiths
772 REAL (KIND = DBL) :: b210x, b210y, b210z, b120x, b120y, b120z, & !components of bezier surface
773 & b021x, b021y, b021z, b012x, b012y, b012z, &
774 & b102x, b102y, b102z, b201x, b201y, b201z
775 REAL (KIND = DBL) :: ox, oy, oz !components of bezier surface
776 REAL (KIND = DBL) :: n1, n2, n3, n4, n5, n6 !modules of normal vectors
777 REAL (KIND = DBL) :: v1x, v1y, v1z !components of bezier surface
778 REAL (KIND = DBL) :: v12, v23, v31 !components weiths
779 REAL (KIND = DBL) :: h110x, h110y, h110z, h011x, h011y, h011z, & !components of bezier surface
780 & h101x, h101y, h101z
781 !-----
782 ! Prepare the normal vector of vertices.
783 !-----
784 nx1= -y1*z1 + y6*z1 + y1*z4 - y6*z4 - y1*z6 + y4*z6
785 ny1= x4*z1 - x6*z1 - x1*z4 + x6*z4 - x1*z6 + x4*z6
786 nz1= -x4*y1 + x6*y1 + x1*y4 - x6*y4 - x1*y6 + x4*y6
787 nx2= -y5*z2 + y4*z2 + y2*z5 - y4*z5 - y2*z4 + y5*z4
788 ny2= x5*z2 - x4*z2 - x2*z5 + x4*z5 + x2*z4 - x5*z4
789 nz2= -x5*y2 + x4*y2 + x2*y5 - x4*y5 - x2*y4 + x5*y4
790 nx3= -y6*z3 + y5*z3 + y3*z6 - y5*z6 - y3*z5 + y6*z5
791 ny3= x6*z3 - x5*z3 - x3*z6 + x5*z6 + x3*z5 - x6*z5
792 nz3= -x6*y3 + x5*y3 + x3*y6 - x5*y6 - x3*y5 + x6*y5
793 !-----
794 n1= DSQRT(nx1**2+ny1**2+nz1**2)
795 n2= DSQRT(nx2**2+ny2**2+nz2**2)
796 n3= DSQRT(nx3**2+ny3**2+nz3**2)
797 !-----
798 nx1= nx1/n1
799 ny1= ny1/n1
800 nz1= nz1/n1
801 nx2= nx2/n2
802 ny2= ny2/n2
803 nz2= nz2/n2
804 nx3= nx3/n3
805 ny3= ny3/n3
806 nz3= nz3/n3
807 !-----
808 w12=(x2-x1)*nx1+(y2-y1)*ny1+(z2-z1)*nz1
809 w13=(x3-x1)*nx1+(y3-y1)*ny1+(z3-z1)*nz1
810 w21=(x1-x2)*nx2+(y1-y2)*ny2+(z1-z2)*nz2
811 w31=(x1-x3)*nx3+(y1-y3)*ny3+(z1-z3)*nz3
812 w23=(x3-x2)*nx2+(y3-y2)*ny2+(z3-z2)*nz2
813 w32=(x2-x3)*nx3+(y2-y3)*ny3+(z2-z3)*nz3
814 !-----
815 b210x=(2*x1+x2-w12*nx1)/3.000
816 b210y=(2*y1+y2-w12*ny1)/3.000
817 b210z=(2*z1+z2-w12*nz1)/3.000
818 !-----
819 b120x=(2*x2+x1-w21*nx2)/3.000
820 b120y=(2*y2+y1-w21*ny2)/3.000
821 b120z=(2*z2+z1-w21*nz2)/3.000
822 !-----
823 b021x=(2*x2+x3-w23*nx2)/3.000
824 b021y=(2*y2+y3-w23*ny2)/3.000
825 b021z=(2*z2+z3-w23*nz2)/3.000
826 !-----
827 b012x=(2*x3+x2-w32*nx3)/3.000
828 b012y=(2*y3+y2-w32*ny3)/3.000
829 b012z=(2*z3+z2-w32*nz3)/3.000
830 !-----
831 b102x=(2*x3+x1-w31*nx3)/3.000
832 b102y=(2*y3+y1-w31*ny3)/3.000
833 b102z=(2*z3+z1-w31*nz3)/3.000

```

```

834 |-----|
835 | b201x=(2*x1+x3-w13*nx1)/3.000
836 | b201y=(2*y1+y3-w13*ny1)/3.000
837 | b201z=(2*z1+z3-w13*nz1)/3.000
838 |-----|
839 | ex=(b210x+b120x+b021x+b012x+b102x+b201x)/6.000
840 | ey=(b210y+b120y+b021y+b012y+b102y+b201y)/6.000
841 | ez=(b210z+b120z+b021z+b012z+b102z+b201z)/6.000
842 |-----|
843 | vx=(x1+x2+x3)/3.000
844 | vy=(y1+y2+y3)/3.000
845 | vz=(z1+z2+z3)/3.000
846 |-----|
847 | xx=ex+(ex-vx)/2.000
848 | yy=ey+(ey-vy)/2.000
849 | zz=ez+(ez-vz)/2.000
850 |-----|
851 | v12=2.000*((x2-x1)*(nx1+nx2)+(y2-y1)*(ny1+ny2)+(z2-z1)*(nz1+nz2))/((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1)+(z2-z1)*(z2-z1))
852 | v23=2.000*((x3-x2)*(nx2+nx3)+(y3-y2)*(ny2+ny3)+(z3-z2)*(nz2+nz3))/((x3-x2)*(x3-x2)+(y3-y2)*(y3-y2)+(z3-z2)*(z3-z2))
853 | v31=2.000*((x1-x3)*(nx3+nx1)+(y1-y3)*(ny1+ny3)+(z1-z3)*(nz1+nz3))/((x1-x3)*(x1-x3)+(y1-y3)*(y1-y3)+(z1-z3)*(z1-z3))
854 |-----|
855 | h110x = nx1 + nx2 - v12*(x2 - x1)
856 | h110y = ny1 + ny2 - v12*(y2 - y1)
857 | h110z = nz1 + nz2 - v12*(z2 - z1)
858 | h011x = nx2 + nx3 - v23*(x3 - x2)
859 | h011y = ny2 + ny3 - v23*(y3 - y2)
860 | h011z = nz2 + nz3 - v23*(z3 - z2)
861 | h101x = nx3 + nx1 - v31*(x1 - x3)
862 | h101y = ny3 + ny1 - v31*(y1 - y3)
863 | h101z = nz3 + nz1 - v31*(z1 - z3)
864 |-----|
865 | n4 = DSQR(h110x**2 + h110y**2 + h110z**2)
866 | n5 = DSQR(h011x**2 + h011y**2 + h011z**2)
867 | n6 = DSQR(h101x**2 + h101y**2 + h101z**2)
868 |-----|
869 | nx4 = h110x/n4
870 | ny4 = h110y/n4
871 | nz4 = h110z/n4
872 | nx5 = h011x/n5
873 | ny5 = h011y/n5
874 | nz5 = h011z/n5
875 | nx6 = h101x/n6
876 | ny6 = h101y/n6
877 | nz6 = h101z/n6
878 |-----|
879 | vnx = nx1 + nx2 + nx3 + nx4 + nx5 + nx6
880 | vny = ny1 + ny2 + ny3 + ny4 + ny5 + ny6
881 | vnz = nz1 + nz2 + nz3 + nz4 + nz5 + nz6
882 | hs = DSQR(vnx**2 + vny**2 + vnz**2)
883 |-----|
884 | vnx = vnx/hs
885 | vny = vny/hs
886 | vnz = vnz/hs
887 |-----|
888 | END SUBROUTINE interp_p2
889 |-----|
890 | SUBROUTINE interp_p3(
891 | x1, y1, z1, &
892 | x2, y2, z2, &
893 | x3, y3, z3, &
894 | x4, y4, z4, &
895 | x5, y5, z5, &
896 | x6, y6, z6, &
897 | al, be, ga, &
898 | xi, eta, &
899 | x, y, z, &
900 | DxDx1, DyDy1, DzDz1, &
901 | DxDet, DyDet, DzDet, &
902 | vnx, vny, vnz, &

```

```

903 | &
904 | hs)
905 |-----|
906 | This subroutine interpolate over an element to compute geometrical variables including the following:
907 | 1) Position vector
908 | 2) Tangential vectors in the xi and eta directions
909 | 3) Unit normal vector
910 | 4) Area and volume of each element
911 |-----|
912 | USE Mod_SharedVars, ONLY: DBL
913 | IMPLICIT NONE
914 |-----|
915 | Variables
916 | REAL (KIND = DBL), INTENT(IN) :: x1, y1, z1, & |coordinates of first node of the element
917 | x2, y2, z2, & |coordinates of second node of the element
918 | x3, y3, z3, & |coordinates of third node of the element
919 | x4, y4, z4, & |coordinates of fourth node of the element
920 | x5, y5, z5, & |coordinates of fifth node of the element
921 | x6, y6, z6 |coordinates of sixth node of the element
922 | REAL (KIND = DBL), INTENT(IN) :: al, be, ga |constants alpha, beta and gamma
923 | REAL (KIND = DBL), INTENT(IN) :: xi, eta |variables to integrate over a triangle
924 | REAL (KIND = DBL), INTENT(OUT) :: x, y, z |coordinates of the f(x,y,z)=f(xi,eta)
925 | REAL (KIND = DBL), INTENT(OUT) :: DxDx1, DyDy1, DzDz1, & |Derivates of the tangential vectors over the
926 | DxDet, DyDet, DzDet |element
927 | REAL (KIND = DBL), INTENT(OUT) :: vnx, vny, vnz |normal vector of the element
928 | REAL (KIND = DBL), INTENT(OUT) :: hs, es, xs |surface metric
929 |-----|
930 | Variables inside the subroutine
931 |-----|
932 | REAL (KIND = DBL) :: alc, bec, gac, xs, es, hhs |complements of alpha, beta and
933 | gamma |
934 | REAL (KIND = DBL) :: alalc, bebec, gagac |other constants of alpha, beta and gamma
935 | REAL (KIND = DBL) :: phi1, ph2, ph3, ph4, ph5, ph6 |components phi(xi,eta)
936 | REAL (KIND = DBL) :: dph1, dph2, dph3, dph4, dph5, dph6 |derivate of phi(xi,eta) wrt xi
937 | REAL (KIND = DBL) :: pph1, pph2, pph3, pph4, pph5, pph6 |derivate of phi(xi,eta) wrt eta
938 |-----|
939 | Prepare the constants of this subroutine.
940 |-----|
941 | alc = 1.000-al
942 | bec = 1.000-be
943 | gac = 1.000-ga
944 | alalc = al*alc
945 | bebec = be*bec
946 | gagac = ga*gac
947 |-----|
948 | In this part, the subroutine evaluates basis functions, it obtains the tangential and normal vectors.
949 | This is based on Numerical Computation in Science and Engineering, C.Pozrikidis, 1998, pp.305-312.
950 |-----|
951 | ph2 = xi*(xi-al+eta*(al-ga))/gac/alc
952 | ph3 = eta*(eta-be+xi*(be-ga-1.000)/ga)/bec
953 | ph4 = xi*(1.000-xi-eta)/alalc
954 | ph5 = xi*eta
955 | ph6 = eta*(1.000-xi-eta)/bebec
956 |-----|
957 | ph1 = 1.000-ph2-ph3-ph4-ph5-ph6
958 |-----|
959 | Interpolate the position vector (x, y, z)
960 |-----|
961 | x = x1*ph1 + x2*ph2 + x3*ph3 + x4*ph4 + x5*ph5 + x6*ph6
962 | y = y1*ph1 + y2*ph2 + y3*ph3 + y4*ph4 + y5*ph5 + y6*ph6
963 | z = z1*ph1 + z2*ph2 + z3*ph3 + z4*ph4 + z5*ph5 + z6*ph6
964 |-----|
965 | In this part suroutine evaluates xi derivatives of basis functions
966 |-----|
967 | dph2 = (2.000*xi-al-eta*(al-ga))/gac/alc
968 | dph3 = eta*(be-ga-1.000)/(ga*bec)
969 | dph4 = (1.000-2.000*xi-eta)/alalc
970 | dph5 = eta/gagac
971 | dph6 = -eta/bebec
972 | dph1 = -dph2-dph3-dph4-dph5-dph6
973 |-----|
974 | Compute dx/dxi from xi derivatives of phi

```



```

973 |-----|
974 | DxDxi = x1*dph1 + x2*dph2 + x3*dph3 + x4*dph4 + x5*dph5 + x6*dph6
975 | DyDxi = y1*dph1 + y2*dph2 + y3*dph3 + y4*dph4 + y5*dph5 + y6*dph6
976 | DzDxi = z1*dph1 + z2*dph2 + z3*dph3 + z4*dph4 + z5*dph5 + z6*dph6
977 | xs = DSQRT(DxDxi*DxDxi + DyDxi*DyDxi + DzDxi*DzDxi)
978 |-----|
979 | Normalization of normal vector
980 |-----|
981 | DxDxi = DxDxi
982 | DyDxi = DyDxi
983 | DzDxi = DzDxi
984 |-----|
985 | In this part suroutine evaluates eta derivatives of basis functions
986 |-----|
987 | pph2 = xi*(al-ga)/(alc*gac)
988 | pph3 = (2.000*eta-be+xi*(berga-1.000)/ga)/bec
989 | pph4 = -xi/alalc
990 | pph5 = xi/gagac
991 | pph6 = (1.000-xi-2.000*eta)/bebec
992 | pph1 = -pph2-pph3-pph4-pph5-pph6
993 |-----|
994 | Compute dx/deta from eta derivatives of phi
995 |-----|
996 | DxDet = x1*pph1 + x2*pph2 + x3*pph3 + x4*pph4 + x5*pph5 + x6*pph6
997 | DyDet = y1*pph1 + y2*pph2 + y3*pph3 + y4*pph4 + y5*pph5 + y6*pph6
998 | DzDet = z1*pph1 + z2*pph2 + z3*pph3 + z4*pph4 + z5*pph5 + z6*pph6
999 | es = DSQRT(DxDet*DxDet + DyDet*DyDet + DzDet*DzDet)
1000 |-----|
1001 | Normalization of normal vector
1002 |-----|
1003 | DxDet = DxDet
1004 | DyDet = DyDet
1005 | DzDet = DzDet
1006 |-----|
1007 | Normal vector vn = (DxDxi)*(DxDeta)
1008 | Surface metric hs = norm(vn)
1009 |-----|
1010 | vnz = DxDxi * DxDet - DyDet * DzDxi
1011 | vny = DzDxi * DxDet - DzDet * DxDxi
1012 | vnx = DxDxi * DyDet - DxDet * DyDxi
1013 | hs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
1014 |-----|
1015 | Normalization of normal vector
1016 |-----|
1017 | vnx = vnx/hs
1018 | vny = vny/hs
1019 | vnz = vnz/hs
1020 | !hs= es*xs
1021 |-----|
1022 | vnx = (DyDxi/xs) * (DzDet/es) - (DyDet/es) * (DzDxi/xs)
1023 | vny = (DzDxi/xs) * (DxDet/es) - (DzDet/es) * (DxDxi/xs)
1024 | vnz = (DxDxi/xs) * (DyDet/es) - (DxDet/es) * (DyDxi/xs)
1025 | hs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
1026 |-----|
1027 | Normalization of normal vector
1028 |-----|
1029 | vnx = vnx/hhs
1030 | vny = vny/hhs
1031 | vnz = vnz/hhs
1032 | !hs= es*xs
1033 |-----|
1034 | END SUBROUTINE Interp_p3
1035 |-----|
1036 | SUBROUTINE Interp_p4( x1, y1, z1, &
1037 | & x2, y2, z2, &
1038 | & x3, y3, z3, &
1039 | & x4, y4, z4, &
1040 | & x5, y5, z5, &
1041 | & x6, y6, z6, &
1042 | &

```

```

1043 | & al, be, ga, &
1044 | & xi, eta, &
1045 | & curv, stride )
1046 |-----|
1047 | This subroutine interpolate over an element to compute geometrical variables including the following:
1048 | 1) Position vector
1049 | 2) Tangential vectors in the xi and eta directions
1050 | 3) Unit normal vector
1051 | 4) Area and volume of each element
1052 |-----|
1053 | USE Mod_SharedVars, ONLY: DBL
1054 |-----|
1055 | IMPLICIT NONE
1056 |-----|
1057 | Variables
1058 | INTEGER, INTENT(IN) :: stride | stride
1059 | REAL (KIND = DBL), INTENT(IN) :: xi, y1, z1, & | coordinates of first node of the element
1060 | & x2, y2, z2, & | coordinates of second node of the element
1061 | & x3, y3, z3, & | coordinates of third node of the element
1062 | & x4, y4, z4, & | coordinates of fourth node of the element
1063 | & x5, y5, z5, & | coordinates of fifth node of the element
1064 | & x6, y6, z6 | coordinates of sixth node of the element
1065 | REAL (KIND = DBL), INTENT(IN) :: al, be, ga | constants alpha, beta and gamma
1066 | REAL (KIND = DBL), INTENT(IN) :: xi, eta | variables to integrate over a triangle
1067 | REAL (KIND = DBL), INTENT(OUT) :: curv | mean curvature
1068 |-----|
1069 | Variables inside the subroutine
1070 |-----|
1071 | REAL (KIND = DBL) :: x, y, z | coordinates of the f(x,y,z)= F(xi,eta)
1072 | REAL (KIND = DBL) :: vnx, vny, vnz | normal vector of the element
1073 | REAL (KIND = DBL) :: hs | surface metric
1074 | REAL (KIND = DBL) :: alc, bec, gac | complements of alpha, beta and gamma
1075 | REAL (KIND = DBL) :: xs, es, hhs | labs value of derivatives wrt eta and xi
1076 | REAL (KIND = DBL) :: alalc, bebec, gagac | other constants of alpha, beta and gamma
1077 | REAL (KIND = DBL) :: phi1, phi2, phi3, phi4, phi5, phi6 | components phi(xi,eta)
1078 | REAL (KIND = DBL) :: dphi1, dphi2, dphi3, dphi4, dphi5, dphi6 | derivative of phi(xi,eta) wrt xi
1079 | REAL (KIND = DBL) :: pph1, pph2, pph3, pph4, pph5, pph6 | derivative of phi(xi,eta) wrt eta
1080 | REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi | Derivates wrt xi
1081 | REAL (KIND = DBL) :: DxDet, DyDet, DzDet | Derivates wrt eta
1082 | REAL (KIND = DBL) :: DxDxi2, DyDxi2, DzDxi2 | Derivates wrt xi and xi
1083 | REAL (KIND = DBL) :: DxDet2, DyDet2, DzDet2 | Derivates wrt eta and eta
1084 | REAL (KIND = DBL) :: DxDxi2t, DyDxi2t, DzDxi2t | Derivates wrt xi and eta
1085 | REAL (KIND = DBL) :: pph21, pph22, pph23, pph24, pph25, pph26 | second derivate of phi(xi,eta) wrt eta
1086 | REAL (KIND = DBL) :: dph21, dph22, dph23, dph24, dph25, dph26 | second derivate of phi(xi,eta) wrt xi
1087 | REAL (KIND = DBL) :: pdphi1, pdphi2, pdphi3, pdphi4, pdphi5, pdphi6 | second derivate of phi(xi,eta) wrt eta
1088 | REAL (KIND = DBL) :: EK1, FK1, GK1 | constants of first Fundamental form
1089 | REAL (KIND = DBL) :: EK2, FK2, GK2 | constants of first Fundamental form
1090 | REAL (KIND = DBL) :: EK3, FK3, GK3 | constants of first Fundamental form
1091 | REAL (KIND = DBL) :: LK1, MK1, NK1 | constants of second Fundamental form
1092 | REAL (KIND = DBL) :: LK2, MK2, NK2 | constants of second Fundamental form
1093 | REAL (KIND = DBL) :: LK3, MK3, NK3 | constants of second Fundamental form
1094 | REAL (KIND = DBL) :: EK, FK, GK | constants of first Fundamental form
1095 | REAL (KIND = DBL) :: LK, MK, NK | constants of second Fundamental form
1096 | REAL (KIND = DBL) :: strider | constants of real of stride
1097 |-----|
1098 | Prepare the constants of this subroutine.
1099 |-----|
1100 | alc = 1.000-al
1101 | bec = 1.000-be
1102 | gac = 1.000-ga
1103 | alalc = al*alc
1104 | bebec = be*bec
1105 | gagac = ga*gac
1106 |-----|
1107 | In this part, the subroutine evaluates basis functions, it obtains the tangential and normal vectors.
1108 | This is based on Numerical Computation in Science and Engineering, C.Pozrikidis, 1998, pp.305-312
1109 |-----|
1110 | phi2 = xi *(xi - al + eta*(al - ga)/gac)/alc
1111 | phi3 = eta*(eta - be + xi*(be + ga - 1.000)/ga)/bec
1112 | phi4 = xi *(1.000 - xi - eta)/alalc
1113 | phi5 = xi*eta/gagac

```

```

1114 ph6 = eta*(1.000 - x1 - eta)/bebec
1115 ph1 = 1.000-ph2-ph3-ph4-ph5-ph6
1116 -----
1117 ! Interpolate the position vector (x, y, z)
1118 -----
1119 x = x1*ph1 + x2*ph2 + x3*ph3 + x4*ph4 + x5*ph5 + x6*ph6
1120 y = y1*ph1 + y2*ph2 + y3*ph3 + y4*ph4 + y5*ph5 + y6*ph6
1121 z = z1*ph1 + z2*ph2 + z3*ph3 + z4*ph4 + z5*ph5 + z6*ph6
1122 -----
1123 ! In this part suroutine evaluates xi derivatives of basis functions
1124 -----
1125 dph2 = (2.000*x1 - a1 + eta*(a1 - ga)/gac)/alc
1126 dph3 = eta*(be + ga - 1.000)/(ga*bec)
1127 dph4 = (1.000 - 2.000*x1 - eta)/alalc
1128 dph5 = eta/gagac
1129 dph6 = -eta/bebec
1130 dph1 = -dph2 - dph3 - dph4 - dph5 - dph6
1131 -----
1132 ! Compute dx/dx1 from x1 derivatives of phi
1133 -----
1134 DxDx1 = x1*dph1 + x2*dph2 + x3*dph3 + x4*dph4 + x5*dph5 + x6*dph6
1135 DyDx1 = y1*dph1 + y2*dph2 + y3*dph3 + y4*dph4 + y5*dph5 + y6*dph6
1136 DzDx1 = z1*dph1 + z2*dph2 + z3*dph3 + z4*dph4 + z5*dph5 + z6*dph6
1137 xs = DSQRT(DxDx1*DxDx1 + DyDx1*DyDx1 + DzDx1*DzDx1)
1138 -----
1139 ! In this part suroutine evaluates eta derivatives of basis functions
1140 -----
1141 pph2 = xi*(a1 - ga)/(alc*gac)
1142 pph3 = (2.000*eta - be + xi*(be + ga - 1.000)/ga)/bec
1143 pph4 = -xi/alalc
1144 pph5 = xi/gagac
1145 pph6 = (1.000 - xi - 2.000*eta)/bebec
1146 pph1 = - pph2 - pph3 - pph4 - pph5 - pph6
1147 -----
1148 ! Compute dx/deta from eta derivatives of phi
1149 -----
1150 DxDet = x1*pph1 + x2*pph2 + x3*pph3 + x4*pph4 + x5*pph5 + x6*pph6
1151 DyDet = y1*pph1 + y2*pph2 + y3*pph3 + y4*pph4 + y5*pph5 + y6*pph6
1152 DzDet = z1*pph1 + z2*pph2 + z3*pph3 + z4*pph4 + z5*pph5 + z6*pph6
1153 es = DSQRT(DxDet*DxDet + DyDet*DyDet + DzDet*DzDet)
1154 -----
1155 ! Normal vector vn = (DxDx1)/(DxDeta)
1156 ! Surface metric hs = norm(vn)
1157 -----
1158 vnx = DyDx1 * DzDet - DyDet * DzDx1
1159 vny = DzDx1 * DxDet - DxDet * DxDx1
1160 vnz = DxDx1 * DyDet - DxDet * DyDx1
1161 hns = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
1162 -----
1163 ! Normalization of normal vector
1164 -----
1165 vnx = vnx/hns
1166 vny = vny/hns
1167 vnz = vnz/hns
1168 hs = hns
1169 -----
1170 ! In this part, the subroutine computes the mean curvature using the analysis of Stoker Chapter IV.
1171 !First the vectors E, F and G are computed to estimate the First Fundamental Form
1172 -----
1173 EK1 = DxDx1**2
1174 EK2 = DyDx1**2
1175 EK3 = DzDx1**2
1176 -----
1177 FK1 = DxDx1*DxDet
1178 FK2 = DyDx1*DyDet
1179 FK3 = DzDx1*DzDet
1180 -----
1181 GK1 = DxDet**2
1182 GK2 = DyDet**2
1183 GK3 = DzDet**2
1184 -----
1185 ! Evaluation of E, F, and G

```

```

1186 -----
1187 EK = EK1 + EK2 + EK3
1188 FK = FK1 + FK2 + FK3
1189 GK = GK1 + GK2 + GK3
1190 -----
1191 ! Second derivatives of the element.
1192 !In this part suroutine evaluates the second derivatives wrt xi
1193 -----
1194 dph22 = (2.000)/alc
1195 dph23 = 0.000
1196 dph24 = -2.000/alalc
1197 dph25 = 0.000
1198 dph26 = 0.000
1199 dph21 = -dph22 - dph23 - dph24 - dph25 - dph26
1200 -----
1201 ! Compute dx/dx1**2 of phi
1202 -----
1203 DxDx12 = x1*dph21 + x2*dph22 + x3*dph23 + x4*dph24 + x5*dph25 + x6*dph26
1204 DyDx12 = y1*dph21 + y2*dph22 + y3*dph23 + y4*dph24 + y5*dph25 + y6*dph26
1205 DzDx12 = z1*dph21 + z2*dph22 + z3*dph23 + z4*dph24 + z5*dph25 + z6*dph26
1206 -----
1207 !In this part suroutine evaluates the second derivatives wrt xi then wrt eta
1208 -----
1209 pdph2 = ((a1 - ga)/gac*alc)
1210 pdph3 = (be + ga - 1.000)/(ga*bec)
1211 pdph4 = (-1.000)/alalc
1212 pdph5 = 1.000/gagac
1213 pdph6 = -1.000/bebec
1214 pdph1 = -pdph2 - pdph3 - pdph4 - pdph5 - pdph6
1215 -----
1216 ! Compute dx/dxidet of phi
1217 -----
1218 DxDxiet = x1*pdph1 + x2*pdph2 + x3*pdph3 + x4*pdph4 + x5*pdph5 + x6*pdph6
1219 DyDxiet = y1*pdph1 + y2*pdph2 + y3*pdph3 + y4*pdph4 + y5*pdph5 + y6*pdph6
1220 DzDxiet = z1*pdph1 + z2*pdph2 + z3*pdph3 + z4*pdph4 + z5*pdph5 + z6*pdph6
1221 -----
1222 ! In this part suroutine evaluates eta second derivatives of phi
1223 -----
1224 pph22 = 0.000
1225 pph23 = (2.000)/bec
1226 pph24 = 0.000
1227 pph25 = 0.000
1228 pph26 = (-2.000)/bebec
1229 pph21 = - pph22 - pph23 - pph24 - pph25 - pph26
1230 -----
1231 ! Compute dx/det**2 from eta derivatives of phi
1232 -----
1233 DxDet2 = x1*pph21 + x2*pph22 + x3*pph23 + x4*pph24 + x5*pph25 + x6*pph26
1234 DyDet2 = y1*pph21 + y2*pph22 + y3*pph23 + y4*pph24 + y5*pph25 + y6*pph26
1235 DzDet2 = z1*pph21 + z2*pph22 + z3*pph23 + z4*pph24 + z5*pph25 + z6*pph26
1236 -----
1237 ! In this part, the subroutine computes the mean curvature using the analysis of Stoker Chapter IV.
1238 !First the vectors L, M and N are computed to estimate the Second Fundamental Form
1239 -----
1240 LK1 = DxDx12*vnx / DSQRT(EK*GK - FK**2)
1241 LK2 = DyDx12*vny / DSQRT(EK*GK - FK**2)
1242 LK3 = DzDx12*vnz / DSQRT(EK*GK - FK**2)
1243 -----
1244 MK1 = DxDxiet*vnx / DSQRT(EK*GK - FK**2)
1245 MK2 = DyDxiet*vny / DSQRT(EK*GK - FK**2)
1246 MK3 = DzDxiet*vnz / DSQRT(EK*GK - FK**2)
1247 -----
1248 NK1 = DxDet2*vnx / DSQRT(EK*GK - FK**2)

```

```
1249 NK2 = DyDet2*vny / DSQRT(EK*GK - FK**2)
1250 NK3 = DzDet2*vnz / DSQRT(EK*GK - FK**2)
1251 !-----
1252 ! Evaluation of L, M and W
1253 !-----
1254 LK = LK1 + LK2 + LK3
1255 MK = MK1 + MK2 + MK3
1256 NK = NK1 + NK2 + NK3
1257 !-----
1258 ! Mean Curvature is obtained
1259 !-----
1260 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)))
1261 ! curv = DSIG(0.5D0*((EK*NK + GK*LK - 2.0D0*FK*MK) / (EK*GK - FK**2)),((LK*NK - MK*MK) / (EK*GK - FK**2)))
1262 strider = REAL(stride)
1263 curv = ((-1.0D0)*curv* hs
1264 !-----
1265 END SUBROUTINE interp_p4
1266 !-----
1267 ! Subroutine INTERP_E is a new method to obtain the normal vectors on every node on dorp's surface. The
1268 !name has the end_E which is an abbreviation of element. Firstly, this decision aids to differentiate this
1269 !method of subroutine INTERP_P. Second idea was to continue with this notation or manner to name subroutines to
1270 !have similar environment names. Finally INTERP_E is very important because this subroutine is fundamental to
1271 !advanced the dynamic of dorp deformation in the time. Therefore, this subroutine is in this new MODULE
1272 !-----
1273 END MODULE Mod_Nodal_Interp
1274
```

```

1
2 MODULE Mod_Gauss_Coefs
3 =====
4 ! Version: 0.5 created on 26 / IX / 2007
5 !
6 !
7 ! Version: 0.7 created on -- / III / 2010
8 !
9 !
10 ! Version: 0.9 created on 21 / 08 / 2012
11 ! Version: 1.0 created on 14 / 11 / 2012
12 !
13 !
14 CONTAINS
15
16 SUBROUTINE Gauss_Legendre(NGL)
17 =====
18 ! This Subroutine obtains abscissas and weights for the Gauss-Legendre quadrature with NGL points.
19 ! The value NGL or NGL as this code is the number of Gauss-Legendre base points for singular integrals.
20 ! SYMBOLS:
21 ! NGL = 1,2,3,4,5,6,8,12 and 20
22 ! Default value is 20
23 =====
24
25 USE Mod_SharedVars, ONLY: DBL, UDat, ULog, ZZ, MW
26 =====
27
28 IMPLICIT NONE
29 =====
30
31 ! Variables
32 INTEGER, INTENT(INOUT) :: NGL ! order of Gauss-Legendre quadrature
33
34 ! The next step is use a Case statement to obtain the coeficientes.
35 SELECT CASE(NGL)
36
37 CASE (1)
38 ALLOCATE(ZZ(NGL), MW(NGL))
39 ZZ = 0.000
40 MW = 0.000
41
42 ZZ(1) = 0.000
43 MW(1) = 2.000
44
45 CASE (2)
46 ALLOCATE(ZZ(NGL), MW(NGL))
47 ZZ = 0.000
48 MW = 0.000
49
50 ZZ(1) = -0.5773502691896257645000
51 ZZ(2) = -ZZ(1)
52
53 MW(1) = 1.000
54 MW(2) = 1.000
55
56 CASE (3)
57 ALLOCATE(ZZ(NGL), MW(NGL))
58 ZZ = 0.000
59 MW = 0.000
60
61 ZZ(1) = -0.7745966692414833770300
62 ZZ(2) = 0.000
63 ZZ(3) = -ZZ(1)
64
65 MW(1) = 0.5555555555555555555500
66 MW(2) = 0.8888888888888888888800
67 MW(3) = 0.5555555555555555555500
68
69 CASE (4)
70 ALLOCATE(ZZ(NGL), MW(NGL))
71 ZZ = 0.000
72 MW = 0.000

```

```

73 |-----
74 ZZ(1) = -0.8611363115948525752200
75 ZZ(2) = -0.3399810435848562648000
76 ZZ(3) = -ZZ(2)
77 ZZ(4) = -ZZ(1)
78 |-----
79 MW(1) = 0.3478548451374538573700
80 MW(2) = 0.6521451548625461426200
81 MW(3) = MW(2)
82 MW(4) = MW(1)
83 |-----
84 CASE (5)
85 ALLOCATE(ZZ(NGL), MW(NGL))
86 ZZ = 0.000
87 MW = 0.000
88 |-----
89 ZZ(1) = -0.9061798459386639927900
90 ZZ(2) = -0.5304693101056830910300
91 ZZ(3) = 0.000
92 ZZ(4) = -ZZ(2)
93 ZZ(5) = -ZZ(1)
94 |-----
95 MW(1) = 0.2369268850561890875100
96 MW(2) = 0.47862870493664680400
97 MW(3) = 0.5688888888888888888900
98 MW(4) = MW(2)
99 MW(5) = MW(1)
100 |-----
101 CASE (6)
102 ALLOCATE(ZZ(NGL), MW(NGL))
103 ZZ = 0.000
104 MW = 0.000
105 |-----
106 ZZ(1) = -0.932469514203152000
107 ZZ(2) = -0.661209386466265000
108 ZZ(3) = -0.238619186083197000
109 ZZ(4) = -ZZ(3)
110 ZZ(5) = -ZZ(2)
111 ZZ(6) = -ZZ(1)
112 |-----
113 MW(1) = 0.171324492379170000
114 MW(2) = 0.360761573040139000
115 MW(3) = 0.467913934572691000
116 MW(4) = MW(3)
117 MW(5) = MW(2)
118 MW(6) = MW(1)
119 |-----
120 CASE (8)
121 ALLOCATE(ZZ(NGL), MW(NGL))
122 ZZ = 0.000
123 MW = 0.000
124 |-----
125 ZZ(1) = -0.9602898564975362316800
126 ZZ(2) = -0.7966664774136267395900
127 ZZ(3) = -0.5255324099163289858100
128 ZZ(4) = -0.1834346424956498049300
129 ZZ(5) = -ZZ(4)
130 ZZ(6) = -ZZ(3)
131 ZZ(7) = -ZZ(2)
132 ZZ(8) = -ZZ(1)
133 |-----
134 MW(1) = 0.1012285362903762591500
135 MW(2) = 0.2223810344533744705400
136 MW(3) = 0.3137066458778872873300
137 MW(4) = 0.3626837837836198296000
138 MW(5) = MW(4)
139 MW(6) = MW(3)
140 MW(7) = MW(2)
141 MW(8) = MW(1)
142 |-----
143 CASE (12)
144 ALLOCATE(ZZ(NGL), MW(NGL))

```

```

145 ZZ = 0.000
146 Ww = 0.000
147 -----
148 ZZ(1) = -0.9815666324671900
149 ZZ(2) = -0.98411725637047500
150 ZZ(3) = -0.76990267419430500
151 ZZ(4) = -0.58731795428661700
152 ZZ(5) = -0.36783149899818000
153 ZZ(6) = -0.12523340851146900
154 ZZ(7) = -ZZ(6)
155 ZZ(8) = -ZZ(5)
156 ZZ(9) = -ZZ(4)
157 ZZ(10) = -ZZ(3)
158 ZZ(11) = -ZZ(2)
159 ZZ(12) = -ZZ(1)
160 -----
161 Ww(1) = 0.04717533638651200
162 Ww(2) = 0.10693932595953100
163 Ww(3) = 0.16087832854334600
164 Ww(4) = 0.20316742672306600
165 Ww(5) = 0.23349253653835500
166 Ww(6) = 0.24914704581340300
167 Ww(7) = Ww(6)
168 Ww(8) = Ww(5)
169 Ww(9) = Ww(4)
170 Ww(10) = Ww(3)
171 Ww(11) = Ww(2)
172 Ww(12) = Ww(1)
173 -----
174 CASE (20)
175 ALLOCATE (ZZ(NGL), Ww(NGL))
176 ZZ = 0.000
177 Ww = 0.000
178 -----
179 ZZ(1) = -0.99312859918509492478600
180 ZZ(2) = -0.96397192727791379126800
181 ZZ(3) = -0.912344282513259058600
182 ZZ(4) = -0.83911697182221882339500
183 ZZ(5) = -0.74633190646015079261400
184 ZZ(6) = -0.63605368072651502545300
185 ZZ(7) = -0.51086700195082709800400
186 ZZ(8) = -0.37370608871541956067300
187 ZZ(9) = -0.22778585114164507808000
188 ZZ(10) = -0.07652652113349733375500
189 ZZ(11) = -ZZ(10)
190 ZZ(12) = -ZZ(9)
191 ZZ(13) = -ZZ(8)
192 ZZ(14) = -ZZ(7)
193 ZZ(15) = -ZZ(6)
194 ZZ(16) = -ZZ(5)
195 ZZ(17) = -ZZ(4)
196 ZZ(18) = -ZZ(3)
197 ZZ(19) = -ZZ(2)
198 ZZ(20) = -ZZ(1)
199 -----
200 Ww(1) = 0.01761400713915211831200
201 Ww(2) = 0.04060142980038694133100
202 Ww(3) = 0.06267204833410906357000
203 Ww(4) = 0.08327674157670474872500
204 Ww(5) = 0.10193011981724043503700
205 Ww(6) = 0.11819453196151841731200
206 Ww(7) = 0.13168863844917662609800
207 Ww(8) = 0.14209610931838205132900
208 Ww(9) = 0.14917298647260374678800
209 Ww(10) = 0.15275338713072585069800
210 Ww(11) = Ww(10)
211 Ww(12) = Ww(9)
212 Ww(13) = Ww(8)
213 Ww(14) = Ww(7)
214 Ww(15) = Ww(6)
215 Ww(16) = Ww(5)
216 Ww(17) = Ww(4)

```

```

217 Ww(18) = Ww(3)
218 Ww(19) = Ww(2)
219 Ww(20) = Ww(1)
220 -----
221 CASE DEFAULT
222 NGL = 20
223 ALLOCATE (ZZ(NGL), Ww(NGL))
224 ZZ = 0.000
225 Ww = 0.000
226 -----
227 ZZ(1) = -0.99312859918509492478600
228 ZZ(2) = -0.96397192727791379126800
229 ZZ(3) = -0.912344282513259058600
230 ZZ(4) = -0.83911697182221882339500
231 ZZ(5) = -0.74633190646015079261400
232 ZZ(6) = -0.63605368072651502545300
233 ZZ(7) = -0.51086700195082709800400
234 ZZ(8) = -0.37370608871541956067300
235 ZZ(9) = -0.22778585114164507808000
236 ZZ(10) = -0.07652652113349733375500
237 ZZ(11) = -ZZ(10)
238 ZZ(12) = -ZZ(9)
239 ZZ(13) = -ZZ(8)
240 ZZ(14) = -ZZ(7)
241 ZZ(15) = -ZZ(6)
242 ZZ(16) = -ZZ(5)
243 ZZ(17) = -ZZ(4)
244 ZZ(18) = -ZZ(3)
245 ZZ(19) = -ZZ(2)
246 ZZ(20) = -ZZ(1)
247 Ww(1) = 0.01761400713915211831200
248 Ww(2) = 0.04060142980038694133100
249 Ww(3) = 0.06267204833410906357000
250 Ww(4) = 0.08327674157670474872500
251 Ww(5) = 0.10193011981724043503700
252 Ww(6) = 0.11819453196151841731200
253 Ww(7) = 0.13168863844917662609800
254 Ww(8) = 0.14209610931838205132900
255 Ww(9) = 0.14917298647260374678800
256 Ww(10) = 0.15275338713072585069800
257 Ww(11) = Ww(10)
258 Ww(12) = Ww(9)
259 Ww(13) = Ww(8)
260 Ww(14) = Ww(7)
261 Ww(15) = Ww(6)
262 Ww(16) = Ww(5)
263 Ww(17) = Ww(4)
264 Ww(18) = Ww(3)
265 Ww(19) = Ww(2)
266 Ww(20) = Ww(1)
267 -----
268 WRITE (Ulog,*)
269 WRITE (Ulog,*) ' Gauss_LegENDre:'
270 WRITE (Ulog,*) ' Chosen number of Gaussian points'
271 WRITE (Ulog,*) ' is not available; it was taken NGL=20'
272
273 END SELECT
274 END SUBROUTINE Gauss_Legendre
275 =====
276
277 SUBROUTINE Gauss_Trg(minc)
278 ! This subroutine obtains abscissas and weights for Gaussian integration over a triangle.
279 ! Integration is performed with respect to the triangle barycentric coordinates
280 ! SYMBOLS:
281 ! minc: order of the quadrature choose from 1,3,4,5,7,9,12,13
282 ! Default value is 13
283 =====
284
285 USE Mod_SharedVars, ONLY: DBL, Ulog, xiq, etq, wq
286
287 IMPLICIT NONE
288 =====

```

```

289 ! Variables
290 -----
291 INTEGER, INTENT(INOUT) :: mint ! order of the triangle quadrature
292 -----
293 ! Variables inside the subroutine
294 -----
295 REAL (KIND = DBL) :: a1, be, ga, de ! constants with the weith values
296 REAL (KIND = DBL) :: rh, qa, ru
297 REAL (KIND = DBL) :: o1, o2, o3, o4
298 -----
299 ! The next step is use a Case statement to obtain the coeficientes.
300 SELECT CASE (mint)
301 CASE(1)
302   ALLOCATE(xiq(mint), etq(mint), wq(mint))
303   xiq = 0.000
304   etq = 0.000
305   wq = 0.000
306 -----
307   xiq(1) = 1.000/3.000
308   etq(1) = 1.000/3.000
309   wq(1) = 1.000
310 -----
311 CASE(2)
312   ALLOCATE(xiq(mint), etq(mint), wq(mint))
313   xiq = 0.000
314   etq = 0.000
315   wq = 0.000
316 -----
317   xiq(1) = 1.000/6.000
318   xiq(2) = 2.000/3.000
319   xiq(3) = 1.000/6.000
320   etq(1) = 1.000/6.000
321   etq(2) = 1.000/6.000
322   etq(3) = 2.000/3.000
323   wq(1) = 1.000/3.000
324   wq(2) = wq(1)
325   wq(3) = wq(1)
326 -----
327 CASE(4)
328   ALLOCATE(xiq(mint), etq(mint), wq(mint))
329   xiq = 0.000
330   etq = 0.000
331   wq = 0.000
332 -----
333   xiq(1) = 1.000/3.000
334   xiq(2) = 1.000/5.000
335   xiq(3) = 3.000/5.000
336   xiq(4) = 1.000/5.000
337   etq(1) = 1.000/3.000
338   etq(2) = 1.000/5.000
339   etq(3) = 1.000/5.000
340   etq(4) = 3.000/5.000
341   wq(1) = -27.000/48.000
342   wq(2) = 25.000/48.000
343   wq(3) = 25.000/48.000
344   wq(4) = 25.000/48.000
345 -----
346 CASE(6)
347   ALLOCATE(xiq(mint), etq(mint), wq(mint))
348   xiq = 0.000
349   etq = 0.000
350   wq = 0.000
351 -----
352   a1 = 0.81684757298045900
353   be = 0.4459488991596500
354   ga = 0.1881838181680700
355   de = 0.09157621350977100
356   o1 = 0.10995174365532200
357   o2 = 0.22338158967801100
358 -----
359   xiq(1) = de
360   xiq(2) = a1

```

```

361   xiq(3) = de
362   xiq(4) = be
363   xiq(5) = ga
364   xiq(6) = be
365   etq(1) = de
366   etq(2) = de
367   etq(3) = a1
368   etq(4) = be
369   etq(5) = be
370   etq(6) = ga
371   wq(1) = o1
372   wq(2) = o1
373   wq(3) = o1
374   wq(4) = o2
375   wq(5) = o2
376   wq(6) = o2
377 -----
378 CASE(7)
379   ALLOCATE(xiq(mint), etq(mint), wq(mint))
380   xiq = 0.000
381   etq = 0.000
382   wq = 0.000
383 -----
384   a1 = 0.7974269853308700
385   be = 0.47014206410511500
386   ga = 0.05971587178977000
387   de = 0.10128650732456000
388   o1 = 0.12593918054482700
389   o2 = 0.13239415278850600
390 -----
391   xiq(1) = de
392   xiq(2) = a1
393   xiq(3) = de
394   xiq(4) = be
395   xiq(5) = ga
396   xiq(6) = be
397   xiq(7) = 1.000/3.000
398   etq(1) = de
399   etq(2) = de
400   etq(3) = a1
401   etq(4) = be
402   etq(5) = be
403   etq(6) = ga
404   etq(7) = 1.000/3.000
405   wq(1) = o1
406   wq(2) = o1
407   wq(3) = o1
408   wq(4) = o2
409   wq(5) = o2
410   wq(6) = o2
411   wq(7) = 0.22500
412 -----
413 CASE(9)
414   ALLOCATE(xiq(mint), etq(mint), wq(mint))
415   xiq = 0.000
416   etq = 0.000
417   wq = 0.000
418 -----
419   a1 = 0.12494950323323200
420   qa = 0.16540992738984100
421   rh = 0.79711265186007100
422   de = 0.43752524838338400
423   ru = 0.03747742075008800
424   o1 = 0.20595050476088700
425   o2 = 0.06369141428622300
426 -----
427   xiq(1) = de
428   xiq(2) = a1
429   xiq(3) = de
430   xiq(4) = qa
431   xiq(5) = ru
432   xiq(6) = rh

```



```

433      xiq(7) = qa
434      xiq(8) = ru
435      xiq(9) = rh
436      etq(1) = de
437      etq(2) = de
438      etq(3) = al
439      etq(4) = ru
440      etq(5) = qa
441      etq(6) = qa
442      etq(7) = rh
443      etq(8) = rh
444      etq(9) = ru
445      wq(1) = o1
446      wq(2) = o1
447      wq(3) = o1
448      wq(4) = o2
449      wq(5) = o2
450      wq(6) = o2
451      wq(7) = o2
452      wq(8) = o2
453      wq(9) = o2
454 -----
455      CASE(12)
456      ALLOCATE(xiq(mint), etq(mint), wq(mint))
457      xiq = 0.000
458      etq = 0.000
459      wq = 0.000
460 -----
461      al = 0.87382197101699600
462      be = 0.24928674517091000
463      ga = 0.50142650965817900
464      de = 0.06308991449150000
465      rh = 0.63650249912139000
466      qa = 0.31035245103378500
467      ru = 0.05314584984481600
468      o1 = 0.05084499637020700
469      o2 = 0.11678627572637900
470      o3 = 0.00285107561837400
471 -----
472      xiq(1) = de
473      xiq(2) = al
474      xiq(3) = de
475      xiq(4) = be
476      xiq(5) = ga
477      xiq(6) = be
478      xiq(7) = qa
479      xiq(8) = ru
480      xiq(9) = rh
481      xiq(10) = qa
482      xiq(11) = ru
483      xiq(12) = rh
484 -----
485      etq(1) = de
486      etq(2) = de
487      etq(3) = al
488      etq(4) = be
489      etq(5) = be
490      etq(6) = ga
491      etq(7) = ru
492      etq(8) = qa
493      etq(9) = qa
494      etq(10) = rh
495      etq(11) = rh
496      etq(12) = ru
497 -----
498      wq(1) = o1
499      wq(2) = o1
500      wq(3) = o1
501      wq(4) = o2
502      wq(5) = o2
503      wq(6) = o2
504      wq(7) = o3

```

```

505      wq(8) = o3
506      wq(9) = o3
507      wq(10) = o3
508      wq(11) = o3
509      wq(12) = o3
510 -----
511      CASE(13)
512      ALLOCATE(xiq(mint), etq(mint), wq(mint))
513      xiq = 0.000
514      etq = 0.000
515      wq = 0.000
516 -----
517      al = 0.47930006784192300
518      be = 0.06513010290221600
519      ga = 0.86973979419556800
520      de = 0.26034596607903800
521      rh = 0.63844418856980900
522      qa = 0.31286549600087500
523      ru = 0.04869031542531600
524      o1 = 0.17561525743320400
525      o2 = 0.05334723560883900
526      o3 = 0.07711376089025700
527      o4 = 0.1495700446767000
528 -----
529      xiq(1) = de
530      xiq(2) = al
531      xiq(3) = de
532      xiq(4) = be
533      xiq(5) = ga
534      xiq(6) = be
535      xiq(7) = qa
536      xiq(8) = ru
537      xiq(9) = rh
538      xiq(10) = qa
539      xiq(11) = ru
540      xiq(12) = rh
541      xiq(13) = 1.000/3.000
542 -----
543      etq(1) = de
544      etq(2) = de
545      etq(3) = al
546      etq(4) = be
547      etq(5) = be
548      etq(6) = ga
549      etq(7) = ru
550      etq(8) = qa
551      etq(9) = qa
552      etq(10) = rh
553      etq(11) = rh
554      etq(12) = ru
555      etq(13) = 1.000/3.000
556 -----
557      wq(1) = o1
558      wq(2) = o1
559      wq(3) = o1
560      wq(4) = o2
561      wq(5) = o2
562      wq(6) = o2
563      wq(7) = o3
564      wq(8) = o3
565      wq(9) = o3
566      wq(10) = o3
567      wq(11) = o3
568      wq(12) = o3
569      wq(13) = o4
570 -----
571      CASE DEFAULT
572      mint = 13
573      ALLOCATE(xiq(mint), etq(mint), wq(mint))
574      xiq = 0.000
575      etq = 0.000
576      wq = 0.000

```

```
577 !-----
578      a1 = 0.47930806784192300
579      be = 0.0651301029021600
580      ga = 0.8697397941955000
581      de = 0.26034596607903800
582      rh = 0.63844418856980900
583      qa = 0.31286549600487500
584      ru = 0.04869081542531600
585      o1 = 0.17561525743330400
586      o2 = 0.05334723560883900
587      o3 = 0.07711376089025700
588      o4 = -0.1495700444676700
589      xiq(1) = de
590      xiq(2) = a1
591      xiq(3) = de
592      xiq(4) = be
593      xiq(5) = ga
594      xiq(6) = be
595      xiq(7) = qa
596      xiq(8) = ru
597      xiq(9) = rh
598      xiq(10) = qa
599      xiq(11) = ru
600      xiq(12) = rh
601      xiq(13) = 1.000/3.000
602 !-----
603      etq(1) = de
604      etq(2) = de
605      etq(3) = a1
606      etq(4) = be
607      etq(5) = be
608      etq(6) = ga
609      etq(7) = ru
610      etq(8) = qa
611      etq(9) = qa
612      etq(10) = rh
613      etq(11) = rh
614      etq(12) = ru
615      etq(13) = 1.000/3.000
616 !-----
617      wq(1) = o1
618      wq(2) = o1
619      wq(3) = o1
620      wq(4) = o2
621      wq(5) = o2
622      wq(6) = o2
623      wq(7) = o3
624      wq(8) = o3
625      wq(9) = o3
626      wq(10) = o3
627      wq(11) = o3
628      wq(12) = o3
629      wq(13) = o4
630 !-----
631      WRITE (Ulog,*)
632      WRITE (Ulog,*) ' Gauss_Trg1:'
633      WRITE (Ulog,*)
634      WRITE (Ulog,*) ' Number of Gauss triangle quadrature'
635      WRITE (Ulog,*) ' is not available; It was taken mint =13'
636      END SELECT
637      END SUBROUTINE Gauss_Trg1
638 !-----
639 END MODULE Mod_Gauss_Coefs
640
```

```

1 MODULE Mod_Builder_Matrix_Arrays
2 =====
3 !Version 1.0      07 / November / 2012
4 !
5 !-----
6 ! Alfredo Sanjuan Sanjuan, in others words, me :)
7 ! The first action is to generate the matrix form of the single and double integral potential. To obtain this
8 ! the program call the necessary subroutines. Then the code build the matrix system, and finally the system is
9 ! solved with the module Mod_Solution_Ax_b.
10 !-----
11 ! three rows at a time,
12 ! corresponding to the x, y, z
13 ! components of the integral equation
14 !-----
15 CONTAINS
16 =====
17 SUBROUTINE Builder_GM(GM, cdg, nelm, gamma, mint, NGL)
18 !-----
19 ! This subroutine builds the matrix array from the Stokeslets GE(x,>0).
20 !-----
21 USE Mod_SharedVars
22 USE Mod_Prtcl3D_SLP
23 USE Mod_sgf_3d_Fs
24 USE Mod_Prtcl3D_DLP
25 USE Mod_sgf_3d_sfs
26 USE mk195_precision
27 USE omp_lib
28 !-----
29 IMPLICIT NONE
30 !-----
31 ! Variables
32 !-----
33 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: GM !The matrix with the information of
34 !Stokeslets on the surface
35 REAL (KIND = DBL), INTENT(IN) :: cdg !coefficient of SLP
36 INTEGER, INTENT(IN) :: nelm !number of elements
37 REAL (KIND = DBL), INTENT(IN) :: gamma
38 INTEGER, INTENT(IN) :: mint, NGL !order of triangle and Gauss-Legendre quadratures
39 !-----
40 ! Variables inside the subroutine
41 !-----
42 INTEGER :: i, j !counters
43 INTEGER :: inel1, inel2
44 !-----
45 REAL (KIND = DBL) :: GEX, GEY, GEZ !Integrated ij component over the element
46 !-----
47 !The loop over collocation points serves to build the matrix form system
48 GM = 0.000
49 !!!$omp parallel do
50 !! Upper: DO i=1,nelm-1
51 !! inel1 = i + nelm
52 !! inel2 = i + nelm + nelm
53 !!-----
54 !! DO j = i+1, nelm
55 !!-----
56 !! The values of single layer potential are obtained in normal kind.
57 !! CALL Intgr_Trgl(x0(i), y0(i), z0(i), &
58 !! & x0(j), y0(j), z0(j), &
59 !! & j, &
60 !! & GEX, GEY, GEZ, &
61 !! & mint )
62 !!-----
63 !!
64 !! !CALL intr_lin_sing4(x0(i), y0(i), z0(i), &
65 !! ! & j, &
66 !! ! & GEX, GEY, GEZ)
67 !!-----
68 !! GM(i) = (GM(i) + GEX)
69 !! GM(inel1) = (GM(inel1) + GEY)
70 !! GM(inel2) = (GM(inel2) + GEZ)
71 !!-----
72 !! END DO

```

```

73 !-----
74 !! END DO Upper
75 !!!$omp end parallel
76 !-----
77 !!!$omp parallel do
78 !! Lower: DO i = 2, nelm
79 !! inel1 = i + nelm
80 !! inel2 = i + nelm + nelm
81 !! DO j = 1, i-1
82 !!-----
83 !! The values of single layer potential are obtained in normal kind.
84 !! CALL Intgr_Trgl(x0(i), y0(i), z0(i), &
85 !! & x0(j), y0(j), z0(j), &
86 !! & j, &
87 !! & GEX, GEY, GEZ, &
88 !! & mint )
89 !!-----
90 !!
91 !! !CALL intr_lin_sing4(x0(i), y0(i), z0(i), &
92 !! ! & j, &
93 !! ! & GEX, GEY, GEZ)
94 !!-----
95 !! The code will made the algebraic assembly of the matrix
96 !! GM(i) = (GM(i)+GEX)
97 !! GM(inel1) = (GM(inel1)+GEY)
98 !! GM(inel2) = (GM(inel2)+GEZ)
99 !!-----
100 !! END DO
101 !! END DO Lower
102 !!!$omp end parallel
103 !-----
104 !! The values of single layer potential are obtained with an interpolation.
105 !!!$omp parallel do
106 !! DO i=1,nelm
107 !! inel1 = i + nelm
108 !! inel2 = i + nelm + nelm
109 !!-----
110 !! CALL Intgr_Trgl_Sing(x0(i), y0(i), z0(i), &
111 !! & i, &
112 !! & GEX, GEY, GEZ, &
113 !! & mint )
114 !!-----
115 !!
116 !! !CALL intr_lin_sing4(x0(i), y0(i), z0(i), &
117 !! ! & i, &
118 !! ! & GEX, GEY, GEZ)
119 !!-----
120 !! The condition is over and then the code will made the algebraic assembly of the matrix
121 !! GM(i) = (GM(i)+GEX)
122 !! GM(inel1) = (GM(inel1)+GEY)
123 !! GM(inel2) = (GM(inel2)+GEZ)
124 !!-----
125 !! END DO
126 !!!$omp end parallel
127 !-----
128 !! Add the constant cdg to GM
129 !!$omp parallel
130 !!$omp do private(i, inel1, inel2, j, GEX, GEY, GEZ)
131 !! DO i=1, nelm
132 !! inel1 = i + nelm
133 !! inel2 = i + nelm + nelm
134 !!-----
135 !! DO j = 1, nelm
136 !! IF (i=j) THEN
137 !!-----
138 !! CALL Intgr_Trgl_Sing(x0(i), y0(i), z0(i), &
139 !! & i, &
140 !! & GEX, GEY, GEZ, &
141 !! & mint )
142 !!-----
143 !! !CALL intr_lin_sing4(x0(i), y0(i), z0(i), &

```

```

144 ! & i, &
145 ! & GEx, GEy, GEz)
146 -----
147 ELSE
148 -----
149 CALL Intrg_Trgl(x0(1), y0(1), z0(1), &
150 & x0(j), y0(j), z0(j), &
151 & j, &
152 & GEx, GEy, GEz, &
153 & mint)
154 -----
155 !CALL intr_lin_sing4(x0(1), y0(1), z0(1), &
156 ! & j, &
157 ! & GEx, GEy, GEz)
158 -----
159 END IF
160 -----
161 GM(1) = (GM(1) + GEx)
162 GM(inelm) = (GM(inelm) + GEy)
163 GM(inelm2) = (GM(inelm2) + GEz)
164 -----
165 END DO
166 -----
167 END DO
168 !$omp end do
169 !$omp end parallel
170 GM = cdt*gamma*GM
171 -----
172 END SUBROUTINE Builder_GM
173 -----
174 SUBROUTINE Builder_TM(TM, cdt, lmu, nelm, mint, NGL)
175 -----
176 ! This subroutine builds the matrix array from the Stresslets TE(x,x0).
177 -----
178 ! Variables
179 -----
180 ! TM ..... The matrix with the information of Stresslets on the surface
181 ! cdt ..... coefficient
182 ! lmu ..... coefficient lamdamu
183 ! nelm ..... number of elements
184 ! mint ..... order of Gauss' quadrature
185 ! NGL ..... order of triangle quadrature
186 ! x0, y0, z0 ..... coordinates of collocation point of the element
187 ! TExx, TEyy, TEzz ..... Integrated ij component over the element
188 ! TEyx, TEyz, TEzx .....
189 ! TEzx, TEzy, TEzz .....
190 -----
191 USE Mod_SharedVars
192 USE Mod_Prtcl3D_DLP
193 USE Mod_Sgf_3d_sfs
194 USE mkl05_precision
195 USE omp_lib
196 -----
197 IMPLICIT NONE
198 -----
199 ! Variables
200 -----
201 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:, :) :: TM !The matrix with the information of
!Stresslets on the surface
202 REAL (KIND = DBL), INTENT(IN) :: cdt, lmu !coefficients of DLP
203 INTEGER, INTENT(IN) :: nelm !number of elements
204 INTEGER, INTENT(IN) :: mint, NGL !order of triangle and Gauss-Legendre quadratures
205 -----
206 ! Variables inside the subroutine
207 -----
208 INTEGER :: i, j !counters
209 -----
210 INTEGER :: inelm, inelm2
211 INTEGER :: jnelm, jnelm2
212 -----
213 REAL (KIND = DBL) :: TExx, TEyy, TEzz, & !Integrated ij component over the element
& TEyx, TEyz, TEzy, &

```

```

216 & TEzx, TEzy, TEzz
217 -----
218 TM= 0.000
219 -----
220 !The loop over collocations points serves to build the matrix form sistem.
221 -----
222 IF(lmu==1.000)THEN
223 -----
224 !$omp parallel
225 !$omp do private(i, inelm, inelm2)
226 DO i= 1, nelm
227 inelm = i + nelm
228 inelm2= i + nelm + nelm
229 -----
230 ! The values of double layer potential are obtained in normal kind.
231 TM(1,i) = - ((1.000+lmu)/2.000)
232 TM(inelm,inelm) = - ((1.000+lmu)/2.000)
233 TM(inelm2,inelm2) = - ((1.000+lmu)/2.000)
234 END DO
235 !$omp end do
236 !$omp end parallel
237 -----
238 ELSE
239 -----
240 !$omp parallel
241 !$omp do private(i, inelm, inelm2, j, jnelm, jnelm2, TExx, TEyy, TEzz, TEyx, TEzy, TEyz, TEzx, TEzy, TEzz)
242 DO i= 1, nelm
243 inelm = i + nelm
244 inelm2= i + nelm + nelm
245 DO j = 1, nelm
246 jnelm = j + nelm
247 jnelm2 = j + nelm + nelm
248 -----
249 ! The values of double layer potential are obtained in normal kind.
250 IF(i==j)THEN
251 -----
252 CALL Intrg_Trgl_Sing_s(x0(j),y0(j),z0(j), &
253 & j, &
254 & TExx,TEyy,TEzz, &
255 & TEyx,TEyz,TEzy, &
256 & TEzx,TEzy,TEzz, &
257 & mint)
258 -----
259 !CALL intr_lin_sing_s4s(x0(1),y0(1),z0(1), &
260 ! & x0(i),y0(i),z0(i), &
261 ! & i, &
262 ! & TExx, TEyy, TEzz, &
263 ! & TEyx, TEyz, TEzy, &
264 ! & TEzx, TEzy, TEzz )
265 -----
266 TM(i,j) = (cdt*TExx) - ((1.000 + lmu)/2.000)
267 TM(j,inelm) = cdt*TEyy
268 TM(j,jnelm2) = cdt*TEzz
269 TM(jnelm,j) = cdt*TEyx
270 TM(jnelm,jnelm) = (cdt*TEyy) - ((1.000 + lmu)/2.000)
271 TM(jnelm,jnelm2) = cdt*TEyz
272 TM(jnelm2,j) = cdt*TEzx
273 TM(jnelm2,jnelm) = cdt*TEzy
274 TM(jnelm2,jnelm2) = (cdt*TEzz) - ((1.000 + lmu)/2.000)
275 -----
276 !TM(j,j) = ((0.0500*cdt*TExx) - ((1.000+lmu)/2.000))
277 !TM(j,jnelm) = 0.0500*cdt*TEzz
278 !TM(j,jnelm2) = 0.0500*cdt*TEyx
279 !TM(jnelm,j) = 0.0500*cdt*TEyy
280 !TM(jnelm,jnelm) = ((0.0500*cdt*TEyy) - ((1.000+lmu)/2.000))
281 !TM(jnelm,jnelm2) = 0.0500*cdt*TEyz
282 !TM(jnelm2,j) = 0.0500*cdt*TEzx
283 !TM(jnelm2,jnelm) = 0.0500*cdt*TEzy
284 !TM(jnelm2,jnelm2) = ((0.0500*cdt*TEzz) - ((1.000+lmu)/2.000))
285 -----
286 ELSE
287 -----

```

```
288 CALL Intrgr_Trg1_s(x0(i), y0(i), z0(i), &
289 & x0(j), y0(j), z0(j), &
290 & j, &
291 & TExx, TEyx, TEzx, &
292 & TEyy, TEyz, &
293 & TEzx, TEzy, TEzz, &
294 & mint)
-----
295 !CALL intr_lin_sing_s0(x0(j),y0(j),z0(j), &
296 ! & x0(i),y0(i),z0(i), &
297 ! & i, &
298 ! & TExx, TEyx, TEzx, &
299 ! & TEyy, TEyz, &
300 ! & TEzx, TEzy, TEzz )
-----
302 !
303 TM(i,j) = cdt*TExx
304 TM(i,jnelm) = cdt*TEyx
305 TM(i,jnelm2) = cdt*TEzx
306 TM(inelm,j) = cdt*TEyy
307 TM(inelm,jnelm) = cdt*TEyz
308 TM(inelm,jnelm2) = cdt*TEzz
309 TM(inelm2,j) = cdt*TExx
310 TM(inelm2,jnelm) = cdt*TEyx
311 TM(inelm2,jnelm2) = cdt*TEzx
-----
312 !
313 !TM(i,j) = cdt*TExx*1.5D0
314 !TM(i,jnelm) = cdt*TEyx*1.5D0
315 !TM(i,jnelm2) = cdt*TEzx*1.5D0
316 !TM(inelm,j) = cdt*TEyy*1.5D0
317 !TM(inelm,jnelm) = cdt*TEyz*1.5D0
318 !TM(inelm,jnelm2) = cdt*TEzz*1.5D0
319 !TM(inelm2,j) = cdt*TExx*1.5D0
320 !TM(inelm2,jnelm) = cdt*TEyx*1.5D0
321 !TM(inelm2,jnelm2) = cdt*TEzx*1.5D0
-----
322 !
323 END IF
324 END DO
325 END DO
326 !$omp end do
327 !$omp end parallel
328 !=====
329 END IF
330 !=====
331 END SUBROUTINE Builder_TM
332 !=====
333 END MODULE Mod_Builder_Matrix_Arrays
```

D:\Darth Vader\Escritorio\prtcl mkl\Mod Prtcl 3D slp.f90 1

```
1 MODULE Mod_Prtcl3D_SLP
2 =====
3 ! Version: 0.5 created on 26 / IX / 2007
4 !
5 ! C. Pozrikidis
6 =====
7 ! Version: 0.7 created on -- / III / 2010
8 !
9 ! Marco Antonio Reyes Huesca
10 =====
11 ! Version: 0.8 created on 22 / 03 / 2012
12 !
13 ! Version: 0.9 created on 03 / 09 / 2012
14 !
15 ! Version: 1.0 created on 12 / 11 / 2012
16 !
17 ! Alfredo Sanjuan Sanjuan
18 =====
19 CONTAINS
20 =====
21 SUBROUTINE Intr_Trgl(x0, y0, z0, &
22 & x, y, z, &
23 & xi, yi, zi, &
24 & k, &
25 & GEx, GEy, GEz, &
26 & mint)
27 =====
28 ! This subroutine integrates the Green's function over a non-singular triangle numbered k
29
30 USE Mod_SharedVars, ONLY: DBL, ULog, eps, Pi, &
31 & p, ne, n, nbe, are1, &
32 & alphaQ, betaQ, gammaQ, &
33 & xi0, etaQ, wq, &
34 & Ns, Np
35
36 USE Mod_sgf_3d_fs
37 USE Mod_Nodal_Interp
38
39 IMPLICIT NONE
40
41 ! Variables
42 =====
43 REAL (KIND = DBL), INTENT(IN) :: x0, y0, z0 !coordinates of collocation point of analysis
44 REAL (KIND = DBL), INTENT(IN) :: xi, yi, zi !coordinates of collocation point of the element
45 INTEGER, INTENT(IN) :: k !element index
46 REAL (KIND = DBL), INTENT(OUT) :: GEx, GEy, GEz !Integrated ij component over the element
47 INTEGER, INTENT(IN) :: mint !order of triangle quadrature
48
49 ! Variables inside the subroutine
50 =====
51 INTEGER :: i, j !counters
52 INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
53
54 REAL (KIND = DBL) :: xi, eta !variables to integrate over a triangle
55 REAL (KIND = DBL) :: x, y, z !coordinates of the f(x,y,z)= F(xi,eta)
56 REAL (KIND = DBL) :: DxDxi, DyDxi, DzDxi, & !Derivates of the tangential vectors over the element
57 & DxDet, DyDet, DzDet
58 REAL (KIND = DBL) :: vnx, vny, vnz !normal vector coordinates of the element
59 REAL (KIND = DBL) :: hs !surface metric on a triangle
60 REAL (KIND = DBL) :: fc !integration weigh coefficient
61 REAL (KIND = DBL) :: Gxx, Gxy, Gxz, & !free-space Green's function of Stokeslet. integrated ij
62 & Gyx, Gyy, Gyz, & !component over the element
63 & Gzx, Gzy, Gzz
64
65 ! Initialize
66 Gxx = 0.0D0
67 Gxy = 0.0D0
68 Gxz = 0.0D0
69 Gyx = 0.0D0
70 Gyy = 0.0D0
71 Gyz = 0.0D0
72 Gzx = 0.0D0
73 Gzy = 0.0D0
74 Gzz = 0.0D0
75
76 ! prepare the array
77 =====
```

D:\Darth Vader\Escritorio\prtcl mkl\Mod Prtcl 3D slp.f90 2

```
73 GEx = 0.0D0
74 GEy = 0.0D0
75 GEz = 0.0D0
76
77 ! vertices of the kth triangle
78 =====
79 i1 = n(k,1)
80 i2 = n(k,2)
81 i3 = n(k,3)
82 i4 = n(k,4)
83 i5 = n(k,5)
84 i6 = n(k,6)
85
86 DO i = 1, mint
87 xi = xiq(i)
88 eta = etq(i)
89 CALL intr_lin_sing2(p(i1,1), p(i1,2), p(i1,3), &
90 & p(i2,1), p(i2,2), p(i2,3), &
91 & p(i3,1), p(i3,2), p(i3,3), &
92 & p(i4,1), p(i4,2), p(i4,3), &
93 & p(i5,1), p(i5,2), p(i5,3), &
94 & p(i6,1), p(i6,2), p(i6,3), &
95 & alphaQ(k), betaQ(k), gammaQ(k), &
96 & xi, eta, &
97 & x, y, z, &
98 & DxDxi, DyDxi, DzDxi, &
99 & DxDet, DyDet, DzDet, &
100 & vnx, vny, vnz, &
101 & hs)
102
103 !! Call subroutine sgf_fs
104
105 CALL sgf_3d_fs(x, y, z, &
106 & x0, y0, z0, &
107 & Gxx, Gxy, Gxz, &
108 & Gyx, Gyy, Gyz, &
109 & Gzx, Gzy, Gzz )
110
111 !! Computes the integral
112
113 fc = wq(1)*hs*0.5D0
114
115 GEx = GEx + (Gxx*vnx+Gxy*vny+Gxz*vnz)*fc
116 GEy = GEy + (Gxy*vnx+Gyy*vny+Gyz*vnz)*fc
117 GEz = GEz + (Gxz*vnx+Gyz*vny+Gzz*vnz)*fc
118
119 END DO
120
121 CALL intr_lin_sing2(x0, y0, z0, &
122 & x, y, z, &
123 & p(i1,1),p(i1,2),p(i1,3), &
124 & p(i4,1),p(i4,2),p(i4,3), &
125 & k, mint, &
126 & GEx, GEy, GEz)
127
128 CALL intr_lin_sing2(x0, y0, z0, &
129 & x, y, z, &
130 & p(i4,1),p(i4,2),p(i4,3), &
131 & p(i2,1),p(i2,2),p(i2,3), &
132 & k, mint, &
133 & GEx, GEy, GEz)
134
135 CALL intr_lin_sing2(x0, y0, z0, &
136 & x, y, z, &
137 & p(i2,1),p(i2,2),p(i2,3), &
138 & p(i5,1),p(i5,2),p(i5,3), &
139 & k, mint, &
140 & GEx, GEy, GEz)
141
142 CALL intr_lin_sing2(x0, y0, z0, &
143 & x, y, z, &
144 & p(i5,1),p(i5,2),p(i5,3), &
```



```

145      & p(13,1),p(13,2),p(13,3), &
146      & k, mint, &
147      & GEx, GEy, GEz)
-----
149 CALL intr_lin_sing2( x0, y0, z0, &
150      & x, y, z, &
151      & p(13,1),p(13,2),p(13,3), &
152      & p(16,1),p(16,2),p(16,3), &
153      & k, mint, &
154      & GEx, GEy, GEz)
-----
156 CALL intr_lin_sing2( x0, y0, z0, &
157      & x, y, z, &
158      & p(16,1),p(16,2),p(16,3), &
159      & p(11,1),p(11,2),p(11,3), &
160      & k, mint, &
161      & GEx, GEy, GEz)
-----
163 ! Done
164 !
165 END SUBROUTINE Intgr_Trgl
-----
167 USE Mod_SharedVars, ONLY: DBL, ULog, eps, &
168      & P1, p, ne, n, nbe, &
169      & arel, crvmel, &
170      & xiq, etq, wq, &
171      & Ns, Np
-----
173 ! Integrate the green's function over the kth singular quadratic triangle.
174 ! This is done by breaking up the SINGULAR triangle into six flat triangles, and THEN integrating individually
175 ! over the flat triangles in local polar coordinates. The singularity
-----
177 USE Mod_SharedVars, ONLY: DBL, ULog, eps, &
178      & P1, p, ne, n, nbe, &
179      & arel, crvmel, &
180      & xiq, etq, wq, &
181      & Ns, Np
-----
183 IMPLICIT NONE
-----
185 REAL (KIND = DBL), INTENT(IN) :: x0, y0, z0 !coordinates of collocationpoint of the element
186 INTEGER, INTENT(IN) :: k !element index
187 REAL (KIND = DBL), INTENT(OUT) :: GEx, GEy, GEz !Integrated ij component over the element
188 INTEGER, INTENT(IN) :: mint !order of triangle quadrature
-----
190 ! Variables inside the subroutine
-----
192 INTEGER :: i, j ! i, j ..... Counters
193 INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
194 ! INTEGER :: mint !order of triangle quadrature
-----
196 ! Obtain the information of the nodes in each element.
-----
198 i1 = n(k,1)
199 i2 = n(k,2)
200 i3 = n(k,3)
201 i4 = n(k,4)
202 i5 = n(k,5)
203 i6 = n(k,6)
-----
205 ! Initialize the array
206 GEx = 0.0D0
207 GEy = 0.0D0
208 GEz = 0.0D0
-----
210 ! Integrate over three flat triangles
-----
212 CALL intr_lin_sing( x0, y0, z0, &
213      & p(i1,1),p(i1,2),p(i1,3), &
214      & p(i2,1),p(i2,2),p(i2,3), &
215      & GEx, GEy, GEz, &
216      & GEyx, GEyy, GEyz, &

```

```

217      & GEzx, GEzy, GEzz, &
218      & NGL
-----
220 CALL intr_lin_sing( x0, y0, z0, &
221      & p(i2,1),p(i2,2),p(i2,3), &
222      & p(i3,1),p(i3,2),p(i3,3), &
223      & GEx, GEy, GEz, &
224      & GEyx, GEyy, GEyz, &
225      & GEzx, GEzy, GEzz, &
226      & NGL
-----
228 CALL intr_lin_sing( x0, y0, z0, &
229      & p(i3,1),p(i3,2),p(i3,3), &
230      & p(i1,1),p(i1,2),p(i1,3), &
231      & GEx, GEy, GEz, &
232      & GEyx, GEyy, GEyz, &
233      & GEzx, GEzy, GEzz, &
234      & NGL
-----
236 ! Integrate over six flat triangles
-----
238 CALL intr_lin_sing( x0, y0, z0, &
239      & p(i1,1),p(i1,2),p(i1,3), &
240      & p(i4,1),p(i4,2),p(i4,3), &
241      & k, &
242      & GEx, GEy, GEz, &
243      & GEyx, GEyy, GEyz, &
244      & GEzx, GEzy, GEzz, &
245      & mint
-----
247 CALL intr_lin_sing( x0, y0, z0, &
248      & p(i4,1),p(i4,2),p(i4,3), &
249      & p(i2,1),p(i2,2),p(i2,3), &
250      & k, &
251      & GEx, GEy, GEz, &
252      & GEyx, GEyy, GEyz, &
253      & GEzx, GEzy, GEzz, &
254      & mint
-----
256 CALL intr_lin_sing( x0, y0, z0, &
257      & p(i2,1),p(i2,2),p(i2,3), &
258      & p(i5,1),p(i5,2),p(i5,3), &
259      & k, &
260      & GEx, GEy, GEz, &
261      & GEyx, GEyy, GEyz, &
262      & GEzx, GEzy, GEzz, &
263      & mint
-----
264 CALL intr_lin_sing( x0, y0, z0, &
265      & p(i5,1),p(i5,2),p(i5,3), &
266      & p(i3,1),p(i3,2),p(i3,3), &
267      & k, &
268      & GEx, GEy, GEz, &
269      & GEyx, GEyy, GEyz, &
270      & GEzx, GEzy, GEzz, &
271      & mint
-----
274 CALL intr_lin_sing( x0, y0, z0, &
275      & p(i3,1),p(i3,2),p(i3,3), &
276      & p(i6,1),p(i6,2),p(i6,3), &
277      & k, &
278      & GEx, GEy, GEz, &
279      & GEyx, GEyy, GEyz, &
280      & GEzx, GEzy, GEzz, &
281      & mint
-----
283 CALL intr_lin_sing( x0, y0, z0, &
284      & p(i6,1),p(i6,2),p(i6,3), &
285      & p(i1,1),p(i1,2),p(i1,3), &
286      & k, &
287      & GEx, GEy, GEz, &
288      & GEyx, GEyy, GEyz, &

```

```

289      & GEZx, GEzy, GEZz, &
290      & mint
291 -----
292      IGEy = GEy*crvml(k)
293      IGEz = GEz*crvml(k)
294 -----
295      END SUBROUTINE Intr_TrgL_Sing
296 -----
297 -----
298 -----
299 SUBROUTINE intr_lin_sing(x1, y1, z1, &
300      & x2, y2, z2, &
301      & x3, y3, z3, &
302      & k,
303      & GEz, GEy, GEz, &
304      & mint)
305 -----
306 ! Integrates the Green's function over a flat triangle in local polar coordinates with origin at singular
307 ! point: (x1,y1,z1). The subroutine is based from Pozriquidis 2002 pp. 119-120 in the technique 5.2.8 - 5.2.11
308 -----
309 USE Mod_SharedVars, ONLY: DBL, ULog, eps, Pi, &
310      & ZZ, Ww,
311      & Hs, Np, crvml, xiq, etq, wq
312 -----
313 USE Mod_sgf_3d_fs
314 -----
315 IMPLICIT NONE
316 -----
317 ! Variables
318 -----
319 REAL (KIND = DBL), INTENT(IN) :: x1, y1, z1 !coordinates of flat triangle vertice
320 REAL (KIND = DBL), INTENT(IN) :: x2, y2, z2
321 REAL (KIND = DBL), INTENT(IN) :: x3, y3, z3
322 INTEGER, INTENT(IN) :: k !element index
323 REAL (KIND = DBL), INTENT(INOUT) :: GEz, GEy, GEz !Integrated ij component over the element
324 INTEGER, INTENT(IN) :: mint !order of Gauss-Legendre quadrature
325 -----
326 ! Variables inside the subroutine
327 -----
328 INTEGER :: i, j !counters
329 INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain the node number in the vertices of each element
330 -----
331 REAL (KIND = DBL) :: pi4, piq !constants to integrate over a circle element in Phi and r
332 REAL (KIND = DBL) :: xi, et, zt !constants of weigh to integrate over a flat triangle
333 REAL (KIND = DBL) :: dx, dy !surface metrics
334 REAL (KIND = DBL) :: vnx, vny, vnz !triangle normal vector
335 REAL (KIND = DBL) :: area, hs !triangle area and surface metric on a flat triangle
336 REAL (KIND = DBL) :: ph, cph, sph !variable Phi, Cos(phi) and Sin(phi)
337 REAL (KIND = DBL) :: r, rmax, rmaxh !variable r and values of variable r to integrate
338 REAL (KIND = DBL) :: x, y, z !coordinates of f(x1, et, zt)
339 REAL (KIND = DBL) :: cf, cf1, cf2 !integrate weights
340 REAL (KIND = DBL) :: asm, bsm !triangle area computed by numerical integration and its derivate
341 REAL (KIND = DBL) :: B, C, ABC !integrate arc coefficients
342 REAL (KIND = DBL) :: b1, b2, c1 !dummy integrate arc constants
343 REAL (KIND = DBL), DIMENSION(3) ::xxi, eet, zzt !variables of weigh over in triangle (xi,eta)
344 REAL (KIND = DBL) :: Gxx, Gxy, Gxz, & !free-space Green's function of Stokeslet. integrated ij component
345      & Gyx, Gyy, Gyz, &
346      & Gzx, Gzy, Gzz
347 REAL (KIND = DBL) :: Rxx, Rxy, Rxz, & !Stokeslets Dummy coefficients
348      & Ryx, Ryy, Ryz, &
349      & Rzx, Rzy, Rzz
350 REAL (KIND = DBL) :: Mxx, Mxy, Mxz, & !Stokeslets Dummy coefficients
351      & Myx, Myy, Myz, &
352      & Mzx, Mzy, Mzz
353 -----
354 ! constants
355 -----
356 piq = 0.25D0*pi
357 pi4 = 4.0D0*pi
358 -----
359 ! Initialize
360 xxi(1) = 0.5D0

```

```

361 eet(1) = 0.5D0
362 zzt(1) = 0.0D0
363 xxi(2) = 0.5D0
364 eet(2) = 0.0D0
365 zzt(2) = 0.5D0
366 xxi(3) = 0.0D0
367 eet(3) = 0.5D0
368 zzt(3) = 0.5D0
369 -----
370 lxxi(1) = 1.00D / 6.00D
371 leet(1) = 1.00D / 6.00D
372 lzzt(1) = 2.00D / 3.00D
373 lxxi(2) = 2.00D / 3.00D
374 leet(2) = 1.00D / 6.00D
375 lzzt(2) = 1.00D / 6.00D
376 lxxi(3) = 1.00D / 6.00D
377 leet(3) = 2.00D / 3.00D
378 lzzt(3) = 1.00D / 6.00D
379 -----
380 Gxx = 0.0D0
381 Gxy = 0.0D0
382 Gxz = 0.0D0
383 Gyx = 0.0D0
384 Gyy = 0.0D0
385 Gyz = 0.0D0
386 Gzx = 0.0D0
387 Gzy = 0.0D0
388 Gzz = 0.0D0
389 -----
390 Rxx = 0.0D0
391 Rxy = 0.0D0
392 Rxz = 0.0D0
393 Ryx = 0.0D0
394 Ryy = 0.0D0
395 Ryz = 0.0D0
396 Rzx = 0.0D0
397 Rzy = 0.0D0
398 Rzz = 0.0D0
399 -----
400 Mxx = 0.0D0
401 Mxy = 0.0D0
402 Mxz = 0.0D0
403 Myx = 0.0D0
404 Myy = 0.0D0
405 Myz = 0.0D0
406 Mzx = 0.0D0
407 Mzy = 0.0D0
408 Mzz = 0.0D0
409 -----
410 ! compute surface metric: hs
411 -----
412 dx = DSQRT( ((x2-x1)**2)+((y2-y1)**2)+((z2-z1)**2) )
413 dy = DSQRT( ((x3-x1)**2)+((y3-y1)**2)+((z3-z1)**2) )
414 -----
415 vnx = ((y2-y1)*(z3-z1))-((z2-z1)*(y3-y1))
416 vny = ((z2-z1)*(x3-x1))-((x2-x1)*(z3-z1))
417 vnz = ((x2-x1)*(y3-y1))-((y2-y1)*(x3-x1))
418 -----
419 hs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
420 vnx = vnx/hs
421 vny = vny/hs
422 vnz = vnz/hs
423 -----
424 ! Surface metric on a flat triangle
425 area = hs/2.0D0
426 -----
427 ! Initialize
428 -----
429 ! Triangle area
430 asm = 0.0D0
431 -----
432 !!

```

```

433 !! Apply the forulas from Pozrikidis (2002, p. 119)
434 ! b1=0.000
435 ! b2=0.000
436 ! c1=0.000
437 ! ABC=0.000
438 ! B=0.000
439 ! C=0.000
440 ! b1=((x3-x1)*(x2-x1))+((y3-y1)*(y2-y1))+((z3-z1)*(z2-z1))
441 ! b2=DSQRT((x2-x1)**2+(y2-y1)**2+(z2-z1)**2)
442 ! c1=DSQRT((x3-x1)**2+(y3-y1)**2+(z3-z1)**2)
443 ! B= b1/b2**2
444 ! C= c1**2/b2**2
445 !-----
446 !! Apply the double quadrature
447 !-----
448 !! Integration wrt phi
449 !-----
450 ! Cycle1: DO i = 1, NGL
451 !   phi = pi*(1.000 + zz(i))
452 !   cph = DCOS(phi)
453 !   sph = DSIN(phi)
454 !   rmax = 1.000/(cph*sph)
455 !   rmaxh = 0.500*rmax
456 !   ABC= (cph**2 + (B*DSIN(2*phi)) + (C*(sph**2)))
457 !-----
458 !! Derivative of asm
459 ! bsm = 0.000
460 !-----
461 !! Integration wrt r
462 ! Cycle2: DO j=1,NGL
463 !   r = rmaxh*(1.000+zz(j))
464 !   x1 = r*cph
465 !   et = r*sph
466 !   zt = 1.000-x1-et
467 !-----
468 !   x = x1*zt + x2*x1 + x3*et
469 !   y = y1*zt + y2*x1 + y3*et
470 !   z = z1*zt + z2*x1 + z3*et
471 !-----
472 !   CALL sgf_3d_fsing(x, y, z, &
473 !     & x1, y1, z1, &
474 !     & Mxx, Myx, Mxz, &
475 !     & Myx, Myy, Myz, &
476 !     & Mzx, Mzy, Mzz, &
477 !     & ABC)
478 !   IWRITE(*,*) Mxx
479 !   IWRITE(*,*) Mxx
480 !-----
481 !   cf1 = ww(j)*r
482 !   IWRITE(*,*) cf1
483 !   IWRITE(*,*) cf1
484 !-----
485 !   bsm = bsm + cf1
486 !-----
487 !   Rxx = Rxx + cf1*Mxx
488 !   Rxy = Rxy + cf1*Mxy
489 !   Rxz = Rxz + cf1*Mxz
490 !   Ryx = Ryx + cf1*Myx
491 !   Ryy = Ryy + cf1*Myy
492 !   Ryz = Ryz + cf1*Myz
493 !   Rzx = Rzx + cf1*Mzx
494 !   Rzy = Rzy + cf1*Mzy
495 !   Rzz = Rzz + cf1*Mzz
496 !-----
497 !   END DO Cycle2
498 !-----
499 !   cf2 = ww(1)*rmaxh
500 !-----
501 !   IWRITE(*,*) cf2
502 !   IWRITE(*,*) cf2
503 !-----
504 !   asm = asm + bsm*cf2

```

```

505 !-----
506 !   Gxx = Gxx + (cf2*Rxx/DSQRT(ABC))
507 !   Gxy = Gxy + (cf2*Rxy/DSQRT(ABC))
508 !   Gxz = Gxz + (cf2*Rxz/DSQRT(ABC))
509 !   Gyx = Gyx + (cf2*Ryx/DSQRT(ABC))
510 !   Gyy = Gyy + (cf2*Ryy/DSQRT(ABC))
511 !   Gyz = Gyz + (cf2*Ryz/DSQRT(ABC))
512 !   Gzx = Gzx + (cf2*Rzx/DSQRT(ABC))
513 !   Gzy = Gzy + (cf2*Rzy/DSQRT(ABC))
514 !   Gzz = Gzz + (cf2*Rzz/DSQRT(ABC))
515 !-----
516 !! initialize again to the next triangle
517 !   Rxx = 0.000
518 !   Rxy = 0.000
519 !   Rxz = 0.000
520 !   Ryx = 0.000
521 !   Ryy = 0.000
522 !   Ryz = 0.000
523 !   Rzx = 0.000
524 !   Rzy = 0.000
525 !   Rzz = 0.000
526 !-----
527 !! END DO Cycle1
528 !-----
529 !! complete the quadrature
530 ! cf = hs/piq
531 !-----
532 !! asm = asm*cf
533 ! GEX = GEX + (cf*(Gxx*vnx+Gyx*vny+Gzx*vmz))/b2
534 ! GEY = GEY + (cf*(Gxy*vnx+Gyy*vny+Gzy*vmz))/b2
535 ! GEZ = GEZ + (cf*(Gxz*vnx+Gyz*vny+Gzz*vmz))/b2
537 !-----
538 !-----
539 !   xi = 1.000/3.000 !r*cph
540 !   et = 1.000/3.000 !r*sph
541 !   zt = 1.000/3.000 !1.000-x1-et
542 !-----
543 !   x = x1*zt + x2*x1 + x3*et
544 !   y = y1*zt + y2*x1 + y3*et
545 !   z = z1*zt + z2*x1 + z3*et
546 !-----
547 !   CALL sgf_3d_fs(x, y, z, &
548 !     & x1, y1, z1, &
549 !     & Mxx, Myx, Mxz, &
550 !     & Myx, Myy, Myz, &
551 !     & Mzx, Mzy, Mzz, )
552 !-----
553 !   cf1 = 1.000 !ww(j)*r
554 !-----
555 !   bsm = bsm + cf1
556 !-----
557 !   Rxx = Rxx + cf1*Mxx
558 !   Rxy = Rxy + cf1*Mxy
559 !   Rxz = Rxz + cf1*Mxz
560 !   Ryx = Ryx + cf1*Myx
561 !   Ryy = Ryy + cf1*Myy
562 !   Ryz = Ryz + cf1*Myz
563 !   Rzx = Rzx + cf1*Mzx
564 !   Rzy = Rzy + cf1*Mzy
565 !   Rzz = Rzz + cf1*Mzz
566 !-----
567 !   cf2 = 1.000 !ww(1)*rmaxh
568 !-----
569 !   asm = asm + bsm*cf2
570 !-----
571 !   Gxx = Gxx + (cf2*Rxx) !((cf2*Rxx/DSQRT(cph**2+(B*DSIN(2*phi)))+(C*sph**2)))
572 !   Gxy = Gxy + (cf2*Rxy) !((cf2*Rxy/DSQRT(cph**2+(B*DSIN(2*phi)))+(C*sph**2)))
573 !   Gxz = Gxz + (cf2*Rxz) !((cf2*Rxz/DSQRT(cph**2+(B*DSIN(2*phi)))+(C*sph**2)))
574 !   Gyx = Gyx + (cf2*Ryx) !((cf2*Ryx/DSQRT(cph**2+(B*DSIN(2*phi)))+(C*sph**2)))
575 !   Gyy = Gyy + (cf2*Ryy) !((cf2*Ryy/DSQRT(cph**2+(B*DSIN(2*phi)))+(C*sph**2)))
576 !   Gyz = Gyz + (cf2*Ryz) !((cf2*Ryz/DSQRT(cph**2+(B*DSIN(2*phi)))+(C*sph**2)))

```

```

577 ! Gzx = Gzx + (cf2*Rzx) !((cf2*Rzx/DSQRT(cph**2+(B*DSIN(2*ph)))+(C*sph**2)))
578 ! Gzy = Gzy + (cf2*Rzy) !((cf2*Rzy/DSQRT(cph**2+(B*DSIN(2*ph)))+(C*sph**2)))
579 ! Gzz = Gzz + (cf2*Rzz) !((cf2*Rzz/DSQRT(cph**2+(B*DSIN(2*ph)))+(C*sph**2)))
580 !-----
581 ! initialize again to the next triangle
582 ! Rxx = 0.000
583 ! Rxy = 0.000
584 ! Rxz = 0.000
585 ! Ryx = 0.000
586 ! Ryy = 0.000
587 ! Ryz = 0.000
588 ! Rzx = 0.000
589 ! Rzy = 0.000
590 ! Rzz = 0.000
591 !-----
592 !-----
593 DO i = 1, mint
594   xi = xiq(i)
595   et = etq(i)
596   zt = 1.000/3.000 * wq(i)
597   cf1 = wq(i)
598 !-----
599 ! DO i = 1, 3
600 !   xi = xxi(i)
601 !   et = eet(i)
602 !   zt = zzt(i)
603 !   cf1 = 1.000/3.000 * wq(i)
604 !-----
605   x = xi*zt + x2*xi + x3*et
606   y = y1*zt + y2*xi + y3*et
607   z = z1*zt + z2*xi + z3*et
608 !-----
609   CALL sgf_3d_fs(x, y, z, &
610     & x1, y1, z1, &
611     & Rxx, Rxy, Rxz, &
612     & Ryx, Ryy, Ryz, &
613     & Rzx, Rzy, Rzz )
614 !-----
615 !   cf1 = 1.000/3.000
616 !-----
617   bsm=0.000
618   bsm=cf1
619 !-----
620   Rxx = Rxx*cf1
621   Rxy = Rxy*cf1
622   Rxz = Rxz*cf1
623   Ryx = Ryx*cf1
624   Ryy = Ryy*cf1
625   Ryz = Ryz*cf1
626   Rzx = Rzx*cf1
627   Rzy = Rzy*cf1
628   Rzz = Rzz*cf1
629 !-----
630   asm = asm + bsm
631 !-----
632   Gxx = Gxx + Rxx
633   Gxy = Gxy + Rxy
634   Gxz = Gxz + Rxz
635   Gyx = Gyx + Ryx
636   Gyy = Gyy + Ryy
637   Gyz = Gyz + Ryz
638   Gzx = Gzx + Rzx
639   Gzy = Gzy + Rzy
640   Gzz = Gzz + Rzz
641 !-----
642   END DO
643 !-----
644 ! complete the quadrature
645 !-----
646   cf = area
647 !-----
648   asm = asm*cf

```

```

649 !-----
650 GEX = GEX + cf*(Gxx*vnx+Gxy*vny+Gzx*vnz) *crvml(k)
651 GEY = GEY + cf*(Gxy*vnx+Gyy*vny+Gzy*vnz) *crvml(k)
652 GEZ = GEZ + cf*(Gxz*vnx+Gyz*vny+Gzz*vnz) *crvml(k)
653 !-----
654 !-----
655 ! IF all went well, asm should be equal to "area"
656 ! WRITE (Ulog,100) 1,area,asm
657 !-----
658 ! 100 FORMAT (1x,i3,2(f10.5))
659 !-----
660 END SUBROUTINE intr_lin_sing
661 !-----
662 SUBROUTINE intr_lin_sing2(x0, y0, z0, &
663   & x1, y1, z1, &
664   & x2, y2, z2, &
665   & x3, y3, z3, &
666   & k, mint, &
667   & GEX, GEY, GEZ)
668 !-----
669 ! Integrates the Green's function over a flat triangle in local polar coordinates with origin at singular
670 ! point: (x1,y1,z1). The subroutine is based from Pozriquidis 2002 pp. 119-120 in the technique 5.2.8 - 5.2.11
671 !-----
672 USE Mod_SharedVars, ONLY: DBL, ULog, eps, Pi, &
673   & Ns, Np, crvml, xiq, etq, wq
674 !-----
675 USE Mod_sgf_3d_fs
676 !-----
677 IMPLICIT NONE
678 !-----
679 ! Variables
680 REAL (KIND = DBL), INTENT(IN) :: x0, y0, z0
681 REAL (KIND = DBL), INTENT(IN) :: x1, y1, z1 !coordinates of flat triangle vertice
682 REAL (KIND = DBL), INTENT(IN) :: x2, y2, z2
683 REAL (KIND = DBL), INTENT(IN) :: x3, y3, z3
684 REAL (KIND = DBL), INTENT(IN) :: k !element index
685 INTEGER, INTENT(IN) :: k !element index
686 INTEGER, INTENT(IN) :: mint !integrated ij component over the element
687 REAL (KIND = DBL), INTENT(INOUT) :: GEX, GEY, GEZ
688 & GEyx, GEyy, GEyz, &
689 & GEzx, GEzy, GEzz !coordinates of collocationpoint of the element
690 !REAL (KIND = DBL), INTENT(IN) :: a, b, c
691 ! Variables inside the subroutine
692 !-----
693 INTEGER :: i, j !counters
694 REAL (KIND = DBL) :: x1, et, zt !constants of weight to integrate over a flat triangle
695 REAL (KIND = DBL) :: dx, dy !surface metrics
696 REAL (KIND = DBL) :: vnx, vny, vnz !triangle normal vector
697 REAL (KIND = DBL) :: area, hs !triangle area and surface metric on a flat triangle
698 REAL (KIND = DBL) :: x, y, z !coordinates of f(x1, et, zt)
699 REAL (KIND = DBL) :: cf, cf1, cf2 !integrate weights
700 REAL (KIND = DBL) :: asm, bsm !triangle area computed by numerical integration and its derivate
701 REAL (KIND = DBL), DIMENSION(3) :: xxi, eet, zzt !variables of weight over in triangle (xi,eta)
702 REAL (KIND = DBL) :: Gxx, Gxy, Gxz, & !Free-space Green's function of Stokeslet. integrated ij component
703 & Gyx, Gyy, Gyz, & !over the element
704 & Gzx, Gzy, Gzz
705 REAL (KIND = DBL) :: Rxx, Rxy, Rxz, & !Stokeslets Dummy coefficients
706 & Ryx, Ryy, Ryz, &
707 & Rzx, Rzy, Rzz
708 !-----
709 ! Initialize
710 xxi(1) = 0.500
711 eet(1) = 0.500
712 zzt(1) = 0.000
713 xxi(2) = 0.500
714 eet(2) = 0.000
715 zzt(2) = 0.500
716 xxi(3) = 0.000
717 eet(3) = 0.500
718 zzt(3) = 0.500
719 !-----
720 !

```

```

721 Gxx = 0.000
722 Gxy = 0.000
723 Gxz = 0.000
724 Gyx = 0.000
725 Gyy = 0.000
726 Gyz = 0.000
727 Gzx = 0.000
728 Gzy = 0.000
729 Gzz = 0.000
-----
730
731 Rxx = 0.000
732 Rxy = 0.000
733 Rxz = 0.000
734 Ryx = 0.000
735 Ryy = 0.000
736 Ryz = 0.000
737 Rzx = 0.000
738 Rzy = 0.000
739 Rzz = 0.000
740
741
742 ! compute surface metric: hs
743
744 dx = DSQRT( (x2-x1)**2+(y2-y1)**2+(z2-z1)**2 )
745 dy = DSQRT( (x3-x1)**2+(y3-y1)**2+(z3-z1)**2 )
746
747 vnx = (y2-y1)*(z3-z1)-(z2-z1)*(y3-y1)
748 vny = (z2-z1)*(x3-x1)-(x2-x1)*(z3-z1)
749 vnz = (x2-x1)*(y3-y1)-(y2-y1)*(x3-x1)
750
751 hs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
752 vnx = vnx/hs
753 vny = vny/hs
754 vnz = vnz/hs
755
756 ! Surface metric on a flat triangle
757 area = hs/2.000
758
759 ! Initialize
760
761 ! Triangle area
762 asm = 0.000
763
764 DO i = 1, mint
765   xi = xiq(i)
766   et = etq(i)
767   zt = 1.000 - xi - et
768   cf1 = wq(i)
769
770   x = xi*zt + x2*x1 + x3*et
771   y = yi*zt + y2*x1 + y3*et
772   z = zi*zt + z2*x1 + z3*et
773
774   CALL sgf_3d_fs(x, y, z, &
775     & x0, y0, z0, &
776     & Rxx, Rxy, Rxz, &
777     & Ryx, Ryy, Ryz, &
778     & Rzx, Rzy, Rzz )
779
780   cf1 = 1.000/3.000
781
782   bsm=0.000
783   bsm=cf1
784
785   Rxx = Rxx*cf1
786   Rxy = Rxy*cf1
787   Rxz = Rxz*cf1
788   Ryx = Ryx*cf1
789   Ryy = Ryy*cf1
790   Ryz = Ryz*cf1
791   Rzx = Rzx*cf1
792   Rzy = Rzy*cf1

```

```

793 Rzz = Rzz*cf1
794
795 asm = asm + bsm
796
797 Gxx = Gxx + Rxx
798 Gxy = Gxy + Rxy
799 Gxz = Gxz + Rxz
800 Gyx = Gyx + Ryx
801 Gyy = Gyy + Ryy
802 Gyz = Gyz + Ryz
803 Gzx = Gzx + Rzx
804 Gzy = Gzy + Rzy
805 Gzz = Gzz + Rzz
806
807 END DO
808
809 ! complete the quadrature
810
811 cf = area
812
813 asm = asm*cf
814
815 GE = GE + cf*(Gxx*vnx+Gxy*vny+Gxz*vnz) *crvml(k)
816 GEY = GEY + cf*(Gyx*vnx+Gyy*vny+Gzy*vnz) *crvml(k)
817 GEZ = GEZ + cf*(Gxz*vnx+Gyz*vny+Gzz*vnz) *crvml(k)
818
819 ! IF all went well, asm should be equal to "area"
820 WRITE (Ulog,100) i,area,asm
821
822 ! 100 FORMAT (1x,13,2(f10.5))
823
824 END SUBROUTINE intr_lin_sing2
825
826 SUBROUTINE intr_lin_sing3(x0, y0, z0, &
827   & k, &
828   & GE, GEY, GEZ )
829
830 ! This subroutine is a new version stokeslet Subroutine.
831 ! Compute:
832 ! *The value of the Stokeslet over each singular element
833 ! Now, (March/ 09 / 2015) this subroutine was made.
834
835 USE Mod_Nodal_Interp
836 USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog,&
837   & alpha0, beta0, gamma0, &
838   & vnx0, vny0, vnz0, &
839   & ZI, WQ, &
840   & xiq, etq, wq, crvml
841
842 IMPLICIT NONE
843
844 ! Variables
845
846 REAL (KIND = DBL), INTENT(IN) :: x0, y0, z0 !singularity coordinates
847 INTEGER, INTENT(IN) :: k !number of element
848 REAL (KIND = DBL), INTENT(OUT) :: GE, GEY, GEZ !value of stokeslet in the singular element
849
850 ! Variables inside the subroutine
851
852 INTEGER :: i, j !counters
853 INTEGER :: i1, i2, i3, i4, i5, i6 !Indices to obtain node numbers from each element
854
855 REAL (KIND = DBL) :: xi, eta !variables of weigth to integrate over a triangle
856 REAL (KIND = DBL) :: x, y, z !coordinates of the f(x,y,z)= F(xi,eta)
857 REAL (KIND = DBL) :: DxDx1, DyDx1, DzDx1 !coordinates of the tangential vector over the xi axis
858 REAL (KIND = DBL) :: DxDeta, DyDeta, DzDeta !coordinates of the tangential vector over the eta axis
859 REAL (KIND = DBL) :: vnx, vny, vnz !normal vector coordinates of the element
860 REAL (KIND = DBL) :: hs !surface metric on a triangle
861 REAL (KIND = DBL) :: a1, be, ga, alc, bec, gac !integration weigh coefficients
862 REAL (KIND = DBL) :: cf, fill !integration weigh coefficients
863 REAL (KIND = DBL), DIMENSION(6) :: xxi, eet !variables of weigh over in triangle (xi,eta)
864 REAL (KIND = DBL), DIMENSION(6) :: DxDx, DyDx, DzDx !tangential vector over xi axis in triangle (xi,eta)

```

```

865 REAL (KIND = DBL), DIMENSION(6) :: DxDy, DyDe, DzDe !tangential vector over eta axis in triangle (xi,eta)
866 REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz !normal vector in triangle (xi,eta)
867 REAL (KIND = DBL), DIMENSION(6) :: hss !weigth
868 REAL (KIND = DBL) :: bvx1, bvy1, bvz1, & !binormal vectors around in triangle (xi,eta)
869 & bvx2, bvy2, bvz2, &
870 & bvx3, bvy3, bvz3
871 REAL (KIND = DBL) :: yvx, yvy, yvz !vector (y-x0)
872 !-----
873 ! Initialize
874 !-----
875 GEx = 0.000
876 GEy = 0.000
877 GEz = 0.000
878
879 yvx = 0.000
880 yvy = 0.000
881 yvz = 0.000
882 !-----
883 ! vertices of the kth triangle
884 !-----
885 i1 = n(k,1)
886 i2 = n(k,2)
887 i3 = n(k,3)
888 i4 = n(k,4)
889 i5 = n(k,5)
890 i6 = n(k,6)
891 !-----
892
893 al = alphaQ(k)
894 be = betaQ(k)
895 ga = gammaQ(k)
896 alc = 1.000-al
897 bec = 1.000-be
898 gac = 1.000-ga
899 cf = 0.000
900 !-----
901 ! compute the average value of the normal vector the mean curvature as a contour integral using the nifty
902 ! formula (4.2.18) of Pozrikidis (1997)
903 !-----
904 xxi(1) = 0.000
905 eet(1) = 0.000
906 xxi(2) = 1.000
907 eet(2) = 0.000
908 xxi(3) = 0.000
909 eet(3) = 1.000
910 xxi(4) = al
911 eet(4) = 0.000
912 xxi(5) = ga
913 eet(5) = gac
914 xxi(6) = 0.000
915 eet(6) = be
916 DO l = 1, 6
917 xi = xxi(l)
918 eta = eet(l)
919 CALL interp_p(p(i1,1), p(i1,2), p(i1,3), &
920 & p(i2,1), p(i2,2), p(i2,3), &
921 & p(i3,1), p(i3,2), p(i3,3), &
922 & p(i4,1), p(i4,2), p(i4,3), &
923 & p(i5,1), p(i5,2), p(i5,3), &
924 & p(i6,1), p(i6,2), p(i6,3), &
925 & al, be, ga, &
926 & xi, eta, z, &
927 & x, y, z, &
928 & DxDx(i), DyDx(i), DzDx(i), &
929 & DxDy(i), DyDy(i), DzDy(i), &
930 & vx(i), vy(i), vz(i), &
931 & hss(i) )
932 END DO
933 !-----
934 bvx1 = 0.000
935 bvy1 = 0.000
936 bvz1 = 0.000

```

```

937 bvx2 = 0.000
938 bvy2 = 0.000
939 bvz2 = 0.000
940 bvx3 = 0.000
941 bvy3 = 0.000
942 bvz3 = 0.000
943 yvx = 0.000
944 yvy = 0.000
945 yvz = 0.000
946 !-----
947 ! computation of curvature line integral along segment 1-4-2
948 !-----
949 yvx = p(i1,1) - x0
950 yvy = p(i1,2) - y0
951 yvz = p(i1,3) - z0
952 fill = (DSQRT(yvx**2 + yvy**2 + yvz**2))/hss(1)
953 !-----
954 bvx1 = (DyDx(1)*yvz - DzDx(1)*yvy)/fill
955 bvy1 = (DzDx(1)*yvx - DxDx(1)*yvz)/fill
956 bvz1 = (DxDx(1)*yvy - DyDx(1)*yvx)/fill
957 !-----
958 yvx = p(i4,1) - x0
959 yvy = p(i4,2) - y0
960 yvz = p(i4,3) - z0
961 fill = (DSQRT(yvx**2 + yvy**2 + yvz**2))/hss(4)
962 !-----
963 bvx2 = (DyDx(4)*yvz - DzDx(4)*yvy)/fill
964 bvy2 = (DzDx(4)*yvx - DxDx(4)*yvz)/fill
965 bvz2 = (DxDx(4)*yvy - DyDx(4)*yvx)/fill
966 !-----
967 yvx = p(i2,1) - x0
968 yvy = p(i2,2) - y0
969 yvz = p(i2,3) - z0
970 fill = (DSQRT(yvx**2 + yvy**2 + yvz**2))/hss(2)
971 !-----
972 bvx3 = (DyDx(2)*yvz - DzDx(2)*yvy)/fill
973 bvy3 = (DzDx(2)*yvx - DxDx(2)*yvz)/fill
974 bvz3 = (DxDx(2)*yvy - DyDx(2)*yvx)/fill
975 !-----
976 GEx = al*bvx1 + bvx2 + alc*bvx3
977 GEy = al*bvy1 + bvy2 + alc*bvy3
978 GEz = al*bvz1 + bvz2 + alc*bvz3
979 !-----
980 ! computation of curvature line integral along segment 2-5-3
981 !-----
982 bvx1 = 0.000
983 bvy1 = 0.000
984 bvz1 = 0.000
985 bvx2 = 0.000
986 bvy2 = 0.000
987 bvz2 = 0.000
988 bvx3 = 0.000
989 bvy3 = 0.000
990 bvz3 = 0.000
991 !-----
992 yvx = p(i2,1) - x0
993 yvy = p(i2,2) - y0
994 yvz = p(i2,3) - z0
995 fill = (DSQRT(yvx**2 + yvy**2 + yvz**2))/hss(2)
996 !-----
997 bvx1 = (DyDx(2)*yvz - DzDx(2)*yvy)/fill
998 bvy1 = (DzDx(2)*yvx - DxDx(2)*yvz)/fill
999 bvz1 = (DxDx(2)*yvy - DyDx(2)*yvx)/fill
1000 !-----
1001 yvx = p(i5,1) - x0
1002 yvy = p(i5,2) - y0
1003 yvz = p(i5,3) - z0
1004 fill = (DSQRT(yvx**2 + yvy**2 + yvz**2))/hss(5)
1005 !-----
1006 bvx2 = (DyDx(5)*yvz - DzDx(5)*yvy)/fill
1007 bvy2 = (DzDx(5)*yvx - DxDx(5)*yvz)/fill
1008 bvz2 = (DxDx(5)*yvy - DyDx(5)*yvx)/fill

```



```

1009 |-----|
1010 | yvx = p(13,1) - x0
1011 | yvy = p(13,2) - y0
1012 | yvz = p(13,3) - z0
1013 | fill = (DSQRT(yvx**2 + yvy**2 + yvz**2))/hss(3)
1014 |-----|
1015 | bvx3 = (DyDx(3)*yvx - DzDx(3)*yvy)/fill
1016 | bvy3 = (DzDx(3)*yvx - DxDe(3)*yvy)/fill
1017 | bvz3 = (DxDx(3)*yvy - DyDx(3)*yvx)/fill
1018 |-----|
1019 | GEz = GEz - (gac*bvx1 + bvx2 + ga*bvx3)
1020 | GEy = GEy - (gac*bvy1 + bvy2 + ga*bvy3)
1021 | GEz = GEz - (gac*bvz1 + bvz2 + ga*bvz3)
1022 |-----|
1023 | yvx = p(12,1) - x0
1024 | yvy = p(12,2) - y0
1025 | yvz = p(12,3) - z0
1026 | fill = (DSQRT(yvx**2 + yvy**2 + yvz**2))/hss(2)
1027 |-----|
1028 | bvx1 = (DyDe(2)*yvx - DzDe(2)*yvy)/fill
1029 | bvy1 = (DzDe(2)*yvx - DxDe(2)*yvy)/fill
1030 | bvz1 = (DxDx(2)*yvy - DyDe(2)*yvx)/fill
1031 |-----|
1032 | yvx = p(15,1) - x0
1033 | yvy = p(15,2) - y0
1034 | yvz = p(15,3) - z0
1035 | fill = (DSQRT(yvx**2 + yvy**2 + yvz**2))/hss(5)
1036 |-----|
1037 | bvx2 = (DyDe(5)*yvx - DzDe(5)*yvy)/fill
1038 | bvy2 = (DzDe(5)*yvx - DxDe(5)*yvy)/fill
1039 | bvz2 = (DxDx(5)*yvy - DyDe(5)*yvx)/fill
1040 |-----|
1041 | yvx = p(13,1) - x0
1042 | yvy = p(13,2) - y0
1043 | yvz = p(13,3) - z0
1044 | fill = (DSQRT(yvx**2 + yvy**2 + yvz**2))/hss(3)
1045 |-----|
1046 | bvx3 = (DyDe(3)*yvx - DzDe(3)*yvy)/fill
1047 | bvy3 = (DzDe(3)*yvx - DxDe(3)*yvy)/fill
1048 | bvz3 = (DxDx(3)*yvy - DyDe(3)*yvx)/fill
1049 |-----|
1050 | GEz = GEz + (gac*bvx1 + bvx2 + ga*bvx3)
1051 | GEy = GEy + (gac*bvy1 + bvy2 + ga*bvy3)
1052 | GEz = GEz + (gac*bvz1 + bvz2 + ga*bvz3)
1053 |-----|
1054 | computation of curvature line integral along segment 3-5-1
1055 |-----|
1056 | bvx1 = 0.000
1057 | bvy1 = 0.000
1058 | bvz1 = 0.000
1059 | bvx2 = 0.000
1060 | bvy2 = 0.000
1061 | bvz2 = 0.000
1062 | bvx3 = 0.000
1063 | bvy3 = 0.000
1064 | bvz3 = 0.000
1065 |-----|
1066 | yvx = p(13,1) - x0
1067 | yvy = p(13,2) - y0
1068 | yvz = p(13,3) - z0
1069 | fill = (DSQRT(yvx**2 + yvy**2 + yvz**2))/hss(3)
1070 |-----|
1071 | bvx1 = (DyDe(3)*yvx - DzDe(3)*yvy)/fill
1072 | bvy1 = (DzDe(3)*yvx - DxDe(3)*yvy)/fill
1073 | bvz1 = (DxDx(3)*yvy - DyDe(3)*yvx)/fill
1074 |-----|
1075 | yvx = p(16,1) - x0
1076 | yvy = p(16,2) - y0
1077 | yvz = p(16,3) - z0
1078 | fill = (DSQRT(yvx**2 + yvy**2 + yvz**2))/hss(6)
1079 |-----|
1080 | bvx2 = (DyDe(6)*yvx - DzDe(6)*yvy)/fill

```

```

1081 | bvy2 = (DzDe(6)*yvx - DxDe(6)*yvy)/fill
1082 | bvz2 = (DxDx(6)*yvy - DyDe(6)*yvx)/fill
1083 |-----|
1084 | yvx = p(11,1) - x0
1085 | yvy = p(11,2) - y0
1086 | yvz = p(11,3) - z0
1087 | fill = (DSQRT(yvx**2 + yvy**2 + yvz**2))/hss(1)
1088 |-----|
1089 | bvx3 = (DyDe(1)*yvx - DzDe(1)*yvy)/fill
1090 | bvy3 = (DzDe(1)*yvx - DxDe(1)*yvy)/fill
1091 | bvz3 = (DxDx(1)*yvy - DyDe(1)*yvx)/fill
1092 |-----|
1093 | GEz = GEz - be*bvx1 - bvx2 - bec*bvx3
1094 | GEy = GEy - be*bvy1 - bvy2 - bec*bvy3
1095 | GEz = GEz - be*bvz1 - bvz2 - bec*bvz3
1096 |-----|
1097 | GEz = GEz*crvme1(k)
1098 | GEy = GEy*crvme1(k)
1099 | GEz = GEz*crvme1(k)
1100 | END SUBROUTINE intr_lin_sing3
1101 |-----|
1102 | SUBROUTINE intr_lin_sing4(x0, y0, z0, &
1103 | & k, &
1104 | & GEz, GEy, GEz )
1105 |-----|
1106 | This subroutine is a new version stokeslet Subroutine.
1107 | Compute:
1108 | The value of the Stokeslet over each singular element
1109 | Now, (March/ 09 / 2015) this subroutine was made.
1110 |-----|
1111 | USE Mod_Nodal_Interp
1112 | USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, Ulog,&
1113 | & crvme1
1114 |-----|
1115 | IMPLICIT NONE
1116 |-----|
1117 | Variables
1118 |-----|
1119 | REAL (KIND = DBL), INTENT(IN) :: x0, y0, z0 !singulaty coordinates
1120 | INTEGER, INTENT(IN) :: k !number of element
1121 | REAL (KIND = DBL), INTENT(OUT) :: GEz, GEy, GEz !value of stokeslet in the singular element
1122 |-----|
1123 | Variables inside the subroutine
1124 |-----|
1125 | INTEGER :: i, j !counters
1126 | INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
1127 |-----|
1128 | REAL (KIND = DBL) :: cf, fill !integration weigh coefficients
1129 | REAL (KIND = DBL) :: DxDe, DyDe, DzDe !tangential vector over xi axis in triangle (xi,eta)
1130 | REAL (KIND = DBL) :: hss !weight
1131 | REAL (KIND = DBL) :: bvx1, bvy1, bvz1, & !binormal vectors around in triangle (xi,eta)
1132 | & bvx2, bvy2, bvz2, &
1133 | & bvx3, bvy3, bvz3
1134 | REAL (KIND = DBL) :: yvx, yvy, yvz !vector (y-x0)
1135 |-----|
1136 | Initialize
1137 |-----|
1138 | GEz = 0.000
1139 | GEy = 0.000
1140 | GEz = 0.000
1141 |-----|
1142 | yvx = 0.000
1143 | yvy = 0.000
1144 | yvz = 0.000
1145 |-----|
1146 | vertices of the kth triangle
1147 |-----|
1148 | i1 = n(k,1)
1149 | i2 = n(k,2)
1150 | i3 = n(k,3)
1151 | i4 = n(k,4)
1152 | i5 = n(k,5)

```

```

1153 i6 = n(k,6)
1154 -----
1155 cf = 0.000
1156 -----
1157 ! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
1158 ! rule formula
1159 -----
1160 bvx1 = 0.000
1161 bvy1 = 0.000
1162 bvz1 = 0.000
1163 bvx2 = 0.000
1164 bvy2 = 0.000
1165 bvz2 = 0.000
1166 bvx3 = 0.000
1167 bvy3 = 0.000
1168 bvz3 = 0.000
1169 yvx = 0.000
1170 yvy = 0.000
1171 yvz = 0.000
1172 -----
1173 ! computation of curvature line integral along segment 1-4
1174 -----
1175 hss = DSQRT((p(i4,1) - p(i1,1))**2 + (p(i4,2) - p(i1,2))**2 + (p(i4,3) - p(i1,3))**2)
1176 DxDx = (p(i4,1) - p(i1,1))/hss
1177 DyDx = (p(i4,2) - p(i1,2))/hss
1178 DzDx = (p(i4,3) - p(i1,3))/hss
1179 -----
1180 fill = DSQRT((p(i1,1) - x0)**2 + (p(i1,2) - y0)**2 + (p(i1,3) - z0)**2)
1181 yvx = p(i1,1) - x0
1182 yvy = p(i1,2) - y0
1183 yvz = p(i1,3) - z0
1184 -----
1185 bvx1 = (DyDx*yvz - DzDx*yvy)/fill
1186 bvy1 = (DzDx*yvx - DxDx*yvz)/fill
1187 bvz1 = (DxDx*yvy - DyDx*yvx)/fill
1188 -----
1189 fill = DSQRT((p(i4,1) - x0)**2 + (p(i4,2) - y0)**2 + (p(i4,3) - z0)**2)
1190 yvx = p(i4,1) - x0
1191 yvy = p(i4,2) - y0
1192 yvz = p(i4,3) - z0
1193 -----
1194 bvx2 = (DyDx*yvz - DzDx*yvy)/fill
1195 bvy2 = (DzDx*yvx - DxDx*yvz)/fill
1196 bvz2 = (DxDx*yvy - DyDx*yvx)/fill
1197 -----
1198 GEx = 0.5D0*hss*(bvx1 + bvx2)
1199 GEy = 0.5D0*hss*(bvy1 + bvy2)
1200 GEz = 0.5D0*hss*(bvz1 + bvz2)
1201 -----
1202 ! computation of curvature line integral along segment 4-2
1203 -----
1204 hss = DSQRT((p(i2,1) - p(i4,1))**2 + (p(i2,2) - p(i4,2))**2 + (p(i2,3) - p(i4,3))**2)
1205 DxDx = (p(i2,1) - p(i4,1))/hss
1206 DyDx = (p(i2,2) - p(i4,2))/hss
1207 DzDx = (p(i2,3) - p(i4,3))/hss
1208 -----
1209 fill = DSQRT((p(i4,1) - x0)**2 + (p(i4,2) - y0)**2 + (p(i4,3) - z0)**2)
1210 yvx = p(i4,1) - x0
1211 yvy = p(i4,2) - y0
1212 yvz = p(i4,3) - z0
1213 -----
1214 bvx1 = (DyDx*yvz - DzDx*yvy)/fill
1215 bvy1 = (DzDx*yvx - DxDx*yvz)/fill
1216 bvz1 = (DxDx*yvy - DyDx*yvx)/fill
1217 -----
1218 fill = DSQRT((p(i2,1) - x0)**2 + (p(i2,2) - y0)**2 + (p(i2,3) - z0)**2)
1219 yvx = p(i2,1) - x0
1220 yvy = p(i2,2) - y0
1221 yvz = p(i2,3) - z0
1222 -----
1223 bvx2 = (DyDx*yvz - DzDx*yvy)/fill
1224 bvy2 = (DzDx*yvx - DxDx*yvz)/fill

```

```

1225 bvz2 = (DxDx*yvy - DyDx*yvx)/fill
1226 -----
1227 GEx = GEx + 0.5D0*hss*(bvx1 + bvx2)
1228 GEy = GEy + 0.5D0*hss*(bvy1 + bvy2)
1229 GEz = GEz + 0.5D0*hss*(bvz1 + bvz2)
1230 -----
1231 ! computation of curvature line integral along segment 2-5
1232 -----
1233 hss = DSQRT((p(i5,1) - p(i2,1))**2 + (p(i5,2) - p(i2,2))**2 + (p(i5,3) - p(i2,3))**2)
1234 DxDx = (p(i5,1) - p(i2,1))/hss
1235 DyDx = (p(i5,2) - p(i2,2))/hss
1236 DzDx = (p(i5,3) - p(i2,3))/hss
1237 -----
1238 fill = DSQRT((p(i2,1) - x0)**2 + (p(i2,2) - y0)**2 + (p(i2,3) - z0)**2)
1239 yvx = p(i2,1) - x0
1240 yvy = p(i2,2) - y0
1241 yvz = p(i2,3) - z0
1242 -----
1243 bvx1 = (DyDx*yvz - DzDx*yvy)/fill
1244 bvy1 = (DzDx*yvx - DxDx*yvz)/fill
1245 bvz1 = (DxDx*yvy - DyDx*yvx)/fill
1246 -----
1247 fill = DSQRT((p(i5,1) - x0)**2 + (p(i5,2) - y0)**2 + (p(i5,3) - z0)**2)
1248 yvx = p(i5,1) - x0
1249 yvy = p(i5,2) - y0
1250 yvz = p(i5,3) - z0
1251 -----
1252 bvx2 = (DyDx*yvz - DzDx*yvy)/fill
1253 bvy2 = (DzDx*yvx - DxDx*yvz)/fill
1254 bvz2 = (DxDx*yvy - DyDx*yvx)/fill
1255 -----
1256 GEx = GEx + 0.5D0*hss*(bvx1 + bvx2)
1257 GEy = GEy + 0.5D0*hss*(bvy1 + bvy2)
1258 GEz = GEz + 0.5D0*hss*(bvz1 + bvz2)
1259 -----
1260 ! computation of curvature line integral along segment 5-3
1261 -----
1262 hss = DSQRT((p(i3,1) - p(i5,1))**2 + (p(i3,2) - p(i5,2))**2 + (p(i3,3) - p(i5,3))**2)
1263 DxDx = (p(i3,1) - p(i5,1))/hss
1264 DyDx = (p(i3,2) - p(i5,2))/hss
1265 DzDx = (p(i3,3) - p(i5,3))/hss
1266 -----
1267 fill = DSQRT((p(i5,1) - x0)**2 + (p(i5,2) - y0)**2 + (p(i5,3) - z0)**2)
1268 yvx = p(i5,1) - x0
1269 yvy = p(i5,2) - y0
1270 yvz = p(i5,3) - z0
1271 -----
1272 bvx1 = (DyDx*yvz - DzDx*yvy)/fill
1273 bvy1 = (DzDx*yvx - DxDx*yvz)/fill
1274 bvz1 = (DxDx*yvy - DyDx*yvx)/fill
1275 -----
1276 fill = DSQRT((p(i3,1) - x0)**2 + (p(i3,2) - y0)**2 + (p(i3,3) - z0)**2)
1277 yvx = p(i3,1) - x0
1278 yvy = p(i3,2) - y0
1279 yvz = p(i3,3) - z0
1280 -----
1281 bvx2 = (DyDx*yvz - DzDx*yvy)/fill
1282 bvy2 = (DzDx*yvx - DxDx*yvz)/fill
1283 bvz2 = (DxDx*yvy - DyDx*yvx)/fill
1284 -----
1285 GEx = GEx + 0.5D0*hss*(bvx1 + bvx2)
1286 GEy = GEy + 0.5D0*hss*(bvy1 + bvy2)
1287 GEz = GEz + 0.5D0*hss*(bvz1 + bvz2)
1288 -----
1289 ! computation of curvature line integral along segment 3-6
1290 -----
1291 hss = DSQRT((p(i6,1) - p(i3,1))**2 + (p(i6,2) - p(i3,2))**2 + (p(i6,3) - p(i3,3))**2)
1292 DxDx = (p(i6,1) - p(i3,1))/hss
1293 DyDx = (p(i6,2) - p(i3,2))/hss
1294 DzDx = (p(i6,3) - p(i3,3))/hss
1295 -----
1296 fill = DSQRT((p(i3,1) - x0)**2 + (p(i3,2) - y0)**2 + (p(i3,3) - z0)**2)

```

```

1297 yvx = p(13,1) - x0
1298 yvy = p(13,2) - y0
1299 yvz = p(13,3) - z0
1300
1301 bvx1 = (DyDx*yvz - DzDx*yvy)/fill
1302 bvy1 = (DzDx*yvx - DxDx*yvz)/fill
1303 bvz1 = (DxDx*yvy - DyDx*yvx)/fill
1304
1305 fill = DSQRT((p(16,1) - x0)**2 + (p(16,2) - y0)**2 + (p(16,3) - z0)**2)
1306 yvx = p(16,1) - x0
1307 yvy = p(16,2) - y0
1308 yvz = p(16,3) - z0
1309
1310 bvx2 = (DyDx*yvz - DzDx*yvy)/fill
1311 bvy2 = (DzDx*yvx - DxDx*yvz)/fill
1312 bvz2 = (DxDx*yvy - DyDx*yvx)/fill
1313
1314 GEx = GEx + 0.5D0*hss*(bvx1 + bvx2)
1315 GEy = GEy + 0.5D0*hss*(bvy1 + bvy2)
1316 GEz = GEz + 0.5D0*hss*(bvz1 + bvz2)
1317
1318 ! computation of curvature line integral along segment 6-1
1319
1320 hss = DSQRT((p(11,1) - p(16,1))**2 + (p(11,2) - p(16,2))**2 + (p(11,3) - p(16,3))**2)
1321 DxDx = (p(11,1) - p(16,1))/hss
1322 DyDx = (p(11,2) - p(16,2))/hss
1323 DzDx = (p(11,3) - p(16,3))/hss
1324
1325 fill = DSQRT((p(16,1) - x0)**2 + (p(16,2) - y0)**2 + (p(16,3) - z0)**2)
1326 yvx = p(16,1) - x0
1327 yvy = p(16,2) - y0
1328 yvz = p(16,3) - z0
1329
1330 bvx1 = (DyDx*yvz - DzDx*yvy)/fill
1331 bvy1 = (DzDx*yvx - DxDx*yvz)/fill
1332 bvz1 = (DxDx*yvy - DyDx*yvx)/fill
1333
1334 fill = DSQRT((p(11,1) - x0)**2 + (p(11,2) - y0)**2 + (p(11,3) - z0)**2)
1335 yvx = p(11,1) - x0
1336 yvy = p(11,2) - y0
1337 yvz = p(11,3) - z0
1338
1339 bvx2 = (DyDx*yvz - DzDx*yvy)/fill
1340 bvy2 = (DzDx*yvx - DxDx*yvz)/fill
1341 bvz2 = (DxDx*yvy - DyDx*yvx)/fill
1342
1343 GEx = GEx + 0.5D0*hss*(bvx1 + bvx2)
1344 GEy = GEy + 0.5D0*hss*(bvy1 + bvy2)
1345 GEz = GEz + 0.5D0*hss*(bvz1 + bvz2)
1346
1347
1348 ! computation of curvature line integral along segment 1-6
1349
1350 hss = DSQRT((p(16,1) - p(11,1))**2 + (p(16,2) - p(11,2))**2 + (p(16,3) - p(11,3))**2)
1351 DxDx = (p(16,1) - p(11,1))/hss
1352 DyDx = (p(16,2) - p(11,2))/hss
1353 DzDx = (p(16,3) - p(11,3))/hss
1354
1355 fill = DSQRT((p(11,1) - x0)**2 + (p(11,2) - y0)**2 + (p(11,3) - z0)**2)
1356 yvx = p(11,1) - x0
1357 yvy = p(11,2) - y0
1358 yvz = p(11,3) - z0
1359
1360 bvx1 = (DyDx*yvz - DzDx*yvy)/fill
1361 bvy1 = (DzDx*yvx - DxDx*yvz)/fill
1362 bvz1 = (DxDx*yvy - DyDx*yvx)/fill
1363
1364 fill = DSQRT((p(16,1) - x0)**2 + (p(16,2) - y0)**2 + (p(16,3) - z0)**2)
1365 yvx = p(16,1) - x0
1366 yvy = p(16,2) - y0
1367 yvz = p(16,3) - z0
1368

```

```

1369 bvx2 = (DyDx*yvz - DzDx*yvy)/fill
1370 bvy2 = (DzDx*yvx - DxDx*yvz)/fill
1371 bvz2 = (DxDx*yvy - DyDx*yvx)/fill
1372
1373 GEx = GEx + 0.5D0*hss*(bvx1 + bvx2)
1374 GEy = GEy + 0.5D0*hss*(bvy1 + bvy2)
1375 GEz = GEz + 0.5D0*hss*(bvz1 + bvz2)
1376
1377 ! computation of curvature line integral along segment 6-3
1378
1379 hss = DSQRT((p(13,1) - p(16,1))**2 + (p(13,2) - p(16,2))**2 + (p(13,3) - p(16,3))**2)
1380 DxDx = (p(13,1) - p(16,1))/hss
1381 DyDx = (p(13,2) - p(16,2))/hss
1382 DzDx = (p(13,3) - p(16,3))/hss
1383
1384 fill = DSQRT((p(16,1) - x0)**2 + (p(16,2) - y0)**2 + (p(16,3) - z0)**2)
1385 yvx = p(16,1) - x0
1386 yvy = p(16,2) - y0
1387 yvz = p(16,3) - z0
1388
1389 bvx1 = (DyDx*yvz - DzDx*yvy)/fill
1390 bvy1 = (DzDx*yvx - DxDx*yvz)/fill
1391 bvz1 = (DxDx*yvy - DyDx*yvx)/fill
1392
1393 fill = DSQRT((p(13,1) - x0)**2 + (p(13,2) - y0)**2 + (p(13,3) - z0)**2)
1394 yvx = p(13,1) - x0
1395 yvy = p(13,2) - y0
1396 yvz = p(13,3) - z0
1397
1398 bvx2 = (DyDx*yvz - DzDx*yvy)/fill
1399 bvy2 = (DzDx*yvx - DxDx*yvz)/fill
1400 bvz2 = (DxDx*yvy - DyDx*yvx)/fill
1401
1402 GEx = GEx + 0.5D0*hss*(bvx1 + bvx2)
1403 GEy = GEy + 0.5D0*hss*(bvy1 + bvy2)
1404 GEz = GEz + 0.5D0*hss*(bvz1 + bvz2)
1405
1406 ! computation of curvature line integral along segment 3-5
1407
1408 hss = DSQRT((p(15,1) - p(13,1))**2 + (p(15,2) - p(13,2))**2 + (p(15,3) - p(13,3))**2)
1409 DxDx = (p(15,1) - p(13,1))/hss
1410 DyDx = (p(15,2) - p(13,2))/hss
1411 DzDx = (p(15,3) - p(13,3))/hss
1412
1413 fill = DSQRT((p(13,1) - x0)**2 + (p(13,2) - y0)**2 + (p(13,3) - z0)**2)
1414 yvx = p(13,1) - x0
1415 yvy = p(13,2) - y0
1416 yvz = p(13,3) - z0
1417
1418 bvx1 = (DyDx*yvz - DzDx*yvy)/fill
1419 bvy1 = (DzDx*yvx - DxDx*yvz)/fill
1420 bvz1 = (DxDx*yvy - DyDx*yvx)/fill
1421
1422 fill = DSQRT((p(15,1) - x0)**2 + (p(15,2) - y0)**2 + (p(15,3) - z0)**2)
1423 yvx = p(15,1) - x0
1424 yvy = p(15,2) - y0
1425 yvz = p(15,3) - z0
1426
1427 bvx2 = (DyDx*yvz - DzDx*yvy)/fill
1428 bvy2 = (DzDx*yvx - DxDx*yvz)/fill
1429 bvz2 = (DxDx*yvy - DyDx*yvx)/fill
1430
1431 GEx = GEx + 0.5D0*hss*(bvx1 + bvx2)
1432 GEy = GEy + 0.5D0*hss*(bvy1 + bvy2)
1433 GEz = GEz + 0.5D0*hss*(bvz1 + bvz2)
1434
1435 ! computation of curvature line integral along segment 5-2
1436
1437 hss = DSQRT((p(12,1) - p(15,1))**2 + (p(12,2) - p(15,2))**2 + (p(12,3) - p(15,3))**2)
1438 DxDx = (p(12,1) - p(15,1))/hss
1439 DyDx = (p(12,2) - p(15,2))/hss
1440 DzDx = (p(12,3) - p(15,3))/hss

```

```

1441 |-----|
1442 | fill = DSQRT((p(15,1) - x0)**2 + (p(15,2) - y0)**2 + (p(15,3) - z0)**2)
1443 | yvx = p(15,1) - x0
1444 | yvy = p(15,2) - y0
1445 | yvz = p(15,3) - z0
1446 |-----|
1447 | bvx1 = (DyDx*yvz - DzDx*yvy)/fill
1448 | bvy1 = (DzDx*yvx - DxDx*yvz)/fill
1449 | bvz1 = (DxDx*yvy - DyDx*yvx)/fill
1450 |-----|
1451 | fill = DSQRT((p(12,1) - x0)**2 + (p(12,2) - y0)**2 + (p(12,3) - z0)**2)
1452 | yvx = p(12,1) - x0
1453 | yvy = p(12,2) - y0
1454 | yvz = p(12,3) - z0
1455 |-----|
1456 | bvx2 = (DyDx*yvz - DzDx*yvy)/fill
1457 | bvy2 = (DzDx*yvx - DxDx*yvz)/fill
1458 | bvz2 = (DxDx*yvy - DyDx*yvx)/fill
1459 |-----|
1460 | GEx = GEx + 0.5D0*hss*(bvx1 + bvx2)
1461 | GEy = GEy + 0.5D0*hss*(bvy1 + bvy2)
1462 | GEz = GEz + 0.5D0*hss*(bvz1 + bvz2)
1463 |-----|
1464 | computation of curvature line integral along segment 2-4
1465 |-----|
1466 | hss = DSQRT((p(14,1) - p(12,1))**2 + (p(14,2) - p(12,2))**2 + (p(14,3) - p(12,3))**2)
1467 | DxDx = (p(14,1) - p(12,1))/hss
1468 | DyDx = (p(14,2) - p(12,2))/hss
1469 | DzDx = (p(14,3) - p(12,3))/hss
1470 |-----|
1471 | fill = DSQRT((p(12,1) - x0)**2 + (p(12,2) - y0)**2 + (p(12,3) - z0)**2)
1472 | yvx = p(12,1) - x0
1473 | yvy = p(12,2) - y0
1474 | yvz = p(12,3) - z0
1475 |-----|
1476 | bvx1 = (DyDx*yvz - DzDx*yvy)/fill
1477 | bvy1 = (DzDx*yvx - DxDx*yvz)/fill
1478 | bvz1 = (DxDx*yvy - DyDx*yvx)/fill
1479 |-----|
1480 | fill = DSQRT((p(14,1) - x0)**2 + (p(14,2) - y0)**2 + (p(14,3) - z0)**2)
1481 | yvx = p(14,1) - x0
1482 | yvy = p(14,2) - y0
1483 | yvz = p(14,3) - z0
1484 |-----|
1485 | bvx2 = (DyDx*yvz - DzDx*yvy)/fill
1486 | bvy2 = (DzDx*yvx - DxDx*yvz)/fill
1487 | bvz2 = (DxDx*yvy - DyDx*yvx)/fill
1488 |-----|
1489 | GEx = GEx + 0.5D0*hss*(bvx1 + bvx2)
1490 | GEy = GEy + 0.5D0*hss*(bvy1 + bvy2)
1491 | GEz = GEz + 0.5D0*hss*(bvz1 + bvz2)
1492 |-----|
1493 | computation of curvature line integral along segment 4-1
1494 |-----|
1495 | hss = DSQRT((p(11,1) - p(14,1))**2 + (p(11,2) - p(14,2))**2 + (p(11,3) - p(14,3))**2)
1496 | DxDx = (p(11,1) - p(14,1))/hss
1497 | DyDx = (p(11,2) - p(14,2))/hss
1498 | DzDx = (p(11,3) - p(14,3))/hss
1499 |-----|
1500 | fill = DSQRT((p(14,1) - x0)**2 + (p(14,2) - y0)**2 + (p(14,3) - z0)**2)
1501 | yvx = p(14,1) - x0
1502 | yvy = p(14,2) - y0
1503 | yvz = p(14,3) - z0
1504 |-----|
1505 | bvx1 = (DyDx*yvz - DzDx*yvy)/fill
1506 | bvy1 = (DzDx*yvx - DxDx*yvz)/fill
1507 | bvz1 = (DxDx*yvy - DyDx*yvx)/fill
1508 |-----|
1509 | fill = DSQRT((p(11,1) - x0)**2 + (p(11,2) - y0)**2 + (p(11,3) - z0)**2)
1510 | yvx = p(11,1) - x0
1511 | yvy = p(11,2) - y0
1512 | yvz = p(11,3) - z0

```

```

1513 |-----|
1514 | bvx2 = (DyDx*yvz - DzDx*yvy)/fill
1515 | bvy2 = (DzDx*yvx - DxDx*yvz)/fill
1516 | bvz2 = (DxDx*yvy - DyDx*yvx)/fill
1517 |-----|
1518 | GEx = GEx + 0.5D0*hss*(bvx1 + bvx2)
1519 | GEy = GEy + 0.5D0*hss*(bvy1 + bvy2)
1520 | GEz = GEz + 0.5D0*hss*(bvz1 + bvz2)
1521 |-----|
1522 | GEx = 0.5D0*GEx
1523 | GEy = 0.5D0*GEy
1524 | GEz = 0.5D0*GEz
1525 |-----|
1526 | GEx = GEx*crvml(k)
1527 | GEy = GEy*crvml(k)
1528 | GEz = GEz*crvml(k)
1529 |-----|
1530 | END SUBROUTINE intr_lin_sing4
1531 |-----|
1532 |-----|
1533 | SUBROUTINE intr_lin_sing5(x0, y0, z0, &
1534 | & k, &
1535 | & GEx, GEy, GEz )
1536 |-----|
1537 | This subroutine is a new version stokeslet Subroutine.
1538 | Compute:
1539 | *The value of the Stokeslet over each singular element
1540 | How, (March/ 09 / 2015) this subroutine was made.
1541 |-----|
1542 | USE Mod_Nodal_Interp
1543 | USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog,&
1544 | & crvml
1545 |-----|
1546 | IMPLICIT NONE
1547 |-----|
1548 | Variables
1549 |-----|
1550 | REAL (KIND = DBL), INTENT(IN) :: x0, y0, z0 |singularity coordinates
1551 | INTEGER, INTENT(IN) :: k |number of element
1552 | REAL (KIND = DBL), INTENT(OUT) :: GEx, GEy, GEz |value of stokeslet in the singular element
1553 |-----|
1554 | Variables inside the subroutine
1555 |-----|
1556 | INTEGER :: i, j |counters
1557 | INTEGER :: i1, i2, i3, i4, i5, i6 |indices to obtain node numbers from each element
1558 |-----|
1559 | REAL (KIND = DBL) :: cf, fill |integration weigh coefficients
1560 | REAL (KIND = DBL) :: DxDx, DyDx, DzDx |tangential vector over xi axis in triangle (xi,eta)
1561 | REAL (KIND = DBL) :: hss |weight
1562 | REAL (KIND = DBL) :: bvx1, bvy1, bvz1, & |binormal vectors around in triangle (xi,eta)
1563 | & bvx2, bvy2, bvz2, &
1564 | & bvx3, bvy3, bvz3, &
1565 | & bvx4, bvy4, bvz4, &
1566 | & bvx5, bvy5, bvz5, &
1567 | & bvx6, bvy6, bvz6
1568 | REAL (KIND = DBL) :: yx1, yy1, yz1 |vector y1
1569 | REAL (KIND = DBL) :: yx2, yy2, yz2 |vector y2
1570 | REAL (KIND = DBL) :: yx3, yy3, yz3 |vector y3
1571 | REAL (KIND = DBL) :: yx4, yy4, yz4 |vector y4
1572 | REAL (KIND = DBL) :: yx5, yy5, yz5 |vector y5
1573 | REAL (KIND = DBL) :: yx6, yy6, yz6 |vector y6
1574 | REAL (KIND = DBL) :: zvx1, zvy1, zvz1 |vector (z1-x0)
1575 | REAL (KIND = DBL) :: zvx2, zvy2, zvz2 |vector (z2-x0)
1576 | REAL (KIND = DBL) :: zvx3, zvy3, zvz3 |vector (z3-x0)
1577 | REAL (KIND = DBL) :: zvx4, zvy4, zvz4 |vector (z4-x0)
1578 | REAL (KIND = DBL) :: zvx5, zvy5, zvz5 |vector (z5-x0)
1579 | REAL (KIND = DBL) :: zvx6, zvy6, zvz6 |vector (z6-x0)
1580 | REAL (KIND = DBL) :: px2, py2, pz2 |pressure vector
1581 | REAL (KIND = DBL) :: a1, a2, a3 |vector a
1582 | REAL (KIND = DBL) :: be1, be2, be3, be4, be5, be6 |betha angle between z1-x0 and tangent vector
1583 | REAL (KIND = DBL) :: ga1, ga2, ga3, ga4, ga5, ga6 |gamma angle between z1-x0 and tangent vector
1584 | REAL (KIND = DBL) :: e1, e2, e3, e4, e5, e6 |constant

```

```

1585 REAL (KIND = DBL) :: g1, g2, g3, g4, g5, g6 !constant
1586 !-----
1587 ! Initialize
1588 !-----
1589 GEx = 0.000
1590 GEy = 0.000
1591 GEz = 0.000
1592 !-----
1593 !-----
1594 ! vertices of the kth triangle
1595 !-----
1596 i1 = n(k,1)
1597 i2 = n(k,2)
1598 i3 = n(k,3)
1599 i4 = n(k,4)
1600 i5 = n(k,5)
1601 i6 = n(k,6)
1602 !-----
1603 cf = 0.000
1604 !-----
1605 ! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
1606 ! rule formula
1607 !-----
1608 DxDx = p(14,1) - p(11,1)
1609 DyDx = p(14,2) - p(11,2)
1610 DzDx = p(14,3) - p(11,3)
1611 hss = DSQRT(DxDx**2 + DyDx**2 + DzDx**2)
1612 DxDx = DxDx/hss
1613 DyDx = DyDx/hss
1614 DzDx = DzDx/hss
1615 !-----
1616 ! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
1617 ! rule formula
1618 !-----
1619 ! Firstly, to compute all vectors and angles involved.
1620 !-----
1621 xv1 = p(11,1)-x0
1622 yv1 = p(11,2)-y0
1623 zv1 = p(11,3)-z0
1624 xv2 = p(12,1)-x0
1625 yv2 = p(12,2)-y0
1626 zv2 = p(12,3)-z0
1627 xv3 = p(13,1)-x0
1628 yv3 = p(13,2)-y0
1629 zv3 = p(13,3)-z0
1630 xv4 = p(14,1)-x0
1631 yv4 = p(14,2)-y0
1632 zv4 = p(14,3)-z0
1633 xv5 = p(15,1)-x0
1634 yv5 = p(15,2)-y0
1635 zv5 = p(15,3)-z0
1636 xv6 = p(16,1)-x0
1637 yv6 = p(16,2)-y0
1638 zv6 = p(16,3)-z0
1639 !-----
1640 xv1 = xv1 / DSQRT(xv1**2 + yv1**2 + zv1**2)
1641 zv1 = yv1 / DSQRT(xv1**2 + yv1**2 + zv1**2)
1642 xv2 = yv2 / DSQRT(xv2**2 + yv2**2 + zv2**2)
1643 zv2 = yv2 / DSQRT(xv2**2 + yv2**2 + zv2**2)
1644 xv3 = yv3 / DSQRT(xv3**2 + yv3**2 + zv3**2)
1645 zv3 = yv3 / DSQRT(xv3**2 + yv3**2 + zv3**2)
1646 xv4 = yv4 / DSQRT(xv4**2 + yv4**2 + zv4**2)
1647 zv4 = yv4 / DSQRT(xv4**2 + yv4**2 + zv4**2)
1648 xv5 = yv5 / DSQRT(xv5**2 + yv5**2 + zv5**2)
1649 zv5 = yv5 / DSQRT(xv5**2 + yv5**2 + zv5**2)
1650 xv6 = yv6 / DSQRT(xv6**2 + yv6**2 + zv6**2)
1651 zv6 = yv6 / DSQRT(xv6**2 + yv6**2 + zv6**2)
1652 xv5 = yv5 / DSQRT(xv5**2 + yv5**2 + zv5**2)
1653 zv5 = yv5 / DSQRT(xv5**2 + yv5**2 + zv5**2)
1654 xv5 = yv5 / DSQRT(xv5**2 + yv5**2 + zv5**2)
1655 zv5 = yv5 / DSQRT(xv5**2 + yv5**2 + zv5**2)
1656 xv6 = yv6 / DSQRT(xv6**2 + yv6**2 + zv6**2)

```

```

1657 zv6 = yv6 / DSQRT(xv6**2 + yv6**2 + zv6**2)
1658 !-----
1659 ! Angles
1660 !-----
1661 be1 = DACOS( ( ( xv4-xv1 ) * zv1 + ( yv4-yv1 ) * zv1 + ( zv4-zv1 ) * zv1 ) &
1662 & / DSQRT( ( xv4-xv1 ) **2 + ( yv4-yv1 ) **2 + ( zv4-zv1 ) **2 ) )
1663 ga1 = DACOS( ( ( xv4-xv1 ) * zv4 + ( yv4-yv1 ) * zv4 + ( zv4-zv1 ) * zv4 ) &
1664 & / DSQRT( ( xv4-xv1 ) **2 + ( yv4-yv1 ) **2 + ( zv4-zv1 ) **2 ) )
1665 be2 = DACOS( ( ( xv2-xv4 ) * zv4 + ( yv2-yv4 ) * zv4 + ( zv2-zv4 ) * zv4 ) &
1666 & / DSQRT( ( xv2-xv4 ) **2 + ( yv2-yv4 ) **2 + ( zv2-zv4 ) **2 ) )
1667 ga2 = DACOS( ( ( xv2-xv4 ) * zv2 + ( yv2-yv4 ) * zv2 + ( zv2-zv4 ) * zv2 ) &
1668 & / DSQRT( ( xv2-xv4 ) **2 + ( yv2-yv4 ) **2 + ( zv2-zv4 ) **2 ) )
1669 be3 = DACOS( ( ( xv5-xv2 ) * zv2 + ( yv5-yv2 ) * zv2 + ( zv5-zv2 ) * zv2 ) &
1670 & / DSQRT( ( xv5-xv2 ) **2 + ( yv5-yv2 ) **2 + ( zv5-zv2 ) **2 ) )
1671 ga3 = DACOS( ( ( xv5-xv2 ) * zv5 + ( yv5-yv2 ) * zv5 + ( zv5-zv2 ) * zv5 ) &
1672 & / DSQRT( ( xv5-xv2 ) **2 + ( yv5-yv2 ) **2 + ( zv5-zv2 ) **2 ) )
1673 be4 = DACOS( ( ( xv3-xv5 ) * zv5 + ( yv3-yv5 ) * zv5 + ( zv3-zv5 ) * zv5 ) &
1674 & / DSQRT( ( xv3-xv5 ) **2 + ( yv3-yv5 ) **2 + ( zv3-zv5 ) **2 ) )
1675 ga4 = DACOS( ( ( xv3-xv5 ) * zv3 + ( yv3-yv5 ) * zv3 + ( zv3-zv5 ) * zv3 ) &
1676 & / DSQRT( ( xv3-xv5 ) **2 + ( yv3-yv5 ) **2 + ( zv3-zv5 ) **2 ) )
1677 be5 = DACOS( ( ( xv6-xv3 ) * zv3 + ( yv6-yv3 ) * zv3 + ( zv6-zv3 ) * zv3 ) &
1678 & / DSQRT( ( xv6-yv3 ) **2 + ( yv6-yv3 ) **2 + ( zv6-yv3 ) **2 ) )
1679 ga5 = DACOS( ( ( xv6-yv3 ) * zv6 + ( yv6-yv3 ) * zv6 + ( zv6-yv3 ) * zv6 ) &
1680 & / DSQRT( ( xv6-yv3 ) **2 + ( yv6-yv3 ) **2 + ( zv6-yv3 ) **2 ) )
1681 be6 = DACOS( ( ( xv1-yv6 ) * zv6 + ( yv1-yv6 ) * zv6 + ( zv1-yv6 ) * zv6 ) &
1682 & / DSQRT( ( xv1-yv6 ) **2 + ( yv1-yv6 ) **2 + ( zv1-yv6 ) **2 ) )
1683 ga6 = DACOS( ( ( xv1-yv6 ) * zv1 + ( yv1-yv6 ) * zv1 + ( zv1-yv6 ) * zv1 ) &
1684 & / DSQRT( ( xv1-yv6 ) **2 + ( yv1-yv6 ) **2 + ( zv1-yv6 ) **2 ) )
1685 !-----
1686 ! computation of curvature line integral along segment 1
1687 !-----
1688 bx1 = (yv1*yv24 - yv21*yv4)
1689 by1 = (yv21*yv44 - yv11*yv24)
1690 bv1 = (yv1*yv44 - yv11*yv44)
1691 !-----
1692 bx1 = bx1/DSQRT((yv1*yv24 - yv21*yv4)**2 + (yv21*yv44 - yv11*yv24)**2 + (yv1*yv44 - yv11*yv44)**2)
1693 by1 = by1/DSQRT((yv1*yv24 - yv21*yv4)**2 + (yv21*yv44 - yv11*yv24)**2 + (yv1*yv44 - yv11*yv44)**2)
1694 bv1 = bv1/DSQRT((yv1*yv24 - yv21*yv4)**2 + (yv21*yv44 - yv11*yv24)**2 + (yv1*yv44 - yv11*yv44)**2)
1695 !-----
1696 ! computation of curvature line integral along segment 4
1697 !-----
1698 bx2 = (yv4*yv22 - yv44*yv22)
1699 by2 = (yv24*yv22 - yv44*yv22)
1700 bv2 = (yv4*yv22 - yv44*yv22)
1701 !-----
1702 bx2 = bx2/DSQRT((yv4*yv22 - yv44*yv22)**2 + (yv24*yv22 - yv44*yv22)**2 + (yv4*yv22 - yv44*yv22)**2)
1703 by2 = by2/DSQRT((yv4*yv22 - yv44*yv22)**2 + (yv24*yv22 - yv44*yv22)**2 + (yv4*yv22 - yv44*yv22)**2)
1704 bv2 = bv2/DSQRT((yv4*yv22 - yv44*yv22)**2 + (yv24*yv22 - yv44*yv22)**2 + (yv4*yv22 - yv44*yv22)**2)
1705 !-----
1706 ! computation of curvature line integral along segment 2
1707 !-----
1708 bx3 = (yv22*yv25 - yv22*yv25)
1709 by3 = (yv22*yv25 - yv22*yv25)
1710 bv3 = (yv22*yv25 - yv22*yv25)
1711 !-----
1712 bx3 = bx3/DSQRT((yv22*yv25 - yv22*yv25)**2 + (yv22*yv25 - yv22*yv25)**2 + (yv22*yv25 - yv22*yv25)**2)
1713 by3 = by3/DSQRT((yv22*yv25 - yv22*yv25)**2 + (yv22*yv25 - yv22*yv25)**2 + (yv22*yv25 - yv22*yv25)**2)
1714 bv3 = bv3/DSQRT((yv22*yv25 - yv22*yv25)**2 + (yv22*yv25 - yv22*yv25)**2 + (yv22*yv25 - yv22*yv25)**2)
1715 !-----
1716 ! computation of curvature line integral along segment 5
1717 !-----
1718 bx4 = (yv5*yv23 - yv25*yv23)
1719 by4 = (yv25*yv23 - yv25*yv23)
1720 bv4 = (yv5*yv23 - yv25*yv23)
1721 !-----
1722 bx4 = bx4/DSQRT((yv5*yv23 - yv25*yv23)**2 + (yv25*yv23 - yv25*yv23)**2 + (yv5*yv23 - yv25*yv23)**2)
1723 by4 = by4/DSQRT((yv5*yv23 - yv25*yv23)**2 + (yv25*yv23 - yv25*yv23)**2 + (yv5*yv23 - yv25*yv23)**2)
1724 bv4 = bv4/DSQRT((yv5*yv23 - yv25*yv23)**2 + (yv25*yv23 - yv25*yv23)**2 + (yv5*yv23 - yv25*yv23)**2)
1725 !-----
1726 ! computation of curvature line integral along segment 3
1727 !-----
1728 bx5 = (yv3*yv26 - yv23*yv26)

```

```

1729 bvy5 = (yvz3*yvx6 - yvx3*yvz6)
1730 bvz5 = (yvx3*yvy6 - yvy3*yvx6)
1731 -----
1732 bvy5 = bvx5/DSQRT((yvy3*yvz6 - yvz3*yvy6)**2 + (yvz3*yvx6 - yvx3*yvz6)**2 + (yvx3*yvy6 - yvy3*yvx6)**2 )
1733 bvy5 = bvy5/DSQRT((yvy3*yvz6 - yvz3*yvy6)**2 + (yvz3*yvx6 - yvx3*yvz6)**2 + (yvx3*yvy6 - yvy3*yvx6)**2 )
1734 bvz5 = bvz5/DSQRT((yvy3*yvz6 - yvz3*yvy6)**2 + (yvz3*yvx6 - yvx3*yvz6)**2 + (yvx3*yvy6 - yvy3*yvx6)**2 )
1735 -----
1736 ! computation of curvature line integral along segment 6
1737 -----
1738 bvx6 = (yvy6*yvz1 - yvz6*yvy1)
1739 bvy6 = (yvz6*yvx1 - yvx6*yvz1)
1740 bvz6 = (yvx6*yvy1 - yvy6*yvx1)
1741 -----
1742 bvx6 = bvx6/DSQRT((yvy6*yvz1 - yvz6*yvy1)**2 + (yvz6*yvx1 - yvx6*yvz1)**2 + (yvx6*yvy1 - yvy6*yvx1)**2 )
1743 bvy6 = bvy6/DSQRT((yvy6*yvz1 - yvz6*yvy1)**2 + (yvz6*yvx1 - yvx6*yvz1)**2 + (yvx6*yvy1 - yvy6*yvx1)**2 )
1744 bvz6 = bvz6/DSQRT((yvy6*yvz1 - yvz6*yvy1)**2 + (yvz6*yvx1 - yvx6*yvz1)**2 + (yvx6*yvy1 - yvy6*yvx1)**2 )
1745 -----
1746 ! Estimation along point 1
1747 -----
1748 GE = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)*DSIN(be1) *%
1749 & DLOG( DSQRT( ( ((1.000 + DCOS(be1))*(1.000 + DCOS(ga1)))/(DSIN(be1)*DSIN(ga1)) )**2 ) ) *bvz1
1750 GEY = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)*DSIN(be1) *%
1751 & DLOG( DSQRT( ( ((1.000 + DCOS(be1))*(1.000 + DCOS(ga1)))/(DSIN(be1)*DSIN(ga1)) )**2 ) ) *bvy1
1752 GEZ = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)*DSIN(be1) *%
1753 & DLOG( DSQRT( ( ((1.000 + DCOS(be1))*(1.000 + DCOS(ga1)))/(DSIN(be1)*DSIN(ga1)) )**2 ) ) *bvz1
1754 -----
1755 ! Estimation along point 4
1756 -----
1757 GE = GE + DSQRT(yvx4**2 + yvy4**2 + yvz4**2)*DSIN(be2) *%
1758 & DLOG( DSQRT( ( ((1.000 + DCOS(be2))*(1.000 + DCOS(ga2)))/(DSIN(be2)*DSIN(ga2)) )**2 ) ) *bvz2
1759 GEY = GEY + DSQRT(yvx4**2 + yvy4**2 + yvz4**2)*DSIN(be2) *%
1760 & DLOG( DSQRT( ( ((1.000 + DCOS(be2))*(1.000 + DCOS(ga2)))/(DSIN(be2)*DSIN(ga2)) )**2 ) ) *bvy2
1761 GEZ = GEZ + DSQRT(yvx4**2 + yvy4**2 + yvz4**2)*DSIN(be2) *%
1762 & DLOG( DSQRT( ( ((1.000 + DCOS(be2))*(1.000 + DCOS(ga2)))/(DSIN(be2)*DSIN(ga2)) )**2 ) ) *bvz2
1763 -----
1764 ! Estimation along point 2
1765 -----
1766 GE = GE + DSQRT(yvx2**3 + yvy2**2 + yvz2**2)*DSIN(be3) *%
1767 & DLOG( DSQRT( ( ((1.000 + DCOS(be3))*(1.000 + DCOS(ga3)))/(DSIN(be3)*DSIN(ga3)) )**2 ) ) *bvz3
1768 GEY = GEY + DSQRT(yvx2**3 + yvy2**2 + yvz2**2)*DSIN(be3) *%
1769 & DLOG( DSQRT( ( ((1.000 + DCOS(be3))*(1.000 + DCOS(ga3)))/(DSIN(be3)*DSIN(ga3)) )**2 ) ) *bvy3
1770 GEZ = GEZ + DSQRT(yvx2**3 + yvy2**2 + yvz2**2)*DSIN(be3) *%
1771 & DLOG( DSQRT( ( ((1.000 + DCOS(be3))*(1.000 + DCOS(ga3)))/(DSIN(be3)*DSIN(ga3)) )**2 ) ) *bvz3
1772 -----
1773 ! Estimation along point 5
1774 -----
1775 GE = GE + DSQRT(yvx5**2 + yvy5**2 + yvz5**2)*DSIN(be4) *%
1776 & DLOG( DSQRT( ( ((1.000 + DCOS(be4))*(1.000 + DCOS(ga4)))/(DSIN(be4)*DSIN(ga4)) )**2 ) ) *bvz4
1777 GEY = GEY + DSQRT(yvx5**2 + yvy5**2 + yvz5**2)*DSIN(be4) *%
1778 & DLOG( DSQRT( ( ((1.000 + DCOS(be4))*(1.000 + DCOS(ga4)))/(DSIN(be4)*DSIN(ga4)) )**2 ) ) *bvy4
1779 GEZ = GEZ + DSQRT(yvx5**2 + yvy5**2 + yvz5**2)*DSIN(be4) *%
1780 & DLOG( DSQRT( ( ((1.000 + DCOS(be4))*(1.000 + DCOS(ga4)))/(DSIN(be4)*DSIN(ga4)) )**2 ) ) *bvz4
1781 -----
1782 ! Estimation along point 3
1783 -----
1784 GE = GE + DSQRT(yvx3**2 + yvy3**2 + yvz3**2)*DSIN(be5) *%
1785 & DLOG( DSQRT( ( ((1.000 + DCOS(be5))*(1.000 + DCOS(ga5)))/(DSIN(be5)*DSIN(ga5)) )**2 ) ) *bvz5
1786 GEY = GEY + DSQRT(yvx3**2 + yvy3**2 + yvz3**2)*DSIN(be5) *%
1787 & DLOG( DSQRT( ( ((1.000 + DCOS(be5))*(1.000 + DCOS(ga5)))/(DSIN(be5)*DSIN(ga5)) )**2 ) ) *bvy5
1788 GEZ = GEZ + DSQRT(yvx3**2 + yvy3**2 + yvz3**2)*DSIN(be5) *%
1789 & DLOG( DSQRT( ( ((1.000 + DCOS(be5))*(1.000 + DCOS(ga5)))/(DSIN(be5)*DSIN(ga5)) )**2 ) ) *bvz5
1790 -----
1791 ! Estimation along point 6
1792 -----
1793 GE = GE + DSQRT(yvx6**2 + yvy6**2 + yvz6**2)*DSIN(be6) *%
1794 & DLOG( DSQRT( ( ((1.000 + DCOS(be6))*(1.000 + DCOS(ga6)))/(DSIN(be6)*DSIN(ga6)) )**2 ) ) *bvz6
1795 GEY = GEY + DSQRT(yvx6**2 + yvy6**2 + yvz6**2)*DSIN(be6) *%
1796 & DLOG( DSQRT( ( ((1.000 + DCOS(be6))*(1.000 + DCOS(ga6)))/(DSIN(be6)*DSIN(ga6)) )**2 ) ) *bvy6
1797 GEZ = GEZ + DSQRT(yvx6**2 + yvy6**2 + yvz6**2)*DSIN(be6) *%
1798 & DLOG( DSQRT( ( ((1.000 + DCOS(be6))*(1.000 + DCOS(ga6)))/(DSIN(be6)*DSIN(ga6)) )**2 ) ) *bvz6
1799 -----
1800 GE = GE*crvml(k)

```

```

1801 GEY = GEY*crvml(k)
1802 GEZ = GEZ*crvml(k)
1803 -----
1804 END SUBROUTINE intr_1in_sing5
1805 -----
1806 !-----
1807 END MODULE Mod_Prtcl3D_slp
1808

```



```

1 MODULE Mod_sgf_3d_fs
2 =====
3 | Version: 0.5 created on 26 / IX / 2007
4 |
5 | C. Pozrikidis
6 |
7 | Version: 0.7 created on -- / III / 2010
8 |
9 | Marco Antonio Reyes Huesca
10 |
11 | Version: 0.8 created on 22 / 03 / 2012
12 |
13 | Version: 0.9 created on 03 / 09 / 2012
14 |
15 | Version: 1.0 created on 13 / 11 / 2012
16 |
17 | Alfredo Sanjuan Sanjuan
18 =====
19 CONTAINS
20 =====
21 SUBROUTINE sgf_3d_fs(x, y, z, &
22 & x0, y0, z0, &
23 & Gxx, Gxy, Gxz, &
24 & Gyx, Gyy, Gyz, &
25 & Gzx, Gzy, Gzz )
26 =====
27 | Free-space Green's function: Stokeslet Pozrikidis (1992, p. 23)
28 | This new version of sfg_3d_fs only calculates the Green's function: Stokeslet. The stress associated is
29 | calculated in other module.
30 |
31 | USE Mod_SharedVars, ONLY: DBL, ULog
32 |
33 | IMPLICIT NONE
34 |
35 | Variables
36 |
37 | REAL (KIND = DBL), INTENT(IN) :: x, y, z |coordinates of collocation point of the element
38 | REAL (KIND = DBL), INTENT(IN) :: x0, y0, z0 |coordinates of the singularity
39 | REAL (KIND = DBL), INTENT(INOUT) :: Gxx, Gxy, Gxz, & |Free-space Green's function of Stokeslet. integrated
40 | & Gyx, Gyy, Gyz, & |ij component over the element
41 | & Gzx, Gzy, Gzz
42 |
43 | Variables inside the subroutine
44 |
45 | REAL (KIND = DBL) :: dx, dy, dz |differences between the (x,y,z) and (x0,y0,z0) coordinates
46 | REAL (KIND = DBL) :: dxx, dxy, dxz, & |square distance d2=dx2+dy2+dz2
47 | & dyy, dyz, dzz
48 | REAL (KIND = DBL) :: r, r3 |distance r between the (x,y,z) point and (x0,y0,z0) and r**3
49 | REAL (KIND = DBL) :: r1, r13 |inverse of distance r and inverse of r**3
50 |
51 | Initialize
52 | Gxx = 0.000
53 | Gxy = 0.000
54 | Gxz = 0.000
55 | Gyx = 0.000
56 | Gyy = 0.000
57 | Gyz = 0.000
58 | Gzx = 0.000
59 | Gzy = 0.000
60 | Gzz = 0.000
61 |
62 | dx = 0.000
63 | dy = 0.000
64 | dz = 0.000
65 |
66 | dxx = 0.000
67 | dxy = 0.000
68 | dxz = 0.000
69 | dyy = 0.000
70 | dyz = 0.000
71 | dzz = 0.000
72 |
73 | r = 0.000
74 | r3 = 0.000
75 | r1 = 0.000
76 | r13 = 0.000
77 |
78 |
79 |
80 |
81 |
82 |
83 |
84 |
85 |
86 |
87 |
88 |
89 |
90 |
91 |
92 |
93 |
94 |
95 |
96 |
97 |
98 |
99 |
100 |
101 |
102 |
103 |
104 |
105 |
106 |
107 |
108 |
109 |
110 |
111 |
112 |
113 |
114 |
115 |
116 |
117 |
118 |
119 |
120 |
121 |
122 |
123 |
124 |
125 |
126 |
127 |
128 |
129 |
130 |
131 |
132 |
133 |
134 |
135 |
136 |
137 |
138 |
139 |
140 |
141 |
142 |
143 |
144 |

```

```

73 |Start the magic
74 | dx = (x-x0 )
75 | dy = (y-y0 )
76 | dz = (z-z0 )
77 |
78 | dxx = dx*dx
79 | dxy = dx*dy
80 | dxz = dx*dz
81 | dyy = dy*dy
82 | dyz = dy*dz
83 | dzz = dz*dz
84 |
85 | r = DSQRT(dxx + dyy + dzz)
86 | r3 = r**3
87 | r1 = 1.000/r
88 | r13 = 1.000/r3
89 |
90 | Gxx = r1 + dx*r13
91 | Gxy = dx*y*r13
92 | Gxz = dx*z*r13
93 | Gyy = r1 + dyy*r13
94 | Gyz = dy*z*r13
95 | Gzz = r1 + dzz*r13
96 |
97 | Gyx = Gxy
98 | Gzx = Gxz
99 | Gzy = Gyz
100 |
101 | Done
102 |
103 | END SUBROUTINE sgf_3d_fs
104 |
105 |
106 | SUBROUTINE sgf_3d_fsing(x, y, z, &
107 | & x0, y0, z0, &
108 | & Gxx, Gxy, Gxz, &
109 | & Gyx, Gyy, Gyz, &
110 | & Gzx, Gzy, Gzz, &
111 | & ABS)
112 |
113 |
114 | Free-space Green's function: Stokeslet Pozrikidis (1992, p. 23)
115 | This new version of sfg_3d_fs only calculates the Green's function: Stokeslet. The stress associated is
116 | calculated in other module.
117 |
118 | USE Mod_SharedVars, ONLY: DBL, ULog
119 |
120 | IMPLICIT NONE
121 |
122 | Variables
123 |
124 | REAL (KIND = DBL), INTENT(IN) :: x, y, z |coordinates of collocation point of the element
125 | REAL (KIND = DBL), INTENT(IN) :: x0, y0, z0 |coordinates of the singularity
126 | REAL (KIND = DBL), INTENT(INOUT) :: Gxx, Gxy, Gxz, & |Free-space Green's function of Stokeslet. integrated
127 | & Gyx, Gyy, Gyz, & |ij component over the element
128 | & Gzx, Gzy, Gzz
129 | REAL (KIND = DBL), INTENT(IN) :: ABS |var on posrikidis
130 |
131 |
132 | Variables inside the subroutine
133 |
134 | REAL (KIND = DBL) :: dx, dy, dz |differences between the (x,y,z) and (x0,y0,z0) coordinates
135 | REAL (KIND = DBL) :: dxx, dxy, dxz, & |square distance d2=dx2+dy2+dz2
136 | & dyy, dyz, dzz
137 | REAL (KIND = DBL) :: r, r3 |distance r between the (x,y,z) point and (x0,y0,z0) and r**3
138 | REAL (KIND = DBL) :: r1, r13 |inverse of distance r and inverse of r**3
139 |
140 | Initialize
141 | Gxx = 0.000
142 | Gxy = 0.000
143 | Gxz = 0.000
144 | Gyx = 0.000

```

```
145 Gyy = 0.000
146 Gyz = 0.000
147 Gzx = 0.000
148 Gzy = 0.000
149 Gzz = 0.000
150 dx = 0.000
151 dy = 0.000
152 dz = 0.000
153 |-----|
154 dxx = 0.000
155 dxy = 0.000
156 dxz = 0.000
157 dyy = 0.000
158 dyz = 0.000
159 dzz = 0.000
160 |-----|
161 r = 0.000
162 r3 = 0.000
163 r1 = 0.000
164 r13 = 0.000
165 |-----|
166 |Start the magic
167 dx = (x-x0 )
168 dy = (y-y0 )
169 dz = (z-z0 )
170 |-----|
171 dxx = dx*dx
172 dxy = dx*dy
173 dxz = dx*dz
174 dyy = dy*dy
175 dyz = dy*dz
176 dzz = dz*dz
177 |-----|
178 r = DSQRT(dxx + dyy + dzz)
179 r3 = r**3
180 r1 = 1.000/r
181 r13 = 1.000/r3
182 |-----|
183 Gxx = 1.000 + dxx/ABS**2
184 Gxy = dxy/ABS**2
185 Gxz = dxz/ABS**2
186 Gyy = 1.000 + dyy/ABS**2
187 Gyz = dyz/ABS**2
188 Gzz = 1.000 + dzz/ABS**2
189 |-----|
190 Gyx = Gxy
191 Gzx = Gxz
192 Gzy = Gyz
193 |-----|
194 | Done
195 |-----|
196 END SUBROUTINE sgf_3d_fsIng
197 |=====|
198 END MODULE Mod_sgf_3d_fs
199
```

```

1 MODULE Mod_Prtc13D_DLP
2 =====
3 ! Version: 0.5 created on 26 / IX / 2007
4 !
5 !
6 ! Version: 0.7 created on -- / III / 2010
7 !
8 !
9 ! Version: 0.9 created on 22 / 03 / 2012
10 ! Version: 1.0 created on 13 / 10 / 2012
11 !
12 =====
13 CONTAINS
14 =====
15 SUBROUTINE Intrg_Trgl_s(x0, y0, z0, &
16 & x1, y1, z1, &
17 & k, &
18 & TExx, TExy, TEz, &
19 & TEyx, TEyy, TEyz, &
20 & TEzx, TEzy, TEzz, &
21 & mint)
22 =====
23 ! This subroutine integrates the Green's function over a non-singular triangle numbered k
24 !
25 USE Mod_SharedVars, ONLY: DBL, ULog, eps, Pi, &
26 & p, ne, n, nbe, &
27 & alphaQ, betaQ, gammaQ, &
28 & xIQ, etaQ, wq, &
29 & Ns,Np
30 =====
31 USE Mod_sgf_3d_sfs
32 USE Mod_nodal_interp
33 =====
34 IMPLICIT NONE
35 =====
36 ! Variables
37 REAL (KIND = DBL), INTENT(IN) :: x0, y0, z0 !coordinates of collocation point analysis
38 REAL (KIND = DBL), INTENT(IN) :: x1, y1, z1 !coordinates of element collocation point
39 INTEGER, INTENT(IN) :: k !element index
40 REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TEz, & !Integrated ij component over the element
41 & TEyx, TEyy, TEyz, &
42 & TEzx, TEzy, TEzz
43 INTEGER, INTENT(IN) :: mint !order of triangle quadrature
44 =====
45 ! Variables inside the subroutine
46 INTEGER :: i, j !counters
47 INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
48 =====
49 REAL (KIND = DBL) :: x1, eta !constants of weigh to integrate over a triangle
50 REAL (KIND = DBL) :: x, y, z !coordinates of the f(x,y,z)= F(x1,eta)
51 REAL (KIND = DBL) :: DxDx1, DyDx1, DzDx1, & !Derivates of the tangential vectors over the element
52 & DxDet, DyDet, DzDet
53 REAL (KIND = DBL) :: vnx, vny, vnz !normal vector coordinates of the element
54 REAL (KIND = DBL) :: hs !surface metric on a triangle
55 REAL (KIND = DBL) :: fc !integration weigh coefficient
56 REAL (KIND = DBL) :: Txxx, Txyx, Txxz, & !free-space Green's function of Stresslet. integrated ijk
57 & Txyx, Txyy, Txyz, &
58 & Tzxx, Tzxy, Tzzz, &
59 & Txxx, Txyx, Txxz, &
60 & Txyx, Txyy, Txyz, &
61 & Tzxx, Tzxy, Tzzz, &
62 & Tzxx, Tzxy, Tzzz, &
63 & Tzyx, Tzyy, Tzyz, &
64 & Tzxx, Tzxy, Tzzz, &
65 & Tzxx, Tzxy, Tzzz
66 =====
67 ! Initialize
68 TExx = 0.000
69 TExy = 0.000
70 TEz = 0.000
71 =====
72 TExx = 0.000

```

```

73 TExy = 0.000
74 TEz = 0.000
75 =====
76 TExx = 0.000
77 TExy = 0.000
78 TEz = 0.000
79 =====
80 ! vertices of the kth triangle
81 i1 = n(k,1)
82 i2 = n(k,2)
83 i3 = n(k,3)
84 i4 = n(k,4)
85 i5 = n(k,5)
86 i6 = n(k,6)
87 =====
88 ! DO i = 1, mint
89 ! xi = xIQ(i)
90 ! eta = etaQ(i)
91 =====
92 ! CALL interp_p(p(i1,1), p(i1,2), p(i1,3), &
93 & p(i2,1), p(i2,2), p(i2,3), &
94 & p(i3,1), p(i3,2), p(i3,3), &
95 & p(i4,1), p(i4,2), p(i4,3), &
96 & p(i5,1), p(i5,2), p(i5,3), &
97 & p(i6,1), p(i6,2), p(i6,3), &
98 & alphaQ(k), betaQ(k), gammaQ(k), &
99 & xi, eta, &
100 & x, y, z, &
101 & DxDx1, DyDx1, DzDx1, &
102 & DxDet, DyDet, DzDet, &
103 & vnx, vny, vnz, &
104 & hs)
105 =====
106 ! CALL sgf_3d_sfs( x, y, z, &
107 & x0, y0, z0, &
108 & Txxx,Txyx,Txxz, &
109 & Txyx,Txyy,Txyz, &
110 & Tzxx,Tzxy,Tzzz, &
111 & Txxx,Txyx,Txxz, &
112 & Txyx,Txyy,Txyz, &
113 & Tzxx,Tzxy,Tzzz, &
114 & Tzxx,Tzxy,Tzzz, &
115 & Tzyx,Tzyy,Tzyz, &
116 & Tzxx,Tzxy,Tzzz )
117 =====
118 ! fc = 0.500*hs**wq(1)
119 =====
120 ! TExx = TExx + (Txxx*vnx*fc + Txyx*vny*fc + Txxz*vnz*fc)
121 ! TExy = TExy + (Txyx*vnx*fc + Txyy*vny*fc + Txyz*vnz*fc)
122 ! TEz = TEz + (Tzxx*vnx*fc + Tzxy*vny*fc + Tzzz*vnz*fc)
123 ! TEyx = TEyx + (Txyx*vnx*fc + Txyy*vny*fc + Txyz*vnz*fc)
124 ! TEyy = TEyy + (Txyx*vnx*fc + Txyy*vny*fc + Txyz*vnz*fc)
125 ! TEyz = TEyz + (Tzxx*vnx*fc + Tzxy*vny*fc + Tzzz*vnz*fc)
126 ! TEzx = TEzx + (Tzxx*vnx*fc + Tzxy*vny*fc + Tzzz*vnz*fc)
127 ! TEzy = TEzy + (Tzyx*vnx*fc + Tzyy*vny*fc + Tzyz*vnz*fc)
128 ! TEzz = TEzz + (Tzxx*vnx*fc + Tzxy*vny*fc + Tzzz*vnz*fc)
129 ! END DO
130 =====
131 ! Integrate over six flat triangles
132 CALL Intr_lin_sing_s2( x0, y0, z0, &
133 & x1, y1, z1, &
134 & p(i1,1),p(i1,2),p(i1,3), &
135 & p(i4,1),p(i4,2),p(i4,3), &
136 & mint, &
137 & TExx, TExy, TEz, &
138 & TExx, TExy, TEz, &
139 & TExx, TExy, TEz, &
140 & TExx, TExy, TEz, &
141 =====
142 CALL Intr_lin_sing_s2( x0, y0, z0, &
143 & x1, y1, z1, &
144 & p(i4,1),p(i4,2),p(i4,3), &

```

```

145      & p(12,1),p(12,2),p(12,3),&
146      & mint, &
147      & TExx, TEyy, TEzz, &
148      & TEyx, TEyz, TEzx, &
149      & TEzx, TEzy, TEzz )
150 -----
151 CALL intr_lin_sing_s2( x0, y0, z0, &
152 & x1, y1, z1, &
153 & p(12,1),p(12,2),p(12,3),&
154 & p(15,1),p(15,2),p(15,3),&
155 & mint, &
156 & TExx, TEyy, TEzz, &
157 & TEyx, TEyz, TEzx, &
158 & TEzx, TEzy, TEzz )
159 -----
160 CALL intr_lin_sing_s2( x0, y0, z0, &
161 & x1, y1, z1, &
162 & p(15,1),p(15,2),p(15,3),&
163 & p(13,1),p(13,2),p(13,3),&
164 & mint, &
165 & TExx, TEyy, TEzz, &
166 & TEyx, TEyz, TEzx, &
167 & TEzx, TEzy, TEzz )
168 -----
169 CALL intr_lin_sing_s2( x0, y0, z0, &
170 & x1, y1, z1, &
171 & p(13,1),p(13,2),p(13,3),&
172 & p(16,1),p(16,2),p(16,3),&
173 & mint, &
174 & TExx, TEyy, TEzz, &
175 & TEyx, TEyz, TEzx, &
176 & TEzx, TEzy, TEzz )
177 -----
178 CALL intr_lin_sing_s2( x0, y0, z0, &
179 & x1, y1, z1, &
180 & p(16,1),p(16,2),p(16,3),&
181 & p(11,1),p(11,2),p(11,3),&
182 & mint, &
183 & TExx, TEyy, TEzz, &
184 & TEyx, TEyz, TEzx, &
185 & TEzx, TEzy, TEzz )
186 -----
187 ! Done
188 -----
189 END SUBROUTINE Intgr_Trgl_s
190 -----
191 -----
192 SUBROUTINE Intgr_Trgl_sing_s(x0, y0, z0, &
193 & k, &
194 & TExx, TEyy, TEzz, &
195 & TEyx, TEyz, TEzx, &
196 & TEzx, TEzy, TEzz, &
197 & mint )
198 -----
199 ! Integrate the Green's function over the kth singular quadratic triangle. This is done by breaking up the
200 ! singular triangle into six flat triangles, and then integrating individually over the flat triangles in local
201 ! polar coordinates.
202 -----
203 USE Mod_SharedVars, ONLY: DBL, ULog, eps, Pi, &
204 & p, ne, n, nbe, &
205 & x10, etq, nq, &
206 & Ns,Np
207 -----
208 IMPLICIT NONE
209 -----
210 REAL (KIND = DBL), INTENT(IN) :: x0, y0, z0 !coordinates of collocationpoint of the element
211 INTEGER, INTENT(IN) :: k !element index
212 REAL (KIND = DBL), INTENT(OUT) :: TExx, TEyy, TEzz, & !Integrated ij component over the element
213 & TEyx, TEyz, TEzx, &
214 & TEzx, TEzy, TEzz
215 INTEGER, INTENT(IN) :: mint !order of Gauss-Legendre quadrature
216 -----

```

```

217 !-----
218 ! Variables inside the subroutine
219 -----
220 INTEGER :: i, j
221 INTEGER :: i1, i2, i3, i4, i5, i6
222 -----
223 ! launching
224 i1 = n(k,1)
225 i2 = n(k,2)
226 i3 = n(k,3)
227 i4 = n(k,4)
228 i5 = n(k,5)
229 i6 = n(k,6)
230 -----
231 TExx = 0.000
232 TEyy = 0.000
233 TEzz = 0.000
234 -----
235 TEyx = 0.000
236 TEyz = 0.000
237 TEzx = 0.000
238 -----
239 TEzx = 0.000
240 TEzy = 0.000
241 TEzz = 0.000
242 -----
243 ! Integrate over three flat triangles
244 -----
245 ! CALL intr_lin_sing_s( x0, y0, z0, &
246 & p(i1,1),p(i1,2),p(i1,3),&
247 & p(i2,1),p(i2,2),p(i2,3),&
248 & TExx, TEyy, TEzz, &
249 & TEyx, TEyz, TEzx, &
250 & TEzx, TEzy, TEzz, &
251 & cdt, NGL )
252 -----
253 ! CALL intr_lin_sing_s( x0, y0, z0, &
254 & p(i2,1),p(i2,2),p(i2,3),&
255 & p(i3,1),p(i3,2),p(i3,3),&
256 & TExx, TEyy, TEzz, &
257 & TEyx, TEyz, TEzx, &
258 & TEzx, TEzy, TEzz, &
259 & cdt, NGL )
260 -----
261 ! CALL intr_lin_sing_s( x0, y0, z0, &
262 & p(i3,1),p(i3,2),p(i3,3),&
263 & p(i1,1),p(i1,2),p(i1,3),&
264 & TExx, TEyy, TEzz, &
265 & TEyx, TEyz, TEzx, &
266 & TEzx, TEzy, TEzz, &
267 & cdt, NGL )
268 -----
269 ! Integrate over six flat triangles
270 -----
271 CALL intr_lin_sing_s( x0, y0, z0, &
272 & p(i1,1),p(i1,2),p(i1,3),&
273 & p(i4,1),p(i4,2),p(i4,3),&
274 & TExx, TEyy, TEzz, &
275 & TEyx, TEyz, TEzx, &
276 & TEzx, TEzy, TEzz, &
277 & mint )
278 -----
279 CALL intr_lin_sing_s( x0, y0, z0, &
280 & p(i4,1),p(i4,2),p(i4,3),&
281 & p(i2,1),p(i2,2),p(i2,3),&
282 & TExx, TEyy, TEzz, &
283 & TEyx, TEyz, TEzx, &
284 & TEzx, TEzy, TEzz, &
285 & mint )
286 -----
287 CALL intr_lin_sing_s( x0, y0, z0, &
288 & p(i2,1),p(i2,2),p(i2,3),&

```

```

289      & p(15,1),p(15,2),p(15,3), &
290      & TExx, TExy, TExz, &
291      & TEyx, TEyy, TEyz, &
292      & TEzx, TEzy, TEzz, &
293      & mint
294 -----
295 CALL intr_lin_sing_s( x0, y0, z0, &
296      & p(15,1),p(15,2),p(15,3), &
297      & p(13,1),p(13,2),p(13,3), &
298      & TExx, TExy, TExz, &
299      & TEyx, TEyy, TEyz, &
300      & TEzx, TEzy, TEzz, &
301      & mint
302 -----
303 CALL intr_lin_sing_s( x0, y0, z0, &
304      & p(13,1),p(13,2),p(13,3), &
305      & p(16,1),p(16,2),p(16,3), &
306      & TExx, TExy, TExz, &
307      & TEyx, TEyy, TEyz, &
308      & TEzx, TEzy, TEzz, &
309      & mint
310 -----
311 CALL intr_lin_sing_s( x0, y0, z0, &
312      & p(16,1),p(16,2),p(16,3), &
313      & p(11,1),p(11,2),p(11,3), &
314      & TExx, TExy, TExz, &
315      & TEyx, TEyy, TEyz, &
316      & TEzx, TEzy, TEzz, &
317      & mint
318 -----
319 END SUBROUTINE Intr_Trgl_Sing_s
320 -----
321 -----
322 SUBROUTINE intr_lin_sing_s(x1, y1, z1, &
323      & x2, y2, z2, &
324      & x3, y3, z3, &
325      & TExx, TExy, TExz, &
326      & TEyx, TEyy, TEyz, &
327      & TEzx, TEzy, TEzz, &
328      & mint
329 -----
330 ! Integrates the Green's function over a flat triangle in local polar coordinates with origin at the singular
331 ! point: (x1,y1,z1). The subroutine is based from Pozriquidis 2002 pp. 119-120 in the technique 5.2.8 - 5.2.11
332 -----
333 USE Mod_SharedVars, ONLY: DBL, ULog, eps, Pi, &
334      & Zz, Ww, &
335      & Ns, Np, xiq, etq, wq
336 USE Mod_sgf_3d_sfs
337 -----
338 IMPLICIT NONE
339 -----
340 ! Variables
341 -----
342 REAL (KIND = DBL), INTENT(IN) :: x1, y1, z1 !coordinates of flat triangle vertice
343 REAL (KIND = DBL), INTENT(IN) :: x2, y2, z2
344 REAL (KIND = DBL), INTENT(IN) :: x3, y3, z3
345 REAL (KIND = DBL), INTENT(INOUT) :: TExx, TExy, TExz, & !integrated ij component over the element
346      & TEyx, TEyy, TEyz, &
347      & TEzx, TEzy, TEzz
348 INTEGER, INTENT(IN) :: mint !order of Gauss-Legendre quadrature
349 -----
350 ! Variables inside the subroutine
351 -----
352 INTEGER :: i, j !counters
353 INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain the node number in the vertices of each element
354 -----
355 REAL (KIND = DBL) :: pi4, piq !constants to integrate over a circle element in Phi and r
356 REAL (KIND = DBL) :: x1, et, zt !constants of weighth to integrate over a flat triangle
357 REAL (KIND = DBL) :: dx, dy !surface metrics
358 REAL (KIND = DBL) :: vnx, vny, vnz !triangle normal vector
359 REAL (KIND = DBL) :: area, hs !triangle area and surface metric on a flat triangle
360 REAL (KIND = DBL) :: phi, cph, sph !variable Phi, Cos(phi) and Sin(phi)

```

```

361 REAL (KIND = DBL) :: r, rmax, rmaxh !variable r and values of variable r to integrate
362 REAL (KIND = DBL) :: x, y, z !coordinates of f(xi, et, zt)
363 REAL (KIND = DBL) :: cf, cf1, cf2 !integrate weights
364 REAL (KIND = DBL) :: asm, bsm !triangle area and its derivative
365 REAL (KIND = DBL) :: B, C !integrate arc coefficients
366 REAL (KIND = DBL) :: b1, b2, c1 !dummy integrate arc constants
367 REAL (KIND = DBL), DIMENSION(3) :: xxi, eet, zzt !variables of weighth over in triangle (xi,eta)
368 REAL (KIND = DBL) :: Txxx, Txyx, Txxz, & !Free-space Green's function of Stresslet. integrated ijk
369      & Txyx, Txyy, Txyz, &
370      & Txxz, Txyy, Txxz, &
371      & Tyxx, Tyxy, Tyxz, &
372      & Tyxx, Tyyy, Tyyz, &
373      & Tyzx, Tyzy, Tyzz, &
374      & Tzxx, Tzxy, Tzxz, &
375      & Tzxx, Tzzy, Tzyz, &
376      & Tzxx, Tzzy, Tzzz
377 REAL (KIND = DBL) :: Txx, Txy, Txz, & !Stresslets Dummy coefficients
378      & Tyx, Tyy, Tyz, &
379      & Txx, Tzy, Tzz
380 REAL (KIND = DBL) :: Rxx, Rxy, Rxz, & !Stresslets Dummy coefficients
381      & Ryx, Ryy, Ryz, &
382      & Rzx, Rzy, Rzz
383 -----
384 ! constants
385 piq = 0.2500*pi
386 pi4 = 4.000*pi
387 -----
388 ! Initialize
389 xxi(1) = 0.500
390 eet(1) = 0.500
391 zzt(1) = 0.000
392 xxi(2) = 0.500
393 eet(2) = 0.000
394 zzt(2) = 0.500
395 xxi(3) = 0.000
396 eet(3) = 0.500
397 zzt(3) = 0.500
398 -----
399 Txx = 0.000
400 Txy = 0.000
401 Txz = 0.000
402 Tyx = 0.000
403 Tyy = 0.000
404 Tyz = 0.000
405 Tzx = 0.000
406 Tzy = 0.000
407 Tzz = 0.000
408 -----
409 Rxx = 0.000
410 Rxy = 0.000
411 Rxz = 0.000
412 Ryx = 0.000
413 Ryy = 0.000
414 Ryz = 0.000
415 Rzx = 0.000
416 Rzy = 0.000
417 Rzz = 0.000
418 -----
419 ! compute surface metric: hs
420 dx = DSQRT( (x2-x1)**2+(y2-y1)**2+(z2-z1)**2 )
421 dy = DSQRT( (x3-x1)**2+(y3-y1)**2+(z3-z1)**2 )
422 -----
423 vnx = ((y2-y1)*(z3-z1))-((z2-z1)*(y3-y1))
424 vny = ((z2-z1)*(x3-x1))-((x2-x1)*(z3-z1))
425 vnz = ((x2-x1)*(y3-y1))-((y2-y1)*(x3-x1))
426 -----
427 hs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
428 vnx = vnx/hs
429 vny = vny/hs
430 vnz = vnz/hs

```

```

433 |-----|
434 | Surface metric on a flat triangle
435 | area = hs/2.000
436 |-----|
437 | Initialize
438 |-----|
439 | Triangle area
440 | asm = 0.000
441 |-----|
442 | Apply the formulas from Pozrikidis (2002, p. 119)
443 | b1=((x3-x1)*(x2-x1))+((y3-y1)*(y2-y1))+((z3-z1)*(z2-z1))
444 | b2=DSQRT((x2-x1)**2+(y2-y1)**2+(z2-z1)**2)
445 | c1=DSQRT((x3-x1)**2+(y3-y1)**2+(z3-z1)**2)
446 | b= b1/b2**2
447 | C= c1**2/b2**2
448 |-----|
449 | Apply the Double quadrature
450 |-----|
451 | Integration wrt phi
452 | Cycle1: DO i = 1, NGL
453 |   ph = pi*(1.000+zz(1))
454 |   cph = DCOS(ph)
455 |   sph = DSIN(ph)
456 |   rmax = 1.000/(cph+sph)
457 |   rmaxh = 0.500*rmax
458 |-----|
459 | Derivative of asm
460 |   bsm = 0.000
461 |-----|
462 | Integration wrt r
463 | Cycle2: DO j=1,NGL
464 |   r = rmaxh*(1.000+zz(j))
465 |   xi = r*cph
466 |   et = r*sph
467 |   zt = 1.000-xi-et
468 |-----|
469 |   x = x1*zt + x2*xi + x3*et
470 |   y = y1*zt + y2*xi + y3*et
471 |   z = z1*zt + z2*xi + z3*et
472 |-----|
473 |   CALL sfg_3d_sfs( x, y, z, &
474 |     & x1, y1, z1, &
475 |     & Txxx, Txyx, Txxz, &
476 |     & Txyx, Txyy, Txyz, &
477 |     & Txzx, Txzy, Txzz, &
478 |     & Tyxx, Tyxy, Tyxz, &
479 |     & Tyyx, Tyyy, Tyyz, &
480 |     & Tyzx, Tyzy, Tyzz, &
481 |     & Tzxx, Tzxy, Tzxz, &
482 |     & Tzyx, Tzyy, Tzyz, &
483 |     & Tzxx, Tzxy, Tzxx, )
484 |-----|
485 |   cf1 = ww(j)*r
486 |-----|
487 |   bsm = bsm + cf1
488 |-----|
489 |   Rxx = Rxx + (Txxx*vmx + Txyx*vny + Txzx*vnz)*cf1
490 |   Rxy = Rxy + (Txyx*vmx + Txyy*vny + Tyxz*vnz)*cf1
491 |   Rxz = Rxz + (Txzx*vmx + Txzy*vny + Tzzz*vnz)*cf1
492 |   Ryx = Ryx + (Tyxx*vmx + Tyxy*vny + Tyxz*vnz)*cf1
493 |   Ryy = Ryy + (Tyyx*vmx + Tyyy*vny + Tyyz*vnz)*cf1
494 |   Ryz = Ryz + (Tyzx*vmx + Tyzy*vny + Tyzz*vnz)*cf1
495 |   Rzx = Rzx + (Tzxx*vmx + Tzxy*vny + Tzxx*vnz)*cf1
496 |   Rzy = Rzy + (Tzyx*vmx + Tzyy*vny + Tzyz*vnz)*cf1
497 |   Rzz = Rzz + (Tzxx*vmx + Tzzy*vny + Tzzz*vnz)*cf1
498 |   END DO Cycle2
499 |-----|
500 |   cf2 = ww(1)*rmaxh
501 |-----|
502 |   asm = asm + bsm*cf2
503 |-----|
504 |   Txx = Txx + (cf2*Rxx/DSQRT(cph**2+(B*DSIN(2*ph)))+(C*sph**2))

```

```

505 |   Txy = Txy + (cf2*Rxy/DSQRT(cph**2+(B*DSIN(2*ph)))+(C*sph**2))
506 |   Txz = Txz + (cf2*Rxz/DSQRT(cph**2+(B*DSIN(2*ph)))+(C*sph**2))
507 |   Tyx = Tyx + (cf2*Ryx/DSQRT(cph**2+(B*DSIN(2*ph)))+(C*sph**2))
508 |   Tyy = Tyy + (cf2*Ryy/DSQRT(cph**2+(B*DSIN(2*ph)))+(C*sph**2))
509 |   Tyz = Tyz + (cf2*Ryz/DSQRT(cph**2+(B*DSIN(2*ph)))+(C*sph**2))
510 |   Tzx = Tzx + (cf2*Rzx/DSQRT(cph**2+(B*DSIN(2*ph)))+(C*sph**2))
511 |   Tzy = Tzy + (cf2*Rzy/DSQRT(cph**2+(B*DSIN(2*ph)))+(C*sph**2))
512 |   Tzz = Tzz + (cf2*Rzz/DSQRT(cph**2+(B*DSIN(2*ph)))+(C*sph**2))
513 |-----|
514 | initialize again to the next triangle
515 |   Rxx = 0.000
516 |   Rxy = 0.000
517 |   Rxz = 0.000
518 |   Ryx = 0.000
519 |   Ryy = 0.000
520 |   Ryz = 0.000
521 |   Rzx = 0.000
522 |   Rzy = 0.000
523 |   Rzz = 0.000
524 | END DO Cycle1
525 |-----|
526 |-----|
527 | DO i = 1, nint
528 |   xi = xiq(1)
529 |   et = etq(i)
530 |   zt = 1.000 - xi - et
531 |-----|
532 |   x = x1*zt + x2*xi + x3*et
533 |   y = y1*zt + y2*xi + y3*et
534 |   z = z1*zt + z2*xi + z3*et
535 |-----|
536 |   CALL sfg_3d_sfs( x, y, z, &
537 |     & x1, y1, z1, &
538 |     & Txxx, Txyx, Txxz, &
539 |     & Txyx, Txyy, Txyz, &
540 |     & Txzx, Txzy, Txzz, &
541 |     & Tyxx, Tyxy, Tyxz, &
542 |     & Tyyx, Tyyy, Tyyz, &
543 |     & Tyzx, Tyzy, Tyzz, &
544 |     & Tzxx, Tzxy, Tzxx, &
545 |     & Tzyx, Tzyy, Tzyz, &
546 |     & Tzxx, Tzxy, Tzxx, )
547 |-----|
548 |   cf1 = wq(1)
549 |-----|
550 |   bsm=0.000
551 |   bsm = cf1
552 |-----|
553 |   Rxx = (Txxx*vmx + Txyx*vny + Txzx*vnz)*cf1
554 |   Rxy = (Txyx*vmx + Txyy*vny + Tyxz*vnz)*cf1
555 |   Rxz = (Txzx*vmx + Txzy*vny + Tzzz*vnz)*cf1
556 |   Ryx = (Tyxx*vmx + Tyxy*vny + Tyxz*vnz)*cf1
557 |   Ryy = (Tyyx*vmx + Tyyy*vny + Tyyz*vnz)*cf1
558 |   Ryz = (Tyzx*vmx + Tyzy*vny + Tyzz*vnz)*cf1
559 |   Rzx = (Tzxx*vmx + Tzxy*vny + Tzxx*vnz)*cf1
560 |   Rzy = (Tzyx*vmx + Tzyy*vny + Tzyz*vnz)*cf1
561 |   Rzz = (Tzxx*vmx + Tzzy*vny + Tzzz*vnz)*cf1
562 |-----|
563 |   asm = asm + bsm
564 |-----|
565 |   Txx = Txx + Rxx
566 |   Txy = Txy + Rxy
567 |   Txz = Txz + Rxz
568 |   Tyx = Tyx + Ryx
569 |   Tyy = Tyy + Ryy
570 |   Tyz = Tyz + Ryz
571 |   Tzx = Tzx + Rzx
572 |   Tzy = Tzy + Rzy
573 |   Tzz = Tzz + Rzz
574 |-----|
575 | END DO
576 |-----|

```

```

577 ! complete the quadrature
578 !-----
579   cf = area
580 !-----
581   asm = asm*cf
582 !-----
583   TExx = TExx + cf*Txx
584   TExy = TExy + cf*Txy
585   TExz = TExz + cf*Txz
586   TEyx = TEyx + cf*Tyx
587   TEyy = TEyy + cf*Tyy
588   TEyz = TEyz + cf*Tyz
589   TEzx = TEzx + cf*Tzx
590   TEzy = TEzy + cf*Tzy
591   TEzz = TEzz + cf*Tzz
592 !-----
593 ! If all went well, asm should be equal to "area"
594 ! WRITE (Ulog,100) i,area,asm
595 !-----
596 ! 100 FORMAT (1x,i3,2(f10.5))
597 !-----
598 END SUBROUTINE intr_lin_sing_s
599 !-----
600 SUBROUTINE intr_lin_sing_s2(x0, y0, z0, &
601   & x1, y1, z1, &
602   & x2, y2, z2, &
603   & x3, y3, z3, &
604   & mint, &
605   & TExx, TExy, TExz, &
606   & TEyx, TEyy, TEyz, &
607   & TEzx, TEzy, TEzz )
608 !-----
609 ! Integrates the Green's function over a flat triangle in local polar coordinates with origin at the singular
610 ! point: (x1,y1,z1). The subroutine is based from Pozriquidis 2002 pp. 119-120 in the technique 5.2.8 - 5.2.11
611 !-----
612 USE Mod_SharedVars, ONLY: DBL, ULog, eps, P1, &
613   & ZZ, MM, &
614   & Ns, Np, xiq, etq, wq
615 USE Mod_sgf_3d_sfs
616 !-----
617 IMPLICIT NONE
618 !-----
619 ! Variables
620 !-----
621 REAL (KIND = DBL), INTENT(IN) :: x0, y0, z0 !coordinates of collocation point
622 REAL (KIND = DBL), INTENT(IN) :: x1, y1, z1 !coordinates of flat triangle vertice
623 REAL (KIND = DBL), INTENT(IN) :: x2, y2, z2
624 REAL (KIND = DBL), INTENT(IN) :: x3, y3, z3
625 INTEGER, INTENT(IN) :: mint !element index
626 REAL (KIND = DBL), INTENT(INOUT) :: TExx, TExy, TExz, & !integrated ij component over the element
627   & TEyx, TEyy, TEyz, &
628   & TEzx, TEzy, TEzz
629 !-----
630 ! Variables inside the subroutine
631 !-----
632 INTEGER :: i, j !counters
633 INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain the node number in the vertices of each element
634 !-----
635 REAL (KIND = DBL) :: pi4, piq !constants to integrate over a circle element in Phi and r
636 REAL (KIND = DBL) :: x1, et, zt !constants of weight to integrate over a flat triangle
637 REAL (KIND = DBL) :: dx, dy !surface metrics
638 REAL (KIND = DBL) :: vnx, vny, vnz !triangle normal vector
639 REAL (KIND = DBL) :: area, hs !triangle area and surface metric on a flat triangle
640 REAL (KIND = DBL) :: x, y, z !coordinates of f(x1, et, zt)
641 REAL (KIND = DBL) :: cf, cf1, cf2 !integrate weights
642 REAL (KIND = DBL) :: asm, bsm !triangle area and its derivative
643 REAL (KIND = DBL), DIMENSION(3) :: xxi, eet, zzt !variables of weight over in triangle (xi,eta)
644 REAL (KIND = DBL) :: Txxx, Txyx, Txxz, & !Free-space Green's function of Stresslet. integrated ijk
645   & Txyx, Txyy, Txyz, &
646   & Tzxx, Tzxy, Txz, &
647   & Tyxx, Tyxy, Tyxz, &
648   & Tyyx, Tyyy, Tyyz, &

```

```

649   & Tyzx, Tyzy, Tyzz, &
650   & Tzxx, Tzxy, Tzxz, &
651   & Tzyx, Tzyy, Tzyz, &
652   & Tzzx, Tzzy, Tzzz
653 REAL (KIND = DBL) :: Txx, Txy, Txz, & !Stresslets Dummy coefficients
654   & Tyx, Tyy, Tyz, &
655   & Tzx, Tzy, Tzz
656 REAL (KIND = DBL) :: Rxx, Rxy, Rxz, & !Stresslets Dummy coefficients
657   & Ryx, Ryy, Ryz, &
658   & Rzx, Rzy, Rzz
659 !-----
660 ! Initialize
661 xxi(1) = 0.500
662 eet(1) = 0.500
663 zzt(1) = 0.000
664 xxi(2) = 0.500
665 eet(2) = 0.000
666 zzt(2) = 0.500
667 xxi(3) = 0.000
668 eet(3) = 0.500
669 zzt(3) = 0.500
670 !-----
671 !-----
672 !xxi(1) = 1.000 / 6.000
673 !eet(1) = 1.000 / 6.000
674 !zzt(1) = 2.000 / 3.000
675 !xxi(2) = 2.000 / 3.000
676 !eet(2) = 1.000 / 6.000
677 !zzt(2) = 1.000 / 6.000
678 !xxi(3) = 1.000 / 6.000
679 !eet(3) = 2.000 / 3.000
680 !zzt(3) = 1.000 / 6.000
681 !-----
682 Txx = 0.000
683 Txy = 0.000
684 Txz = 0.000
685 Tyx = 0.000
686 Tyy = 0.000
687 Tyz = 0.000
688 Tzx = 0.000
689 Tzy = 0.000
690 Tzz = 0.000
691 !-----
692 Rxx = 0.000
693 Rxy = 0.000
694 Rxz = 0.000
695 Ryx = 0.000
696 Ryy = 0.000
697 Ryz = 0.000
698 Rzx = 0.000
699 Rzy = 0.000
700 Rzz = 0.000
701 !-----
702 ! compute surface metric: hs
703 !-----
704 dx = DSQRT( (x2-x1)**2+(y2-y1)**2+(z2-z1)**2 )
705 dy = DSQRT( (x3-x1)**2+(y3-y1)**2+(z3-z1)**2 )
706 !-----
707 vnx = ((y2-y1)*(z3-z1))-((z2-z1)*(y3-y1))
708 vny = ((z2-z1)*(x3-x1))-((x2-x1)*(z3-z1))
709 vnz = ((x2-x1)*(y3-y1))-((y2-y1)*(x3-x1))
710 !-----
711 hs = DSQRT(vnx*vnx + vny*vny + vnz*vnz)
712 vnx = vnx/hs
713 vny = vny/hs
714 vnz = vnz/hs
715 !-----
716 ! Surface metric on a flat triangle
717 area = hs/2.000
718 !-----
719 ! Initialize
720 !-----

```



```

721 ! Triangle area
722 asm = 0.0D0
723 -----
724 DO i = 1, mint
725   xi = xiq(i)
726   et = etq(i)
727   zt = 1.0D0 - et - xi
728   cf1 = wq(i)
729 -----
730 ! DO i = 1, 3
731 ! xi = xxi(i)
732 ! et = eet(i)
733 ! zt = zzt(i)
734 ! cf1 = 1.000/3.0D0
735 -----
736 x = x1*zt + x2*xi + x3*et
737 y = y1*zt + y2*xi + y3*et
738 z = z1*zt + z2*xi + z3*et
739 -----
740 CALL sgf_3d_sfs( x, y, z, &
741                & x0, y0, z0, &
742                & Txxx, Txy, Txxz, &
743                & Txyx, Txyy, Txyz, &
744                & Txxz, Txy, Txxz, &
745                & Txyx, Txyy, Txyz, &
746                & Tyxx, Tyyy, Tyyz, &
747                & Tyxz, Tyzy, Tyzz, &
748                & Tzxx, Tzxy, Tzxx, &
749                & Tzyx, Tzyy, Tzyz, &
750                & Tzzx, Tzzy, Tzzz )
751 -----
752 bsm = 0.0D0
753 bsm = cf1
754 -----
755 Rxx = (Txxx*vx + Txyx*vy + Txxz*vz)*cf1
756 Rxy = (Txyx*vx + Txyy*vy + Txyz*vz)*cf1
757 Rxz = (Txxz*vx + Txyy*vy + Txxz*vz)*cf1
758 Ryx = (Tyxx*vx + Tyxy*vy + Tyxz*vz)*cf1
759 Ryy = (Tyxx*vx + Tyyy*vy + Tyyz*vz)*cf1
760 Ryz = (Tyxz*vx + Tyzy*vy + Tyzz*vz)*cf1
761 Rzx = (Tzxx*vx + Tzxy*vy + Tzzz*vz)*cf1
762 Rzy = (Tzxy*vx + Tzzy*vy + Tzzz*vz)*cf1
763 Rzz = (Tzxx*vx + Tzzy*vy + Tzzz*vz)*cf1
764 -----
765 asm = asm + bsm
766 -----
767 Txx = Txx + Rxx
768 Txy = Txy + Rxy
769 Txz = Txz + Rxz
770 Tyx = Tyx + Ryx
771 Tyy = Tyy + Ryy
772 Tyz = Tyz + Ryz
773 Tzx = Tzx + Rzx
774 Tzy = Tzy + Rzy
775 Tzz = Tzz + Rzz
776 -----
777 END DO
778 ! complete the quadrature
779 ! -----
780 cf = area
781 -----
782 !
783 asm = asm*cf
784 -----
785 TExx = TExx + cf*Txx
786 TExy = TExy + cf*Txy
787 TEKz = TEKz + cf*Txz
788 TEyx = TEyx + cf*Tyx
789 TEyy = TEyy + cf*Tyy
790 TEyz = TEyz + cf*Tyz
791 TEzx = TEzx + cf*Tzx
792 TEzy = TEzy + cf*Tzy

```

```

793 TEzz = TEzz + cf*Tzz
794 ! -----
795 ! IF all went well, asm should be equal to "area"
796 ! WRITE (Ulog,100) i,area,asm
797 ! -----
798 ! 100 FORMAT (1x,i3,2(f10.5))
799 ! -----
800 END SUBROUTINE intr_lin_sing_s2
801 -----
802 SUBROUTINE intr_lin_sing_s4(x0e, y0e, z0e, &
803                            & k, &
804                            & TExx, TExy, TEKz, &
805                            & TEyx, TEyy, TEyz, &
806                            & TEzx, TEzy, TEzz )
807 -----
808 ! This subroutine is a new version stokeslet Subroutine.
809 ! Compute:
810 ! *The value of the Stokeslet over each singular element
811 ! Now, (March/ 09 / 2015) this subroutine was made.
812 ! -----
813 !
814 USE Mod_Modal_Interp
815 USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog, pi, eps, vnx0, vny0, vnz0, x0, y0, z0
816 ! -----
817 IMPLICIT NONE
818 ! Variables
819 ! -----
820 REAL (KIND = DBL), INTENT(IN) :: x0e, y0e, z0e !singularty coordinates
821 REAL (KIND = DBL), INTENT(IN) :: x0e, y0e, z0e !singularty coordinates element
822 INTEGER, INTENT(IN) :: k !number of element
823 REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TEKz !value of stresslet in the singular element
824 REAL (KIND = DBL), INTENT(OUT) :: TEyx, TEyy, TEyz !value of stresslet in the singular element
825 REAL (KIND = DBL), INTENT(OUT) :: TEzx, TEzy, TEzz !value of stresslet in the singular element
826 ! -----
827 ! Variables inside the subroutine
828 ! -----
829 INTEGER :: i, j !counters
830 INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
831 ! -----
832 REAL (KIND = DBL) :: cf, fill1, fill2, fill3 !integration weigh coefficients
833 REAL (KIND = DBL) :: DxDx, DyDy, DzDz !tangential vector around the triangle (xi,eta)
834 REAL (KIND = DBL) :: hss !length
835 REAL (KIND = DBL) :: cpi !selection to add the solid angle
836 REAL (KIND = DBL) :: modx0 !length
837 REAL (KIND = DBL) :: bvx1, bvy1, bvz1, & !vectoreial product
838 & bvx2, bvy2, bvz2, &
839 & bvx3, bvy3, bvz3
840 REAL (KIND = DBL) :: QExx, QExy, QExz, & !Iq tensor
841 & QEyx, QEyy, QEyz, &
842 & QEzx, QEzy, QEzz
843 REAL (KIND = DBL) :: PEy, PEz, PE !Iip tensor
844 REAL (KIND = DBL) :: yv1, yv11, yv21 !vector (y1-x0)
845 REAL (KIND = DBL) :: yv2, yv21, yv22 !vector (y2-x0)
846 REAL (KIND = DBL) :: yv3, yv31, yv32 !vector (y2-x0)
847 REAL (KIND = DBL) :: pv1, pv2, pv2 !vector (y2-x0)
848 REAL (KIND = DBL) :: a1, a2, a3 !vector a
849 REAL (KIND = DBL) :: ay1, ay2, ay3 !vector dot product a*y1, a*y2
850 REAL (KIND = DBL) :: by1, by2, by3 !product yi*(yi+a.yi)
851 ! -----
852 ! Initialize
853 ! -----
854 !
855 TExx = 0.0D0
856 TExy = 0.0D0
857 TEKz = 0.0D0
858 TEyx = 0.0D0
859 TEyy = 0.0D0
860 TEyz = 0.0D0
861 TEzx = 0.0D0
862 TEzy = 0.0D0
863 TEzz = 0.0D0
864 ! -----

```

```

865 QExx = 0.000
866 QExy = 0.000
867 QExz = 0.000
868 QEyx = 0.000
869 QEyy = 0.000
870 QEyz = 0.000
871 QEzx = 0.000
872 QEzy = 0.000
873 QEzz = 0.000
874
875 PEx = 0.000
876 PEy = 0.000
877 PEz = 0.000
878 PE = 0.000
879
880 a1 = x0e-x00
881 a2 = y0e-y00
882 a3 = z0e-z00
883 modx0= DSQRT(a1**2+a2**2+a3**2)
884
885 IF (modx0 >= 1.000 + eps) THEN
886   cpl = 0.000
887 ELSE IF (modx0 <= 1.000 - eps) THEN
888   cpl = -8.000*pi
889 ELSE
890   cpl = -4.000*pi
891 END IF
892 a1 = -x0e
893 a2 = -y0e
894 a3 = -z0e
895 modx0= DSQRT(a1**2+a2**2+a3**2)
896 a1 = a1/modx0
897 a2 = a2/modx0
898 a3 = a3/modx0
899
900 !OPEN (9,'file="TE.out"')
901
902 ! vertices of the kth triangle
903
904 i1 = n(k,1)
905 i2 = n(k,2)
906 i3 = n(k,3)
907 i4 = n(k,4)
908 i5 = n(k,5)
909 i6 = n(k,6)
910
911 cf = 0.000
912
913 ! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
914 ! rule formula
915
916 bvx1 = 0.000
917 bvy1 = 0.000
918 bvz1 = 0.000
919 bvx2 = 0.000
920 bvy2 = 0.000
921 bvz2 = 0.000
922 bvx3 = 0.000
923 bvy3 = 0.000
924 bvz3 = 0.000
925 yvx1 = 0.000
926 yvy1 = 0.000
927 yvz1 = 0.000
928 yvx2 = 0.000
929 yvy2 = 0.000
930 yvz2 = 0.000
931 yvx3 = 0.000
932 yvy3 = 0.000
933 yvz3 = 0.000
934
935 ! computation of curvature line integral along segment 1-4
936

```

```

937 Dx0x = p(14,1) - p(11,1)
938 Dy0x = p(14,2) - p(11,2)
939 Dz0x = p(14,3) - p(11,3)
940
941 hss = DSQRT(Dx0x**2 + Dy0x**2 + Dz0x**2)
942 Dx0x = Dx0x/hss
943 Dy0x = Dy0x/hss
944 Dz0x = Dz0x/hss
945
946 yvx1 = p(11,1) - x00
947 yvy1 = p(11,2) - y00
948 yvz1 = p(11,3) - z00
949 fill = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
950
951 bvx1 = (Dy0x*yvz1 - Dz0x*yvy1)
952 bvy1 = (Dz0x*yvx1 - Dx0x*yvz1)
953 bvz1 = (Dx0x*yvy1 - Dy0x*yvx1)
954
955 yvx3 = p(14,1) - x00
956 yvy3 = p(14,2) - y00
957 yvz3 = p(14,3) - z00
958 fill3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
959
960 bvx3 = (Dy0x*yvz3 - Dz0x*yvy3)
961 bvy3 = (Dz0x*yvx3 - Dx0x*yvz3)
962 bvz3 = (Dx0x*yvy3 - Dy0x*yvx3)
963
964 QExx = QExx + hss*( - (bvx1*(yvx1/fill**3)) - (bvz3*(yvx3/fill3**3)) )/2.000
965 QEyy = QEyy + hss*( - (bvy1*(yvy1/fill**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
966 QExz = QExz + hss*( - (bvz1*(yvx1/fill**3)) - (bvz3*(yvx3/fill3**3)) )/2.000
967 QEyx = QEyx + hss*( - (bvx1*(yvy1/fill**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
968 QEyy = QEyy + hss*( - (bvy1*(yvy1/fill**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
969 QEyz = QEyz + hss*( - (bvz1*(yvy1/fill**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
970 QEzx = QEzx + hss*( - (bvx1*(yvz1/fill**3)) - (bvz3*(yvz3/fill3**3)) )/2.000
971 QEzy = QEzy + hss*( - (bvy1*(yvz1/fill**3)) - (bvz3*(yvz3/fill3**3)) )/2.000
972 QEzz = QEzz + hss*( - (bvz1*(yvz1/fill**3)) - (bvz3*(yvz3/fill3**3)) )/2.000
973
974 ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
975 ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
976
977 by1 = fill*(fill+ay1)
978 by3 = fill3*(fill3+ay3)
979
980 PEx = ( (a1*bvx1/by1) + (a3*bvx3/by3) ) * a1
981 PEy = ( (a1*bvy1/by1) + (a3*bvy3/by3) ) * a2
982 PEz = ( (a1*bvz1/by1) + (a3*bvz3/by3) ) * a3
983
984 PE = PE + hss*( PEx + PEy + PEz )/2.000
985
986 ! computation of curvature line integral along segment 4-2
987
988 bvx1 = 0.000
989 bvy1 = 0.000
990 bvz1 = 0.000
991 bvx2 = 0.000
992 bvy2 = 0.000
993 bvz2 = 0.000
994 bvx3 = 0.000
995 bvy3 = 0.000
996 bvz3 = 0.000
997 yvx1 = 0.000
998 yvy1 = 0.000
999 yvz1 = 0.000
1000 yvx2 = 0.000
1001 yvy2 = 0.000
1002 yvz2 = 0.000
1003 yvx3 = 0.000
1004 yvy3 = 0.000
1005 yvz3 = 0.000
1006
1007 Dx0x = p(12,1) - p(14,1)
1008 Dy0x = p(12,2) - p(14,2)

```

```

1009 DzDx = p(12,3) - p(14,3)
1010 -----
1011 hss = DSQRT(DxDx**2 + DyDx**2 + DzDx**2)
1012 DxDx = Dx/Dx/hss
1013 DyDx = Dy/Dx/hss
1014 DzDx = Dz/Dx/hss
1015 -----
1016 yvx1 = p(14,1) - x00
1017 yvy1 = p(14,2) - y00
1018 yvz1 = p(14,3) - z00
1019 fill1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
1020 -----
1021 bvx1 = (DyDx*yvz1 - DzDx*yvy1)
1022 bvy1 = (DzDx*yvx1 - DxDx*yvz1)
1023 bvz1 = (DxDx*yvy1 - DyDx*yvx1)
1024 -----
1025 yvx3 = p(12,1) - x00
1026 yvy3 = p(12,2) - y00
1027 yvz3 = p(12,3) - z00
1028 fill3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
1029 -----
1030 bvx3 = (DyDx*yvz3 - DzDx*yvy3)
1031 bvy3 = (DzDx*yvx3 - DxDx*yvz3)
1032 bvz3 = (DxDx*yvy3 - DyDx*yvx3)
1033 -----
1034 QExx = QExx + hss*( - (bv1*(yvx1/fill1**3)) - (bv3*(yvx3/fill3**3)) )/2.000
1035 QEyx = QEyx + hss*( - (bvy1*(yvx1/fill1**3)) - (bv3*(yvx3/fill3**3)) )/2.000
1036 QExz = QExz + hss*( - (bv21*(yvx1/fill1**3)) - (bv23*(yvx3/fill3**3)) )/2.000
1037 QEyx = QEyx + hss*( - (bv1*(yvy1/fill1**3)) - (bv3*(yvy3/fill3**3)) )/2.000
1038 QEyy = QEyy + hss*( - (bvy1*(yvy1/fill1**3)) - (bv3*(yvy3/fill3**3)) )/2.000
1039 QEyz = QEyz + hss*( - (bv21*(yvy1/fill1**3)) - (bv23*(yvy3/fill3**3)) )/2.000
1040 QExx = QExx + hss*( - (bv1*(yvx1/fill1**3)) - (bv3*(yvx3/fill3**3)) )/2.000
1041 QEyx = QEyx + hss*( - (bvy1*(yvx1/fill1**3)) - (bv3*(yvx3/fill3**3)) )/2.000
1042 QExz = QExz + hss*( - (bv21*(yvx1/fill1**3)) - (bv23*(yvx3/fill3**3)) )/2.000
1043 -----
1044 ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
1045 ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
1046 -----
1047 by1 = fill1*(fill1+ay1)
1048 by3 = fill3*(fill3+ay3)
1049 -----
1050 PEx = ( (a1*bvx1/by1) + (a2*bvx3/by3) )*a1
1051 PEy = ( (a1*bvy1/by1) + (a2*bvy3/by3) )*a2
1052 PEz = ( (a1*bvz1/by1) + (a2*bvz3/by3) )*a3
1053 -----
1054 PE = PE + hss*( PEx + PEy + PEz )/2.000
1055 -----
1056 ! computation of curvature line integral along segment 2-5
1057 -----
1058 bvx1 = 0.000
1059 bvy1 = 0.000
1060 bvz1 = 0.000
1061 bvx2 = 0.000
1062 bvy2 = 0.000
1063 bvz2 = 0.000
1064 bvx3 = 0.000
1065 bvy3 = 0.000
1066 bvz3 = 0.000
1067 yvx1 = 0.000
1068 yvy1 = 0.000
1069 yvx2 = 0.000
1070 yvy2 = 0.000
1071 yvx3 = 0.000
1072 yvy3 = 0.000
1073 yvx3 = 0.000
1074 yvy3 = 0.000
1075 yvz3 = 0.000
1076 -----
1077 DxDx = p(15,1) - p(12,1)
1078 DyDx = p(15,2) - p(12,2)
1079 DzDx = p(15,3) - p(12,3)
1080 -----

```

```

1081 hss = DSQRT(DxDx**2 + DyDx**2 + DzDx**2)
1082 DxDx = Dx/Dx/hss
1083 DyDx = Dy/Dx/hss
1084 DzDx = Dz/Dx/hss
1085 -----
1086 yvx1 = p(12,1) - x00
1087 yvy1 = p(12,2) - y00
1088 yvz1 = p(12,3) - z00
1089 fill1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
1090 -----
1091 bvx1 = (DyDx*yvz1 - DzDx*yvy1)
1092 bvy1 = (DzDx*yvx1 - DxDx*yvz1)
1093 bvz1 = (DxDx*yvy1 - DyDx*yvx1)
1094 -----
1095 yvx3 = p(15,1) - x00
1096 yvy3 = p(15,2) - y00
1097 yvz3 = p(15,3) - z00
1098 fill3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
1099 -----
1100 bvx3 = (DyDx*yvz3 - DzDx*yvy3)
1101 bvy3 = (DzDx*yvx3 - DxDx*yvz3)
1102 bvz3 = (DxDx*yvy3 - DyDx*yvx3)
1103 -----
1104 QExx = QExx + hss*( - (bv1*(yvx1/fill1**3)) - (bv3*(yvx3/fill3**3)) )/2.000
1105 QEyx = QEyx + hss*( - (bvy1*(yvx1/fill1**3)) - (bv3*(yvx3/fill3**3)) )/2.000
1106 QExz = QExz + hss*( - (bv21*(yvx1/fill1**3)) - (bv23*(yvx3/fill3**3)) )/2.000
1107 QEyx = QEyx + hss*( - (bv1*(yvy1/fill1**3)) - (bv3*(yvy3/fill3**3)) )/2.000
1108 QEyy = QEyy + hss*( - (bvy1*(yvy1/fill1**3)) - (bv3*(yvy3/fill3**3)) )/2.000
1109 QEyz = QEyz + hss*( - (bv21*(yvy1/fill1**3)) - (bv23*(yvy3/fill3**3)) )/2.000
1110 QExx = QExx + hss*( - (bv1*(yvx1/fill1**3)) - (bv3*(yvx3/fill3**3)) )/2.000
1111 QEyx = QEyx + hss*( - (bvy1*(yvx1/fill1**3)) - (bv3*(yvx3/fill3**3)) )/2.000
1112 QExz = QExz + hss*( - (bv21*(yvx1/fill1**3)) - (bv23*(yvx3/fill3**3)) )/2.000
1113 -----
1114 ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
1115 ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
1116 -----
1117 by1 = fill1*(fill1+ay1)
1118 by3 = fill3*(fill3+ay3)
1119 -----
1120 PEx = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
1121 PEy = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
1122 PEz = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
1123 -----
1124 PE = PE + hss*( PEx + PEy + PEz )/2.000
1125 -----
1126 ! computation of curvature line integral along segment 5-3
1127 -----
1128 bvx1 = 0.000
1129 bvy1 = 0.000
1130 bvz1 = 0.000
1131 bvx2 = 0.000
1132 bvy2 = 0.000
1133 bvz2 = 0.000
1134 bvx3 = 0.000
1135 bvy3 = 0.000
1136 bvz3 = 0.000
1137 yvx1 = 0.000
1138 yvy1 = 0.000
1139 yvz1 = 0.000
1140 yvx2 = 0.000
1141 yvy2 = 0.000
1142 yvz2 = 0.000
1143 yvx3 = 0.000
1144 yvy3 = 0.000
1145 yvz3 = 0.000
1146 -----
1147 DxDx = p(13,1) - p(15,1)
1148 DyDx = p(13,2) - p(15,2)
1149 DzDx = p(13,3) - p(15,3)
1150 -----
1151 hss = DSQRT(DxDx**2 + DyDx**2 + DzDx**2)
1152 DxDx = Dx/Dx/hss

```

```

1153 DyDx = DyDx/hss
1154 DzDx = DzDx/hss
1155 -----
1156 yvx1 = p(15,1) - x00
1157 yvy1 = p(15,2) - y00
1158 yvz1 = p(15,3) - z00
1159 fill1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
1160 -----
1161 bvx1 = (DyDx*yvz1 - DzDx*yvy1)
1162 bvy1 = (DzDx*yvx1 - DxDx*yvz1)
1163 bvz1 = (DxDx*yvy1 - DyDx*yvx1)
1164 -----
1165 yvx3 = p(13,1) - x00
1166 yvy3 = p(13,2) - y00
1167 yvz3 = p(13,3) - z00
1168 fill3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
1169 -----
1170 bvx3 = (DyDx*yvz3 - DzDx*yvy3)
1171 bvy3 = (DzDx*yvx3 - DxDx*yvz3)
1172 bvz3 = (DxDx*yvy3 - DyDx*yvx3)
1173 -----
1174 QExx = QExx + hss*( - (bvx1*(yvx1/fill1**3)) - (bvz3*(yvx3/fill3**3)) )/2.000
1175 QEyy = QEyy + hss*( - (bvy1*(yvy1/fill1**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
1176 QExz = QExz + hss*( - (bvz1*(yvx1/fill1**3)) - (bvz3*(yvx3/fill3**3)) )/2.000
1177 QEyx = QEyx + hss*( - (bvx1*(yvy1/fill1**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
1178 QEyy = QEyy + hss*( - (bvy1*(yvy1/fill1**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
1179 QEyz = QEyz + hss*( - (bvz1*(yvy1/fill1**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
1180 QExx = QExx + hss*( - (bvx1*(yvx1/fill1**3)) - (bvz3*(yvx3/fill3**3)) )/2.000
1181 QEzy = QEzy + hss*( - (bvy1*(yvy1/fill1**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
1182 QExz = QExz + hss*( - (bvz1*(yvx1/fill1**3)) - (bvz3*(yvx3/fill3**3)) )/2.000
1183 -----
1184 ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
1185 ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
1186 -----
1187 by1 = fill1*(fill1+ay1)
1188 by3 = fill3*(fill3+ay3)
1189 -----
1190 PEx = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
1191 PEy = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
1192 PEz = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
1193 -----
1194 PE = PE + hss*( PEx + PEy + PEz )/2.000
1195 -----
1196 ! computation of curvature line integral along segment 3-6
1197 -----
1198 bvx1 = 0.000
1199 bvy1 = 0.000
1200 bvz1 = 0.000
1201 bvx2 = 0.000
1202 bvy2 = 0.000
1203 bvz2 = 0.000
1204 bvx3 = 0.000
1205 bvy3 = 0.000
1206 bvz3 = 0.000
1207 yvx1 = 0.000
1208 yvy1 = 0.000
1209 yvz1 = 0.000
1210 yvx2 = 0.000
1211 yvy2 = 0.000
1212 yvz2 = 0.000
1213 yvx3 = 0.000
1214 yvy3 = 0.000
1215 yvz3 = 0.000
1216 -----
1217 DxDx = p(16,1) - p(13,1)
1218 DyDx = p(16,2) - p(13,2)
1219 DzDx = p(16,3) - p(13,3)
1220 -----
1221 hss = DSQRT(DxDx**2 + DyDx**2 + DzDx**2)
1222 DxDx = DxDx/hss
1223 DyDx = DyDx/hss
1224 DzDx = DzDx/hss

```

```

1225 -----
1226 yvx1 = p(13,1) - x00
1227 yvy1 = p(13,2) - y00
1228 yvz1 = p(13,3) - z00
1229 fill1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
1230 -----
1231 bvx1 = (DyDx*yvz1 - DzDx*yvy1)
1232 bvy1 = (DzDx*yvx1 - DxDx*yvz1)
1233 bvz1 = (DxDx*yvy1 - DyDx*yvx1)
1234 -----
1235 yvx3 = p(16,1) - x00
1236 yvy3 = p(16,2) - y00
1237 yvz3 = p(16,3) - z00
1238 fill3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
1239 -----
1240 bvx3 = (DyDx*yvz3 - DzDx*yvy3)
1241 bvy3 = (DzDx*yvx3 - DxDx*yvz3)
1242 bvz3 = (DxDx*yvy3 - DyDx*yvx3)
1243 -----
1244 QExx = QExx + hss*( - (bvx1*(yvx1/fill1**3)) - (bvz3*(yvx3/fill3**3)) )/2.000
1245 QEyy = QEyy + hss*( - (bvy1*(yvy1/fill1**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
1246 QExz = QExz + hss*( - (bvz1*(yvx1/fill1**3)) - (bvz3*(yvx3/fill3**3)) )/2.000
1247 QEyx = QEyx + hss*( - (bvx1*(yvy1/fill1**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
1248 QEyy = QEyy + hss*( - (bvy1*(yvy1/fill1**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
1249 QEyz = QEyz + hss*( - (bvz1*(yvy1/fill1**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
1250 QExx = QExx + hss*( - (bvx1*(yvx1/fill1**3)) - (bvz3*(yvx3/fill3**3)) )/2.000
1251 QEzy = QEzy + hss*( - (bvy1*(yvy1/fill1**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
1252 QExz = QExz + hss*( - (bvz1*(yvx1/fill1**3)) - (bvz3*(yvx3/fill3**3)) )/2.000
1253 -----
1254 ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
1255 ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
1256 -----
1257 by1 = fill1*(fill1+ay1)
1258 by3 = fill3*(fill3+ay3)
1259 -----
1260 PEx = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
1261 PEy = ( (a1*bvy1/by1) + (a3*bvy3/by3) )*a2
1262 PEz = ( (a1*bvz1/by1) + (a3*bvz3/by3) )*a3
1263 -----
1264 PE = PE + hss*( PEx + PEy + PEz )/2.000
1265 -----
1266 ! computation of curvature line integral along segment 6-1
1267 -----
1268 bvx1 = 0.000
1269 bvy1 = 0.000
1270 bvz1 = 0.000
1271 bvx2 = 0.000
1272 bvy2 = 0.000
1273 bvz2 = 0.000
1274 bvx3 = 0.000
1275 bvy3 = 0.000
1276 bvz3 = 0.000
1277 yvx1 = 0.000
1278 yvy1 = 0.000
1279 yvz1 = 0.000
1280 yvx2 = 0.000
1281 yvy2 = 0.000
1282 yvz2 = 0.000
1283 yvx3 = 0.000
1284 yvy3 = 0.000
1285 yvz3 = 0.000
1286 -----
1287 DxDx = p(11,1) - p(16,1)
1288 DyDx = p(11,2) - p(16,2)
1289 DzDx = p(11,3) - p(16,3)
1290 -----
1291 hss = DSQRT(DxDx**2 + DyDx**2 + DzDx**2)
1292 DxDx = DxDx/hss
1293 DyDx = DyDx/hss
1294 DzDx = DzDx/hss
1295 -----
1296 yvx1 = p(16,1) - x00

```

```

1297 yvy1 = p(16,2) - y00
1298 yvz1 = p(16,3) - z00
1299 fil1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
-----
1301 bvx1 = (DyDx*yvz1 - DzDx*yvy1)
1302 bvy1 = (DzDx*yvx1 - DxDx*yvz1)
1303 bvz1 = (DxDx*yvy1 - DyDx*yvx1)
-----
1304 yvx3 = p(11,1) - x00
1306 yvy3 = p(11,2) - y00
1307 yvz3 = p(11,3) - z00
1308 fil3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
-----
1310 bvx3 = (DyDx*yvz3 - DzDx*yvy3)
1311 bvy3 = (DzDx*yvx3 - DxDx*yvz3)
1312 bvz3 = (DxDx*yvy3 - DyDx*yvx3)
-----
1313 QExx = QExx + hss*( - (bvz1*(yvx1/fil1**3)) - (bvz3*(yvx3/fil3**3)) )/2.000
1314 QExy = QExy + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.000
1315 QExz = QExz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.000
1316 QExx = QExx + hss*( - (bvz1*(yvx1/fil1**3)) - (bvz3*(yvx3/fil3**3)) )/2.000
1317 QExy = QExy + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.000
1318 QExz = QExz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.000
1319 QExy = QExy + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.000
1320 QExz = QExz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.000
1321 QExy = QExy + hss*( - (bvz1*(yvy1/fil1**3)) - (bvz3*(yvy3/fil3**3)) )/2.000
1322 QExz = QExz + hss*( - (bvz1*(yvz1/fil1**3)) - (bvz3*(yvz3/fil3**3)) )/2.000
-----
1323 ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
1324 ay2 = (a1*yvx2 + a2*yvy2 + a3*yvz2)
1325 ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
-----
1327 by1 = fil1*(fil1+ay1)
1328 by3 = fil3*(fil3+ay3)
-----
1330 PEx = ( a1*bvx1/by1 ) + ( a3*bvx3/by3 ) *a1
1331 PEy = ( a1*bvy1/by1 ) + ( a3*bvy3/by3 ) *a2
1332 PEz = ( a1*bvz1/by1 ) + ( a3*bvz3/by3 ) *a3
-----
1334 PE = PE + hss*( PEx + PEy + PEz )/2.000
1335 ! Close(9)
-----
1336 TExx = 2.000*( QExx + cpi -PE )
1337 TExy = 2.000*( QExy )
1338 TExz = 2.000*( QExz )
1339 TEyx = 2.000*( QExy )
1340 TEyy = 2.000*( QExy + cpi -PE )
1341 TEyz = 2.000*( QExz )
1342 TEzx = 2.000*( QExx )
1343 TEzy = 2.000*( QExy )
1344 TEzz = 2.000*( QExz + cpi -PE )
-----
1346 !
1347 END SUBROUTINE intr_lin_sing_s4
1348 !=====
1349 SUBROUTINE intr_lin_sing_s4s(x0e, y0e, z0e, &
1350 & x0, y00, z00, &
1351 & k,
1352 & TExx, TExy, TExz, &
1353 & TEyx, TEyy, TEyz, &
1354 & TEzx, TEzy, TEzz )
1355 !=====
1356 ! This subroutine is a new version stokeslet Subroutine.
1357 !Compute:
1358 ! *The value of the Stokeslet over each singular element
1359 ! Now, (March/ 09 / 2015) this subroutine was made.
1360 !=====
1361 USE Mod_Nodal_Interp
1362 USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, Ulog, pi, eps, vnx0, vny0, vnz0, x0, y0, z0
1363 !=====
1364 IMPLICIT NONE
1365 !=====
1366 ! Variables

```

```

1367 !-----
1368 REAL (KIND = DBL), INTENT(IN) :: x00, y00, z00 !singulartiy coordinates
1369 REAL (KIND = DBL), INTENT(IN) :: x0e, y0e, z0e !singulartiy coordinates element
1370 INTEGER, INTENT(IN) :: k !number of element
1371 REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TExz !value of stresslet in the singular element
1372 REAL (KIND = DBL), INTENT(OUT) :: TEyx, TEyy, TEyz !value of stresslet in the singular element
1373 REAL (KIND = DBL), INTENT(OUT) :: TEzx, TEzy, TEzz !value of stresslet in the singular element
1374 !-----
1375 ! Variables inside the subroutine
1376 !-----
1377 INTEGER :: i, j !Counters
1378 INTEGER :: i1, i2, i3, i4, i5, i6 !Indices to obtain node numbers from each element
1379 !-----
1380 REAL (KIND = DBL) :: cf, fil1, fil2, fil3 !integration weigh coefficients
1381 REAL (KIND = DBL) :: Dx0x, Dy0x, Dz0x !tangential vector around the triangle (xi,eta)
1382 REAL (KIND = DBL) :: hss !weight
1383 REAL (KIND = DBL) :: cpi !Selection to add the solid angle
1384 REAL (KIND = DBL) :: modx0 !weight
1385 REAL (KIND = DBL) :: bvx1, bvy1, bvz1, & !vectorreal product
1386 & bvx2, bvy2, bvz2, &
1387 & bvx3, bvy3, bvz3
1388 REAL (KIND = DBL) :: QExx, QExy, QExz, & !Iq tensor
1389 & QExy, QExy, QExz, &
1390 & QExz, QExy, QExz
1391 REAL (KIND = DBL) :: PEx, PEy, PEz, PE !Ij tensor
1392 REAL (KIND = DBL) :: yvx1, yvy1, yvz1 !vector (y1-x0)
1393 REAL (KIND = DBL) :: yvx2, yvy2, yvz2 !vector (y2-x0)
1394 REAL (KIND = DBL) :: yvx3, yvy3, yvz3 !vector (y2-x0)
1395 REAL (KIND = DBL) :: px2, py2, pz2 !vector a
1396 REAL (KIND = DBL) :: a1, a2, a3 !vector a
1397 REAL (KIND = DBL) :: ay1, ay2, ay3 !vector dot product a*y1, a*y2
1398 REAL (KIND = DBL) :: by1, by2, by3 !product y1*(y1+a.y1)
1399 !-----
1400 ! Initialize
1401 !-----
1402 TExx = 0.000
1403 TExy = 0.000
1404 TExz = 0.000
1405 TEyx = 0.000
1406 TEyy = 0.000
1407 TEyz = 0.000
1408 TEzx = 0.000
1409 TEzy = 0.000
1410 TEzz = 0.000
1411 QExx = 0.000
1412 QExy = 0.000
1413 QExz = 0.000
1414 QExx = 0.000
1415 QExy = 0.000
1416 QExy = 0.000
1417 QExz = 0.000
1418 QExz = 0.000
1419 QExy = 0.000
1420 QExz = 0.000
1421 PEx = 0.000
1422 PEy = 0.000
1423 PEz = 0.000
1424 PEz = 0.000
1425 PE = 0.000
1426
1427 a1 = -x0(k)
1428 a2 = -y0(k)
1429 a3 = -z0(k)
1430 modx0= DSQRT(a1**2+a2**2+a3**2)
1431
1432 cpi = 4.000*pi
1433
1434 !IF ((ABS(a1) <= eps) .AND. &
1435 & (ABS(a2) <= eps) .AND. &
1436 & (ABS(a3) <= eps)) THEN
1437 ! a1 = x0(k)
1438 ! a2 = y0(k)

```

```

1439 ! a3 = z0(k)
1440 !ELSE
1441 !a1 = x0e-x00
1442 !a2 = y0e-y00
1443 !a3 = z0e-z00
1444 !END IF
1445 !
1446 a1 = a1/modx0
1447 a2 = a2/mody0
1448 a3 = a3/modz0
1449
1450 !OPEN (9,file="TE.out")
1451
1452 ! vertices of the kth triangle
1453 !-----
1454 i1 = n(k,1)
1455 i2 = n(k,2)
1456 i3 = n(k,3)
1457 i4 = n(k,4)
1458 i5 = n(k,5)
1459 i6 = n(k,6)
1460 !-----
1461 cf = 0.000
1462
1463 ! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
1464 ! rule formula
1465 !-----
1466 bvx1 = 0.000
1467 bvy1 = 0.000
1468 bvz1 = 0.000
1469 bvx2 = 0.000
1470 bvy2 = 0.000
1471 bvz2 = 0.000
1472 bvx3 = 0.000
1473 bvy3 = 0.000
1474 bvz3 = 0.000
1475 yvx1 = 0.000
1476 yvy1 = 0.000
1477 yvz1 = 0.000
1478 yvx2 = 0.000
1479 yvy2 = 0.000
1480 yvz2 = 0.000
1481 yvx3 = 0.000
1482 yvy3 = 0.000
1483 yvz3 = 0.000
1484
1485 ! computation of curvature line integral along segment 1-4
1486 !-----
1487 Dx0x = p(i4,1) - p(i1,1)
1488 Dy0x = p(i4,2) - p(i1,2)
1489 Dz0x = p(i4,3) - p(i1,3)
1490
1491 hss = DSQRT(Dx0x**2 +Dy0x**2 +Dz0x**2)
1492 Dx0x = Dx0x/hss
1493 Dy0x = Dy0x/hss
1494 Dz0x = Dz0x/hss
1495 !-----
1496 yvx1 = p(i1,1) - x00
1497 yvy1 = p(i1,2) - y00
1498 yvz1 = p(i1,3) - z00
1499 fill1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
1500 !-----
1501 bvx1 = (Dy0x*yvz1 - Dz0x*yvy1)
1502 bvy1 = (Dz0x*yvx1 - Dx0x*yvz1)
1503 bvz1 = (Dx0x*yvy1 - Dy0x*yvx1)
1504 !-----
1505 yvx3 = p(i4,1) - x00
1506 yvy3 = p(i4,2) - y00
1507 yvz3 = p(i4,3) - z00
1508 fill3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
1509 !-----
1510 bvx3 = (Dy0x*yvz3 - Dz0x*yvy3)

```

```

1511 bvy3 = (Dz0x*yvx3 - Dx0x*yvz3)
1512 bvz3 = (Dx0x*yvy3 - Dy0x*yvx3)
1513 !-----
1514 QExx = QExx + hss*( - (bvz1*(yvx1/fill1**3)) - (bvx3*(yvx3/fill3**3)) )/2.000
1515 QExy = QExy + hss*( - (bvy1*(yvx1/fill1**3)) - (bvy3*(yvx3/fill3**3)) )/2.000
1516 QExz = QExz + hss*( - (bvz1*(yvy1/fill1**3)) - (bvz3*(yvx3/fill3**3)) )/2.000
1517 QEyx = QEyx + hss*( - (bvz1*(yvy1/fill1**3)) - (bvx3*(yvy3/fill3**3)) )/2.000
1518 QEyy = QEyy + hss*( - (bvy1*(yvy1/fill1**3)) - (bvy3*(yvy3/fill3**3)) )/2.000
1519 QEyz = QEyz + hss*( - (bvz1*(yvy1/fill1**3)) - (bvz3*(yvy3/fill3**3)) )/2.000
1520 QExz = QExz + hss*( - (bvz1*(yvz1/fill1**3)) - (bvx3*(yvz3/fill3**3)) )/2.000
1521 QEzy = QEzy + hss*( - (bvy1*(yvz1/fill1**3)) - (bvy3*(yvz3/fill3**3)) )/2.000
1522 QEzz = QEzz + hss*( - (bvz1*(yvz1/fill1**3)) - (bvz3*(yvz3/fill3**3)) )/2.000
1523 !-----
1524 ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
1525 ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
1526 !-----
1527 by1 = fill1*(fill1+ay1)
1528 by3 = fill3*(fill3+ay3)
1529 !-----
1530 PEx = ( (a1*bvx1/by1) + (a3*bvx3/by3) ) *a1
1531 PEy = ( (a1*bvy1/by1) + (a3*bvy3/by3) ) *a2
1532 PEz = ( (a1*bvz1/by1) + (a3*bvz3/by3) ) *a3
1533 !-----
1534 PE = PE + hss*( PEx + PEy + PEz )/2.000
1535
1536 ! computation of curvature line integral along segment 4-2
1537 !-----
1538 bvx1 = 0.000
1539 bvy1 = 0.000
1540 bvz1 = 0.000
1541 bvx2 = 0.000
1542 bvy2 = 0.000
1543 bvz2 = 0.000
1544 bvx3 = 0.000
1545 bvy3 = 0.000
1546 bvz3 = 0.000
1547 yvx1 = 0.000
1548 yvy1 = 0.000
1549 yvz1 = 0.000
1550 yvx2 = 0.000
1551 yvy2 = 0.000
1552 yvz2 = 0.000
1553 yvx3 = 0.000
1554 yvy3 = 0.000
1555 yvz3 = 0.000
1556 !-----
1557 Dx0x = p(i2,1) - p(i4,1)
1558 Dy0x = p(i2,2) - p(i4,2)
1559 Dz0x = p(i2,3) - p(i4,3)
1560 !-----
1561 hss = DSQRT(Dx0x**2 +Dy0x**2 +Dz0x**2)
1562 Dx0x = Dx0x/hss
1563 Dy0x = Dy0x/hss
1564 Dz0x = Dz0x/hss
1565 !-----
1566 yvx1 = p(i4,1) - x00
1567 yvy1 = p(i4,2) - y00
1568 yvz1 = p(i4,3) - z00
1569 fill1 = DSQRT(yvx1**2 + yvy1**2 +yvz1**2)
1570 !-----
1571 bvx1 = (Dy0x*yvz1 - Dz0x*yvy1)
1572 bvy1 = (Dz0x*yvx1 - Dx0x*yvz1)
1573 bvz1 = (Dx0x*yvy1 - Dy0x*yvx1)
1574 !-----
1575 yvx3 = p(i2,1) - x00
1576 yvy3 = p(i2,2) - y00
1577 yvz3 = p(i2,3) - z00
1578 fill3 = DSQRT(yvx3**2 + yvy3**2 +yvz3**2)
1579 !-----
1580 bvx3 = (Dy0x*yvz3 - Dz0x*yvy3)
1581 bvy3 = (Dz0x*yvx3 - Dx0x*yvz3)
1582 bvz3 = (Dx0x*yvy3 - Dy0x*yvx3)

```

```

1583 -----
1584 QExx = QExx + hss*( - (bvxl*(yvx1/fil1**3)) - (bvxl*(yvx3/fil3**3)) )/2.000
1585 QExy = QExy + hss*( - (bvyl*(yvx1/fil1**3)) - (bvyl*(yvx3/fil3**3)) )/2.000
1586 QExz = QExz + hss*( - (bvzl*(yvx1/fil1**3)) - (bvzl*(yvx3/fil3**3)) )/2.000
1587 QEvx = QEvx + hss*( - (bvxl*(yvy1/fil1**3)) - (bvxl*(yvy3/fil3**3)) )/2.000
1588 QEyy = QEyy + hss*( - (bvyl*(yvy1/fil1**3)) - (bvyl*(yvy3/fil3**3)) )/2.000
1589 QEyz = QEyz + hss*( - (bvzl*(yvy1/fil1**3)) - (bvzl*(yvy3/fil3**3)) )/2.000
1590 QEzx = QEzx + hss*( - (bvxl*(y vz1/fil1**3)) - (bvxl*(y vz3/fil3**3)) )/2.000
1591 QEzy = QEzy + hss*( - (bvyl*(y vz1/fil1**3)) - (bvyl*(y vz3/fil3**3)) )/2.000
1592 QEzz = QEzz + hss*( - (bvzl*(y vz1/fil1**3)) - (bvzl*(y vz3/fil3**3)) )/2.000
1593 -----
1594 ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
1595 ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
1596 -----
1597 by1 = fil1*(fil1+ay1)
1598 by3 = fil3*(fil3+ay3)
1599 -----
1600 PEx = ( (a1*bvxl/by1) + (a3*bvx3/by3) )*a1
1601 PEy = ( (a1*bvyl/by1) + (a3*bvy3/by3) )*a2
1602 PEz = ( (a1*bvzl/by1) + (a3*bvz3/by3) )*a3
1603 -----
1604 PE = PE + hss*( PEx + PEy + PEz )/2.000
1605 -----
1606 ! computation of curvature line integral along segment 2-5
1607 -----
1608 bvx1 = 0.000
1609 bvy1 = 0.000
1610 bvz1 = 0.000
1611 bvx2 = 0.000
1612 bvy2 = 0.000
1613 bvz2 = 0.000
1614 bvx3 = 0.000
1615 bvy3 = 0.000
1616 bvz3 = 0.000
1617 yvx1 = 0.000
1618 yvy1 = 0.000
1619 yvz1 = 0.000
1620 yvx2 = 0.000
1621 yvy2 = 0.000
1622 yvz2 = 0.000
1623 yvx3 = 0.000
1624 yvy3 = 0.000
1625 yvz3 = 0.000
1626 -----
1627 DxDx = p(15,1) - p(12,1)
1628 DyDy = p(15,2) - p(12,2)
1629 DzDz = p(15,3) - p(12,3)
1630 -----
1631 hss = DSQRT(DxDx**2 +DyDy**2 +DzDz**2)
1632 DxDx = DxDx/hss
1633 DyDy = DyDy/hss
1634 DzDz = DzDz/hss
1635 -----
1636 yvx1 = p(12,1) - x00
1637 yvy1 = p(12,2) - y00
1638 yvz1 = p(12,3) - z00
1639 fil1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
1640 -----
1641 bvx1 = (DyDy*yvz1 - DzDz*yvy1)
1642 bvy1 = (DzDz*yvx1 - DxDx*yvz1)
1643 bvz1 = (DxDx*yvy1 - DyDy*yvx1)
1644 -----
1645 yvx3 = p(15,1) - x00
1646 yvy3 = p(15,2) - y00
1647 yvz3 = p(15,3) - z00
1648 fil3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
1649 -----
1650 bvx3 = (DyDy*yvz3 - DzDz*yvy3)
1651 bvy3 = (DzDz*yvx3 - DxDx*yvz3)
1652 bvz3 = (DxDx*yvy3 - DyDy*yvx3)
1653 -----
1654 QExx = QExx + hss*( - (bvxl*(yvx1/fil1**3)) - (bvxl*(yvx3/fil3**3)) )/2.000

```

```

1655 QExy = QExy + hss*( - (bvyl*(yvx1/fil1**3)) - (bvyl*(yvx3/fil3**3)) )/2.000
1656 QExz = QExz + hss*( - (bvzl*(yvx1/fil1**3)) - (bvzl*(yvx3/fil3**3)) )/2.000
1657 QEyx = QEyx + hss*( - (bvxl*(yvy1/fil1**3)) - (bvxl*(yvy3/fil3**3)) )/2.000
1658 QEyy = QEyy + hss*( - (bvyl*(yvy1/fil1**3)) - (bvyl*(yvy3/fil3**3)) )/2.000
1659 QEyz = QEyz + hss*( - (bvzl*(yvy1/fil1**3)) - (bvzl*(yvy3/fil3**3)) )/2.000
1660 QEzx = QEzx + hss*( - (bvxl*(y vz1/fil1**3)) - (bvxl*(y vz3/fil3**3)) )/2.000
1661 QEzy = QEzy + hss*( - (bvyl*(y vz1/fil1**3)) - (bvyl*(y vz3/fil3**3)) )/2.000
1662 QEzz = QEzz + hss*( - (bvzl*(y vz1/fil1**3)) - (bvzl*(y vz3/fil3**3)) )/2.000
1663 -----
1664 ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
1665 ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
1666 -----
1667 by1 = fil1*(fil1+ay1)
1668 by3 = fil3*(fil3+ay3)
1669 -----
1670 PEx = ( (a1*bvxl/by1) + (a3*bvx3/by3) )*a1
1671 PEy = ( (a1*bvyl/by1) + (a3*bvy3/by3) )*a2
1672 PEz = ( (a1*bvzl/by1) + (a3*bvz3/by3) )*a3
1673 -----
1674 PE = PE + hss*( PEx + PEy + PEz )/2.000
1675 -----
1676 ! computation of curvature line integral along segment 5-3
1677 -----
1678 bvx1 = 0.000
1679 bvy1 = 0.000
1680 bvz1 = 0.000
1681 bvx2 = 0.000
1682 bvy2 = 0.000
1683 bvz2 = 0.000
1684 bvx3 = 0.000
1685 bvy3 = 0.000
1686 bvz3 = 0.000
1687 yvx1 = 0.000
1688 yvy1 = 0.000
1689 yvz1 = 0.000
1690 yvx2 = 0.000
1691 yvy2 = 0.000
1692 yvz2 = 0.000
1693 yvx3 = 0.000
1694 yvy3 = 0.000
1695 yvz3 = 0.000
1696 -----
1697 DxDx = p(13,1) - p(15,1)
1698 DyDy = p(13,2) - p(15,2)
1699 DzDz = p(13,3) - p(15,3)
1700 -----
1701 hss = DSQRT(DxDx**2 +DyDy**2 +DzDz**2)
1702 DxDx = DxDx/hss
1703 DyDy = DyDy/hss
1704 DzDz = DzDz/hss
1705 -----
1706 yvx1 = p(15,1) - x00
1707 yvy1 = p(15,2) - y00
1708 yvz1 = p(15,3) - z00
1709 fil1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
1710 -----
1711 bvx1 = (DyDy*yvz1 - DzDz*yvy1)
1712 bvy1 = (DzDz*yvx1 - DxDx*yvz1)
1713 bvz1 = (DxDx*yvy1 - DyDy*yvx1)
1714 -----
1715 yvx3 = p(13,1) - x00
1716 yvy3 = p(13,2) - y00
1717 yvz3 = p(13,3) - z00
1718 fil3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
1719 -----
1720 bvx3 = (DyDy*yvz3 - DzDz*yvy3)
1721 bvy3 = (DzDz*yvx3 - DxDx*yvz3)
1722 bvz3 = (DxDx*yvy3 - DyDy*yvx3)
1723 -----
1724 QExx = QExx + hss*( - (bvxl*(yvx1/fil1**3)) - (bvxl*(yvx3/fil3**3)) )/2.000
1725 QExy = QExy + hss*( - (bvyl*(yvx1/fil1**3)) - (bvyl*(yvx3/fil3**3)) )/2.000
1726 QExz = QExz + hss*( - (bvzl*(yvx1/fil1**3)) - (bvzl*(yvx3/fil3**3)) )/2.000

```



```

1727 QExx = QExx + hss*( - (bvxl*(yyvl/f111**3)) - (bvxl*(yyvl/f111**3)) )/2.000
1728 QEyy = QEyy + hss*( - (bvyl*(yyvl/f111**3)) - (bvyl*(yyvl/f111**3)) )/2.000
1729 QEyz = QEyz + hss*( - (bvzl*(yyvl/f111**3)) - (bvzl*(yyvl/f111**3)) )/2.000
1730 QExz = QExz + hss*( - (bvxl*(yvz1/f111**3)) - (bvxl*(yvz1/f111**3)) )/2.000
1731 QEzy = QEzy + hss*( - (bvyl*(yvz1/f111**3)) - (bvyl*(yvz1/f111**3)) )/2.000
1732 QEzz = QEzz + hss*( - (bvzl*(yvz1/f111**3)) - (bvzl*(yvz1/f111**3)) )/2.000
1733 -----
1734 ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
1735 ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
1736 -----
1737 by1 = f111*(f111+ay1)
1738 by3 = f113*(f113+ay3)
1739 -----
1740 PEx = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
1741 PEy = ( (a1*bvyl/by1) + (a3*bvyl/by3) )*a2
1742 PEz = ( (a1*bvzl/by1) + (a3*bvzl/by3) )*a3
1743 -----
1744 PE = PE + hss*( PEx + PEy + PEz )/2.000
1745 -----
1746 ! computation of curvature line integral along segment 3-6
1747 -----
1748 bvx1 = 0.000
1749 bvy1 = 0.000
1750 bvz1 = 0.000
1751 bvx2 = 0.000
1752 bvy2 = 0.000
1753 bvz2 = 0.000
1754 bvx3 = 0.000
1755 bvy3 = 0.000
1756 bvz3 = 0.000
1757 yvx1 = 0.000
1758 yvy1 = 0.000
1759 yvz1 = 0.000
1760 yvx2 = 0.000
1761 yvy2 = 0.000
1762 yvz2 = 0.000
1763 yvx3 = 0.000
1764 yvy3 = 0.000
1765 yvz3 = 0.000
1766 -----
1767 DxDx = p(16,1) - p(13,1)
1768 DyDx = p(16,2) - p(13,2)
1769 DzDx = p(16,3) - p(13,3)
1770 -----
1771 hss = DSQR(DxDx**2 + DyDx**2 + DzDx**2)
1772 DxDx = DxDx/hss
1773 DyDx = DyDx/hss
1774 DzDx = DzDx/hss
1775 -----
1776 yvx1 = p(13,1) - x00
1777 yvy1 = p(13,2) - y00
1778 yvz1 = p(13,3) - z00
1779 f111 = DSQR(yvx1**2 + yvy1**2 + yvz1**2)
1780 -----
1781 bvx1 = (DyDx*yvz1 - DzDx*yvy1)
1782 bvy1 = (DzDx*yvx1 - DxDx*yvz1)
1783 bvz1 = (DxDx*yvy1 - DyDx*yvx1)
1784 -----
1785 yvx3 = p(16,1) - x00
1786 yvy3 = p(16,2) - y00
1787 yvz3 = p(16,3) - z00
1788 f113 = DSQR(yvx3**2 + yvy3**2 + yvz3**2)
1789 -----
1790 bvx3 = (DyDx*yvz3 - DzDx*yvy3)
1791 bvy3 = (DzDx*yvx3 - DxDx*yvz3)
1792 bvz3 = (DxDx*yvy3 - DyDx*yvx3)
1793 -----
1794 QExx = QExx + hss*( - (bvxl*(yyvl/f111**3)) - (bvxl*(yyvl/f111**3)) )/2.000
1795 QEyy = QEyy + hss*( - (bvyl*(yyvl/f111**3)) - (bvyl*(yyvl/f111**3)) )/2.000
1796 QExz = QExz + hss*( - (bvxl*(yvz1/f111**3)) - (bvxl*(yvz1/f111**3)) )/2.000
1797 QEzy = QEzy + hss*( - (bvyl*(yvz1/f111**3)) - (bvyl*(yvz1/f111**3)) )/2.000
1798 QEzz = QEzz + hss*( - (bvzl*(yvz1/f111**3)) - (bvzl*(yvz1/f111**3)) )/2.000

```

```

1799 QEyz = QEyz + hss*( - (bvyl*(yyvl/f111**3)) - (bvyl*(yyvl/f111**3)) )/2.000
1800 QExz = QExz + hss*( - (bvxl*(yvz1/f111**3)) - (bvxl*(yvz1/f111**3)) )/2.000
1801 QEzy = QEzy + hss*( - (bvyl*(yvz1/f111**3)) - (bvyl*(yvz1/f111**3)) )/2.000
1802 QEzz = QEzz + hss*( - (bvzl*(yvz1/f111**3)) - (bvzl*(yvz1/f111**3)) )/2.000
1803 -----
1804 ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
1805 ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
1806 -----
1807 by1 = f111*(f111+ay1)
1808 by3 = f113*(f113+ay3)
1809 -----
1810 PEx = ( (a1*bvx1/by1) + (a3*bvx3/by3) )*a1
1811 PEy = ( (a1*bvyl/by1) + (a3*bvyl/by3) )*a2
1812 PEz = ( (a1*bvzl/by1) + (a3*bvzl/by3) )*a3
1813 -----
1814 PE = PE + hss*( PEx + PEy + PEz )/2.000
1815 -----
1816 ! computation of curvature line integral along segment 6-1
1817 -----
1818 bvx1 = 0.000
1819 bvy1 = 0.000
1820 bvz1 = 0.000
1821 bvx2 = 0.000
1822 bvy2 = 0.000
1823 bvz2 = 0.000
1824 bvx3 = 0.000
1825 bvy3 = 0.000
1826 bvz3 = 0.000
1827 yvx1 = 0.000
1828 yvy1 = 0.000
1829 yvz1 = 0.000
1830 yvx2 = 0.000
1831 yvy2 = 0.000
1832 yvz2 = 0.000
1833 yvx3 = 0.000
1834 yvy3 = 0.000
1835 yvz3 = 0.000
1836 -----
1837 DxDx = p(11,1) - p(16,1)
1838 DyDx = p(11,2) - p(16,2)
1839 DzDx = p(11,3) - p(16,3)
1840 -----
1841 hss = DSQR(DxDx**2 + DyDx**2 + DzDx**2)
1842 DxDx = DxDx/hss
1843 DyDx = DyDx/hss
1844 DzDx = DzDx/hss
1845 -----
1846 yvx1 = p(16,1) - x00
1847 yvy1 = p(16,2) - y00
1848 yvz1 = p(16,3) - z00
1849 f111 = DSQR(yvx1**2 + yvy1**2 + yvz1**2)
1850 -----
1851 bvx1 = (DyDx*yvz1 - DzDx*yvy1)
1852 bvy1 = (DzDx*yvx1 - DxDx*yvz1)
1853 bvz1 = (DxDx*yvy1 - DyDx*yvx1)
1854 -----
1855 yvx3 = p(11,1) - x00
1856 yvy3 = p(11,2) - y00
1857 yvz3 = p(11,3) - z00
1858 f113 = DSQR(yvx3**2 + yvy3**2 + yvz3**2)
1859 -----
1860 bvx3 = (DyDx*yvz3 - DzDx*yvy3)
1861 bvy3 = (DzDx*yvx3 - DxDx*yvz3)
1862 bvz3 = (DxDx*yvy3 - DyDx*yvx3)
1863 -----
1864 QExx = QExx + hss*( - (bvxl*(yyvl/f111**3)) - (bvxl*(yyvl/f111**3)) )/2.000
1865 QEyy = QEyy + hss*( - (bvyl*(yyvl/f111**3)) - (bvyl*(yyvl/f111**3)) )/2.000
1866 QExz = QExz + hss*( - (bvxl*(yvz1/f111**3)) - (bvxl*(yvz1/f111**3)) )/2.000
1867 QEzy = QEzy + hss*( - (bvyl*(yvz1/f111**3)) - (bvyl*(yvz1/f111**3)) )/2.000
1868 QEzz = QEzz + hss*( - (bvzl*(yvz1/f111**3)) - (bvzl*(yvz1/f111**3)) )/2.000
1869 QEyz = QEyz + hss*( - (bvyl*(yyvl/f111**3)) - (bvyl*(yyvl/f111**3)) )/2.000
1870 QExz = QExz + hss*( - (bvxl*(yvz1/f111**3)) - (bvxl*(yvz1/f111**3)) )/2.000

```

```

1871 QExy = QExy + hss*( - (bv1*(yz1/fill1**3)) - (bv3*(yz3/fil3**3)) )/2.000
1872 QExz = QExz + hss*( - (bv2*(yz1/fill1**3)) - (bv3*(yz3/fil3**3)) )/2.000
1873 -----
1874 ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
1875 ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
1876 -----
1877 by1 = fill1*(fill1+ay1)
1878 by3 = fill3*(fill3+ay3)
1879 -----
1880 PEx = ( a1*bvx1/by1 ) + ( a3*bvx3/by3 ) *a1
1881 PEy = ( a1*bvy1/by1 ) + ( a3*bvy3/by3 ) *a2
1882 PEz = ( a1*bvz1/by1 ) + ( a3*bvz3/by3 ) *a3
1883 -----
1884 PE = PE + hss*( PEx + PEy + PEz )/2.000
1885 !
1886 Close(9)
1887 -----
1888 TExx = 2.000*( QExx + cpi -PE )
1889 TExy = 2.000*( QExy )
1890 TExz = 2.000*( QExz )
1891 TExy = 2.000*( QExy )
1892 TExz = 2.000*( QExz )
1893 TExy = 2.000*( QExy + cpi -PE )
1894 TExz = 2.000*( QExz )
1895 TExy = 2.000*( QExy )
1896 TExz = 2.000*( QExz + cpi -PE )
1897 -----
1898 END SUBROUTINE intr_lin_sing_s4s
1899 -----
1900 SUBROUTINE intr_lin_sing_s5(x10, y10, z10, &
1901 & k, &
1902 & TExx, TExy, TExz, &
1903 & TEyx, TEyy, TEyz, &
1904 & TEzx, TEzy, TEzz )
1905 ! This subroutine is a new version stokeslet Subroutine.
1906 ! compute:
1907 ! *The value of the Stokeslet over each singular element
1908 ! Now, (March/ 09 / 2015) this subroutine was made.
1909 !-----
1910 USE Mod_Nodal_Interp
1911 USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, Ulog, pi, eps, x0, y0, z0
1912 !-----
1913 IMPLICIT NONE
1914 !-----
1915 ! Variables
1916 !-----
1917 REAL (KIND = DBL), INTENT(IN) :: x10, y10, z10 !singularity coordinates
1918 INTEGER, INTENT(IN) :: k !number of element
1919 REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TExz !value of stresslet in the singular element
1920 REAL (KIND = DBL), INTENT(OUT) :: TEyx, TEyy, TEyz !value of stresslet in the singular element
1921 REAL (KIND = DBL), INTENT(OUT) :: TEzx, TEzy, TEzz !value of stresslet in the singular element
1922 !-----
1923 ! Variables inside the subroutine
1924 !-----
1925 INTEGER :: i, j !counters
1926 INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
1927 !-----
1928 REAL (KIND = DBL) :: cf, fill1, fill2, fill3 !integration weigh coefficients
1929 REAL (KIND = DBL) :: DxDx, DyDx, DzDx !tangential vector around the triangle (xi,eta)
1930 REAL (KIND = DBL) :: hss !weighth
1931 REAL (KIND = DBL) :: cpi !selection to add the solid angle
1932 REAL (KIND = DBL) :: modx0 !weighth
1933 REAL (KIND = DBL) :: bxv1, bvy1, bvz1, & !vectoreal product
1934 & bxv2, bvy2, bvz2, &
1935 & bxv3, bvy3, bvz3
1936 REAL (KIND = DBL) :: QExx, QExy, QExz, & !Iq tensor
1937 & QEyx, QEyy, QEyz, &
1938 & QExz, QEzy, QEzz
1939 REAL (KIND = DBL) :: PEx, PEy, PEz, PE !Ip tensor
1940 REAL (KIND = DBL) :: yvx1, yvy1, yvz1 !vector y1
1941 REAL (KIND = DBL) :: yvx2, yvy2, yvz2 !vector y2
1942 REAL (KIND = DBL) :: yvx3, yvy3, yvz3 !vector y3

```

```

1943 REAL (KIND = DBL) :: yvx4, yvy4, yvz4 !vector y4
1944 REAL (KIND = DBL) :: yvx5, yvy5, yvz5 !vector y5
1945 REAL (KIND = DBL) :: yvx6, yvy6, yvz6 !vector y6
1946 REAL (KIND = DBL) :: zvx1, zvy1, zvz1 !vector (z1-x0)
1947 REAL (KIND = DBL) :: zvx2, zvy2, zvz2 !vector (z2-x0)
1948 REAL (KIND = DBL) :: zvx3, zvy3, zvz3 !vector (z3-x0)
1949 REAL (KIND = DBL) :: zvx4, zvy4, zvz4 !vector (z4-x0)
1950 REAL (KIND = DBL) :: zvx5, zvy5, zvz5 !vector (z5-x0)
1951 REAL (KIND = DBL) :: zvx6, zvy6, zvz6 !vector (z6-x0)
1952 REAL (KIND = DBL) :: zcx1, zcy1, zcz1 !vector z1 ^ z1+1
1953 REAL (KIND = DBL) :: zcx2, zcy2, zcz2 !vector z1 ^ z1+1
1954 REAL (KIND = DBL) :: zcx3, zcy3, zcz3 !vector z1 ^ z1+1
1955 REAL (KIND = DBL) :: zcx4, zcy4, zcz4 !vector z1 ^ z1+1
1956 REAL (KIND = DBL) :: zcx5, zcy5, zcz5 !vector z1 ^ z1+1
1957 REAL (KIND = DBL) :: zcx6, zcy6, zcz6 !vector z1 ^ z1+1
1958 REAL (KIND = DBL) :: zd1, zd2, zd3, zd4, zd5, zd6 !vector z1 . z1+1
1959 REAL (KIND = DBL) :: px2, py2, pz2 !pressure vector
1960 REAL (KIND = DBL) :: a1, a2, a3 !vector a
1961 REAL (KIND = DBL) :: a11, a12, a13, a14, a15, a16 !alpha angle between z1 and z1+1 from x0
1962 REAL (KIND = DBL) :: th1, th2, th3, th4, th5, th6 !theta angle between vector a and m1 vector
1963 REAL (KIND = DBL) :: e1, e2, e3, e4, e5, e6 !constant
1964 REAL (KIND = DBL) :: f1, f2, f3, f4, f5, f6 !constant
1965 REAL (KIND = DBL) :: g1, g2, g3, g4, g5, g6 !constant
1966 REAL (KIND = DBL) :: ay1, ay2, ay3 !vector dot product a*y1, a*y2
1967 REAL (KIND = DBL) :: by1, by2, by3 !product yi*(yi+a.yi)
1968 !-----
1969 ! Initialize
1970 !-----
1971 TExx = 0.000
1972 TExy = 0.000
1973 TExz = 0.000
1974 TEyx = 0.000
1975 TEyy = 0.000
1976 TEyz = 0.000
1977 TEzx = 0.000
1978 TEzy = 0.000
1979 TEzz = 0.000
1980
1981 QExx = 0.000
1982 QExy = 0.000
1983 QExz = 0.000
1984 QEyx = 0.000
1985 QEyy = 0.000
1986 QEyz = 0.000
1987 QExz = 0.000
1988 QEzy = 0.000
1989 QEzz = 0.000
1990
1991 PEx = 0.000
1992 PEy = 0.000
1993 PEz = 0.000
1994 PE = 0.000
1995
1996 modx0= DSQRT((x0(k)-x10)**2+(y0(k)-y10)**2+(z0(k)-z10)**2)
1997
1998 IF (modx0 >= 1.000 + eps) THEN
1999 cpi = 0.000
2000 ELSE IF (modx0 <= 1.000 + eps) THEN
2001 cpi = -8.000*pi
2002 ELSE
2003 cpi = -4.000*pi
2004 end if
2005
2006 IF ((ABS(x10-x0(k)) <= eps) .AND. &
2007 & (ABS(y10-y0(k)) <= eps) .AND. &
2008 & (ABS(z10-z0(k)) <= eps)) THEN
2009 a1 = x10
2010 a2 = y10
2011 a3 = z10
2012 ELSE
2013 a1 = x10-x0(k)
2014 a2 = y10-y0(k)

```

```

2015      a3 = z10-z0(k)
2016      END IF
2017
2018      la1 = x00-y0(k)
2019      la2 = y00-y0(k)
2020      la3 = z00-z0(k)
2021      modx0= DSQRT(a1**2+a2**2+a3**2)
2022      a1 = a1/modx0
2023      a2 = a2/modx0
2024      a3 = a3/modx0
2025
2026
2027      !OPEN (9,file="TE.out")
2028
2029 !-----
2029 ! vertices of the kth triangle
2030 !-----
2031      i1 = n(k,1)
2032      i2 = n(k,2)
2033      i3 = n(k,3)
2034      i4 = n(k,4)
2035      i5 = n(k,5)
2036      i6 = n(k,6)
2037
2038      cf = 0.000
2039
2040 ! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
2041 ! rule formula
2042 !-----
2043 ! Firstly, to compute all vectors and angles involved.
2044 !-----
2045      yvx1 = p(i1,1)-x10
2046      yvy1 = p(i1,2)-y10
2047      yvz1 = p(i1,3)-z10
2048      yvx2 = p(i2,1)-x10
2049      yvy2 = p(i2,2)-y10
2050      yvz2 = p(i2,3)-z10
2051      yvx3 = p(i3,1)-x10
2052      yvy3 = p(i3,2)-y10
2053      yvz3 = p(i3,3)-z10
2054      yvx4 = p(i4,1)-x10
2055      yvy4 = p(i4,2)-y10
2056      yvz4 = p(i4,3)-z10
2057      yvx5 = p(i5,1)-x10
2058      yvy5 = p(i5,2)-y10
2059      yvz5 = p(i5,3)-z10
2060      yvx6 = p(i6,1)-x10
2061      yvy6 = p(i6,2)-y10
2062      yvz6 = p(i6,3)-z10
2063 !-----
2064      zvx1 = yvx1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
2065      zvy1 = yvy1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
2066      zvz1 = yvz1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
2067      zvx2 = yvx2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
2068      zvy2 = yvy2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
2069      zvz2 = yvz2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
2070      zvx3 = yvx3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
2071      zvy3 = yvy3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
2072      zvz3 = yvz3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
2073      zvx4 = yvx4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
2074      zvy4 = yvy4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
2075      zvz4 = yvz4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
2076      zvx5 = yvx5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
2077      zvy5 = yvy5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
2078      zvz5 = yvz5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
2079      zvx6 = yvx6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
2080      zvy6 = yvy6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
2081      zvz6 = yvz6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
2082 !-----
2083 ! Cross products.
2084
2085      zcx1 = zvy1*zvz4 - zvz1*zvy4
2086      zcy1 = zvz1*zvx4 - zvx1*zvz4

```

```

2087      zcz1 = zvx1*zvy4 - zvy1*zvx4
2088      zcx2 = zvy4*zvz2 - zvz4*zvy2
2089      zcy2 = zvz4*zvx2 - zvx4*zvz2
2090      zcz2 = zvx4*zvy2 - zvy4*zvx2
2091      zcx3 = zvy2*zvz5 - zvz2*zvy5
2092      zcy3 = zvz2*zvx5 - zvx2*zvz5
2093      zcz3 = zvx2*zvy5 - zvy2*zvx5
2094      zcx4 = zvy5*zvz3 - zvz5*zvy3
2095      zcy4 = zvz5*zvx3 - zvx5*zvz3
2096      zcz4 = zvx5*zvy3 - zvy5*zvx3
2097      zcx5 = zvy3*zvz6 - zvz3*zvy6
2098      zcy5 = zvz3*zvx6 - zvx3*zvz6
2099      zcz5 = zvx3*zvy6 - zvy3*zvx6
2100      zcx6 = zvy6*zvz1 - zvz6*zvy1
2101      zcy6 = zvz6*zvx1 - zvx6*zvz1
2102      zcz6 = zvx6*zvy1 - zvy6*zvx1
2103 !-----
2104 ! Dot products.
2105 !-----
2106      zd1 = zvx1*zvx4 + zvy1*zvy4 + zvx1*zvy4
2107      zd2 = zvx4*zvx2 + zvy4*zvy2 + zvx4*zvy2
2108      zd3 = zvx2*zvx5 + zvy2*zvy5 + zvx2*zvy5
2109      zd4 = zvx5*zvx3 + zvy5*zvy3 + zvx5*zvy3
2110      zd5 = zvx3*zvx6 + zvy3*zvy6 + zvx3*zvy6
2111      zd6 = zvx6*zvx1 + zvy6*zvy1 + zvx6*zvy1
2112 !-----
2113 ! Angles
2114 !-----
2115      a11 = DACOS( zvx1*zvx4 + zvy1*zvy4 + zvx1*zvy4 )
2116      a12 = DACOS( zvx4*zvx2 + zvy4*zvy2 + zvx4*zvy2 )
2117      a13 = DACOS( zvx2*zvx5 + zvy2*zvy5 + zvx2*zvy5 )
2118      a14 = DACOS( zvx5*zvx3 + zvy5*zvy3 + zvx5*zvy3 )
2119      a15 = DACOS( zvx3*zvx6 + zvy3*zvy6 + zvx3*zvy6 )
2120      a16 = DACOS( zvx6*zvx1 + zvy6*zvy1 + zvx6*zvy1 )
2121 !-----
2122      th1 = DACOS( a1*zvx1 + a2*zvy1 + a3*zvz1 )
2123      th2 = DACOS( a1*zvx4 + a2*zvy4 + a3*zvz4 )
2124      th3 = DACOS( a1*zvx2 + a2*zvy2 + a3*zvz2 )
2125      th4 = DACOS( a1*zvx5 + a2*zvy5 + a3*zvz5 )
2126      th5 = DACOS( a1*zvx3 + a2*zvy3 + a3*zvz3 )
2127      th6 = DACOS( a1*zvx6 + a2*zvy6 + a3*zvz6 )
2128 !-----
2129 ! e and g constants
2130 !-----
2131      e1 = ( DCOS( th2 ) - DCOS( th1 )*DCOS( a11 ) )/DSIN( a11 )
2132      e2 = ( DCOS( th3 ) - DCOS( th2 )*DCOS( a12 ) )/DSIN( a12 )
2133      e3 = ( DCOS( th4 ) - DCOS( th3 )*DCOS( a13 ) )/DSIN( a13 )
2134      e4 = ( DCOS( th5 ) - DCOS( th4 )*DCOS( a14 ) )/DSIN( a14 )
2135      e5 = ( DCOS( th6 ) - DCOS( th5 )*DCOS( a15 ) )/DSIN( a15 )
2136      e6 = ( DCOS( th1 ) - DCOS( th6 )*DCOS( a16 ) )/DSIN( a16 )
2137      f1 = DCOS( th1 )
2138      f2 = DCOS( th2 )
2139      f3 = DCOS( th3 )
2140      f4 = DCOS( th4 )
2141      f5 = DCOS( th5 )
2142      f6 = DCOS( th6 )
2143      g1 = DSQRT( 1.000 - e1**2 - f1**2 )
2144      g2 = DSQRT( 1.000 - e2**2 - f2**2 )
2145      g3 = DSQRT( 1.000 - e3**2 - f3**2 )
2146      g4 = DSQRT( 1.000 - e4**2 - f4**2 )
2147      g5 = DSQRT( 1.000 - e5**2 - f5**2 )
2148      g6 = DSQRT( 1.000 - e6**2 - f6**2 )
2149 !-----
2150 ! computation of curvature line integral along segment 1
2151 !-----
2152      QExx = ( ( zvx1 + zvx4 ) * zcx1 ) / ( 1.000 + zd1 )
2153      QEyy = ( ( zvx1 + zvx4 ) * zcy1 ) / ( 1.000 + zd1 )
2154      QExz = ( ( zvx1 + zvx4 ) * zcz1 ) / ( 1.000 + zd1 )
2155      QEyx = ( ( zvy1 + zvy4 ) * zcx1 ) / ( 1.000 + zd1 )
2156      QEyy = ( ( zvy1 + zvy4 ) * zcy1 ) / ( 1.000 + zd1 )
2157      QEyz = ( ( zvy1 + zvy4 ) * zcz1 ) / ( 1.000 + zd1 )
2158      QExz = ( ( zvx1 + zvx4 ) * zcx1 ) / ( 1.000 + zd1 )

```

```

2159 QEzy = ( zvx1 + zvx4)*zcy1 / ( 1.000 + zdl )
2160 QEzz = ( zvx1 + zvx4)*zcz1 / ( 1.000 + zdl )
-----
2161 PEK = ( 2.000*a1*zcx1/g1 )*( DATAN(( 1.000 - f1)*DTAN(al1/2) + e1 / g1) - DATAN(e1/g1) )
2162 PEY = ( 2.000*a2*zcy1/g1 )*( DATAN(( 1.000 - f1)*DTAN(al1/2) + e1 / g1) - DATAN(e1/g1) )
2163 PEZ = ( 2.000*a3*zcz1/g1 )*( DATAN(( 1.000 - f1)*DTAN(al1/2) + e1 / g1) - DATAN(e1/g1) )
-----
2166 ! computation of curvature line integral along segment 4
2167
2168 QExx = QExx + ( zvx4 + zvx2)*zcx2 / ( 1.000 + zd2 )
2169 QExy = QExy + ( zvx4 + zvx2)*zcy2 / ( 1.000 + zd2 )
2170 QExz = QExz + ( zvx4 + zvx2)*zcz2 / ( 1.000 + zd2 )
2171 QEyx = QEyx + ( zvy4 + zvy2)*zcx2 / ( 1.000 + zd2 )
2172 QEyy = QEyy + ( zvy4 + zvy2)*zcy2 / ( 1.000 + zd2 )
2173 QEyz = QEyz + ( zvy4 + zvy2)*zcz2 / ( 1.000 + zd2 )
2174 QEzx = QEzx + ( zvx4 + zvx2)*zcx2 / ( 1.000 + zd2 )
2175 QEzy = QEzy + ( zvx4 + zvx2)*zcy2 / ( 1.000 + zd2 )
2176 QEzz = QEzz + ( zvx4 + zvx2)*zcz2 / ( 1.000 + zd2 )
-----
2178 PEK = PEK + ( 2.000*a1*zcx2/g2 )*( DATAN(( 1.000 - f2)*DTAN(al2/2) + e2 / g2) - DATAN(e2/g2) )
2179 PEY = PEY + ( 2.000*a2*zcy2/g2 )*( DATAN(( 1.000 - f2)*DTAN(al2/2) + e2 / g2) - DATAN(e2/g2) )
2180 PEZ = PEZ + ( 2.000*a3*zcz2/g2 )*( DATAN(( 1.000 - f2)*DTAN(al2/2) + e2 / g2) - DATAN(e2/g2) )
-----
2182 ! computation of curvature line integral along segment 2
2183
2184 QExx = QExx + ( zvx2 + zvx5)*zcx3 / ( 1.000 + zd3 )
2185 QExy = QExy + ( zvx2 + zvx5)*zcy3 / ( 1.000 + zd3 )
2186 QExz = QExz + ( zvx2 + zvx5)*zcz3 / ( 1.000 + zd3 )
2187 QEyx = QEyx + ( zvy2 + zvy5)*zcx3 / ( 1.000 + zd3 )
2188 QEyy = QEyy + ( zvy2 + zvy5)*zcy3 / ( 1.000 + zd3 )
2189 QEyz = QEyz + ( zvy2 + zvy5)*zcz3 / ( 1.000 + zd3 )
2190 QEzx = QEzx + ( zvx2 + zvx5)*zcx3 / ( 1.000 + zd3 )
2191 QEzy = QEzy + ( zvx2 + zvx5)*zcy3 / ( 1.000 + zd3 )
2192 QEzz = QEzz + ( zvx2 + zvx5)*zcz3 / ( 1.000 + zd3 )
-----
2194 PEK = PEK + ( 2.000*a1*zcx3/g3 )*( DATAN(( 1.000 - f3)*DTAN(al3/3) + e3 / g3) - DATAN(e3/g3) )
2195 PEY = PEY + ( 2.000*a2*zcy3/g3 )*( DATAN(( 1.000 - f3)*DTAN(al3/3) + e3 / g3) - DATAN(e3/g3) )
2196 PEZ = PEZ + ( 2.000*a3*zcz3/g3 )*( DATAN(( 1.000 - f3)*DTAN(al3/3) + e3 / g3) - DATAN(e3/g3) )
-----
2198 ! computation of curvature line integral along segment 5
2199
2200 QExx = QExx + ( zvx5 + zvx3)*zcx4 / ( 1.000 + zd4 )
2201 QExy = QExy + ( zvx5 + zvx3)*zcy4 / ( 1.000 + zd4 )
2202 QExz = QExz + ( zvx5 + zvx3)*zcz4 / ( 1.000 + zd4 )
2203 QEyx = QEyx + ( zvy5 + zvy3)*zcx4 / ( 1.000 + zd4 )
2204 QEyy = QEyy + ( zvy5 + zvy3)*zcy4 / ( 1.000 + zd4 )
2205 QEyz = QEyz + ( zvy5 + zvy3)*zcz4 / ( 1.000 + zd4 )
2206 QEzx = QEzx + ( zvx5 + zvx3)*zcx4 / ( 1.000 + zd4 )
2207 QEzy = QEzy + ( zvx5 + zvx3)*zcy4 / ( 1.000 + zd4 )
2208 QEzz = QEzz + ( zvx5 + zvx3)*zcz4 / ( 1.000 + zd4 )
-----
2210 PEK = PEK + ( 2.000*a1*zcx4/g4 )*( DATAN(( 1.000 - f4)*DTAN(al4/2) + e4 / g4) - DATAN(e4/g4) )
2211 PEY = PEY + ( 2.000*a2*zcy4/g4 )*( DATAN(( 1.000 - f4)*DTAN(al4/2) + e4 / g4) - DATAN(e4/g4) )
2212 PEZ = PEZ + ( 2.000*a3*zcz4/g4 )*( DATAN(( 1.000 - f4)*DTAN(al4/2) + e4 / g4) - DATAN(e4/g4) )
-----
2214 ! computation of curvature line integral along segment 3
2215
2216 QExx = QExx + ( zvx3 + zvx6)*zcx5 / ( 1.000 + zd5 )
2217 QExy = QExy + ( zvx3 + zvx6)*zcy5 / ( 1.000 + zd5 )
2218 QExz = QExz + ( zvx3 + zvx6)*zcz5 / ( 1.000 + zd5 )
2219 QEyx = QEyx + ( zvy3 + zvy6)*zcx5 / ( 1.000 + zd5 )
2220 QEyy = QEyy + ( zvy3 + zvy6)*zcy5 / ( 1.000 + zd5 )
2221 QEyz = QEyz + ( zvy3 + zvy6)*zcz5 / ( 1.000 + zd5 )
2222 QEzx = QEzx + ( zvx3 + zvx6)*zcx5 / ( 1.000 + zd5 )
2223 QEzy = QEzy + ( zvx3 + zvx6)*zcy5 / ( 1.000 + zd5 )
2224 QEzz = QEzz + ( zvx3 + zvx6)*zcz5 / ( 1.000 + zd5 )
-----
2226 PEK = PEK + ( 2.000*a1*zcx5/g5 )*( DATAN(( 1.000 - f5)*DTAN(al5/2) + e5 / g5) - DATAN(e5/g5) )
2227 PEY = PEY + ( 2.000*a2*zcy5/g5 )*( DATAN(( 1.000 - f5)*DTAN(al5/2) + e5 / g5) - DATAN(e5/g5) )
2228 PEZ = PEZ + ( 2.000*a3*zcz5/g5 )*( DATAN(( 1.000 - f5)*DTAN(al5/2) + e5 / g5) - DATAN(e5/g5) )
-----
2229 ! computation of curvature line integral along segment 6

```

```

2231
2232 QExx = QExx + ( zvx6 + zvx1)*zcx6 / ( 1.000 + zd6 )
2233 QExy = QExy + ( zvx6 + zvx1)*zcy6 / ( 1.000 + zd6 )
2234 QExz = QExz + ( zvx6 + zvx1)*zcz6 / ( 1.000 + zd6 )
2235 QEyx = QEyx + ( zvy6 + zvy1)*zcx6 / ( 1.000 + zd6 )
2236 QEyy = QEyy + ( zvy6 + zvy1)*zcy6 / ( 1.000 + zd6 )
2237 QEyz = QEyz + ( zvy6 + zvy1)*zcz6 / ( 1.000 + zd6 )
2238 QEzx = QEzx + ( zvx6 + zvx1)*zcx6 / ( 1.000 + zd6 )
2239 QEzy = QEzy + ( zvx6 + zvx1)*zcy6 / ( 1.000 + zd6 )
2240 QEzz = QEzz + ( zvx6 + zvx1)*zcz6 / ( 1.000 + zd6 )
-----
2242 PEK = PEK + ( 2.000*a1*zcx6/g6 )*( DATAN(( 1.000 - f6)*DTAN(al6/2) + e6 / g6) - DATAN(e6/g6) )
2243 PEY = PEY + ( 2.000*a2*zcy6/g6 )*( DATAN(( 1.000 - f6)*DTAN(al6/2) + e6 / g6) - DATAN(e6/g6) )
2244 PEZ = PEZ + ( 2.000*a3*zcz6/g6 )*( DATAN(( 1.000 - f6)*DTAN(al6/2) + e6 / g6) - DATAN(e6/g6) )
-----
2246 TExx = 2.000*( QExx + cpi + PE )
2247 TEzy = 2.000*( QEzy )
2248 TEzx = 2.000*( QExz )
2249 TEyx = 2.000*( QEyx )
2250 TEyy = 2.000*( QEyy + cpi + PE )
2251 TEyz = 2.000*( QEyz )
2252 TEzx = 2.000*( QEzx )
2253 TEzy = 2.000*( QEzy )
2254 TEzz = 2.000*( QEzz + cpi + PE )
-----
2255 !
2256 END SUBROUTINE intr_lin_sing_s5
2257 !
2258 SUBROUTINE intr_lin_sing_s6(x00, y00, z00, &
2259 & k,
2260 & TExx, TEzy, TEzx, &
2261 & TEyx, TEyy, TEyz, &
2262 & TEzx, TEzy, TEzz )
2263 !
2264 ! This subroutine is a new version stokeslet Subroutine.
2265 ! Compute:
2266 ! *The value of the Stokeslet over each singular element
2267 ! How, (March/ 09 / 2015) this subroutine was made.
2268 !
2269 USE Mod_Nodal_Interp
2270 USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog, pi, eps, vnx0, vny0, vnz0, x0, y0, z0
2271 !
2272 IMPLICIT NONE
2273 !
2274 ! Variables
2275 !
2276 REAL (KIND = DBL), INTENT(IN) :: x00, y00, z00 !singularly coordinates
2277 INTEGER, INTENT(IN) :: k !number of element
2278 REAL (KIND = DBL), INTENT(OUT) :: TExx, TEzy, TEzx !value of stresslet in the singular element
2279 REAL (KIND = DBL), INTENT(OUT) :: TEyx, TEyy, TEyz !value of stresslet in the singular element
2280 REAL (KIND = DBL), INTENT(OUT) :: TEzx, TEzy, TEzz !value of stresslet in the singular element
2281 !
2282 ! Variables inside the subroutine
2283 !
2284 INTEGER :: i, j !counters
2285 INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
2286 !
2287 REAL (KIND = DBL) :: cf, fill1, fill2, fill3 !integration weigh coefficients
2288 REAL (KIND = DBL) :: DxDx, DyDy, DzDz !tangential vector around the triangle (xi,eta)
2289 REAL (KIND = DBL) :: hss !weight
2290 REAL (KIND = DBL) :: cpi !selection to add the solid angle
2291 REAL (KIND = DBL) :: modx0 !weight
2292 REAL (KIND = DBL) :: bvx1, bvy1, bvz1, & !vectorial product
2293 & bvx2, bvy2, bvz2, &
2294 & bvx3, bvy3, bvz3, &
2295 REAL (KIND = DBL) :: QExx, QEzy, QEzx, & !Iij tensor
2296 & QEyx, QEyy, QEyz, &
2297 & QEzx, QEzy, QEzz
2298 REAL (KIND = DBL) :: PEK, PEY, PEZ, PE !Ip tensor
2299 REAL (KIND = DBL) :: yvx1, yvy1, yvz1 !vector (y1-x0)
2300 REAL (KIND = DBL) :: yvx2, yvy2, yvz2 !vector (y2-x0)
2301 REAL (KIND = DBL) :: yvx3, yvy3, yvz3 !vector (y2-x0)
2302 REAL (KIND = DBL) :: px2, py2, pz2 !vector (y2-x0)

```

```

2303 REAL (KIND = DBL) :: a1, a2, a3          !vector a
2304 REAL (KIND = DBL) :: ay1, ay2, ay3      !vector dot product a*y1, a*y2
2305 REAL (KIND = DBL) :: by1, by2, by3      !product yi*(yi+a.yi)
2306 !-----
2307 ! Initialize
2308 !-----
2309 TExx = 0.000
2310 TExy = 0.000
2311 TExz = 0.000
2312 TEyx = 0.000
2313 TEyy = 0.000
2314 TEyz = 0.000
2315 TEzx = 0.000
2316 TEzy = 0.000
2317 TEzz = 0.000
2318
2319 QExx = 0.000
2320 QExy = 0.000
2321 QExz = 0.000
2322 QEyx = 0.000
2323 QEyy = 0.000
2324 QEyz = 0.000
2325 QExz = 0.000
2326 QEzy = 0.000
2327 QEzz = 0.000
2328
2329 PEx = 0.000
2330 PEy = 0.000
2331 PEz = 0.000
2332 PE = 0.000
2333
2334 modx0= DSQRT((x0(k)-x00)**2+(y0(k)-y00)**2+(z0(k)-z00)**2)
2335
2336 IF (modx0 >= 1.000 + eps) THEN
2337   cp1 = 0.000
2338 ELSE IF (modx0 <= 1.000 + eps) THEN
2339   cp1 = -8.000*pi
2340 ELSE
2341   cp1 = -4.000*pi
2342 end if
2343
2344 IF ((ABS(x00-x0(k)) <= eps) .AND. &
2345    & (ABS(y00-y0(k)) <= eps) .AND. &
2346    & (ABS(z00-z0(k)) <= eps)) THEN
2347   a1 = x00
2348   a2 = y00
2349   a3 = z00
2350 ELSE
2351   a1 = x00-x0(k)
2352   a2 = y00-y0(k)
2353   a3 = z00-z0(k)
2354 END IF
2355 modx0= DSQRT(a1**2+a2**2+a3**2)
2356 a1 = a1/modx0
2357 a2 = a2/modx0
2358 a3 = a3/modx0
2359
2360 !OPEN (9,file="TE.out")
2361 !-----
2362 ! vertices of the kth triangle
2363 !-----
2364 i1 = n(k,1)
2365 i2 = n(k,2)
2366 i3 = n(k,3)
2367 i4 = n(k,4)
2368 i5 = n(k,5)
2369 i6 = n(k,6)
2370 !-----
2371 cf = 0.000
2372 !-----
2373 ! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
2374 ! rule formula

```

```

2375 !-----
2376 bvx1 = 0.000
2377 bvy1 = 0.000
2378 bvz1 = 0.000
2379 bvx2 = 0.000
2380 bvy2 = 0.000
2381 bvz2 = 0.000
2382 bvx3 = 0.000
2383 bvy3 = 0.000
2384 bvz3 = 0.000
2385 yvx1 = 0.000
2386 yvy1 = 0.000
2387 yvz1 = 0.000
2388 yvx2 = 0.000
2389 yvy2 = 0.000
2390 yvz2 = 0.000
2391 yvx3 = 0.000
2392 yvy3 = 0.000
2393 yvz3 = 0.000
2394 !-----
2395 ! computation of curvature line integral along segment 1-6
2396 !-----
2397 DxDx = p(16,1) - p(11,1)
2398 DyDx = p(16,2) - p(11,2)
2399 DzDx = p(16,3) - p(11,3)
2400 !-----
2401 px2 = p(11,1) + 0.500*DxDx
2402 py2 = p(11,2) + 0.500*DyDx
2403 pz2 = p(11,3) + 0.500*DzDx
2404 !-----
2405 hss = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
2406 DxDx = DxDx/hss
2407 DyDx = DyDx/hss
2408 DzDx = DzDx/hss
2409 !-----
2410 yvx1 = p(11,1) - x00
2411 yvy1 = p(11,2) - y00
2412 yvz1 = p(11,3) - z00
2413 fil1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
2414 !-----
2415 bvx1 = (DyDx*yvz1 - DzDx*yvy1)
2416 bvy1 = (DxDx*yvx1 - Dx0x*yvz1)
2417 bvz1 = (DxDx*yvy1 - DyDx*yvx1)
2418 !-----
2419 yvx2 = px2 - x00
2420 yvy2 = py2 - y00
2421 yvz2 = pz2 - z00
2422 fil2 = DSQRT(yvx2**2 + yvy2**2 + yvz2**2)
2423 !-----
2424 bvx2 = (DyDx*yvz2 - DzDx*yvy2)
2425 bvy2 = (DxDx*yvx2 - Dx0x*yvz2)
2426 bvz2 = (DxDx*yvy2 - DyDx*yvx2)
2427 !-----
2428 yvx3 = p(16,1) - x00
2429 yvy3 = p(16,2) - y00
2430 yvz3 = p(16,3) - z00
2431 fil3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
2432 !-----
2433 bvx3 = (DyDx*yvz3 - DzDx*yvy3)
2434 bvy3 = (DxDx*yvx3 - Dx0x*yvz3)
2435 bvz3 = (DxDx*yvy3 - DyDx*yvx3)
2436 !-----
2437 QExx = QExx + hss*( - (bvz1*(yvx1/fil1**3)) - 4.000*(bvz2*(yvx2/fil2**3)) - (bvz3*(yvx3/fil3**3)) )/3.
2438 QExy = QExy + hss*( - (bvy1*(yvx1/fil1**3)) - 4.000*(bvy2*(yvx2/fil2**3)) - (bvy3*(yvx3/fil3**3)) )/3.
2439 QExz = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - 4.000*(bvz2*(yvx2/fil2**3)) - (bvz3*(yvx3/fil3**3)) )/3.
2440 QEyx = QEyx + hss*( - (bvz1*(yvy1/fil1**3)) - 4.000*(bvz2*(yvy2/fil2**3)) - (bvz3*(yvy3/fil3**3)) )/3.
2441 QEyy = QEyy + hss*( - (bvy1*(yvy1/fil1**3)) - 4.000*(bvy2*(yvy2/fil2**3)) - (bvy3*(yvy3/fil3**3)) )/3.
2442 QEvz = QEvz + hss*( - (bvz1*(yvy1/fil1**3)) - 4.000*(bvz2*(yvy2/fil2**3)) - (bvz3*(yvy3/fil3**3)) )/3.

```

```

2442 QEyz = QEyz + hss*( - (bvz1*(yvz1/fil1**3)) - 4.000*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
000
2443 QExx = QExx + hss*( - (bvz1*(yvz1/fil1**3)) - 4.000*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
000
2444 QEzy = QEzy + hss*( - (bvz1*(yvz1/fil1**3)) - 4.000*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
000
2445 QEzz = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - 4.000*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
000
-----
2446 !
2447 !a1 = (bvz1 + bvz2 + bvz3)/3.000
2448 !a2 = (bvz1 + bvz2 + bvz3)/3.000
2449 !a3 = (bvz1 + bvz2 + bvz3)/3.000
2450 !modx0= DSQRT(a1**2+a2**2+a3**2)
2451 !a1 = a1/modx0
2452 !a2 = a2/modx0
2453 !a3 = a3/modx0
2454 !ay1 = (a1*yvz1 + a2*yvz1 + a3*yvz1)
2455 !ay2 = (a1*yvz2 + a2*yvz2 + a3*yvz2)
2456 !ay3 = (a1*yvz3 + a2*yvz3 + a3*yvz3)
2457 !by1 = DSQRT(( fil1*(fil1+ay1) + 0.3 )**2)
2458 !by2 = DSQRT(( fil2*(fil2+ay2) + 0.3 )**2)
2459 !by3 = DSQRT(( fil3*(fil3+ay3) + 0.3 )**2)
2460 !b1 = DSQRT(( fil1*(fil1+ay1) )**2)
2461 !b2 = DSQRT(( fil2*(fil2+ay2) )**2)
2462 !b3 = DSQRT(( fil3*(fil3+ay3) )**2)
2463 !WRITE (9,*) by1
2464 !WRITE (9,*) by2
2465 !WRITE (9,*) by3
2466 !PEX = ( (a1*bvz1/by1) + 4.000*(a2*bvz2/by2) + (a3*bvz3/by3) )*a1
2467 !PEy = ( (a1*bvz1/by1) + 4.000*(a2*bvz2/by2) + (a3*bvz3/by3) )*a2
2468 !PEz = ( (a1*bvz1/by1) + 4.000*(a2*bvz2/by2) + (a3*bvz3/by3) )*a3
2469 !fay1 = (a1*yvz1 + a2*yvz1 + a3*yvz1)
2470 !fay2 = (a1*yvz2 + a2*yvz2 + a3*yvz2)
2471 !fay3 = (a1*yvz3 + a2*yvz3 + a3*yvz3)
2472 !fby1 = fil1*(fil1+ay1)
2473 !fby2 = fil2*(fil2+ay2)
2474 !fby3 = fil3*(fil3+ay3)
2475 !fPEX = ( (bvz1/by1) + 4.000*(bvz2/by2) + (bvz3/by3) )*a1
2476 !fPEy = ( (bvz1/by1) + 4.000*(bvz2/by2) + (bvz3/by3) )*a2
2477 !fPEz = ( (bvz1/by1) + 4.000*(bvz2/by2) + (bvz3/by3) )*a3
2478 !PEX = ( (bvz1**2/(fil1**2)) + 4.000*(bvz2**2/(fil2**2)) + (bvz3**2/(fil3**2)) )
2479 !PEy = ( (bvz1**2/(fil1**2)) + 4.000*(bvz2**2/(fil2**2)) + (bvz3**2/(fil3**2)) )
2480 !PEz = ( (bvz1**2/(fil1**2)) + 4.000*(bvz2**2/(fil2**2)) + (bvz3**2/(fil3**2)) )
2481 !PE = hss*( PEX + PEy + PEz )/3.000
-----
2482 !
2483 ! computation of curvature line integral along segment 6-3
2484 !
2485 !bvz1 = 0.000
2486 !bvz2 = 0.000
2487 !bvz3 = 0.000
2488 !bvz1 = 0.000
2489 !bvz2 = 0.000
2490 !bvz3 = 0.000
2491 !bvz1 = 0.000
2492 !bvz2 = 0.000
2493 !bvz3 = 0.000
2494 !yvz1 = 0.000
2495 !yvz2 = 0.000
2496 !yvz3 = 0.000
2497 !yvz1 = 0.000
2498 !yvz2 = 0.000
2499 !yvz3 = 0.000
2500 !yvz1 = 0.000
2501 !yvz2 = 0.000
2502 !yvz3 = 0.000
-----
2503 !
2504 !DxDx = p(13,1) - p(16,1)
2505 !DyDx = p(13,2) - p(16,2)
2506 !DzDx = p(13,3) - p(16,3)
-----
2508 !px2 = p(16,1) + 0.500*DxDx
2509 !py2 = p(16,2) + 0.500*DyDx

```

```

2510 !p2 = p(16,3) + 0.500*DzDx
2511 !
2512 !hss = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
2513 !DxDx = DxDx/hss
2514 !DyDx = DyDx/hss
2515 !DzDx = DzDx/hss
-----
2517 !yvz1 = p(16,1) - x00
2518 !yvz1 = p(16,2) - y00
2519 !yvz1 = p(16,3) - z00
2520 !fil1 = DSQRT(yvz1**2 + yvz1**2 + yvz2**2)
-----
2522 !bvz1 = (DyDx*yvz1 - DzDx*yvz1)
2523 !bvz1 = (DxDx*yvz1 - DxDx*yvz1)
2524 !bvz1 = (DxDx*yvz1 - DyDx*yvz1)
-----
2525 !
2526 !yvz2 = px2 - x00
2527 !yvz2 = py2 - y00
2528 !yvz2 = pz2 - z00
2529 !fil2 = DSQRT(yvz2**2 + yvz2**2 + yvz2**2)
-----
2530 !
2531 !bvz2 = (DyDx*yvz2 - DzDx*yvz2)
2532 !bvz2 = (DxDx*yvz2 - DxDx*yvz2)
2533 !bvz2 = (DxDx*yvz2 - DyDx*yvz2)
-----
2534 !
2535 !yvz3 = p(13,1) - x00
2536 !yvz3 = p(13,2) - y00
2537 !yvz3 = p(13,3) - z00
2538 !fil3 = DSQRT(yvz3**2 + yvz3**2 + yvz3**2)
-----
2539 !
2540 !bvz3 = (DyDx*yvz3 - DzDx*yvz3)
2541 !bvz3 = (DxDx*yvz3 - DxDx*yvz3)
2542 !bvz3 = (DxDx*yvz3 - DyDx*yvz3)
-----
2543 !
2544 !QExx = QExx + hss*( - (bvz1*(yvz1/fil1**3)) - 4.000*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
000
2545 !QExy = QExy + hss*( - (bvz1*(yvz1/fil1**3)) - 4.000*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
000
2546 !QExz = QExz + hss*( - (bvz1*(yvz1/fil1**3)) - 4.000*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
000
2547 !QEyx = QEyx + hss*( - (bvz1*(yvz1/fil1**3)) - 4.000*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
000
2548 !QEyy = QEyy + hss*( - (bvz1*(yvz1/fil1**3)) - 4.000*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
000
2549 !QEyz = QEyz + hss*( - (bvz1*(yvz1/fil1**3)) - 4.000*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
000
2550 !QExx = QExx + hss*( - (bvz1*(yvz1/fil1**3)) - 4.000*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
000
2551 !QEzy = QEzy + hss*( - (bvz1*(yvz1/fil1**3)) - 4.000*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
000
2552 !QEzz = QEzz + hss*( - (bvz1*(yvz1/fil1**3)) - 4.000*(bvz2*(yvz2/fil2**3)) - (bvz3*(yvz3/fil3**3)) )/3.
000
-----
2553 !
2554 !a1 = (bvz1 + bvz2 + bvz3)/3.000
2555 !a2 = (bvz1 + bvz2 + bvz3)/3.000
2556 !a3 = (bvz1 + bvz2 + bvz3)/3.000
2557 !modx0= DSQRT(a1**2+a2**2+a3**2)
2558 !a1 = a1/modx0
2559 !a2 = a2/modx0
2560 !a3 = a3/modx0
2561 !ay1 = (a1*yvz1 + a2*yvz1 + a3*yvz1)
2562 !ay2 = (a1*yvz2 + a2*yvz2 + a3*yvz2)
2563 !ay3 = (a1*yvz3 + a2*yvz3 + a3*yvz3)
2564 !by1 = DSQRT(( fil1*(fil1+ay1) + 0.3 )**2)
2565 !by2 = DSQRT(( fil2*(fil2+ay2) + 0.3 )**2)
2566 !by3 = DSQRT(( fil3*(fil3+ay3) + 0.3 )**2)
2567 !b1 = DSQRT(( fil1*(fil1+ay1) )**2)
2568 !b2 = DSQRT(( fil2*(fil2+ay2) )**2)
2569 !b3 = DSQRT(( fil3*(fil3+ay3) )**2)
2570 !WRITE (9,*) by1
2571 !WRITE (9,*) by2
2572 !WRITE (9,*) by3

```

```

2573 PEX = ( (a1*bvx1/by1) + 4.000*(a2*bvx2/by2) + (a3*bvx3/by3) )*a1
2574 PEY = ( (a1*bvy1/by1) + 4.000*(a2*bvy2/by2) + (a3*bvy3/by3) )*a2
2575 PEZ = ( (a1*bvz1/bv1) + 4.000*(a2*bvz2/bv2) + (a3*bvz3/bv3) )*a3
2576 !ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
2577 !ay2 = (a1*yvx2 + a2*yvy2 + a3*yvz2)
2578 !ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
2579 !by1 = fill*(fill+ay1)
2580 !by2 = fill2*(fill2+ay2)
2581 !by3 = fill3*(fill3+ay3)
2582 !PEX = ( (bvx1/by1) + 4.000*(bvz2/by2) + (bvz3/by3) )*a1
2583 !PEY = ( (bvy1/by1) + 4.000*(bvz2/by2) + (bvz3/by3) )*a2
2584 !PEZ = ( (bvz1/bv1) + 4.000*(bvz2/bv2) + (bvz3/bv3) )*a3
2585 !PEX = ( (bvz1**2/(fill**2)) + 4.000*(bvz2**2/(fill2**2)) + (bvz3**2/(fill3**2)) )
2586 !PEY = ( (bvz1**2/(fill**2)) + 4.000*(bvz2**2/(fill2**2)) + (bvz3**2/(fill3**2)) )
2587 !PEZ = ( (bvz1**2/(fill**2)) + 4.000*(bvz2**2/(fill2**2)) + (bvz3**2/(fill3**2)) )
2588 PE = PE + hss*( PEX + PEY + PEZ )/3.000
-----
2589 ! computation of curvature line integral along segment 2-5
2591 !-----
2592 !bx1 = 0.000
2593 !by1 = 0.000
2594 !bv1 = 0.000
2595 !bv2 = 0.000
2596 !bv3 = 0.000
2597 !bvz2 = 0.000
2598 !bvz3 = 0.000
2599 !bvz3 = 0.000
2600 !bvz3 = 0.000
2601 !yvx1 = 0.000
2602 !yvy1 = 0.000
2603 !yvz1 = 0.000
2604 !yvz2 = 0.000
2605 !yvz2 = 0.000
2606 !yvz2 = 0.000
2607 !yvz3 = 0.000
2608 !yvz3 = 0.000
2609 !yvz3 = 0.000
2610 !-----
2611 !DxDx = p(15,1) - p(13,1)
2612 !DyDx = p(15,2) - p(13,2)
2613 !DzDx = p(15,3) - p(13,3)
2614 !-----
2615 !px2 = p(13,1) + 0.500*DxDx
2616 !py2 = p(13,2) + 0.500*DyDx
2617 !pz2 = p(13,3) + 0.500*DzDx
2618 !-----
2619 !hss = DSQRT(DxDx**2 + DyDx**2 + DzDx**2)
2620 !DxDx = DxDx/hss
2621 !DyDx = DyDx/hss
2622 !DzDx = DzDx/hss
2623 !-----
2624 !yvz1 = p(13,1) - x00
2625 !yvz1 = p(13,2) - y00
2626 !yvz1 = p(13,3) - z00
2627 !fill = DSQRT((yvz1**2 + yvz1**2 + yvz1**2))
2628 !-----
2629 !bvz1 = (DyDx*yvz1 - DzDx*yvz1)
2630 !bvz1 = (DzDx*yvz1 - DxDx*yvz1)
2631 !bvz1 = (DxDx*yvz1 - DyDx*yvz1)
2632 !-----
2633 !yvz2 = pz2 - x00
2634 !yvz2 = pz2 - y00
2635 !yvz2 = pz2 - z00
2636 !fill2 = DSQRT((yvz2**2 + yvz2**2 + yvz2**2))
2637 !-----
2638 !bvz2 = (DyDx*yvz2 - DzDx*yvz2)
2639 !bvz2 = (DzDx*yvz2 - DxDx*yvz2)
2640 !bvz2 = (DxDx*yvz2 - DyDx*yvz2)
2641 !-----
2642 !yvz3 = p(15,1) - x00
2643 !yvz3 = p(15,2) - y00
2644 !yvz3 = p(15,3) - z00

```

```

2645 fill3 = DSQRT((yvz3**2 + yvz3**2 + yvz3**2))
2646 !-----
2647 !bvz3 = (DyDx*yvz3 - DzDx*yvz3)
2648 !bvz3 = (DzDx*yvz3 - DxDx*yvz3)
2649 !bvz3 = (DxDx*yvz3 - DyDx*yvz3)
2650 !-----
2651 !QExx = QExx + hss*( - (bvz1*(yvz1/fill1**3)) - 4.000*(bvz2*(yvz2/fill2**3)) - (bvz3*(yvz3/fill3**3)) )/3.
000
2652 !QExy = QExy + hss*( - (bvz1*(yvz1/fill1**3)) - 4.000*(bvz2*(yvz2/fill2**3)) - (bvz3*(yvz3/fill3**3)) )/3.
000
2653 !QExz = QExz + hss*( - (bvz1*(yvz1/fill1**3)) - 4.000*(bvz2*(yvz2/fill2**3)) - (bvz3*(yvz3/fill3**3)) )/3.
000
2654 !QEyx = QEyx + hss*( - (bvz1*(yvz1/fill1**3)) - 4.000*(bvz2*(yvz2/fill2**3)) - (bvz3*(yvz3/fill3**3)) )/3.
000
2655 !QEyy = QEyy + hss*( - (bvz1*(yvz1/fill1**3)) - 4.000*(bvz2*(yvz2/fill2**3)) - (bvz3*(yvz3/fill3**3)) )/3.
000
2656 !QEyz = QEyz + hss*( - (bvz1*(yvz1/fill1**3)) - 4.000*(bvz2*(yvz2/fill2**3)) - (bvz3*(yvz3/fill3**3)) )/3.
000
2657 !QEzx = QEzx + hss*( - (bvz1*(yvz1/fill1**3)) - 4.000*(bvz2*(yvz2/fill2**3)) - (bvz3*(yvz3/fill3**3)) )/3.
000
2658 !QEzy = QEzy + hss*( - (bvz1*(yvz1/fill1**3)) - 4.000*(bvz2*(yvz2/fill2**3)) - (bvz3*(yvz3/fill3**3)) )/3.
000
2659 !QEzz = QEzz + hss*( - (bvz1*(yvz1/fill1**3)) - 4.000*(bvz2*(yvz2/fill2**3)) - (bvz3*(yvz3/fill3**3)) )/3.
000
2660 !-----
2661 !a1 = (bvz1 + bvz2 + bvz3)/3.000
2662 !a2 = (bvz1 + bvz2 + bvz3)/3.000
2663 !a3 = (bvz1 + bvz2 + bvz3)/3.000
2664 !modx0 = DSQRT(a1**2+a2**2+a3**2)
2665 !a1 = a1/modx0
2666 !a2 = a2/modx0
2667 !a3 = a3/modx0
2668 !ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
2669 !ay2 = (a1*yvx2 + a2*yvy2 + a3*yvz2)
2670 !ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
2671 !by1 = DSQRT(( fill1*(fill1+ay1) + 0.3 )**2)
2672 !by2 = DSQRT(( fill2*(fill2+ay2) + 0.3 )**2)
2673 !by3 = DSQRT(( fill3*(fill3+ay3) + 0.3 )**2)
2674 !b1 = DSQRT(( fill1*(fill1+ay1) )**2)
2675 !b2 = DSQRT(( fill2*(fill2+ay2) )**2)
2676 !b3 = DSQRT(( fill3*(fill3+ay3) )**2)
2677 !WRITE (9,*) by1
2678 !WRITE (9,*) by2
2679 !WRITE (9,*) by3
2680 !PEX = ( (a1*bvx1/by1) + 4.000*(a2*bvx2/by2) + (a3*bvx3/by3) )*a1
2681 !PEY = ( (a1*bvy1/by1) + 4.000*(a2*bvy2/by2) + (a3*bvy3/by3) )*a2
2682 !PEZ = ( (a1*bvz1/bv1) + 4.000*(a2*bvz2/bv2) + (a3*bvz3/bv3) )*a3
2683 !ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
2684 !ay2 = (a1*yvx2 + a2*yvy2 + a3*yvz2)
2685 !ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
2686 !by1 = fill1*(fill1+ay1)
2687 !by2 = fill2*(fill2+ay2)
2688 !by3 = fill3*(fill3+ay3)
2689 !PEX = ( (bvz1/by1) + 4.000*(bvz2/by2) + (bvz3/by3) )*a1
2690 !PEY = ( (bvz1/by1) + 4.000*(bvz2/by2) + (bvz3/by3) )*a2
2691 !PEZ = ( (bvz1/bv1) + 4.000*(bvz2/bv2) + (bvz3/bv3) )*a3
2692 !PEX = ( (bvz1**2/(fill**2)) + 4.000*(bvz2**2/(fill2**2)) + (bvz3**2/(fill3**2)) )
2693 !PEY = ( (bvz1**2/(fill**2)) + 4.000*(bvz2**2/(fill2**2)) + (bvz3**2/(fill3**2)) )
2694 !PEZ = ( (bvz1**2/(fill**2)) + 4.000*(bvz2**2/(fill2**2)) + (bvz3**2/(fill3**2)) )
2695 PE = PE + hss*( PEX + PEY + PEZ )/3.000
2696 !-----
2697 ! computation of curvature line integral along segment 5-2
2698 !-----
2699 !bx1 = 0.000
2700 !by1 = 0.000
2701 !bv1 = 0.000
2702 !bv2 = 0.000
2703 !bv3 = 0.000
2704 !bvz2 = 0.000
2705 !bvz3 = 0.000
2706 !bvz3 = 0.000
2707 !bvz3 = 0.000

```



```

2708 yvx1 = 0.000
2709 yvy1 = 0.000
2710 yvz1 = 0.000
2711 yvx2 = 0.000
2712 yvy2 = 0.000
2713 yvz2 = 0.000
2714 yvx3 = 0.000
2715 yvy3 = 0.000
2716 yvz3 = 0.000
-----
2718 Dx0x = p(12,1) - p(15,1)
2719 Dy0x = p(12,2) - p(15,2)
2720 Dz0x = p(12,3) - p(15,3)
-----
2722 px2 = p(15,1) + 0.500*DxDx
2723 py2 = p(15,2) + 0.500*DyDx
2724 pz2 = p(15,3) + 0.500*DzDx
-----
2726 hss = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
2727 Dx0x = Dx0x/hss
2728 Dy0x = Dy0x/hss
2729 Dz0x = Dz0x/hss
-----
2731 yvx1 = p(15,1) - x00
2732 yvy1 = p(15,2) - y00
2733 yvz1 = p(15,3) - z00
2734 fill1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
-----
2736 bvx1 = (Dy0x*yvz1 - Dz0x*yvy1)
2737 bvy1 = (Dz0x*yvx1 - Dx0x*yvz1)
2738 bvz1 = (Dx0x*yvy1 - Dy0x*yvx1)
-----
2740 yvx2 = px2 - x00
2741 yvy2 = py2 - y00
2742 yvz2 = pz2 - z00
2743 fill2 = DSQRT(yvx2**2 + yvy2**2 + yvz2**2)
-----
2744 bvx2 = (Dy0x*yvz2 - Dz0x*yvy2)
2745 bvy2 = (Dz0x*yvx2 - Dx0x*yvz2)
2746 bvz2 = (Dx0x*yvy2 - Dy0x*yvx2)
-----
2749 yvx3 = p(12,1) - x00
2750 yvy3 = p(12,2) - y00
2751 yvz3 = p(12,3) - z00
2752 fill3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
-----
2754 bvx3 = (Dy0x*yvz3 - Dz0x*yvy3)
2755 bvy3 = (Dz0x*yvx3 - Dx0x*yvz3)
2756 bvz3 = (Dx0x*yvy3 - Dy0x*yvx3)
-----
2758 QExx = QExx + hss*( - (bvx1*(yvx1/fill1**3) - 4.000*(bvz2*(yvx2/fill2**3) - (bvz3*(yvx3/fill3**3) )/3.
2759 QExy = QExy + hss*( - (bvy1*(yvx1/fill1**3) - 4.000*(bvz2*(yvx2/fill2**3) - (bvz3*(yvx3/fill3**3) )/3.
2760 QExz = QExz + hss*( - (bvz1*(yvx1/fill1**3) - 4.000*(bvz2*(yvx2/fill2**3) - (bvz3*(yvx3/fill3**3) )/3.
2761 QEyx = QEyx + hss*( - (bvx1*(yvy1/fill1**3) - 4.000*(bvz2*(yvy2/fill2**3) - (bvz3*(yvy3/fill3**3) )/3.
2762 QEyy = QEyy + hss*( - (bvy1*(yvy1/fill1**3) - 4.000*(bvz2*(yvy2/fill2**3) - (bvz3*(yvy3/fill3**3) )/3.
2763 QEyz = QEyz + hss*( - (bvz1*(yvy1/fill1**3) - 4.000*(bvz2*(yvy2/fill2**3) - (bvz3*(yvy3/fill3**3) )/3.
2764 QExx = QExx + hss*( - (bvx1*(yvx1/fill1**3) - 4.000*(bvz2*(yvx2/fill2**3) - (bvz3*(yvx3/fill3**3) )/3.
2765 QEzy = QEzy + hss*( - (bvy1*(yvy1/fill1**3) - 4.000*(bvz2*(yvy2/fill2**3) - (bvz3*(yvy3/fill3**3) )/3.
2766 QEzz = QEzz + hss*( - (bvz1*(yvx1/fill1**3) - 4.000*(bvz2*(yvx2/fill2**3) - (bvz3*(yvx3/fill3**3) )/3.
-----
2767 !a1 = (bvx1 + bvy2 + bvz3)/3.000
2768 !a2 = (bvy1 + bvy2 + bvy3)/3.000
2769 !a3 = (bvz1 + bvz2 + bvz3)/3.000
2770

```

```

2771 !modx0= DSQRT(a1**2+a2**2+a3**2)
2772 !a1 = a1/modx0
2773 !a2 = a2/modx0
2774 !a3 = a3/modx0
2775 ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
2776 ay2 = (a1*yvx2 + a2*yvy2 + a3*yvz2)
2777 ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
2778 !by1 = DSQRT(( fill1*(fill1+ay1) + 0.3 )**2)
2779 !by2 = DSQRT(( fill2*(fill2+ay2) + 0.3 )**2)
2780 !by3 = DSQRT(( fill3*(fill3+ay3) + 0.3 )**2)
2781 by1 = DSQRT(( fill1*(fill1+ay1) )**2)
2782 by2 = DSQRT(( fill2*(fill2+ay2) )**2)
2783 by3 = DSQRT(( fill3*(fill3+ay3) )**2)
2784 !WRITE (9,*) by1
2785 !WRITE (9,*) by2
2786 !WRITE (9,*) by3
2787 !PEX = ( (a1*bvx1/by1) + 4.000*(a2*bvx2/by2) + (a3*bvx3/by3) ) *a1
2788 !PEY = ( (a1*bvy1/by1) + 4.000*(a2*bvy2/by2) + (a3*bvy3/by3) ) *a2
2789 !PEZ = ( (a1*bvz1/by1) + 4.000*(a2*bvz2/by2) + (a3*bvz3/by3) ) *a3
2790 !ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
2791 !ay2 = (a1*yvx2 + a2*yvy2 + a3*yvz2)
2792 !ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
2793 !by1 = fill1*(fill1+ay1)
2794 !by2 = fill2*(fill2+ay2)
2795 !by3 = fill3*(fill3+ay3)
2796 !PEX = ( (bvx1/by1) + 4.000*(bvz2/by2) + (bvz3/by3) ) *a1
2797 !PEY = ( (bvy1/by1) + 4.000*(bvz2/by2) + (bvz3/by3) ) *a2
2798 !PEZ = ( (bvz1/by1) + 4.000*(bvz2/by2) + (bvz3/by3) ) *a3
2799 ! PEX = ( (bvz1**2/(fill1**2)) + 4.000*(bvz2**2/(fill2**2)) + (bvz3**2/(fill3**2)) )
2800 ! PEY = ( (bvy1**2/(fill1**2)) + 4.000*(bvz2**2/(fill2**2)) + (bvz3**2/(fill3**2)) )
2801 ! PEZ = ( (bvz1**2/(fill1**2)) + 4.000*(bvz2**2/(fill2**2)) + (bvz3**2/(fill3**2)) )
2802 ! PE = PE + hss*( PEX + PEY + PEZ )/3.000
2803 !-----
2804 ! computation of curvature line Integral along segment 2-4
2805 !-----
2806 bvx1 = 0.000
2807 bvy1 = 0.000
2808 bvz1 = 0.000
2809 bvx2 = 0.000
2810 bvy2 = 0.000
2811 bvz2 = 0.000
2812 bvx3 = 0.000
2813 bvy3 = 0.000
2814 bvz3 = 0.000
2815 yvx1 = 0.000
2816 yvy1 = 0.000
2817 yvz1 = 0.000
2818 yvx2 = 0.000
2819 yvy2 = 0.000
2820 yvz2 = 0.000
2821 yvx3 = 0.000
2822 yvy3 = 0.000
2823 yvz3 = 0.000
2824 !-----
2825 Dx0x = p(14,1) - p(12,1)
2826 Dy0x = p(14,2) - p(12,2)
2827 Dz0x = p(14,3) - p(12,3)
-----
2829 px2 = p(12,1) + 0.500*DxDx
2830 py2 = p(12,2) + 0.500*DyDx
2831 pz2 = p(12,3) + 0.500*DzDx
2832 !-----
2833 hss = DSQRT(DxDx**2 +DyDx**2 +DzDx**2)
2834 Dx0x = Dx0x/hss
2835 Dy0x = Dy0x/hss
2836 Dz0x = Dz0x/hss
2837 !-----
2838 yvx1 = p(12,1) - x00
2839 yvy1 = p(12,2) - y00
2840 yvz1 = p(12,3) - z00
2841 fill1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
2842 !-----

```

```

2843      bvx1 = (DyDx*yvz1 - DzDx*yvy1)
2844      bvy1 = (DzDx*yvx1 - DxDx*yvz1)
2845      bvz1 = (DxDx*yvy1 - DyDx*yvx1)
-----
2847      yvx2 = px2 - x00
2848      yvy2 = py2 - y00
2849      yvz2 = pz2 - z00
2850      fil2 = DSQRT(yvx2**2 + yvy2**2 + yvz2**2)
-----
2852      bvx2 = (DyDx*yvz2 - DzDx*yvy2)
2853      bvy2 = (DzDx*yvx2 - DxDx*yvz2)
2854      bvz2 = (DxDx*yvy2 - DyDx*yvx2)
-----
2856      yvx3 = p(14,1) - x00
2857      yvy3 = p(14,2) - y00
2858      yvz3 = p(14,3) - z00
2859      fil3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
-----
2861      bvx3 = (DyDx*yvz3 - DzDx*yvy3)
2862      bvy3 = (DzDx*yvx3 - DxDx*yvz3)
2863      bvz3 = (DxDx*yvy3 - DyDx*yvx3)
-----
2864      QExx = QExx + hss*( - (bvz1*(yvx1/fil1**3)) - 4.000*(bvz2*(yvx2/fil2**3)) - (bvz3*(yvx3/fil3**3)) )/3.
2865      000
2866      QExy = QExy + hss*( - (bvz1*(yvx1/fil1**3)) - 4.000*(bvz2*(yvx2/fil2**3)) - (bvz3*(yvx3/fil3**3)) )/3.
2867      000
2868      QExz = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - 4.000*(bvz2*(yvx2/fil2**3)) - (bvz3*(yvx3/fil3**3)) )/3.
2869      000
2870      QEyx = QEyx + hss*( - (bvz1*(yvy1/fil1**3)) - 4.000*(bvz2*(yvy2/fil2**3)) - (bvz3*(yvy3/fil3**3)) )/3.
2871      000
2872      QEyz = QEyz + hss*( - (bvz1*(yvy1/fil1**3)) - 4.000*(bvz2*(yvy2/fil2**3)) - (bvz3*(yvy3/fil3**3)) )/3.
2873      000
2874      QEzy = QEzy + hss*( - (bvz1*(yvy1/fil1**3)) - 4.000*(bvz2*(yvy2/fil2**3)) - (bvz3*(yvy3/fil3**3)) )/3.
2875      000
2876      QEzz = QEzz + hss*( - (bvz1*(yvy1/fil1**3)) - 4.000*(bvz2*(yvy2/fil2**3)) - (bvz3*(yvy3/fil3**3)) )/3.
2877      000
2878      ia1 = (bvx1 + bvz2 + bvz3)/3.000
2879      ia2 = (bvy1 + bvy2 + bvy3)/3.000
2880      ia3 = (bvz1 + bvz2 + bvz3)/3.000
2881      imodx0= DSQRT(a1**2+a2**2+a3**2)
2882      ia1 = a1/modx0
2883      ia2 = a2/modx0
2884      ia3 = a3/modx0
2885      ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
2886      ay2 = (a1*yvx2 + a2*yvy2 + a3*yvz2)
2887      ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
2888      lby1 = DSQRT(( fil1*(fil1+ay1) + 0.3 )**2)
2889      lby2 = DSQRT(( fil2*(fil2+ay2) + 0.3 )**2)
2890      lby3 = DSQRT(( fil3*(fil3+ay3) + 0.3 )**2)
2891      lby1 = DSQRT(( fil1*(fil1+ay1) )**2)
2892      lby2 = DSQRT(( fil2*(fil2+ay2) )**2)
2893      lby3 = DSQRT(( fil3*(fil3+ay3) )**2)
2894      lwrite(9,*) by1
2895      lwrite(9,*) by2
2896      lwrite(9,*) by3
2897      PEx = ( (a1*bvx1/by1) + 4.000*(a2*bvx2/by2) + (a3*bvx3/by3) )*a1
2898      PEy = ( (a1*bvy1/by1) + 4.000*(a2*bvy2/by2) + (a3*bvy3/by3) )*a2
2899      PEz = ( (a1*bvz1/by1) + 4.000*(a2*bvz2/by2) + (a3*bvz3/by3) )*a3
2900      lay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
2901      lay2 = (a1*yvx2 + a2*yvy2 + a3*yvz2)
2902      lay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
2903      lby1 = fil1*(fil1+ay1)
2904      lby2 = fil2*(fil2+ay2)
2905      lby3 = fil3*(fil3+ay3)
2906      lPEX = ( (bvz1/by1) + 4.000*(bvz2/by2) + (bvz3/by3) )*a1
2907      lPEY = ( (bvz1/by1) + 4.000*(bvz2/by2) + (bvz3/by3) )*a2
2908      lPEZ = ( (bvz1/by1) + 4.000*(bvz2/by2) + (bvz3/by3) )*a3

```

```

2906 !      PEx = ( (bvz1**2/(fil1**2)) + 4.000*(bvz2**2/(fil2**2)) + (bvz3**2/(fil3**2)) )
2907 !      PEy = ( (bvz1**2/(fil1**2)) + 4.000*(bvz2**2/(fil2**2)) + (bvz3**2/(fil3**2)) )
2908 !      PEz = ( (bvz1**2/(fil1**2)) + 4.000*(bvz2**2/(fil2**2)) + (bvz3**2/(fil3**2)) )
2909      PE = PE + hss*( PEx + PEy + PEz )/3.000
-----
2911 ! computation of curvature line integral along segment 4-1
2912 !
2913      bvx1 = 0.000
2914      bvy1 = 0.000
2915      bvz1 = 0.000
2916      bvx2 = 0.000
2917      bvy2 = 0.000
2918      bvz2 = 0.000
2919      bvx3 = 0.000
2920      bvy3 = 0.000
2921      bvz3 = 0.000
2922      yvx1 = 0.000
2923      yvy1 = 0.000
2924      yvz1 = 0.000
2925      yvx2 = 0.000
2926      yvy2 = 0.000
2927      yvz2 = 0.000
2928      yvx3 = 0.000
2929      yvy3 = 0.000
2930      yvz3 = 0.000
2931 !
2932      DxDx = p(11,1) - p(14,1)
2933      DyDx = p(11,2) - p(14,2)
2934      DzDx = p(11,3) - p(14,3)
2935 !
2936      px2 = p(14,1) + 0.500*DxDx
2937      py2 = p(14,2) + 0.500*DyDx
2938      pz2 = p(14,3) + 0.500*DzDx
2939 !
2940      hss = DSQRT(DxDx**2 + DyDx**2 + DzDx**2)
2941      DxDx = DxDx/hss
2942      DyDx = DyDx/hss
2943      DzDx = DzDx/hss
2944 !
2945      yvx1 = p(14,1) - x00
2946      yvy1 = p(14,2) - y00
2947      yvz1 = p(14,3) - z00
2948      fil1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
2949 !
2950      bvx1 = (DyDx*yvz1 - DzDx*yvy1)
2951      bvy1 = (DzDx*yvx1 - DxDx*yvz1)
2952      bvz1 = (DxDx*yvy1 - DyDx*yvx1)
2953 !
2954      yvx2 = px2 - x00
2955      yvy2 = py2 - y00
2956      yvz2 = pz2 - z00
2957      fil2 = DSQRT(yvx2**2 + yvy2**2 + yvz2**2)
2958 !
2959      bvx2 = (DyDx*yvz2 - DzDx*yvy2)
2960      bvy2 = (DzDx*yvx2 - DxDx*yvz2)
2961      bvz2 = (DxDx*yvy2 - DyDx*yvx2)
2962 !
2963      yvx3 = p(11,1) - x00
2964      yvy3 = p(11,2) - y00
2965      yvz3 = p(11,3) - z00
2966      fil3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
2967 !
2968      bvx3 = (DyDx*yvz3 - DzDx*yvy3)
2969      bvy3 = (DzDx*yvx3 - DxDx*yvz3)
2970      bvz3 = (DxDx*yvy3 - DyDx*yvx3)
2971 !
2972      QExx = QExx + hss*( - (bvz1*(yvx1/fil1**3)) - 4.000*(bvz2*(yvx2/fil2**3)) - (bvz3*(yvx3/fil3**3)) )/3.
2973      000
2974      QExy = QExy + hss*( - (bvz1*(yvx1/fil1**3)) - 4.000*(bvz2*(yvx2/fil2**3)) - (bvz3*(yvx3/fil3**3)) )/3.
2975      000
2976      QExz = QExz + hss*( - (bvz1*(yvx1/fil1**3)) - 4.000*(bvz2*(yvx2/fil2**3)) - (bvz3*(yvx3/fil3**3)) )/3.
2977      000

```

```

2975 QExx = QExx + hss*( - (bv1*(yv1/fil1**3)) - 4.000*(bv2*(yv2/fil2**3)) - (bv3*(yv3/fil3**3)) )/3.
000
2976 QEyy = QEyy + hss*( - (bv1*(yv1/fil1**3)) - 4.000*(bv2*(yv2/fil2**3)) - (bv3*(yv3/fil3**3)) )/3.
000
2977 QEyz = QEyz + hss*( - (bv21*(yv1/fil1**3)) - 4.000*(bv2*(yv2/fil2**3)) - (bv23*(yv3/fil3**3)) )/3.
000
2978 QExz = QExz + hss*( - (bv1*(yv1/fil1**3)) - 4.000*(bv2*(yv2/fil2**3)) - (bv3*(yv3/fil3**3)) )/3.
000
2979 QEzy = QEzy + hss*( - (bv1*(yv1/fil1**3)) - 4.000*(bv2*(yv2/fil2**3)) - (bv3*(yv3/fil3**3)) )/3.
000
2980 QEzz = QEzz + hss*( - (bv21*(yv1/fil1**3)) - 4.000*(bv2*(yv2/fil2**3)) - (bv23*(yv3/fil3**3)) )/3.
000
2981
2982 !a1 = (bv1 + bv2 + bv3)/3.000
2983 !a2 = (bv1 + bv2 + bv3)/3.000
2984 !a3 = (bv21 + bv22 + bv23)/3.000
2985 !modx0= DSQRT(a1**2+a2**2+a3**2)
2986 !a1 = a1/modx0
2987 !a2 = a2/modx0
2988 !a3 = a3/modx0
2989 !ay1 = (a1*yv1 + a2*yv2 + a3*yv3)
2990 !ay2 = (a1*yv2 + a2*yv3 + a3*yv1)
2991 !ay3 = (a1*yv3 + a2*yv1 + a3*yv2)
2992 !by1 = DSQRT(( fil1*(fil1+ay1) + 0.3 )**2)
2993 !by2 = DSQRT(( fil2*(fil2+ay2) + 0.3 )**2)
2994 !by3 = DSQRT(( fil3*(fil3+ay3) + 0.3 )**2)
2995 !by1 = DSQRT(( fil1*(fil1+ay1) )**2)
2996 !by2 = DSQRT(( fil2*(fil2+ay2) )**2)
2997 !by3 = DSQRT(( fil3*(fil3+ay3) )**2)
2998 !WRITE (9,*) by1
2999 !WRITE (9,*) by2
3000 !WRITE (9,*) by3
3001 !PEX = ( (a1*bv1/by1) + 4.000*(a2*bv2/by2) + (a3*bv3/by3) )*a1
3002 !PEY = ( (a1*bv1/by1) + 4.000*(a2*bv2/by2) + (a3*bv3/by3) )*a2
3003 !PEZ = ( (a1*bv1/by1) + 4.000*(a2*bv2/by2) + (a3*bv3/by3) )*a3
3004 !lay1 = (a1*yv1 + a2*yv2 + a3*yv3)
3005 !lay2 = (a1*yv2 + a2*yv3 + a3*yv1)
3006 !lay3 = (a1*yv3 + a2*yv1 + a3*yv2)
3007 !by1 = fil1*(fil1+ay1)
3008 !by2 = fil2*(fil2+ay2)
3009 !by3 = fil3*(fil3+ay3)
3010 !PEX = ( (bv1/by1) + 4.000*(bv2/by2) + (bv3/by3) )*a1
3011 !PEY = ( (bv1/by1) + 4.000*(bv2/by2) + (bv3/by3) )*a2
3012 !PEZ = ( (bv1/by1) + 4.000*(bv2/by2) + (bv3/by3) )*a3
3013 ! PEX = ( (bv1**2/(fil1**2)) + 4.000*(bv2**2/(fil2**2)) + (bv3**2/(fil3**2)) )
3014 ! PEY = ( (bv1**2/(fil1**2)) + 4.000*(bv2**2/(fil2**2)) + (bv3**2/(fil3**2)) )
3015 ! PEZ = ( (bv1**2/(fil1**2)) + 4.000*(bv2**2/(fil2**2)) + (bv3**2/(fil3**2)) )
3016 ! PE = PE + hss*( PEX + PEY + PEZ )/3.000
3017 !
3018 !
3019 !TExx = 2.000*( -QExx + cpi -PE )
3020 !TEyy = 2.000*( -QEyy )
3021 !TEzz = 2.000*( -QEzz )
3022 !TEyx = 2.000*( -QExy )
3023 !TEyy = 2.000*( -QEyy + cpi -PE )
3024 !TEyz = 2.000*( -QEyz )
3025 !TEzx = 2.000*( -QExz )
3026 !TEzy = 2.000*( -QEzy )
3027 !TEzz = 2.000*( -QEzz + cpi -PE )
3028 !
3029 !END SUBROUTINE intr_lin_sing_s6
3030 !
3031 !SUBROUTINE intr_lin_sing_s7(x10, y10, z10, &
3032 ! & k, &
3033 ! & TExx, TEyy, TEzz, &
3034 ! & TEyx, TEyz, &
3035 ! & TEzx, TEzy, TEzz )
3036 !
3037 ! This subroutine is a new version stokeslet Subroutine.
3038 ! [compute:
3039 ! *The value of the Stokeslet over each singular element
3040 ! Now, (March/ 09 / 2015) this subroutine was made.

```

```

3041 !
3042 !USE Mod_Nodal_Interp
3043 !USE Mod_SharedVars, ONLY: DBL, p, ne, n, mbe, ULog, pi, eps, x0, y0, z0
3044 !
3045 !IMPLICIT NONE
3046 !
3047 ! Variables
3048 !
3049 !REAL (KIND = DBL), INTENT(IN) :: x10, y10, z10 !singularity coordinates
3050 !INTEGER, INTENT(IN) :: k !number of element
3051 !REAL (KIND = DBL), INTENT(OUT) :: TExx, TEyy, TEzz !value of stresslet in the singular element
3052 !REAL (KIND = DBL), INTENT(OUT) :: TEyx, TEzy, TEzx !value of stresslet in the singular element
3053 !REAL (KIND = DBL), INTENT(OUT) :: TEzx, TEzy, TEzz !value of stresslet in the singular element
3054 !
3055 ! Variables inside the subroutine
3056 !
3057 !INTEGER :: i, j !counters
3058 !INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
3059 !
3060 !REAL (KIND = DBL) :: cf, fil1, fil2, fil3 !integration weigh coefficients
3061 !REAL (KIND = DBL) :: DxDx, DyDx, DzDx !tangential vector around the triangle (xi,eta)
3062 !REAL (KIND = DBL) :: hss !weight
3063 !REAL (KIND = DBL) :: cpi !selection to add the solid angle
3064 !REAL (KIND = DBL) :: modx0 !weight
3065 !REAL (KIND = DBL) :: bv1, bv11, bv21, & !vectoreial product
3066 ! & bv22, bv23, bv24, &
3067 ! & bv33, bv31, bv32
3068 !REAL (KIND = DBL) :: QExx, QEyy, QEzz, & !liq tensor
3069 ! & QEyx, QEzy, QEzx, &
3070 ! & QEzx, QEzy, QEzz
3071 !REAL (KIND = DBL) :: PEX, PEY, PEz, PE !lp tensor
3072 !REAL (KIND = DBL) :: yv1, yv2, yv3 !vector y1
3073 !REAL (KIND = DBL) :: yv4, yv5, yv6 !vector y2
3074 !REAL (KIND = DBL) :: yv7, yv8, yv9 !vector y3
3075 !REAL (KIND = DBL) :: yv10, yv11, yv12 !vector y4
3076 !REAL (KIND = DBL) :: yv13, yv14, yv15 !vector y5
3077 !REAL (KIND = DBL) :: yv16, yv17, yv18 !vector y6
3078 !REAL (KIND = DBL) :: zvx1, zvy1, zvz1 !vector (21-x0)
3079 !REAL (KIND = DBL) :: zvx2, zvy2, zvz2 !vector (22-x0)
3080 !REAL (KIND = DBL) :: zvx3, zvy3, zvz3 !vector (23-x0)
3081 !REAL (KIND = DBL) :: zvx4, zvy4, zvz4 !vector (24-x0)
3082 !REAL (KIND = DBL) :: zvx5, zvy5, zvz5 !vector (25-x0)
3083 !REAL (KIND = DBL) :: zvx6, zvy6, zvz6 !vector (26-x0)
3084 !REAL (KIND = DBL) :: zcx1, zcy1, zcz1 !vector z1 ^ zi+1
3085 !REAL (KIND = DBL) :: zcx2, zcy2, zcz2 !vector z1 ^ zi+1
3086 !REAL (KIND = DBL) :: zcx3, zcy3, zcz3 !vector z1 ^ zi+1
3087 !REAL (KIND = DBL) :: zcx4, zcy4, zcz4 !vector z1 ^ zi+1
3088 !REAL (KIND = DBL) :: zcx5, zcy5, zcz5 !vector z1 ^ zi+1
3089 !REAL (KIND = DBL) :: zcx6, zcy6, zcz6 !vector z1 ^ zi+1
3090 !REAL (KIND = DBL) :: zd1, zd2, zd3, zd4, zd5, zd6 !vector z1 . zi+1
3091 !REAL (KIND = DBL) :: pz2, py2, pz2 !pressure vector
3092 !REAL (KIND = DBL) :: a1, a2, a3 !vector a
3093 !REAL (KIND = DBL) :: a11, a12, a13, a14, a15, a16 !alpha angle between zi and zi+1 from x0
3094 !REAL (KIND = DBL) :: th1, th2, th3, th4, th5, th6 !theta angle between vector a and mi vector
3095 !REAL (KIND = DBL) :: e1, e2, e3, e4, e5, e6 !constant
3096 !REAL (KIND = DBL) :: f1, f2, f3, f4, f5, f6 !constant
3097 !REAL (KIND = DBL) :: g1, g2, g3, g4, g5, g6 !constant
3098 !REAL (KIND = DBL) :: ay1, ay2, ay3 !vector dot product a*y1, a*y2
3099 !REAL (KIND = DBL) :: by1, by2, by3 !product yi*(yi+a.yi)
3100 !
3101 ! Initialize
3102 !
3103 !TExx = 0.000
3104 !TEyy = 0.000
3105 !TEzz = 0.000
3106 !TEyx = 0.000
3107 !TEyy = 0.000
3108 !TEyz = 0.000
3109 !TEzx = 0.000
3110 !TEzy = 0.000
3111 !TEzz = 0.000
3112 !

```

```

3113 QExx = 0.000
3114 QExy = 0.000
3115 QExz = 0.000
3116 QEyx = 0.000
3117 QEyy = 0.000
3118 QEyz = 0.000
3119 QEzx = 0.000
3120 QEzy = 0.000
3121 QEzz = 0.000
3122
3123 PEx = 0.000
3124 PEy = 0.000
3125 PEz = 0.000
3126 PE = 0.000
3127
3128 modx0= DSQRT((x0(k)-x10)**2+(y0(k)-y10)**2+(z0(k)-z10)**2)
3129
3130 IF (modx0 >= 1.000 + eps) THEN
3131   cpi = 0.000
3132 ELSE IF (modx0 <= 1.000 + eps) THEN
3133   cpi = -8.000*pi
3134 ELSE
3135   cpi = -4.000*pi
3136 END IF
3137
3138 IF ((ABS(x10-x0(k)) <= eps) .AND. &
3139    & (ABS(y10-y0(k)) <= eps) .AND. &
3140    & (ABS(z10-z0(k)) <= eps)) THEN
3141   a1 = x10
3142   a2 = y10
3143   a3 = z10
3144 ELSE
3145   a1 = x10-x0(k)
3146   a2 = y10-y0(k)
3147   a3 = z10-z0(k)
3148 END IF
3149
3150 !a1 = x00-x0(k)
3151 !a2 = y00-y0(k)
3152 !a3 = z00-z0(k)
3153 modx0= DSQRT(a1**2+a2**2+a3**2)
3154 a1 = a1/modx0
3155 a2 = a2/modx0
3156 a3 = a3/modx0
3157
3158 !OPEN (9,file="TE.out")
3159 !-----
3160 ! vertices of the kth triangle
3161 !-----
3162 i1 = n(k,1)
3163 i2 = n(k,2)
3164 i3 = n(k,3)
3165 i4 = n(k,4)
3166 i5 = n(k,5)
3167 i6 = n(k,6)
3168
3169 cf = 0.000
3170 !-----
3171 ! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
3172 ! rule formula
3173 !-----
3174 ! Firstly, to compute all vectors and angles involved.
3175 !-----
3176 yvx1 = p(i1,1)-x10
3177 yvy1 = p(i1,2)-y10
3178 yvz1 = p(i1,3)-z10
3179 yvx2 = p(i2,1)-x10
3180 yvy2 = p(i2,2)-y10
3181 yvz2 = p(i2,3)-z10
3182 yvx3 = p(i3,1)-x10
3183 yvy3 = p(i3,2)-y10
3184 yvz3 = p(i3,3)-z10

```

```

3185 yvx4 = p(i4,1)-x10
3186 yvy4 = p(i4,2)-y10
3187 yvz4 = p(i4,3)-z10
3188 yvx5 = p(i5,1)-x10
3189 yvy5 = p(i5,2)-y10
3190 yvz5 = p(i5,3)-z10
3191 yvx6 = p(i6,1)-x10
3192 yvy6 = p(i6,2)-y10
3193 yvz6 = p(i6,3)-z10
3194 !-----
3195 zvx1 = yvx1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
3196 zvy1 = yvy1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
3197 zvz1 = yvz1 / DSQRT(yvx1**2 + yvy1**2 + yvz1**2 )
3198 zvx2 = yvx2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
3199 zvy2 = yvy2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
3200 zvz2 = yvz2 / DSQRT(yvx2**2 + yvy2**2 + yvz2**2 )
3201 zvx3 = yvx3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
3202 zvy3 = yvy3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
3203 zvz3 = yvz3 / DSQRT(yvx3**2 + yvy3**2 + yvz3**2 )
3204 zvx4 = yvx4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
3205 zvy4 = yvy4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
3206 zvz4 = yvz4 / DSQRT(yvx4**2 + yvy4**2 + yvz4**2 )
3207 zvx5 = yvx5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
3208 zvy5 = yvy5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
3209 zvz5 = yvz5 / DSQRT(yvx5**2 + yvy5**2 + yvz5**2 )
3210 zvx6 = yvx6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
3211 zvy6 = yvy6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
3212 zvz6 = yvz6 / DSQRT(yvx6**2 + yvy6**2 + yvz6**2 )
3213 !-----
3214 ! Cross products.
3215 !-----
3216 zcx1 = zvy1*zvz6 - zvx1*zvy6
3217 zcy1 = zvx1*zvx6 - zvx1*zvz6
3218 zcz1 = zvx1*zvy6 - zvy1*zvx6
3219 zcx2 = zvy6*zvz3 - zvx6*zvy3
3220 zcy2 = zvx6*zvx3 - zvx6*zvz3
3221 zcz2 = zvx6*zvy3 - zvy6*zvx3
3222 zcx3 = zvy3*zvz5 - zvx3*zvy5
3223 zcy3 = zvx3*zvx5 - zvx3*zvz5
3224 zcz3 = zvx3*zvy5 - zvy3*zvx5
3225 zcx4 = zvy5*zvz2 - zvx5*zvy2
3226 zcy4 = zvx5*zvx2 - zvx5*zvz2
3227 zcz4 = zvx5*zvy2 - zvy5*zvx2
3228 zcx5 = zvy2*zvz4 - zvx2*zvy4
3229 zcy5 = zvx2*zvx4 - zvx2*zvz4
3230 zcz5 = zvx2*zvy4 - zvy2*zvx4
3231 zcx6 = zvy4*zvz1 - zvx4*zvy1
3232 zcy6 = zvx4*zvx1 - zvx4*zvz1
3233 zcz6 = zvx4*zvy1 - zvy4*zvx1
3234 !-----
3235 ! Dot products.
3236 !-----
3237 zd1 = zvx1*zvx6 + zvy1*zvy6 + zvx1*zvy6
3238 zd2 = zvx6*zvx3 + zvy6*zvy3 + zvx6*zvy3
3239 zd3 = zvx3*zvx5 + zvy3*zvy5 + zvx3*zvy5
3240 zd4 = zvx5*zvx2 + zvy5*zvy2 + zvx5*zvy2
3241 zd5 = zvx2*zvx4 + zvy2*zvy4 + zvx2*zvy4
3242 zd6 = zvx4*zvx1 + zvy4*zvy1 + zvx4*zvy1
3243 !-----
3244 ! Angles
3245 !-----
3246 a11 = DACOS( zvx1*zvx6 + zvy1*zvy6 + zvx1*zvy6 )
3247 a12 = DACOS( zvx6*zvx3 + zvy6*zvy3 + zvx6*zvy3 )
3248 a13 = DACOS( zvx3*zvx5 + zvy3*zvy5 + zvx3*zvy5 )
3249 a14 = DACOS( zvx5*zvx2 + zvy5*zvy2 + zvx5*zvy2 )
3250 a15 = DACOS( zvx2*zvx4 + zvy2*zvy4 + zvx2*zvy4 )
3251 a16 = DACOS( zvx4*zvx1 + zvy4*zvy1 + zvx4*zvy1 )
3252 !-----
3253 th1 = DACOS( a1*zvx1 + a2*zvy1 + a3*zvz1 )
3254 th2 = DACOS( a1*zvx6 + a2*zvy6 + a3*zvz6 )
3255 th3 = DACOS( a1*zvx3 + a2*zvy3 + a3*zvz3 )
3256 th4 = DACOS( a1*zvx5 + a2*zvy5 + a3*zvz5 )

```

```

3257 th5 = DACOS( a1*zvx2 + a2*zvy2 + a3*zvz2 )
3258 th6 = DACOS( a1*zvx4 + a2*zvy4 + a3*zvz4 )
3259 ! e and g constants
3261 !-----
3262 e1 = ( DCOS( th2 ) - DCOS( th1 ) * DCOS( a11 ) ) / DSIN( a11 )
3263 e2 = ( DCOS( th3 ) - DCOS( th2 ) * DCOS( a12 ) ) / DSIN( a12 )
3264 e3 = ( DCOS( th4 ) - DCOS( th3 ) * DCOS( a13 ) ) / DSIN( a13 )
3265 e4 = ( DCOS( th5 ) - DCOS( th4 ) * DCOS( a14 ) ) / DSIN( a14 )
3266 e5 = ( DCOS( th6 ) - DCOS( th5 ) * DCOS( a15 ) ) / DSIN( a15 )
3267 e6 = ( DCOS( th1 ) - DCOS( th6 ) * DCOS( a16 ) ) / DSIN( a16 )
3268 f1 = DCOS( th1 )
3269 f2 = DCOS( th2 )
3270 f3 = DCOS( th3 )
3271 f4 = DCOS( th4 )
3272 f5 = DCOS( th5 )
3273 f6 = DCOS( th6 )
3274 g1 = DSQRT( 1.000 - e1**2 - f1**2 )
3275 g2 = DSQRT( 1.000 - e2**2 - f2**2 )
3276 g3 = DSQRT( 1.000 - e3**2 - f3**2 )
3277 g4 = DSQRT( 1.000 - e4**2 - f4**2 )
3278 g5 = DSQRT( 1.000 - e5**2 - f5**2 )
3279 g6 = DSQRT( 1.000 - e6**2 - f6**2 )
3280 !-----
3281 ! computation of curvature line integral along segment 1
3282 !-----
3283 QExx = ( (zvx1 + zvx6)*zcx1 ) / ( 1.000 + zd1 )
3284 QExy = ( (zvx1 + zvx6)*zcy1 ) / ( 1.000 + zd1 )
3285 QExz = ( (zvx1 + zvx6)*zcx1 ) / ( 1.000 + zd1 )
3286 QEyx = ( (zvy1 + zvy6)*zcx1 ) / ( 1.000 + zd1 )
3287 QEyy = ( (zvy1 + zvy6)*zcy1 ) / ( 1.000 + zd1 )
3288 QEyz = ( (zvy1 + zvy6)*zcx1 ) / ( 1.000 + zd1 )
3289 QExz = ( (zvx1 + zvx6)*zcx1 ) / ( 1.000 + zd1 )
3290 QEyz = ( (zvy1 + zvy6)*zcy1 ) / ( 1.000 + zd1 )
3291 QEzz = ( (zvx1 + zvx6)*zcx1 ) / ( 1.000 + zd1 )
3292 !-----
3293 PE = PEX + ( 2.000*a1*zcx1/g1 )*( DATAN(( 1.000 - f1)*DTAN(a11/2) + e1 / g1) - DATAN(e1/g1) )
3294 PEY = PEY + ( 2.000*a2*zcy1/g1 )*( DATAN(( 1.000 - f1)*DTAN(a11/2) + e1 / g1) - DATAN(e1/g1) )
3295 PEZ = PEZ + ( 2.000*a3*zcx1/g1 )*( DATAN(( 1.000 - f1)*DTAN(a11/2) + e1 / g1) - DATAN(e1/g1) )
3296 !-----
3297 ! computation of curvature line integral along segment 6
3298 !-----
3299 QExx = QExx + ( (zvx6 + zvx3)*zcx2 ) / ( 1.000 + zd2 )
3300 QExy = QExy + ( (zvx6 + zvx3)*zcy2 ) / ( 1.000 + zd2 )
3301 QExz = QExz + ( (zvx6 + zvx3)*zcx2 ) / ( 1.000 + zd2 )
3302 QEyx = QEyx + ( (zvy6 + zvy3)*zcx2 ) / ( 1.000 + zd2 )
3303 QEyy = QEyy + ( (zvy6 + zvy3)*zcy2 ) / ( 1.000 + zd2 )
3304 QEyz = QEyz + ( (zvy6 + zvy3)*zcx2 ) / ( 1.000 + zd2 )
3305 QExz = QExz + ( (zvx6 + zvx3)*zcx2 ) / ( 1.000 + zd2 )
3306 QEyz = QEyz + ( (zvx6 + zvx3)*zcy2 ) / ( 1.000 + zd2 )
3307 QEzz = QEzz + ( (zvx6 + zvx3)*zcx2 ) / ( 1.000 + zd2 )
3308 !-----
3309 PEX = PEX + ( 2.000*a1*zcx2/g2 )*( DATAN(( 1.000 - f2)*DTAN(a12/2) + e2 / g2) - DATAN(e2/g2) )
3310 PEY = PEY + ( 2.000*a2*zcy2/g2 )*( DATAN(( 1.000 - f2)*DTAN(a12/2) + e2 / g2) - DATAN(e2/g2) )
3311 PEZ = PEZ + ( 2.000*a3*zcx2/g2 )*( DATAN(( 1.000 - f2)*DTAN(a12/2) + e2 / g2) - DATAN(e2/g2) )
3312 !-----
3313 ! computation of curvature line integral along segment 3
3314 !-----
3315 QExx = QExx + ( (zvx3 + zvx5)*zcx3 ) / ( 1.000 + zd3 )
3316 QExy = QExy + ( (zvx3 + zvx5)*zcy3 ) / ( 1.000 + zd3 )
3317 QExz = QExz + ( (zvx3 + zvx5)*zcx3 ) / ( 1.000 + zd3 )
3318 QEyx = QEyx + ( (zvy3 + zvy5)*zcx3 ) / ( 1.000 + zd3 )
3319 QEyy = QEyy + ( (zvy3 + zvy5)*zcy3 ) / ( 1.000 + zd3 )
3320 QEyz = QEyz + ( (zvy3 + zvy5)*zcx3 ) / ( 1.000 + zd3 )
3321 QExz = QExz + ( (zvx3 + zvx5)*zcx3 ) / ( 1.000 + zd3 )
3322 QEyz = QEyz + ( (zvx3 + zvx5)*zcy3 ) / ( 1.000 + zd3 )
3323 QEzz = QEzz + ( (zvx3 + zvx5)*zcx3 ) / ( 1.000 + zd3 )
3324 !-----
3325 PEX = PEX + ( 2.000*a1*zcx3/g3 )*( DATAN(( 1.000 - f3)*DTAN(a13/3) + e3 / g3) - DATAN(e3/g3) )
3326 PEY = PEY + ( 2.000*a2*zcy3/g3 )*( DATAN(( 1.000 - f3)*DTAN(a13/3) + e3 / g3) - DATAN(e3/g3) )
3327 PEZ = PEZ + ( 2.000*a3*zcx3/g3 )*( DATAN(( 1.000 - f3)*DTAN(a13/3) + e3 / g3) - DATAN(e3/g3) )
3328 !-----

```

```

3329 ! computation of curvature line integral along segment 5
3330 !-----
3331 QExx = QExx + ( (zvx5 + zvx2)*zcx4 ) / ( 1.000 + zd4 )
3332 QExz = QExz + ( (zvx5 + zvx2)*zcy4 ) / ( 1.000 + zd4 )
3333 QExz = QExz + ( (zvx5 + zvx2)*zcx4 ) / ( 1.000 + zd4 )
3334 QEyx = QEyx + ( (zvy5 + zvy2)*zcx4 ) / ( 1.000 + zd4 )
3335 QEyy = QEyy + ( (zvy5 + zvy2)*zcy4 ) / ( 1.000 + zd4 )
3336 QEyz = QEyz + ( (zvy5 + zvy2)*zcx4 ) / ( 1.000 + zd4 )
3337 QExz = QExz + ( (zvx5 + zvx2)*zcx4 ) / ( 1.000 + zd4 )
3338 QEyz = QEyz + ( (zvx5 + zvx2)*zcy4 ) / ( 1.000 + zd4 )
3339 QEzz = QEzz + ( (zvx5 + zvx2)*zcx4 ) / ( 1.000 + zd4 )
3340 !-----
3341 PEX = PEX + ( 2.000*a1*zcx4/g4 )*( DATAN(( 1.000 - f4)*DTAN(a14/2) + e4 / g4) - DATAN(e4/g4) )
3342 PEY = PEY + ( 2.000*a2*zcy4/g4 )*( DATAN(( 1.000 - f4)*DTAN(a14/2) + e4 / g4) - DATAN(e4/g4) )
3343 PEZ = PEZ + ( 2.000*a3*zcx4/g4 )*( DATAN(( 1.000 - f4)*DTAN(a14/2) + e4 / g4) - DATAN(e4/g4) )
3344 !-----
3345 ! computation of curvature line integral along segment 2
3346 !-----
3347 QExx = QExx + ( (zvx2 + zvx4)*zcx5 ) / ( 1.000 + zd5 )
3348 QExy = QExy + ( (zvx2 + zvx4)*zcy5 ) / ( 1.000 + zd5 )
3349 QExz = QExz + ( (zvx2 + zvx4)*zcx5 ) / ( 1.000 + zd5 )
3350 QEyx = QEyx + ( (zvy2 + zvy4)*zcx5 ) / ( 1.000 + zd5 )
3351 QEyy = QEyy + ( (zvy2 + zvy4)*zcy5 ) / ( 1.000 + zd5 )
3352 QEyz = QEyz + ( (zvy2 + zvy4)*zcx5 ) / ( 1.000 + zd5 )
3353 QExz = QExz + ( (zvx2 + zvx4)*zcx5 ) / ( 1.000 + zd5 )
3354 QEyz = QEyz + ( (zvx2 + zvx4)*zcy5 ) / ( 1.000 + zd5 )
3355 QEzz = QEzz + ( (zvx2 + zvx4)*zcx5 ) / ( 1.000 + zd5 )
3356 !-----
3357 PEX = PEX + ( 2.000*a1*zcx5/g5 )*( DATAN(( 1.000 - f5)*DTAN(a15/2) + e5 / g5) - DATAN(e5/g5) )
3358 PEY = PEY + ( 2.000*a2*zcy5/g5 )*( DATAN(( 1.000 - f5)*DTAN(a15/2) + e5 / g5) - DATAN(e5/g5) )
3359 PEZ = PEZ + ( 2.000*a3*zcx5/g5 )*( DATAN(( 1.000 - f5)*DTAN(a15/2) + e5 / g5) - DATAN(e5/g5) )
3360 !-----
3361 ! computation of curvature line integral along segment 4
3362 !-----
3363 QExx = QExx + ( (zvx4 + zvx1)*zcx6 ) / ( 1.000 + zd6 )
3364 QExy = QExy + ( (zvx4 + zvx1)*zcy6 ) / ( 1.000 + zd6 )
3365 QExz = QExz + ( (zvx4 + zvx1)*zcx6 ) / ( 1.000 + zd6 )
3366 QEyx = QEyx + ( (zvy4 + zvy1)*zcx6 ) / ( 1.000 + zd6 )
3367 QEyy = QEyy + ( (zvy4 + zvy1)*zcy6 ) / ( 1.000 + zd6 )
3368 QEyz = QEyz + ( (zvy4 + zvy1)*zcx6 ) / ( 1.000 + zd6 )
3369 QExz = QExz + ( (zvx4 + zvx1)*zcx6 ) / ( 1.000 + zd6 )
3370 QEyz = QEyz + ( (zvx4 + zvx1)*zcy6 ) / ( 1.000 + zd6 )
3371 QEzz = QEzz + ( (zvx4 + zvx1)*zcx6 ) / ( 1.000 + zd6 )
3372 !-----
3373 PEX = PEX + ( 2.000*a1*zcx6/g6 )*( DATAN(( 1.000 - f6)*DTAN(a16/2) + e6 / g6) - DATAN(e6/g6) )
3374 PEY = PEY + ( 2.000*a2*zcy6/g6 )*( DATAN(( 1.000 - f6)*DTAN(a16/2) + e6 / g6) - DATAN(e6/g6) )
3375 PEZ = PEZ + ( 2.000*a3*zcx6/g6 )*( DATAN(( 1.000 - f6)*DTAN(a16/2) + e6 / g6) - DATAN(e6/g6) )
3376 !-----
3377 TExx = 2.000*( QExx + cpi + PE )
3378 TExy = 2.000*( QEYx )
3379 TExz = 2.000*( QExz )
3380 TEyx = 2.000*( QEYx )
3381 TEyy = 2.000*( QEYy + cpi + PE )
3382 TEyz = 2.000*( QEYz )
3383 TExz = 2.000*( QExz )
3384 TEyz = 2.000*( QEYz )
3385 TEzz = 2.000*( QEzz + cpi + PE )
3386 !-----
3387 END SUBROUTINE intr_lin_sing_s7
3388 !-----
3389 SUBROUTINE intr_lin_sing_s8(x0e, y0e, z0e, &
& x00, y00, z00, &
& k, &
& TExx, TExy, TExz, &
& TEyx, TEyy, TEyz, &
& TExz, TEyz, TEzz )
3395 !-----
3396 ! This subroutine is a new version stokeslet Subroutine.
3397 !Compute:
3398 ! *The value of the Stokeslet over each singular element
3399 !Now, (March/ 09 / 2015) this subroutine was made.
3400 !-----

```

```

3401 USE Mod_Nodal_Interp
3402 USE Mod_Prtcl_3D_Geo, ONLY: abc
3403 USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, Ulog, &
& alpha0, beta0, gamma0, &
3405 & arel, crwml, &
3406 & vnx0, vny0, vnz0, &
3407 & ZZ, &
3408 & xiq, etq, wq, pl, eps, x0, y0, z0
3409
3410 IMPLICIT NONE
3411
3412 ! Variables
3413 REAL (KIND = DBL), INTENT(IN) :: x00, y00, z00 !singularity coordinates
3414 REAL (KIND = DBL), INTENT(IN) :: x0e, ye0, z0e !singularity coordinates element
3415 INTEGER, INTENT(IN) :: k !number of element
3416 REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TExz !value of stresslet in the singular element
3417 REAL (KIND = DBL), INTENT(OUT) :: TEyx, TEyy, TEyz !value of stresslet in the singular element
3418 REAL (KIND = DBL), INTENT(OUT) :: TEzx, TEzy, TEzz !value of stresslet in the singular element
3419
3420 ! Variables inside the subroutine
3421
3422 INTEGER :: i, j !counters
3423 INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
3424
3425 REAL (KIND = DBL) :: cf, fill1, fill2, fill3 !integration weigh coefficients
3426 REAL (KIND = DBL) :: hss !length
3427 REAL (KIND = DBL) :: cpi !selection to add the solid angle
3428 REAL (KIND = DBL) :: modx0 !length
3429 REAL (KIND = DBL) :: bvx1, bvy1, bvz1, & !vectoreal product
& bvx2, bvy2, bvz2, &
3431 & bvx3, bvy3, bvz3
3432 REAL (KIND = DBL) :: QExx, QExy, QExz, & !Iq tensor
& QEyx, QEyy, QEyz, &
3434 & QEzx, QEzy, QEzz
3435 REAL (KIND = DBL) :: PEx, PEy, PEz, PE !Iip tensor
3436 REAL (KIND = DBL) :: yx1, yy1, yz1 !vector (y1-x0)
3437 REAL (KIND = DBL) :: yx2, yy2, yz2 !vector (y2-x0)
3438 REAL (KIND = DBL) :: yx3, yy3, yz3 !vector (y3-x0)
3439 REAL (KIND = DBL) :: px2, py2, pz2 !vector a
3440 REAL (KIND = DBL) :: a1, a2, a3 !vector a
3441 REAL (KIND = DBL) :: ay1, ay2, ay3 !vector dot product a*y1, a*y2
3442 REAL (KIND = DBL) :: by1, by2, by3 !product yi*(yi+a.yi)
3443
3444 REAL (KIND = DBL) :: xi, eta !variables of weigh to integrate over a triangle
3445 REAL (KIND = DBL) :: x, y, z !coordinates of the f(x,y,z)= F(xi,eta)
3446 REAL (KIND = DBL) :: DxDx1, DyDx1, DzDx1 !coordinates of the tangential vector over the xi axis
3447 REAL (KIND = DBL) :: DxDeta, DyDeta, DzDeta !coordinates of the tangential vector over the eta axis
3448 REAL (KIND = DBL) :: vnx, vny, vnz !normal vector coordinates of the element
3449 REAL (KIND = DBL) :: hs, xs, es !surface metric on a triangle
3450 REAL (KIND = DBL) :: al, be, ga, alc, bec, gac !integration weigh coefficients
3451 REAL (KIND = DBL) :: fill !integration weigh coefficients
3452 REAL (KIND = DBL), DIMENSION(6) :: xxi, eet !variables of weigh over xi axis in triangle (xi,eta)
3453 REAL (KIND = DBL), DIMENSION(6) :: DxDx, DyDx, DzDx !tangential vector over xi axis in triangle (xi,eta)
3454 REAL (KIND = DBL), DIMENSION(6) :: DxDeta, DyDeta, DzDeta !tangential vector over eta axis in triangle (xi,eta)
3455 REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz !normal vector in triangle (xi,eta)
3456
3457
3458 ! Initialize
3459
3460 TExx = 0.000
3461 TEyx = 0.000
3462 TEzx = 0.000
3463 TEyy = 0.000
3464 TEyz = 0.000
3465 TEzz = 0.000
3466 TEzx = 0.000
3467 TEzy = 0.000
3468 TEzz = 0.000
3469
3470 QExx = 0.000
3471 QExy = 0.000
3472 QExz = 0.000

```

```

3473 QEyx = 0.000
3474 QEyy = 0.000
3475 QEyz = 0.000
3476 QExx = 0.000
3477 QEzy = 0.000
3478 QEzz = 0.000
3479
3480 PEx = 0.000
3481 PEy = 0.000
3482 PEz = 0.000
3483 PE = 0.000
3484
3485 a1 = x0e-x00
3486 a2 = ye0-y00
3487 a3 = z0e-z00
3488 modx0= DSQRT(a1**2+a2**2+a3**2)
3489
3490 IF (modx0 >= 1.000 + eps) THEN
3491 cpi = 0.000
3492 ELSE IF (modx0 <= 1.000 - eps) THEN
3493 cpi = -8.000*pi
3494 ELSE
3495 cpi = -4.000*pi
3496 end if
3497
3498 a1 = x00-x0e
3499 a2 = y00-y0e
3500 a3 = z00-z0e
3501 modx0= DSQRT(a1**2+a2**2+a3**2)
3502
3503 a1 = a1/modx0
3504 a2 = a2/modx0
3505 a3 = a3/modx0
3506
3507 IOPEN (9,file="TE.out")
3508
3509 ! vertices of the kth triangle
3510
3511 i1 = n(k,1)
3512 i2 = n(k,2)
3513 i3 = n(k,3)
3514 i4 = n(k,4)
3515 i5 = n(k,5)
3516 i6 = n(k,6)
3517 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, &
3519 & p(i3,1)-x00, p(i3,2)-y00, p(i3,3)-z00, &
& p(i4,1)-x00, p(i4,2)-y00, p(i4,3)-z00, &
3521 & p(i5,1)-x00, p(i5,2)-y00, p(i5,3)-z00, &
& p(i6,1)-x00, p(i6,2)-y00, p(i6,3)-z00, &
3523 & a1, be, ga)
3524 alc = 1.000-a1
3525 bec = 1.000-be
3526 gac = 1.000-ga
3527
3528 cf = 0.000
3529
3530 ! compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
3531 ! rule formula
3532
3533 DO i = 1, 13
3534 x1 = xiq(i)
3535 eta = etq(i)
3536 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, &
3538 & p(i3,1)-x00, p(i3,2)-y00, p(i3,3)-z00, &
& p(i4,1)-x00, p(i4,2)-y00, p(i4,3)-z00, &
3540 & p(i5,1)-x00, p(i5,2)-y00, p(i5,3)-z00, &
& p(i6,1)-x00, p(i6,2)-y00, p(i6,3)-z00, &
3542 & a1, be, g0, &
& xi, eta, &
3544 & x, y, z, &

```

```

3545      &      DxDx1,  DyDx1,  DzDx1,  &
3546      &      DxDet,  DyDet,  DzDet,  &
3547      &      vnx,    vny,    vnz,    &
3548      &      hs)
3549      cf = hs*wq(1)
3550      END DO
3551      cf=0.5D0*cf
3552      |-----|
3553      | computation of curvature line integral along segment 1-4
3554      | formula (4.2.10) of Pozrikidis (1997)
3555      |-----|
3556      xxi(1) = 0.000
3557      eet(1) = 0.000
3558      xxi(2) = 1.000
3559      eet(2) = 0.000
3560      xxi(3) = 0.000
3561      eet(3) = 1.000
3562      xxi(4) = al
3563      eet(4) = 0.000
3564      xxi(5) = ga
3565      eet(5) = gac
3566      xxi(6) = 0.000
3567      eet(6) = be
3568      DO i = 1, 6
3569      xi = xxi(i)
3570      eta = eet(i)
3571      CALL interp_p(p(11,1)-x00, p(11,2)-y00, p(11,3)-z00, &
3572      &      p(12,1)-x00, p(12,2)-y00, p(12,3)-z00, &
3573      &      p(13,1)-x00, p(13,2)-y00, p(13,3)-z00, &
3574      &      p(14,1)-x00, p(14,2)-y00, p(14,3)-z00, &
3575      &      p(15,1)-x00, p(15,2)-y00, p(15,3)-z00, &
3576      &      p(16,1)-x00, p(16,2)-y00, p(16,3)-z00, &
3577      &      al,    be,    ga,    &
3578      &      xi,    eta,    &
3579      &      x,    y,    z,    &
3580      &      DxDx(i), DyDx(i), DzDx(i), &
3581      &      DDxDet(i), DyDe(i), DZDe(i), &
3582      &      vx(i), vy(i), vz(i), &
3583      &      hs)
3584      END DO
3585      |-----|
3586      bvx1 = 0.000
3587      bvy1 = 0.000
3588      bvz1 = 0.000
3589      bvx2 = 0.000
3590      bvy2 = 0.000
3591      bvz2 = 0.000
3592      bvx3 = 0.000
3593      bvy3 = 0.000
3594      bvz3 = 0.000
3595      yvx1 = 0.000
3596      yvy1 = 0.000
3597      yvz1 = 0.000
3598      yvx2 = 0.000
3599      yvy2 = 0.000
3600      yvz2 = 0.000
3601      yvx3 = 0.000
3602      yvy3 = 0.000
3603      yvz3 = 0.000
3604      fill1 = 0.000
3605      fill2 = 0.000
3606      fill3 = 0.000
3607      ay1 = 0.000
3608      ay2 = 0.000
3609      ay3 = 0.000
3610      |-----|
3611      | computation of curvature line integral along segment 1-4-2
3612      |-----|
3613      yvx1 = p(11,1) - x00
3614      yvy1 = p(11,2) - y00
3615      yvz1 = p(11,3) - z00
3616      fill1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)

```

```

3617      |-----|
3618      yvx2 = p(14,1) - x00
3619      yvy2 = p(14,2) - y00
3620      yvz2 = p(14,3) - z00
3621      fill2 = DSQRT(yvx2**2 + yvy2**2 + yvz2**2)
3622      |-----|
3623      yvx3 = p(12,1) - x00
3624      yvy3 = p(12,2) - y00
3625      yvz3 = p(12,3) - z00
3626      fill3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
3627      |-----|
3628      ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
3629      ay2 = (a1*yvx2 + a2*yvy2 + a3*yvz2)
3630      ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
3631      |-----|
3632      by1 = ( fill1*(fill1+ay1) )
3633      by2 = ( fill2*(fill2+ay2) )
3634      by3 = ( fill3*(fill3+ay3) )
3635      |-----|
3636      bvx1 = yvy1*DzDx(1)-yvz1*DyDx(1)
3637      bvy1 = yvz1*DxDx(1)-yvx1*DzDx(1)
3638      bvz1 = yvx1*DyDx(1)-yvy1*DxDx(1)
3639      |-----|
3640      bvx2 = yvy2*DzDx(4)-yvz2*DyDx(4)
3641      bvy2 = yvz2*DxDx(4)-yvx2*DzDx(4)
3642      bvz2 = yvx2*DyDx(4)-yvy2*DxDx(4)
3643      |-----|
3644      bvx3 = yvy3*DzDx(2)-yvz3*DyDx(2)
3645      bvy3 = yvz3*DxDx(2)-yvx3*DzDx(2)
3646      bvz3 = yvx3*DyDx(2)-yvy3*DxDx(2)
3647      |-----|
3648      QExx = ((a1*yvx1*bvx1)/(fill1**3)) + ((yvx2*bvx2)/(fill2**3)) + ((a1c*yvx3*bvx3)/(fill3**3))
3649      QExy = ((a1*yvx1*bvy1)/(fill1**3)) + ((yvx2*bvy2)/(fill2**3)) + ((a1c*yvx3*bvy3)/(fill3**3))
3650      QExz = ((a1*yvx1*bvz1)/(fill1**3)) + ((yvx2*bvz2)/(fill2**3)) + ((a1c*yvx3*bvz3)/(fill3**3))
3651      QEyx = ((a1*yvy1*bvx1)/(fill1**3)) + ((yvy2*bvx2)/(fill2**3)) + ((a1c*yvy3*bvx3)/(fill3**3))
3652      QEyy = ((a1*yvy1*bvy1)/(fill1**3)) + ((yvy2*bvy2)/(fill2**3)) + ((a1c*yvy3*bvy3)/(fill3**3))
3653      QEyz = ((a1*yvy1*bvz1)/(fill1**3)) + ((yvy2*bvz2)/(fill2**3)) + ((a1c*yvy3*bvz3)/(fill3**3))
3654      QEzx = ((a1*yvz1*bvx1)/(fill1**3)) + ((yvz2*bvx2)/(fill2**3)) + ((a1c*yvz3*bvx3)/(fill3**3))
3655      QEzy = ((a1*yvz1*bvy1)/(fill1**3)) + ((yvz2*bvy2)/(fill2**3)) + ((a1c*yvz3*bvy3)/(fill3**3))
3656      QEzz = ((a1*yvz1*bvz1)/(fill1**3)) + ((yvz2*bvz2)/(fill2**3)) + ((a1c*yvz3*bvz3)/(fill3**3))
3657      |-----|
3658      PEx = ((a1*a1*bvx1)/(by1)) + ((a1*bvx2)/(by2)) + ((a1c*a1*bvx3)/(by3))
3659      PEy = ((a1*a2*bvy1)/(by1)) + ((a2*bvy2)/(by2)) + ((a1c*a2*bvy3)/(by3))
3660      PEz = ((a1*a3*bvz1)/(by1)) + ((a3*bvz2)/(by2)) + ((a1c*a3*bvz3)/(by3))
3661      |-----|
3662      | computation of curvature line integral along segment 2-5-3
3663      |-----|
3664      yvx1 = p(12,1) - x00
3665      yvy1 = p(12,2) - y00
3666      yvz1 = p(12,3) - z00
3667      fill1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
3668      |-----|
3669      yvx2 = p(15,1) - x00
3670      yvy2 = p(15,2) - y00
3671      yvz2 = p(15,3) - z00
3672      fill2 = DSQRT(yvx2**2 + yvy2**2 + yvz2**2)
3673      |-----|
3674      yvx3 = p(13,1) - x00
3675      yvy3 = p(13,2) - y00
3676      yvz3 = p(13,3) - z00
3677      fill3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
3678      |-----|
3679      ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
3680      ay2 = (a1*yvx2 + a2*yvy2 + a3*yvz2)
3681      ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
3682      |-----|
3683      by1 = ( fill1*(fill1+ay1) )
3684      by2 = ( fill2*(fill2+ay2) )
3685      by3 = ( fill3*(fill3+ay3) )
3686      |-----|
3687      bvx1 = yvy1*DzDx(2)-yvz1*DyDx(2)
3688      bvy1 = yvz1*DxDx(2)-yvx1*DzDx(2)

```



```

3689   bvz1 = yvx1*DyDx(2)-yyv1*DxDx(2)
3690
3691   bvz2 = yvz2*DzDx(5)-yvz2*DzDx(5)
3692   bvz2 = yvz2*DzDx(5)-yvz2*DzDx(5)
3693   bvz2 = yvz2*DzDx(5)-yvz2*DzDx(5)
3694
3695   bvz3 = yvy3*DzDx(3)-yvz3*DzDx(3)
3696   bvz3 = yvz3*DzDx(3)-yvz3*DzDx(3)
3697   bvz3 = yvz3*DzDx(3)-yvz3*DzDx(3)
3698
3699   QExx = QExx - ((gac*yvx1*bvz1)/(f11**3)) - ((yvx2*bvz2)/(f112**3)) - ((ga*yvx3*bvz3)/(f113**3))
3700   QExy = QExy - ((gac*yvx1*bv1)/(f11**3)) - ((yvx2*bv2)/(f112**3)) - ((ga*yvx3*bv3)/(f113**3))
3701   QExz = QExz - ((gac*yvx1*bvz1)/(f11**3)) - ((yvx2*bv2)/(f112**3)) - ((ga*yvx3*bvz3)/(f113**3))
3702   QEyx = QEyx - ((gac*yvy1*bvz1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((ga*yvy3*bvz3)/(f113**3))
3703   QEyy = QEyy - ((gac*yvy1*bv1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((ga*yvy3*bv3)/(f113**3))
3704   QEyz = QEyz - ((gac*yvy1*bvz1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((ga*yvy3*bvz3)/(f113**3))
3705   QEzx = QEzx - ((gac*yvz1*bvz1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((ga*yvz3*bvz3)/(f113**3))
3706   QEzy = QEzy - ((gac*yvz1*bv1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((ga*yvz3*bv3)/(f113**3))
3707   QEzz = QEzz - ((gac*yvz1*bvz1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((ga*yvz3*bvz3)/(f113**3))
3708
3709   IQExx = QExx + ((gac*yvx1*bvz1)/(f11**3)) + ((yvx2*bvz2)/(f112**3)) + ((ga*yvx3*bvz3)/(f113**3))
3710   IQExy = QExy + ((gac*yvx1*bv1)/(f11**3)) + ((yvx2*bv2)/(f112**3)) + ((ga*yvx3*bv3)/(f113**3))
3711   IQExz = QExz + ((gac*yvx1*bvz1)/(f11**3)) + ((yvx2*bv2)/(f112**3)) + ((ga*yvx3*bvz3)/(f113**3))
3712   IQEyx = QEyx + ((gac*yvy1*bvz1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((ga*yvy3*bvz3)/(f113**3))
3713   IQEyy = QEyy + ((gac*yvy1*bv1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((ga*yvy3*bv3)/(f113**3))
3714   IQEyz = QEyz + ((gac*yvy1*bvz1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((ga*yvy3*bvz3)/(f113**3))
3715   IQEzx = QEzx + ((gac*yvz1*bvz1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((ga*yvz3*bvz3)/(f113**3))
3716   IQEzy = QEzy + ((gac*yvz1*bv1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((ga*yvz3*bv3)/(f113**3))
3717   IQEzz = QEzz + ((gac*yvz1*bvz1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((ga*yvz3*bvz3)/(f113**3))
3718
3719   PEx = PEx - ((gac*a1*bvz1)/(by1)) - ((a1*bvz2)/(by2)) - ((ga*a1*bvz3)/(by3))
3720   PEy = PEy - ((gac*a2*bv1)/(by1)) - ((a2*bv2)/(by2)) - ((ga*a2*bv3)/(by3))
3721   PEz = PEz - ((gac*a3*bv1)/(by1)) - ((a3*bv2)/(by2)) - ((ga*a3*bv3)/(by3))
3722
3723   IPEX = PEx + ((gac*a1*bvz1)/(by1)) + ((a1*bvz2)/(by2)) + ((ga*a1*bvz3)/(by3))
3724   IPEY = PEy + ((gac*a2*bv1)/(by1)) + ((a2*bv2)/(by2)) + ((ga*a2*bv3)/(by3))
3725   IPEZ = PEz + ((gac*a3*bv1)/(by1)) + ((a3*bv2)/(by2)) + ((ga*a3*bv3)/(by3))
3726
3727   bvz1 = yvz1*DzDe(2)-yvz1*DzDe(2)
3728   bvz1 = yvz1*DzDe(2)-yvz1*DzDe(2)
3729   bvz1 = yvz1*DzDe(2)-yvz1*DzDe(2)
3730
3731   bvz2 = yvz2*DzDe(5)-yvz2*DzDe(5)
3732   bvz2 = yvz2*DzDe(5)-yvz2*DzDe(5)
3733   bvz2 = yvz2*DzDe(5)-yvz2*DzDe(5)
3734
3735   bvz3 = yvz3*DzDe(3)-yvz3*DzDe(3)
3736   bvz3 = yvz3*DzDe(3)-yvz3*DzDe(3)
3737   bvz3 = yvz3*DzDe(3)-yvz3*DzDe(3)
3738
3739   QExx = QExx + ((gac*yvx1*bvz1)/(f11**3)) + ((yvx2*bvz2)/(f112**3)) + ((ga*yvx3*bvz3)/(f113**3))
3740   QExy = QExy + ((gac*yvx1*bv1)/(f11**3)) + ((yvx2*bv2)/(f112**3)) + ((ga*yvx3*bv3)/(f113**3))
3741   QExz = QExz + ((gac*yvx1*bvz1)/(f11**3)) + ((yvx2*bv2)/(f112**3)) + ((ga*yvx3*bvz3)/(f113**3))
3742   QEyx = QEyx + ((gac*yvy1*bvz1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((ga*yvy3*bvz3)/(f113**3))
3743   QEyy = QEyy + ((gac*yvy1*bv1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((ga*yvy3*bv3)/(f113**3))
3744   QEyz = QEyz + ((gac*yvy1*bvz1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((ga*yvy3*bvz3)/(f113**3))
3745   QEzx = QEzx + ((gac*yvz1*bvz1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((ga*yvz3*bvz3)/(f113**3))
3746   QEzy = QEzy + ((gac*yvz1*bv1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((ga*yvz3*bv3)/(f113**3))
3747   QEzz = QEzz + ((gac*yvz1*bvz1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((ga*yvz3*bvz3)/(f113**3))
3748
3749   IQExx = QExx - ((gac*yvx1*bvz1)/(f11**3)) - ((yvx2*bvz2)/(f112**3)) - ((ga*yvx3*bvz3)/(f113**3))
3750   IQExy = QExy - ((gac*yvx1*bv1)/(f11**3)) - ((yvx2*bv2)/(f112**3)) - ((ga*yvx3*bv3)/(f113**3))
3751   IQExz = QExz - ((gac*yvx1*bvz1)/(f11**3)) - ((yvx2*bv2)/(f112**3)) - ((ga*yvx3*bvz3)/(f113**3))
3752   IQEyx = QEyx - ((gac*yvy1*bvz1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((ga*yvy3*bvz3)/(f113**3))
3753   IQEyy = QEyy - ((gac*yvy1*bv1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((ga*yvy3*bv3)/(f113**3))
3754   IQEyz = QEyz - ((gac*yvy1*bvz1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((ga*yvy3*bvz3)/(f113**3))
3755   IQEzx = QEzx - ((gac*yvz1*bvz1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((ga*yvz3*bvz3)/(f113**3))
3756   IQEzy = QEzy - ((gac*yvz1*bv1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((ga*yvz3*bv3)/(f113**3))
3757   IQEzz = QEzz - ((gac*yvz1*bvz1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((ga*yvz3*bvz3)/(f113**3))
3758
3759   PEx = PEx + ((gac*a1*bvz1)/(by1)) + ((a1*bvz2)/(by2)) + ((ga*a1*bvz3)/(by3))
3760   PEY = PEY + ((gac*a2*bv1)/(by1)) + ((a2*bv2)/(by2)) + ((ga*a2*bv3)/(by3))

```

```

3761   PEZ = PEZ + ((gac*a3*bv1)/(by1)) + ((a3*bv2)/(by2)) + ((ga*a3*bv3)/(by3))
3762
3763   IPEX = PEx - ((gac*a1*bvz1)/(by1)) - ((a1*bvz2)/(by2)) - ((ga*a1*bvz3)/(by3))
3764   IPEY = PEY - ((gac*a2*bv1)/(by1)) - ((a2*bv2)/(by2)) - ((ga*a2*bv3)/(by3))
3765   IPEZ = PEZ - ((gac*a3*bv1)/(by1)) - ((a3*bv2)/(by2)) - ((ga*a3*bv3)/(by3))
3766
3767 ! computation of curvature line integral along segment 3-6-1
3768
3769   yvx1 = p(13,1) - x00
3770   yvy1 = p(13,2) - y00
3771   yvz1 = p(13,3) - z00
3772   fil1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
3773
3774   yvx2 = p(16,1) - x00
3775   yvy2 = p(16,2) - y00
3776   yvz2 = p(16,3) - z00
3777   fil2 = DSQRT(yvx2**2 + yvy2**2 + yvz2**2)
3778
3779   yvx3 = p(11,1) - x00
3780   yvy3 = p(11,2) - y00
3781   yvz3 = p(11,3) - z00
3782   fil3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
3783
3784   ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
3785   ay2 = (a1*yvx2 + a2*yvy2 + a3*yvz2)
3786   ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
3787
3788   by1 = DSQRT(( fil1*(fil1+ay1) )**2)
3789   by2 = DSQRT(( fil2*(fil2+ay2) )**2)
3790   by3 = DSQRT(( fil3*(fil3+ay3) )**2)
3791
3792   bvx1 = yvz1*DzDe(3)-yvz1*DzDe(3)
3793   bvy1 = yvz1*DzDe(3)-yvz1*DzDe(3)
3794   bvz1 = yvz1*DzDe(3)-yvz1*DzDe(3)
3795
3796   bvz2 = yvz2*DzDe(6)-yvz2*DzDe(6)
3797   bvz2 = yvz2*DzDe(6)-yvz2*DzDe(6)
3798   bvz2 = yvz2*DzDe(6)-yvz2*DzDe(6)
3799
3800   bvz3 = yvz3*DzDe(1)-yvz3*DzDe(1)
3801   bvz3 = yvz3*DzDe(1)-yvz3*DzDe(1)
3802   bvz3 = yvz3*DzDe(1)-yvz3*DzDe(1)
3803
3804   QExx = QExx - ((be*yvx1*bvz1)/(f11**3)) - ((yvx2*bvz2)/(f112**3)) - ((bec*yvx3*bvz3)/(f113**3))
3805   QExy = QExy - ((be*yvx1*bv1)/(f11**3)) - ((yvx2*bv2)/(f112**3)) - ((bec*yvx3*bv3)/(f113**3))
3806   QExz = QExz - ((be*yvx1*bvz1)/(f11**3)) - ((yvx2*bv2)/(f112**3)) - ((bec*yvx3*bvz3)/(f113**3))
3807   QEyx = QEyx - ((be*yvy1*bvz1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((bec*yvy3*bvz3)/(f113**3))
3808   QEyy = QEyy - ((be*yvy1*bv1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((bec*yvy3*bv3)/(f113**3))
3809   QEyz = QEyz - ((be*yvy1*bvz1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((bec*yvy3*bvz3)/(f113**3))
3810   QEzx = QEzx - ((be*yvz1*bvz1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((bec*yvz3*bvz3)/(f113**3))
3811   QEzy = QEzy - ((be*yvz1*bv1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((bec*yvz3*bv3)/(f113**3))
3812   QEzz = QEzz - ((be*yvz1*bvz1)/(f11**3)) - ((yvz2*bv2)/(f112**3)) - ((bec*yvz3*bvz3)/(f113**3))
3813
3814   IQExx = QExx + ((be*yvx1*bvz1)/(f11**3)) + ((yvx2*bvz2)/(f112**3)) + ((bec*yvx3*bvz3)/(f113**3))
3815   IQExy = QExy + ((be*yvx1*bv1)/(f11**3)) + ((yvx2*bv2)/(f112**3)) + ((bec*yvx3*bv3)/(f113**3))
3816   IQExz = QExz + ((be*yvx1*bvz1)/(f11**3)) + ((yvx2*bv2)/(f112**3)) + ((bec*yvx3*bvz3)/(f113**3))
3817   IQEyx = QEyx + ((be*yvy1*bvz1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((bec*yvy3*bvz3)/(f113**3))
3818   IQEyy = QEyy + ((be*yvy1*bv1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((bec*yvy3*bv3)/(f113**3))
3819   IQEyz = QEyz + ((be*yvy1*bvz1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((bec*yvy3*bvz3)/(f113**3))
3820   IQEzx = QEzx + ((be*yvz1*bvz1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((bec*yvz3*bvz3)/(f113**3))
3821   IQEzy = QEzy + ((be*yvz1*bv1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((bec*yvz3*bv3)/(f113**3))
3822   IQEzz = QEzz + ((be*yvz1*bvz1)/(f11**3)) + ((yvz2*bv2)/(f112**3)) + ((bec*yvz3*bvz3)/(f113**3))
3823
3824   PEx = PEx - ((be*a1*bvz1)/(by1)) - ((a1*bvz2)/(by2)) - ((bec*a1*bvz3)/(by3))
3825   PEY = PEY - ((be*a2*bv1)/(by1)) - ((a2*bv2)/(by2)) - ((bec*a2*bv3)/(by3))
3826   PEZ = PEZ - ((be*a3*bv1)/(by1)) - ((a3*bv2)/(by2)) - ((bec*a3*bv3)/(by3))
3827
3828 ! PEx = PEx + ((be*a1*bvz1)/(by1)) + ((a1*bvz2)/(by2)) + ((bec*a1*bvz3)/(by3))
3829 ! PEY = PEY + ((be*a2*bv1)/(by1)) + ((a2*bv2)/(by2)) + ((bec*a2*bv3)/(by3))
3830 ! PEZ = PEZ + ((be*a3*bv1)/(by1)) + ((a3*bv2)/(by2)) + ((bec*a3*bv3)/(by3))
3831
3832 ! TExx = 2.000* QExx + cp1 -PE )/(cf)

```

```

3833 TExy = 2.000*( QExy )/(2.000*cf)
3834 TExz = 2.000*( QExz )/(2.000*cf)
3835 TExx = 2.000*( QExx )/(2.000*cf)
3836 TEyy = 2.000*( QEyy + cpl -PE )/(cf)
3837 TEyz = 2.000*( QEyz )/(2.000*cf)
3838 TEzx = 2.000*( QEzx )/(2.000*cf)
3839 TEzy = 2.000*( QEzy )/(2.000*cf)
3840 TEzz = 2.000*( QEzz + cpl -PE )/(cf)
3841 ! WRITE(*,*) PEx, PEy, PEz
3842 !WRITE(*,*) TExx, TEyy, TEyz
3843 !WRITE(*,*) TEzx, TEzy, TEzz
3844 !-----
3845 END SUBROUTINE intr_lin_sing_s8
3846 !-----
3847 !-----
3848 SUBROUTINE intr_lin_sing_s9(x0e, y0e, z0e, &
3849 & x00, y00, z00, &
3850 & k, &
3851 & TExx, TExy, TExz, &
3852 & TEyx, TEyy, TEyz, &
3853 & TEzx, TEzy, TEzz )
3854 !-----
3855 ! This subroutine is a new version Stokeslet Subroutine.
3856 !Compute:
3857 ! *The value of the Stokeslet over each singular element
3858 !Now, (March/ 09 / 2015) this subroutine was made.
3859 !-----
3860 USE Mod_Modul_Interp
3861 USE Mod_Prtc13D_Geo, ONLY: abc
3862 USE Mod_SharedVars, ONLY: DBL, p, ne, n, nbe, ULog,&
3863 & alpha0, beta0, gamma0,&
3864 & are1, crvml,&
3865 & vnx0, vny0, vnz0,&
3866 & ZZ, WJ, &
3867 & xiq, etq, Wq, pl, eps, x0, y0, z0
3868 !-----
3869 IMPLICIT NONE
3870 !-----
3871 ! Variables
3872 !-----
3873 REAL (KIND = DBL), INTENT(IN) :: x00, y00, z00 !singularity coordinates
3874 REAL (KIND = DBL), INTENT(IN) :: x0e, y0e, z0e !singularity coordinates element
3875 INTEGER, INTENT(IN) :: k !number of element
3876 REAL (KIND = DBL), INTENT(OUT) :: TExx, TExy, TExz !value of stresslet in the singular element
3877 REAL (KIND = DBL), INTENT(OUT) :: TEyx, TEyy, TEyz !value of stresslet in the singular element
3878 REAL (KIND = DBL), INTENT(OUT) :: TEzx, TEzy, TEzz !value of stresslet in the singular element
3879 !-----
3880 ! Variables inside the subroutine
3881 !-----
3882 INTEGER :: i, j !Counters
3883 INTEGER :: i1, i2, i3, i4, i5, i6 !indices to obtain node numbers from each element
3884 !-----
3885 REAL (KIND = DBL) :: cf, fill1, fill2, fill3 !integration weigh coefficients
3886 REAL (KIND = DBL) :: hss !lweigth
3887 REAL (KIND = DBL) :: cpl !selection to add the solid angle
3888 REAL (KIND = DBL) :: modx0 !lweigth
3889 REAL (KIND = DBL) :: bvx1, bvy1, bvz1, & !vectoreal product
3890 & bvx2, bvy2, bvz2, &
3891 & bvx3, bvy3, bvz3
3892 REAL (KIND = DBL) :: QExx, QExy, QExz, & !Iq tensor
3893 & QEyx, QEyy, QEyz, &
3894 & QExx, QExy, QEzz
3895 REAL (KIND = DBL) :: PEx, PEy, PEz, PE !Ip tensor
3896 REAL (KIND = DBL) :: yvx1, yvy1, yvz1 !vector (y1-x0)
3897 REAL (KIND = DBL) :: yvx2, yvy2, yvz2 !vector (y2-x0)
3898 REAL (KIND = DBL) :: yvx3, yvy3, yvz3 !vector (y3-x0)
3899 REAL (KIND = DBL) :: px2, py2, pz2 !vector (y2-x0)
3900 REAL (KIND = DBL) :: a1, a2, a3 !vector a
3901 REAL (KIND = DBL) :: ay1, ay2, ay3 !vector dot product a*y1, a*y2
3902 REAL (KIND = DBL) :: by1, by2, by3 !product y1*(y1+a.y1)
3903 !-----
3904 REAL (KIND = DBL) :: xi, eta !variables of weigh to integrate over a triangle

```

```

3905 REAL (KIND = DBL) :: x, y, z !coordinates of the f(x,y,z)= F(xi,eta)
3906 REAL (KIND = DBL) :: Dx0xi, Dy0xi, Dz0xi !coordinates of the tangential vector over the xi axis
3907 REAL (KIND = DBL) :: Dx0et, Dy0et, Dz0et !coordinates of the tangential vector over the eta axis
3908 REAL (KIND = DBL) :: vnx, vny, vnz !normal vector coordinates of the element
3909 REAL (KIND = DBL) :: hs, xs, es !surface metric on a triangle
3910 REAL (KIND = DBL) :: al, be, ga, alc, bec, gac !integration weigh coefficients
3911 REAL (KIND = DBL) :: fil !integration weigh coefficients
3912 REAL (KIND = DBL), DIMENSION(6) :: xxi, eet !variables of weigh over in triangle (xi,eta)
3913 REAL (KIND = DBL), DIMENSION(6) :: Dx0x, Dy0x, Dz0x !tangential vector over xi axis in triangle (xi,eta)
3914 REAL (KIND = DBL), DIMENSION(6) :: Dx0e, Dy0e, Dz0e !tangential vector over eta axis in triangle (xi,eta)
3915 REAL (KIND = DBL), DIMENSION(6) :: vx, vy, vz !normal vector in triangle (xi,eta)
3916 !-----
3917 ! Initialize
3918 !-----
3919 TExx = 0.000
3920 TExy = 0.000
3921 TExz = 0.000
3922 TEyx = 0.000
3923 TEyy = 0.000
3924 TEyz = 0.000
3925 TEzx = 0.000
3926 TEzy = 0.000
3927 TEzz = 0.000
3928 !-----
3929 QExx = 0.000
3930 QExy = 0.000
3931 QExz = 0.000
3932 QEyx = 0.000
3933 QEyy = 0.000
3934 QEyz = 0.000
3935 QExx = 0.000
3936 QExy = 0.000
3937 QExz = 0.000
3938 !-----
3939 PEx = 0.000
3940 PEy = 0.000
3941 PEz = 0.000
3942 PE = 0.000
3943 !-----
3944 a1 = -x00
3945 a2 = -y00
3946 a3 = -z00
3947 modx0= DSQRT(a1**2+a2**2+a3**2)
3948 !-----
3949 cpl = 4.000*pi
3950 !-----
3951 a1 = a1/modx0
3952 a2 = a2/modx0
3953 a3 = a3/modx0
3954 !-----
3955 !OPEN (9, file='TE.out')
3956 !-----
3957 ! vertices of the kth triangle
3958 !-----
3959 i1 = n(k,1)
3960 i2 = n(k,2)
3961 i3 = n(k,3)
3962 i4 = n(k,4)
3963 i5 = n(k,5)
3964 i6 = n(k,6)
3965 CALL abc(p(i1,1)-x00, p(i1,1)-y00, p(i1,1)-z00, &
3966 & p(i2,1)-x00, p(i2,1)-y00, p(i2,1)-z00, &
3967 & p(i3,1)-x00, p(i3,1)-y00, p(i3,1)-z00, &
3968 & p(i4,1)-x00, p(i4,1)-y00, p(i4,1)-z00, &
3969 & p(i5,1)-x00, p(i5,1)-y00, p(i5,1)-z00, &
3970 & p(i6,1)-x00, p(i6,1)-y00, p(i6,1)-z00, &
3971 & al, be, ga)
3972 alc = 1.000-a1
3973 bec = 1.000-be
3974 gac = 1.000-ga
3975 !-----
3976 cf = 0.000

```

```

3977 |-----|
3978 | compute the average value of the normal vector the mean curvature as a contour integral using the trapezoidal
3979 | rule formula
3980 |-----|
3981 DO i = 1, 13
3982   xi = xiq(i)
3983   eta = etq(i)
3984   CALL interp_p(p(11,1)-x00, p(11,2)-y00, p(11,3)-z00, &
3985     & p(12,1)-x00, p(12,2)-y00, p(12,3)-z00, &
3986     & p(13,1)-x00, p(13,2)-y00, p(13,3)-z00, &
3987     & p(14,1)-x00, p(14,2)-y00, p(14,3)-z00, &
3988     & p(15,1)-x00, p(15,2)-y00, p(15,3)-z00, &
3989     & p(16,1)-x00, p(16,2)-y00, p(16,3)-z00, &
3990     & al, be, ga, &
3991     & xi, eta, &
3992     & x, y, z, &
3993     & DxDxi, DyDxi, DzDxi, &
3994     & DDxDet, DDyDet, DDzDet, &
3995     & vx, vy, vz, &
3996     & hs)
3997   cf = hs*wq(i)
3998 END DO
3999 | cf=0.5D0*cf
4000 |-----|
4001 | computation of curvature line integral along segment 1-4
4002 | formula (4.2.10) of Pozrikidis (1997)
4003 |-----|
4004 xxi(1) = 0.000
4005 eet(1) = 0.000
4006 xxi(2) = 1.000
4007 eet(2) = 0.000
4008 xxi(3) = 0.000
4009 eet(3) = 1.000
4010 xxi(4) = al
4011 eet(4) = 0.000
4012 xxi(5) = ga
4013 eet(5) = gac
4014 xxi(6) = 0.000
4015 eet(6) = be
4016 DO i = 1, 6
4017   xi = xxi(i)
4018   eta = eet(i)
4019   CALL interp_p(p(11,1)-x00, p(11,2)-y00, p(11,3)-z00, &
4020     & p(12,1)-x00, p(12,2)-y00, p(12,3)-z00, &
4021     & p(13,1)-x00, p(13,2)-y00, p(13,3)-z00, &
4022     & p(14,1)-x00, p(14,2)-y00, p(14,3)-z00, &
4023     & p(15,1)-x00, p(15,2)-y00, p(15,3)-z00, &
4024     & p(16,1)-x00, p(16,2)-y00, p(16,3)-z00, &
4025     & al, be, ga, &
4026     & xi, eta, &
4027     & x, y, z, &
4028     & DxDxi, DyDxi, DzDxi, &
4029     & DDxDet, DDyDet, DDzDet, &
4030     & vx(i), vy(i), vz(i), &
4031     & hs)
4032 END DO
4033 |-----|
4034 bvx1 = 0.000
4035 bvy1 = 0.000
4036 bvz1 = 0.000
4037 bvx2 = 0.000
4038 bvy2 = 0.000
4039 bvz2 = 0.000
4040 bvx3 = 0.000
4041 bvy3 = 0.000
4042 bvz3 = 0.000
4043 yvx1 = 0.000
4044 yvy1 = 0.000
4045 yvz1 = 0.000
4046 yvx2 = 0.000
4047 yvy2 = 0.000
4048 yvz2 = 0.000

```

```

4049 yvx3 = 0.000
4050 yvy3 = 0.000
4051 yvz3 = 0.000
4052 fill1 = 0.000
4053 fill2 = 0.000
4054 fill3 = 0.000
4055 ay1 = 0.000
4056 ay2 = 0.000
4057 ay3 = 0.000
4058 |-----|
4059 | computation of curvature line integral along segment 1-4-2
4060 |-----|
4061 yvx1 = p(11,1) - x00
4062 yvy1 = p(11,2) - y00
4063 yvz1 = p(11,3) - z00
4064 fill1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
4065 |-----|
4066 yvx2 = p(14,1) - x00
4067 yvy2 = p(14,2) - y00
4068 yvz2 = p(14,3) - z00
4069 fill2 = DSQRT(yvx2**2 + yvy2**2 + yvz2**2)
4070 |-----|
4071 yvx3 = p(12,1) - x00
4072 yvy3 = p(12,2) - y00
4073 yvz3 = p(12,3) - z00
4074 fill3 = DSQRT(yvx3**2 + yvy3**2 + yvz3**2)
4075 |-----|
4076 ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
4077 ay2 = (a1*yvx2 + a2*yvy2 + a3*yvz2)
4078 ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
4079 |-----|
4080 by1 = ( fill1*(fill1+ay1) )
4081 by2 = ( fill2*(fill2+ay2) )
4082 by3 = ( fill3*(fill3+ay3) )
4083 |-----|
4084 bvx1 = yvy1*DzDx(1)-yvz1*DyDx(1)
4085 bvy1 = yvz1*DxDx(1)-yvx1*DzDx(1)
4086 bvz1 = yvx1*DyDx(1)-yvy1*DxDx(1)
4087 |-----|
4088 bvx2 = yvy2*DzDx(4)-yvz2*DyDx(4)
4089 bvy2 = yvz2*DxDx(4)-yvx2*DzDx(4)
4090 bvz2 = yvx2*DyDx(4)-yvy2*DxDx(4)
4091 |-----|
4092 bvx3 = yvy3*DzDx(2)-yvz3*DyDx(2)
4093 bvy3 = yvz3*DxDx(2)-yvx3*DzDx(2)
4094 bvz3 = yvx3*DyDx(2)-yvy3*DxDx(2)
4095 |-----|
4096 QExx = ((a1*yvx1*bvx1)/(fill1**3)) + ((yvx2*bvx2)/(fill2**3)) + ((alc*yvx3*bvx3)/(fill3**3))
4097 QExy = ((a1*yvx1*bvy1)/(fill1**3)) + ((yvx2*bvy2)/(fill2**3)) + ((alc*yvx3*bvy3)/(fill3**3))
4098 QExz = ((a1*yvx1*bvz1)/(fill1**3)) + ((yvx2*bvz2)/(fill2**3)) + ((alc*yvx3*bvz3)/(fill3**3))
4099 QEyx = ((a1*yvy1*bvx1)/(fill1**3)) + ((yvy2*bvx2)/(fill2**3)) + ((alc*yvy3*bvx3)/(fill3**3))
4100 QEyx = ((a1*yvy1*bvy1)/(fill1**3)) + ((yvy2*bvy2)/(fill2**3)) + ((alc*yvy3*bvy3)/(fill3**3))
4101 QEyz = ((a1*yvy1*bvz1)/(fill1**3)) + ((yvy2*bvz2)/(fill2**3)) + ((alc*yvy3*bvz3)/(fill3**3))
4102 QEzx = ((a1*yvz1*bvx1)/(fill1**3)) + ((yvx2*bvx2)/(fill2**3)) + ((alc*yvz3*bvx3)/(fill3**3))
4103 QEzy = ((a1*yvz1*bvy1)/(fill1**3)) + ((yvx2*bvy2)/(fill2**3)) + ((alc*yvz3*bvy3)/(fill3**3))
4104 QEzz = ((a1*yvz1*bvz1)/(fill1**3)) + ((yvx2*bvz2)/(fill2**3)) + ((alc*yvz3*bvz3)/(fill3**3))
4105 |-----|
4106 PEx = ((a1*a1*bvx1)/(by1)) + ((a1*bvx2)/(by2)) + ((alc*a1*bvx3)/(by3))
4107 PEy = ((a1*a2*bvy1)/(by1)) + ((a2*bvy2)/(by2)) + ((alc*a2*bvy3)/(by3))
4108 PEz = ((a1*a3*bvz1)/(by1)) + ((a3*bvz2)/(by2)) + ((alc*a3*bvz3)/(by3))
4109 |-----|
4110 | computation of curvature line integral along segment 2-5-3
4111 |-----|
4112 yvx1 = p(12,1) - x00
4113 yvy1 = p(12,2) - y00
4114 yvz1 = p(12,3) - z00
4115 fill1 = DSQRT(yvx1**2 + yvy1**2 + yvz1**2)
4116 |-----|
4117 yvx2 = p(15,1) - x00
4118 yvy2 = p(15,2) - y00
4119 yvz2 = p(15,3) - z00
4120 fill2 = DSQRT(yvx2**2 + yvy2**2 + yvz2**2)

```

```

4121 |-----|
4122 | yvx3 = p(13,1) - x00
4123 | yvy3 = p(13,2) - y00
4124 | yvz3 = p(13,3) - z00
4125 | fil3 = DSQR((yvx3**2 + yvy3**2 + yvz3**2))
4126 |-----|
4127 | ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
4128 | ay2 = (a1*yvx2 + a2*yvy2 + a3*yvz2)
4129 | ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
4130 |-----|
4131 | by1 = ( fil1*(fil1+ay1) )
4132 | by2 = ( fil2*(fil2+ay2) )
4133 | by3 = ( fil3*(fil3+ay3) )
4134 |-----|
4135 | bvx1 = yvy1*DzDx(2)-yvz1*DbDx(2)
4136 | bvy1 = yvz1*DxDx(2)-yvx1*DzDx(2)
4137 | bvz1 = yvx1*DyDx(2)-yvy1*DbDx(2)
4138 |-----|
4139 | bvx2 = yvy2*DzDx(5)-yvz2*DbDx(5)
4140 | bvy2 = yvz2*DxDx(5)-yvx2*DzDx(5)
4141 | bvz2 = yvx2*DyDx(5)-yvy2*DbDx(5)
4142 |-----|
4143 | bvx3 = yvy3*DzDx(3)-yvz3*DbDx(3)
4144 | bvy3 = yvz3*DxDx(3)-yvx3*DzDx(3)
4145 | bvz3 = yvx3*DyDx(3)-yvy3*DbDx(3)
4146 |-----|
4147 | QExx = QExx - ((gac*yvx1*bvx1)/(fil1**3)) - ((yvx2*bvx2)/(fil2**3)) - ((ga*yvx3*bvx3)/(fil3**3))
4148 | QExy = QExy - ((gac*yvx1*bvy1)/(fil1**3)) - ((yvx2*bvy2)/(fil2**3)) - ((ga*yvx3*bvy3)/(fil3**3))
4149 | QExz = QExz - ((gac*yvx1*bvz1)/(fil1**3)) - ((yvx2*bvz2)/(fil2**3)) - ((ga*yvx3*bvz3)/(fil3**3))
4150 | QEyX = QEyX - ((gac*yvy1*bvx1)/(fil1**3)) - ((yvy2*bvx2)/(fil2**3)) - ((ga*yvy3*bvx3)/(fil3**3))
4151 | QEyY = QEyY - ((gac*yvy1*bvy1)/(fil1**3)) - ((yvy2*bvy2)/(fil2**3)) - ((ga*yvy3*bvy3)/(fil3**3))
4152 | QEyZ = QEyZ - ((gac*yvy1*bvz1)/(fil1**3)) - ((yvy2*bvz2)/(fil2**3)) - ((ga*yvy3*bvz3)/(fil3**3))
4153 | QEzX = QEzX - ((gac*yvz1*bvx1)/(fil1**3)) - ((yvz2*bvx2)/(fil2**3)) - ((ga*yvz3*bvx3)/(fil3**3))
4154 | QEzY = QEzY - ((gac*yvz1*bvy1)/(fil1**3)) - ((yvz2*bvy2)/(fil2**3)) - ((ga*yvz3*bvy3)/(fil3**3))
4155 | QEzZ = QEzZ - ((gac*yvz1*bvz1)/(fil1**3)) - ((yvz2*bvz2)/(fil2**3)) - ((ga*yvz3*bvz3)/(fil3**3))
4156 |-----|
4157 | QExx = QExx + ((gac*yvx1*bvx1)/(fil1**3)) + ((yvx2*bvx2)/(fil2**3)) + ((ga*yvx3*bvx3)/(fil3**3))
4158 | QExy = QExy + ((gac*yvx1*bvy1)/(fil1**3)) + ((yvx2*bvy2)/(fil2**3)) + ((ga*yvx3*bvy3)/(fil3**3))
4159 | QExz = QExz + ((gac*yvx1*bvz1)/(fil1**3)) + ((yvx2*bvz2)/(fil2**3)) + ((ga*yvx3*bvz3)/(fil3**3))
4160 | QEyX = QEyX + ((gac*yvy1*bvx1)/(fil1**3)) + ((yvy2*bvx2)/(fil2**3)) + ((ga*yvy3*bvx3)/(fil3**3))
4161 | QEyY = QEyY + ((gac*yvy1*bvy1)/(fil1**3)) + ((yvy2*bvy2)/(fil2**3)) + ((ga*yvy3*bvy3)/(fil3**3))
4162 | QEyZ = QEyZ + ((gac*yvy1*bvz1)/(fil1**3)) + ((yvy2*bvz2)/(fil2**3)) + ((ga*yvy3*bvz3)/(fil3**3))
4163 | QEzX = QEzX + ((gac*yvz1*bvx1)/(fil1**3)) + ((yvz2*bvx2)/(fil2**3)) + ((ga*yvz3*bvx3)/(fil3**3))
4164 | QEzY = QEzY + ((gac*yvz1*bvy1)/(fil1**3)) + ((yvz2*bvy2)/(fil2**3)) + ((ga*yvz3*bvy3)/(fil3**3))
4165 | QEzZ = QEzZ + ((gac*yvz1*bvz1)/(fil1**3)) + ((yvz2*bvz2)/(fil2**3)) + ((ga*yvz3*bvz3)/(fil3**3))
4166 |-----|
4167 | PEx = PEx - ((gac*a1*bvx1)/(by1)) - ((a1*bvx2)/(by2)) - ((ga*a1*bvx3)/(by3))
4168 | PEy = PEy - ((gac*a2*bvy1)/(by1)) - ((a2*bvy2)/(by2)) - ((ga*a2*bvy3)/(by3))
4169 | PEz = PEz - ((gac*a3*bvz1)/(by1)) - ((a3*bvz2)/(by2)) - ((ga*a3*bvz3)/(by3))
4170 |-----|
4171 | PEx = PEx + ((gac*a1*bvx1)/(by1)) + ((a1*bvx2)/(by2)) + ((ga*a1*bvx3)/(by3))
4172 | PEy = PEy + ((gac*a2*bvy1)/(by1)) + ((a2*bvy2)/(by2)) + ((ga*a2*bvy3)/(by3))
4173 | PEz = PEz + ((gac*a3*bvz1)/(by1)) + ((a3*bvz2)/(by2)) + ((ga*a3*bvz3)/(by3))
4174 |-----|
4175 | bvx1 = yvy1*DzDe(2)-yvz1*DyDe(2)
4176 | bvy1 = yvz1*DzDe(2)-yvx1*DyDe(2)
4177 | bvz1 = yvx1*DyDe(2)-yvy1*DzDe(2)
4178 |-----|
4179 | bvx2 = yvy2*DzDe(5)-yvz2*DyDe(5)
4180 | bvy2 = yvz2*DzDe(5)-yvx2*DyDe(5)
4181 | bvz2 = yvx2*DyDe(5)-yvy2*DzDe(5)
4182 |-----|
4183 | bvx3 = yvy3*DzDe(3)-yvz3*DyDe(3)
4184 | bvy3 = yvz3*DzDe(3)-yvx3*DyDe(3)
4185 | bvz3 = yvx3*DyDe(3)-yvy3*DzDe(3)
4186 |-----|
4187 | QExx = QExx + ((gac*yvx1*bvx1)/(fil1**3)) + ((yvx2*bvx2)/(fil2**3)) + ((ga*yvx3*bvx3)/(fil3**3))
4188 | QExy = QExy + ((gac*yvx1*bvy1)/(fil1**3)) + ((yvx2*bvy2)/(fil2**3)) + ((ga*yvx3*bvy3)/(fil3**3))
4189 | QExz = QExz + ((gac*yvx1*bvz1)/(fil1**3)) + ((yvx2*bvz2)/(fil2**3)) + ((ga*yvx3*bvz3)/(fil3**3))
4190 | QEyX = QEyX + ((gac*yvy1*bvx1)/(fil1**3)) + ((yvy2*bvx2)/(fil2**3)) + ((ga*yvy3*bvx3)/(fil3**3))
4191 | QEyY = QEyY + ((gac*yvy1*bvy1)/(fil1**3)) + ((yvy2*bvy2)/(fil2**3)) + ((ga*yvy3*bvy3)/(fil3**3))
4192 | QEyZ = QEyZ + ((gac*yvy1*bvz1)/(fil1**3)) + ((yvy2*bvz2)/(fil2**3)) + ((ga*yvy3*bvz3)/(fil3**3))

```

```

4193 | QEzX = QEzX + ((gac*yvz1*bvx1)/(fil1**3)) + ((yvz2*bvx2)/(fil2**3)) + ((ga*yvz3*bvx3)/(fil3**3))
4194 | QEzY = QEzY + ((gac*yvz1*bvy1)/(fil1**3)) + ((yvz2*bvy2)/(fil2**3)) + ((ga*yvz3*bvy3)/(fil3**3))
4195 | QEzZ = QEzZ + ((gac*yvz1*bvz1)/(fil1**3)) + ((yvz2*bvz2)/(fil2**3)) + ((ga*yvz3*bvz3)/(fil3**3))
4196 |-----|
4197 | QExx = QExx - ((gac*yvx1*bvx1)/(fil1**3)) - ((yvx2*bvx2)/(fil2**3)) - ((ga*yvx3*bvx3)/(fil3**3))
4198 | QExy = QExy - ((gac*yvx1*bvy1)/(fil1**3)) - ((yvx2*bvy2)/(fil2**3)) - ((ga*yvx3*bvy3)/(fil3**3))
4199 | QExz = QExz - ((gac*yvx1*bvz1)/(fil1**3)) - ((yvx2*bvz2)/(fil2**3)) - ((ga*yvx3*bvz3)/(fil3**3))
4200 | QEyX = QEyX - ((gac*yvy1*bvx1)/(fil1**3)) - ((yvy2*bvx2)/(fil2**3)) - ((ga*yvy3*bvx3)/(fil3**3))
4201 | QEyY = QEyY - ((gac*yvy1*bvy1)/(fil1**3)) - ((yvy2*bvy2)/(fil2**3)) - ((ga*yvy3*bvy3)/(fil3**3))
4202 | QEyZ = QEyZ - ((gac*yvy1*bvz1)/(fil1**3)) - ((yvy2*bvz2)/(fil2**3)) - ((ga*yvy3*bvz3)/(fil3**3))
4203 | QEzX = QEzX - ((gac*yvz1*bvx1)/(fil1**3)) - ((yvz2*bvx2)/(fil2**3)) - ((ga*yvz3*bvx3)/(fil3**3))
4204 | QEzY = QEzY - ((gac*yvz1*bvy1)/(fil1**3)) - ((yvz2*bvy2)/(fil2**3)) - ((ga*yvz3*bvy3)/(fil3**3))
4205 | QEzZ = QEzZ - ((gac*yvz1*bvz1)/(fil1**3)) - ((yvz2*bvz2)/(fil2**3)) - ((ga*yvz3*bvz3)/(fil3**3))
4206 |-----|
4207 | PEx = PEx + ((gac*a1*bvx1)/(by1)) + ((a1*bvx2)/(by2)) + ((ga*a1*bvx3)/(by3))
4208 | PEy = PEy + ((gac*a2*bvy1)/(by1)) + ((a2*bvy2)/(by2)) + ((ga*a2*bvy3)/(by3))
4209 | PEz = PEz + ((gac*a3*bvz1)/(by1)) + ((a3*bvz2)/(by2)) + ((ga*a3*bvz3)/(by3))
4210 |-----|
4211 | lPEX = PEx - ((gac*a1*bvx1)/(by1)) - ((a1*bvx2)/(by2)) - ((ga*a1*bvx3)/(by3))
4212 | lPEY = PEy - ((gac*a2*bvy1)/(by1)) - ((a2*bvy2)/(by2)) - ((ga*a2*bvy3)/(by3))
4213 | lPEZ = PEz - ((gac*a3*bvz1)/(by1)) - ((a3*bvz2)/(by2)) - ((ga*a3*bvz3)/(by3))
4214 |-----|
4215 | computation of curvature line integral along segment 3-6-1
4216 |-----|
4217 | yvx1 = p(13,1) - x00
4218 | yvy1 = p(13,2) - y00
4219 | yvz1 = p(13,3) - z00
4220 | fil1 = DSQR((yvx1**2 + yvy1**2 + yvz1**2))
4221 |-----|
4222 | yvx2 = p(16,1) - x00
4223 | yvy2 = p(16,2) - y00
4224 | yvz2 = p(16,3) - z00
4225 | fil2 = DSQR((yvx2**2 + yvy2**2 + yvz2**2))
4226 |-----|
4227 | yvx3 = p(11,1) - x00
4228 | yvy3 = p(11,2) - y00
4229 | yvz3 = p(11,3) - z00
4230 | fil3 = DSQR((yvx3**2 + yvy3**2 + yvz3**2))
4231 |-----|
4232 | ay1 = (a1*yvx1 + a2*yvy1 + a3*yvz1)
4233 | ay2 = (a1*yvx2 + a2*yvy2 + a3*yvz2)
4234 | ay3 = (a1*yvx3 + a2*yvy3 + a3*yvz3)
4235 |-----|
4236 | by1 = DSQR(( fil1*(fil1+ay1) **2))
4237 | by2 = DSQR(( fil2*(fil2+ay2) **2))
4238 | by3 = DSQR(( fil3*(fil3+ay3) **2))
4239 |-----|
4240 | bvx1 = yvy1*DzDe(3)-yvz1*DyDe(3)
4241 | bvy1 = yvz1*DzDe(3)-yvx1*DyDe(3)
4242 | bvz1 = yvx1*DyDe(3)-yvy1*DzDe(3)
4243 |-----|
4244 | bvx2 = yvy2*DzDe(6)-yvz2*DyDe(6)
4245 | bvy2 = yvz2*DzDe(6)-yvx2*DyDe(6)
4246 | bvz2 = yvx2*DyDe(6)-yvy2*DzDe(6)
4247 |-----|
4248 | bvx3 = yvy3*DzDe(3)-yvz3*DyDe(3)
4249 | bvy3 = yvz3*DzDe(3)-yvx3*DyDe(3)
4250 | bvz3 = yvx3*DyDe(3)-yvy3*DzDe(3)
4251 |-----|
4252 | QExx = QExx - ((be*yvx1*bvx1)/(fil1**3)) - ((yvx2*bvx2)/(fil2**3)) - ((bec*yvx3*bvx3)/(fil3**3))
4253 | QExy = QExy - ((be*yvx1*bvy1)/(fil1**3)) - ((yvx2*bvy2)/(fil2**3)) - ((bec*yvx3*bvy3)/(fil3**3))
4254 | QExz = QExz - ((be*yvx1*bvz1)/(fil1**3)) - ((yvx2*bvz2)/(fil2**3)) - ((bec*yvx3*bvz3)/(fil3**3))
4255 | QEyX = QEyX - ((be*yvy1*bvx1)/(fil1**3)) - ((yvy2*bvx2)/(fil2**3)) - ((bec*yvy3*bvx3)/(fil3**3))
4256 | QEyY = QEyY - ((be*yvy1*bvy1)/(fil1**3)) - ((yvy2*bvy2)/(fil2**3)) - ((bec*yvy3*bvy3)/(fil3**3))
4257 | QEyZ = QEyZ - ((be*yvy1*bvz1)/(fil1**3)) - ((yvy2*bvz2)/(fil2**3)) - ((bec*yvy3*bvz3)/(fil3**3))
4258 | QEzX = QEzX - ((be*yvz1*bvx1)/(fil1**3)) - ((yvz2*bvx2)/(fil2**3)) - ((bec*yvz3*bvx3)/(fil3**3))
4259 | QEzY = QEzY - ((be*yvz1*bvy1)/(fil1**3)) - ((yvz2*bvy2)/(fil2**3)) - ((bec*yvz3*bvy3)/(fil3**3))
4260 | QEzZ = QEzZ - ((be*yvz1*bvz1)/(fil1**3)) - ((yvz2*bvz2)/(fil2**3)) - ((bec*yvz3*bvz3)/(fil3**3))
4261 |-----|
4262 | lQExx = QExx + ((be*yvx1*bvx1)/(fil1**3)) + ((yvx2*bvx2)/(fil2**3)) + ((bec*yvx3*bvx3)/(fil3**3))
4263 | lQExy = QExy + ((be*yvx1*bvy1)/(fil1**3)) + ((yvx2*bvy2)/(fil2**3)) + ((bec*yvx3*bvy3)/(fil3**3))
4264 | lQExz = QExz + ((be*yvx1*bvz1)/(fil1**3)) + ((yvx2*bvz2)/(fil2**3)) + ((bec*yvx3*bvz3)/(fil3**3))

```

```
4265 IQEyx = QEyx + ((be*yvy1*bvx1)/(f11**3)) + ((yvy2*bvx2)/(f12**3)) + ((bec*yvy3*bvx3)/(f13**3))
4266 IQEyy = QEyy + ((be*yvy1*bvy1)/(f11**3)) + ((yvy2*bvy2)/(f12**3)) + ((bec*yvy3*bvy3)/(f13**3))
4267 IQEyz = QEyz + ((be*yvy1*bvz1)/(f11**3)) + ((yvy2*bvz2)/(f12**3)) + ((bec*yvy3*bvz3)/(f13**3))
4268 IQEzx = QEzx + ((be*yvz1*bvx1)/(f11**3)) + ((yvz2*bvx2)/(f12**3)) + ((bec*yvz3*bvx3)/(f13**3))
4269 IQEzy = QEzy + ((be*yvz1*bvy1)/(f11**3)) + ((yvz2*bvy2)/(f12**3)) + ((bec*yvz3*bvy3)/(f13**3))
4270 IQEzz = QEzz + ((be*yvz1*bvz1)/(f11**3)) + ((yvz2*bvz2)/(f12**3)) + ((bec*yvz3*bvz3)/(f13**3))
-----
4271 PEx = PEx - ((be*a1*bvx1)/(by1)) - ((a1*bvx2)/(by2)) - ((bec*a1*bvx3)/(by3))
4272 PEy = PEy - ((be*a2*bvy1)/(by1)) - ((a2*bvy2)/(by2)) - ((bec*a2*bvy3)/(by3))
4273 PEz = PEz - ((be*a3*bvz1)/(by1)) - ((a3*bvz2)/(by2)) - ((bec*a3*bvz3)/(by3))
-----
4275 IPEx = PEx + ((be*a1*bvx1)/(by1)) + ((a1*bvx2)/(by2)) + ((bec*a1*bvx3)/(by3))
4276 IPEy = PEy + ((be*a2*bvy1)/(by1)) + ((a2*bvy2)/(by2)) + ((bec*a2*bvy3)/(by3))
4277 IPEz = PEz + ((be*a3*bvz1)/(by1)) + ((a3*bvz2)/(by2)) + ((bec*a3*bvz3)/(by3))
-----
4279 TExx = 2.000*( QEyx + cpi -PE)/(cf)
4280 TEyx = 2.000*( QExy)/(2.000*cf)
4281 TEzx = 2.000*( QEzx)/(2.000*cf)
4282 TEyz = 2.000*( QEyz)/(2.000*cf)
4283 TEyy = 2.000*( QEyy + cpi -PE)/(cf)
4284 TEzy = 2.000*( QEzy)/(2.000*cf)
4285 TEzz = 2.000*( QEzz)/(2.000*cf)
4286 TEzx = 2.000*( QEzx)/(2.000*cf)
4287 TEzy = 2.000*( QEzy)/(2.000*cf)
4288 TEzz = 2.000*( QEzz + cpi -PE)/(cf)
4289 ! WRITE(*,*) PEx, PEy, PEz
4290 !WRITE(*,*) TEyx, TEyy, TEyz
4291 !WRITE(*,*) TEzx, TEzy, TEzz
-----
4292
4293 END SUBROUTINE intr_lin_sing_s9
4294 !=====
4295 END MODULE Mod_Prtcl3D_DLP
4296
```

```

1 MODULE Mod_sgf_3d_sfs
2 !-----
3 | Version: 0.8 created on 22 / 03 / 2012
4 | Version: 0.9 created on 03 / 09 / 2012
5 | Version: 1.0 created on 13 / 11 / 2012
6 !-----
7 |-----
8 |-----
9 |-----
10 |-----
11 |-----
12 |-----
13 |-----
14 |-----
15 |-----
16 |-----
17 |-----
18 |-----
19 |-----
20 |-----
21 |-----
22 | Free-space Green's function: Stresslet Pozrikidis (1992, p. 23)
23 | This new version of sfg_3d_sfs only calculates the Green's function: Stresslet.
24 |-----
25 | USE Mod_SharedVars, ONLY: DBL, ULog
26 | IMPLICIT NONE
27 |-----
28 | Variables
29 |-----
30 | REAL (KIND = DBL), INTENT(IN) :: x, y, z |coordinates of collocation point of the element
31 | REAL (KIND = DBL), INTENT(IN) :: x0, y0, z0 |coordinates of the singularity
32 | REAL (KIND = DBL), INTENT(OUT) :: Txxx,Txxy,Txxz, & |Free-space Green's function of Stokeslet.
33 | & Txyx,Txyy,Txyz, & |Integrate over ijk component over the element
34 | & Tzxx,Tzxy,Tzxz, &
35 | & Txyx,Txyy,Txyz, &
36 | & Tyxx,Tyyy,Tyzz, &
37 | & Tzxx,Tzxy,Tzxz, &
38 | & Tzyx,Tzyy,Tzyz, &
39 | & Tzxx,Tzxy,Tzxz, &
40 | & Tzxx,Tzxy,Tzxz
41 |-----
42 | Variables inside the subroutine
43 |-----
44 | REAL (KIND = DBL) :: dx, dy, dz |differences between the (x,y,z) and (x0,y0,z0) coordinates
45 | REAL (KIND = DBL) :: dxx, dxy, dxz, & |square distance d1*dj=dij
46 | & dyy, dyz, dzz
47 | REAL (KIND = DBL) :: r |distance r between the (x,y,z) point and (x0,y0,z0)
48 | REAL (KIND = DBL) :: r15 |stresslet coefficient
49 |-----
50 | Initialize
51 | dx = 0.000
52 | dy = 0.000
53 | dz = 0.000
54 |-----
55 | dxx = 0.000
56 | dxy = 0.000
57 | dxz = 0.000
58 | dyy = 0.000
59 | dyz = 0.000
60 | dzz = 0.000
61 |-----
62 | Txxx = 0.000
63 | Txyx = 0.000
64 | Txxz = 0.000
65 | Tyxx = 0.000
66 | Txyy = 0.000
67 | Txyz = 0.000
68 | Tzxx = 0.000
69 | Tzyx = 0.000
70 | Tzxy = 0.000
71 |-----
72 | Tyxx = 0.000

```

```

73 | Tyxy = 0.000
74 | Tyxz = 0.000
75 | Tyyx = 0.000
76 | Tyyy = 0.000
77 | Tyzz = 0.000
78 | Tzxx = 0.000
79 | Tzxy = 0.000
80 | Tzyz = 0.000
81 |-----
82 | Tzxx = 0.000
83 | Tzxy = 0.000
84 | Tzxz = 0.000
85 | Tzyx = 0.000
86 | Tzyy = 0.000
87 | Tzyz = 0.000
88 | Tzxx = 0.000
89 | Tzzy = 0.000
90 | Tzzz = 0.000
91 |-----
92 | dx = x-x0
93 | dy = y-y0
94 | dz = z-z0
95 |-----
96 | dxx = dx*dx
97 | dxy = dx*dy
98 | dxz = dx*dz
99 | dyy = dy*dy
100 | dyz = dy*dz
101 | dzz = dz*dz
102 |-----
103 | r = DSQRT(dxx + dyy + dzz)
104 |-----
105 | compute the stress tensor
106 |-----
107 | r15 = -6.000/r**5
108 |-----
109 | Txxx = dxx*dxx * r15
110 | Txyx = dxx*dy * r15
111 | Txxz = dxx*dz * r15
112 | Txyx = Txyx
113 | Txyy = dxy*dy * r15
114 | Txyz = dxy*dz * r15
115 | Tzxx = Tzxx
116 | Tzxy = Tzxy
117 | Tzxx = dxz*dz * r15
118 |-----
119 | Tyxx = Tyxx
120 | Tyxy = Tyxy
121 | Tyxz = Tyxz
122 | Tyyx = Tyxy
123 | Tyyy = dyy*dy * r15
124 | Tyyz = dyy*dz * r15
125 | Tyzx = Tyxz
126 | Tyzy = Tyzy
127 | Tyzz = dyz*dz * r15
128 |-----
129 | Tzxx = Tzxx
130 | Tzxy = Tzxy
131 | Tzxz = Tzxz
132 | Tzyx = Tzyx
133 | Tzyy = Tzyy
134 | Tzyz = Tzyz
135 | Tzxx = Tzxx
136 | Tzzy = Tzzy
137 | Tzzz = dzz*dz * r15
138 |-----
139 | END SUBROUTINE sgf_3d_sfs
140 |-----
141 | END MODULE Mod_sgf_3d_sfs

```

```

1 MODULE Mod_Velocity_Menu
2 =====
3 | Version: 0.9 created on 01 / 03 / 2013
4 |
5 | Alfredo Sanjuan Sanjuan
6 |
7 CONTAINS
8 =====
9 |
10 | SUBROUTINE Infinity_Flow(WF, alpha, g1, Uoo, stride, steps, nelm, h)
11 | This Subroutine obtains the value of velocity field in the interface.
12 | SYMBOLS:
13 |
14 | USE Mod_SharedVars, ONLY: DBL, x0, y0, z0, Ulog
15 | USE Mod_VelFieldTRM
16 |
17 |
18 | IMPLICIT NONE
19 |
20 |
21 | Variables
22 | INTEGER, INTENT(INOUT) :: NVF !selection of velocity field
23 | INTEGER, INTENT(IN) :: nelm !number of elements
24 | INTEGER, INTENT(IN) :: stride, steps !number of elements
25 | REAL(KIND = DBL), INTENT(IN) :: alpha, h !flow parameter
26 | REAL(KIND = DBL), INTENT(IN) :: g1 !intensity of flow
27 | REAL(KIND = DBL) :: tobi !dummy integrate arc constants
28 | REAL(KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: Uoo !arrays of infinity velocity field
29 |
30 | Variables inside the subroutine
31 |
32 |
33 | INTEGER :: i !Counters
34 | tobi = g1*DSQRT(1.000)
35 | The next step is use a Case statement to obtain the coeficientes.
36 | SELECT CASE(NVF)
37 |
38 | CASE (1)
39 | FORALL (i=1:nelm)
40 | Uoo(i) = 0.000
41 | Uoo(i+nelm) = 0.000
42 | Uoo(i+nelm+nelm) = 0.000
43 | END FORALL
44 |
45 | CASE (2)
46 | FORALL (i=1:nelm)
47 | Uoo(i) = 0.000
48 | Uoo(i+nelm) = 1.000
49 | Uoo(i+nelm+nelm) = 0.000
50 | END FORALL
51 |
52 | CASE (3)
53 | FORALL (i=1:nelm)
54 | Uoo(i) = tobi/DSQRT(2.000)
55 | Uoo(i+nelm) = tobi/DSQRT(2.000)
56 | Uoo(i+nelm+nelm) = 0.000
57 | END FORALL
58 |
59 | CASE (4)
60 | FORALL (i=1:nelm)
61 | Uoo(i) = 2.000*tobi/DSQRT(5.000)
62 | Uoo(i+nelm) = tobi/DSQRT(5.000)
63 | Uoo(i+nelm+nelm) = 0.000
64 | END FORALL
65 |
66 | CASE (5) !it is not normalized
67 | FORALL (i=1:nelm)
68 | Uoo(i) = tobi*z0(i)
69 | Uoo(i+nelm) = -tobi*z0(i)
70 | Uoo(i+nelm+nelm) = 0.000
71 | END FORALL
72 |

```

```

73 | CASE (6)
74 | FORALL (i=1:nelm)
75 | Uoo(i) = -tobi*y0(i)
76 | Uoo(i+nelm) = tobi*x0(i)
77 | Uoo(i+nelm+nelm) = 0.000
78 | END FORALL
79 |
80 | CASE (7)
81 | FORALL (i=1:nelm)
82 | !Uoo(i) = ((1.000- DEXP((-100.000*DBLE(stride)/DBLE(steps))))*gi*x0(i)*(alpha - 2.000))/(2.000*DSQRT(4.000
83 | - 2.000*alpha + alpha**2))
84 | !Uoo(i+nelm) = ((1.000- DEXP((-100.000*DBLE(stride)/DBLE(steps))))*gi*y0(i))/(DSQRT(4.000 - 2.000*alpha +
85 | alpha**2))
86 | !Uoo(i+nelm+nelm) = ((1.000- DEXP((-100.000*DBLE(stride)/DBLE(steps))))*-alpha*gi*z0(i))/(2.000*DSQRT(4.000
87 | - 2.000*alpha + alpha**2))
88 | END FORALL
89 |
90 | CASE (8)
91 | FORALL (i=1:nelm)
92 | Uoo(i) = (tobi*x0(i)**(alpha - 2.000))/(2.000*DSQRT(4.000 - 2.000*alpha + alpha**2))
93 | Uoo(i+nelm) = (tobi*y0(i))/(DSQRT(4.000 - 2.000*alpha + alpha**2))
94 | Uoo(i+nelm+nelm) = (-alpha*tobi*z0(i))/(2.000*DSQRT(4.000 - 2.000*alpha + alpha**2))
95 | END FORALL
96 |
97 | CASE (9)
98 | FORALL (i=1:nelm)
99 | Uoo(i) = (tobi/DSQRT(2.000)*(1.000+alpha))**((1.000+alpha)*x0(i) + (1.000-alpha)*y0(i))
100 | Uoo(i+nelm) = (tobi/DSQRT(2.000)*(1.000+alpha))**((-1.000+alpha)*x0(i) + (-1.000-alpha)*y0(i))
101 | Uoo(i+nelm+nelm) = 0.000
102 | END FORALL
103 |
104 | CASE (10)
105 | FORALL (i=1:nelm)
106 | !Uoo(i) = g1*(1.000- DEXP((-100.000*DBLE(stride)/DBLE(steps))))*y0(i)/(1.000+alpha)
107 | !Uoo(i+nelm) = gi*(1.000- DEXP((-100.000*DBLE(stride)/DBLE(steps))))*alpha*x0(i)/(1.000+alpha)
108 | !Uoo(i+nelm+nelm) = 0.000
109 | END FORALL
110 |
111 | CASE (11)
112 | FORALL (i=1:nelm)
113 | Uoo(i) = tobi*y0(i)/(1.000+alpha)
114 | Uoo(i+nelm) = tobi*alpha*x0(i)/(1.000+alpha)
115 | Uoo(i+nelm+nelm) = 0.000
116 | END FORALL
117 |
118 | CASE (12)
119 | FORALL (i=1:nelm)
120 | Uoo(i) = DSIN(z0(i)) + DCOS(y0(i))
121 | Uoo(i+nelm) = DSIN(x0(i))+DCOS(z0(i))
122 | Uoo(i+nelm+nelm) = DSIN(y0(i))+DCOS(x0(i))
123 | END FORALL
124 |
125 | CASE (13)
126 | FORALL (i=1:nelm)
127 | Uoo(i) = tobi*y0(i)**(1.000 - ((-0.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride))))
128 | Uoo(i+nelm) = tobi*x0(i)**(1.000 + ((-0.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride))))
129 | Uoo(i+nelm+nelm) = g1*y0(i)**(1.000 - ((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride))))
130 | Uoo(i) = g1*x0(i)**(1.000 + ((-2.000*alpha) + (((4.000*alpha)/DBLE(steps))*DBLE(stride))))
131 | Uoo(i+nelm+nelm) = 0.000
132 | END FORALL
133 |
134 | CASE (14)
135 | FORALL (i=1:nelm)
136 | Uoo(i) = tobi*(y0(i))**((1.000 - ((2.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride))))
137 | Uoo(i+nelm) = tobi*x0(i)**(1.000 + ((2.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride))))
138 | Uoo(i+nelm+nelm) = g1*(y0(i))**((1.000 - ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride))))
139 | Uoo(i) = g1*x0(i)**(1.000 + ((4.000*alpha) - (((4.000*alpha)/DBLE(steps))*DBLE(stride))))
140 | Uoo(i+nelm+nelm) = 0.000
141 | END FORALL

```



```
142 !-----
143 CASE (14)
144 !$omp parallel
145 !$omp do
146     DO i = 1, nelm
147         CALL VelPsi(x0(i), y0(i), Uoo(i), Uoo(i+nelm))
148         Uoo(i+nelm+nelm) = 0.000
149     END DO
150 !$omp end do
151 !$omp end parallel
152 !-----
153 CASE (15)
154     FORALL (i=1:nelm)
155         Uoo(i) = 0.500*tobi*y0(i)/(1.000+alpha)
156         Uoo(i+nelm) = 0.500*tobi*alpha*x0(i)/(1.000+alpha)
157         Uoo(i+nelm+nelm) = 0.000
158     END FORALL
159 !-----
160 CASE (16)
161     FORALL (i=1:nelm)
162         Uoo(i) = tobi*y0(i)/(1.000+alpha)
163         !Uoo(i+nelm) = g1*x0(i)*((-0.500*alpha) + (((2.500*alpha)/DBLE(steps))*DBLE(stride)))/(1.000+alpha)
164         Uoo(i+nelm) = tobi*x0(i)*alpha/(1.000+alpha)
165         Uoo(i+nelm+nelm) = 0.000
166     END FORALL
167 !-----
168 CASE (17)
169     FORALL (i=1:nelm)
170         Uoo(i) = tobi*y0(i)/(1.000+alpha)
171         !Uoo(i+nelm) = g1*x0(i)*((2.500*alpha) - (((2.500*alpha)/DBLE(steps))*DBLE(stride)))/(1.000+alpha)
172         Uoo(i+nelm) = tobi*x0(i)*alpha/(1.000+alpha)
173         Uoo(i+nelm+nelm) = 0.000
174     END FORALL
175 !-----
176 CASE (18)
177     FORALL (i=1:nelm)
178         Uoo(i) = tobi*(1.000 - DEXP((-100.000*DBLE(stride))/DBLE(steps)))*y0(i)*(1.000 - alpha)
179         Uoo(i+nelm) = tobi*(1.000 - DEXP((-100.000*DBLE(stride))/DBLE(steps)))*x0(i)*(1.000 + alpha)
180         Uoo(i+nelm+nelm) = 0.000
181     END FORALL
182 !-----
183 CASE (71)
184     FORALL (i=1:nelm)
185         Uoo(i) = (tobi*x0(i)*(alpha - 2.000))/(2.000*DSQRT(4.000 - 2.000*alpha + alpha**2))
186         Uoo(i+nelm) = (tobi*y0(i))/(DSQRT(4.000 - 2.000*alpha + alpha**2))
187         Uoo(i+nelm+nelm) = (-alpha*g1*z0(i))/(2.000*DSQRT(4.000 - 2.000*alpha + alpha**2))
188     END FORALL
189 !-----
190 CASE (72)
191     FORALL (i=1:nelm)
192         Uoo(i) = (tobi*x0(i)*(alpha - 2.000))/(2.000*DSQRT(4.000 - 2.000*alpha + alpha**2))
193         Uoo(i+nelm) = (tobi*y0(i))/(DSQRT(4.000 - 2.000*alpha + alpha**2))
194         Uoo(i+nelm+nelm) = (-alpha*g1*z0(i))/(2.000*DSQRT(4.000 - 2.000*alpha + alpha**2))
195     END FORALL
196 !-----
197 CASE DEFAULT
198     NIV = 0
199     FORALL (i=1:nelm)
200         Uoo(i) = 0.000
201         Uoo(i+nelm) = 0.000
202         Uoo(i+nelm+nelm) = 0.000
203     END FORALL
204 !-----
205     WRITE (Ulog,*)
206     WRITE (Ulog,*) ' Velocity menu:'
207     WRITE (Ulog,*) ' Chosen velocity field is not available; It was taken the zero velocity field.'
208 END SELECT
209 END SUBROUTINE Infinity_Flow
210 !=====
211 END MODULE Mod_Velocity_Menu
212
```

D:\Darth Vader\Escritorio\prtcl\_mkl\Mod\_VelField.F90 1

```
1 MODULE Mod_VelFieldTRM
2 =====
3 ! : Version: 0.7 created on -- / III / 2010
4 !
5 !
6 ! : Version: 1.0 created on 12 / 11 / 2012
7 !
8 !
9 ! This module was made to evaluate the solution of Two Roll Mill on surface.
10 =====
11 CONTAINS
12 =====
13 SUBROUTINE VelPsi(x00, y00, VelX, VelY)
14 !
15 ! This subroutine computes the velocity field of analytical solution of TRM.
16 !
17 ! USE Mod_SharedVars , ONLY: DBL, NCS, R, u, vR, ORY, StPnt, a, de, g, M1, M2, A0, B0, C0, D0, Kc, An, &
18 ! & Bn, Cn, Dn, lambda, shrate
19 ! USE Mod_TRMCoeffs
20 !
21 ! IMPLICIT NONE
22 =====
23 ! Variables
24 REAL (KIND = DBL), INTENT(IN) :: x00, y00 !Point's coordinates
25 REAL (KIND = DBL), INTENT(OUT) :: VelX, VelY !Velocities
26 =====
27 ! Variables inside the subroutine
28 INTEGER :: i !number of points
29 REAL (KIND = DBL) :: u, v !Components of velocity vector
30 REAL (KIND = DBL) :: co1, co2 !Complex componentes, co1, real, co2 imaginary
31 REAL (KIND = DBL) :: SumA, SumB !Sumatories to compute the velocity
32 REAL (KIND = DBL) :: x, y !Coordinates of p
33 ! REAL (KIND = DBL) :: VelX, VelY !Components of velocity at the end of computation
34 COMPLEX (KIND = DBL) :: Cmp !Complex number
35 =====
36 ! Initialize
37 x = x00
38 y = y00
39 SumA = x**2 + (y - a)**2
40 co1 = (x**2 + y**2 - a**2)
41 co2 = 2*a*x
42 =====
43 Cmp = DCMPLEX(co1,co2)/SumA
44 =====
45 v = DLOG((co1**2 + co2**2)/SumA**2)/2
46 u = DIMAG(DLOG(Cmp))
47 ! u = DLOG(DATAN(co2/co1))
48 =====
49 SumA = 0.0D0
50 SumB = 0.0D0
51 VelX = 0.0D0
52 VelY = 0.0D0
53 =====
54 IF (v < vr(1) .AND. v > vr(2)) THEN
55 !! DECS PARALLEL
56 DO i = NCS, 2, -1
57 SumA = SumA +
58 & (COS(1*u)*SIN(u) + i*(COSH(v) - COS(u))*SIN(1*u)) &
59 & (AN(1)*COSH((1 + i)*v) + Bn(1)*COSH((-1 + i)*v) + &
60 & Cn(1)*SINH((1 + i)*v) + Dn(1)*SINH((-1 + i)*v))
61
62 SumB = SumB + COS(1*u)*(COSH(v) - COS(u)) &
63 & ((1 + 1)*AN(1)*SINH((1 + 1)*v) + &
64 & (1 - 1)*Bn(1)*SINH((1 - 1)*v) + &
65 & (1 + 1)*Cn(1)*COSH((1 + 1)*v) + &
66 & (1 - 1)*Dn(1)*COSH((1 - 1)*v)) - &
67 & SINH(v)*(AN(1)*COSH((1 + 1)*v) + &
68 & Bn(1)*COSH((1 - 1)*v) + &
69 & Cn(1)*SINH((1 + 1)*v) + &
70 & Dn(1)*SINH((1 - 1)*v))
71 END DO
72 =====
```

D:\Darth Vader\Escritorio\prtcl\_mkl\Mod\_VelField.F90 2

```
73 VelX = -(SIN(u)*SINH(v)* &
74 & (-SIN(u)*(Kc*COS(u) + (A0 - Kc + Bn(1))*COSH(v) + &
75 & (C0 + D0*v)*SINH(v) + (An(1)*(COSH(v) + COSH(3*v))) + &
76 & Cn(1)*(SINH(v) + SINH(3*v))/2)) - &
77 & SumA)/(COSH(v) - COS(u)**2) + &
78 & ((-1 + COS(u)*COSH(v))** &
79 & (B0*(COS(u) - COSH(v)) - 2*Cn(1)*COS(u)* &
80 & COSH(2*v) - (A0 + D0)*SINH(v) - Kc*SINH(v) - &
81 & COSH(v)**(C0 + D0*v + A*AN(1)*COS(u)* &
82 & SINH(v) + (SINH(v)*(A0*COSH(v) + &
83 & C0*SINH(v) + D0*v*SINH(v) + &
84 & COS(u)*(Bn(1) + An(1)*COSH(2*v) + &
85 & Cn(1)*SINH(2*v)))/(-COS(u) + COSH(v)) + &
86 & SumB/(COS(u) - COSH(v)))/(COSH(v) - COS(u));
87 =====
88 VelY = ((1 - COS(u)*COSH(v))** &
89 & (-SIN(u)*(Kc*COS(u) + (A0 - Kc + Bn(1))*COSH(v) + &
90 & (C0 + D0*v)*SINH(v) + &
91 & (An(1)*(COSH(v) + COSH(3*v)) + &
92 & Cn(1)*(SINH(v) + SINH(3*v))/2)) - &
93 & SumA)/(COSH(v) - COS(u)**2) - &
94 & (SIN(u)*SINH(v)*B0*(COS(u) - COSH(v)) - &
95 & 2*Cn(1)*COS(u)*COSH(2*v) - (A0 + D0)*SINH(v) - &
96 & Kc*SINH(v) - COSH(v)*(C0 + D0*v + &
97 & A*AN(1)*COS(u)*SINH(v) + &
98 & (SINH(v)*(A0*COSH(v) + C0*SINH(v) + &
99 & D0*v*SINH(v) + COS(u)*(Bn(1) + &
100 & An(1)*COSH(2*v) + Cn(1)*SINH(2*v)))/ &
101 & (-COS(u) + COSH(v)) + &
102 & SumB/(COS(u) - COSH(v)))/(COSH(v) - COS(u));
103 =====
104 ELSE IF (v >= vr(1)) THEN
105 VelX = -(y - ORY(1))**w(1)
106 VelY = x**w(1)
107 ELSE IF (v <= vr(2)) THEN
108 VelX = -(y - ORY(2))**w(2)
109 VelY = x**w(2)
110 END IF
111 =====
112 ! VelP = (/ VelX, VelY /)
113 !
114 END SUBROUTINE VelPsi
115 =====
116 !FUNCTION TRMill_velx (x)
117 ! This function computes the velocity component x based on analytical solution of TRM velocity field.
118 !
119 ! USE Mod_SharedVars, ONLY: DBL
120 ! USE Mod_TRMCoeffs
121 !
122 ! IMPLICIT NONE
123 !
124 ! Variables
125 ! REAL (KIND = DBL), DIMENSION(2), INTENT(IN) :: P !Point's coordinates
126 ! REAL (KIND = DBL), DIMENSION(2), INTENT(OUT) :: VelP !Velocities
127 !
128 ! Variables inside the subroutine
129 ! INTEGER :: n !number of points
130 ! REAL (KIND = DBL) :: u, v !Components of velocity vector
131 ! REAL (KIND = DBL) :: co1, co2 !Complex componentes, co1, real, co2 imaginary
132 ! REAL (KIND = DBL) :: SumA, SumB !Sumatories to compute the velocity
133 ! REAL (KIND = DBL) :: x, y !Coordinates of p
134 ! REAL (KIND = DBL) :: VelX, VelY !Components of velocity at the end of computation
135 ! COMPLEX (KIND = DBL) :: Cmp !Complex number
136 !
137 ! Initialize
138 ! x = P(1)
139 ! y = P(2)
140 ! SumA = x**2 + (y - a)**2
141 ! co1 = (x**2 + y**2 - a**2)
142 ! co2 = 2*a*x
143 !
144 ! Cmp = DCMPLEX(co1,co2)/SumA
```

```

145 !
146 ! v = DLOG((co1**2 + co2**2)/SumA**2)/2
147 ! u = DIMAG(CDLOG(Cmp))
148 ! u = DLOG(DATAN(co2/co1))
149 !
150 ! SumA = 0.000
151 ! SumB = 0.000
152 ! VeIX = 0.000
153 ! VeIY = 0.000
154 !
155 ! IF (v < vr(1) .AND. v > vr(2)) THEN
156 !! !DEC$ PARALLEL
157 ! DO n = NCS, 2, -1
158 ! SumA = SumA +
159 ! & (COS(n*u)*SIN(u) + n*(COSH(v) - COS(u))*SIN(n*u))* &
160 ! & (AN(n)*COSH((1 + n)*v) + Bn(n)*COSH((-1 + n)*v) + &
161 ! & Cn(n)*SINH((1 + n)*v) + Dn(n)*SINH((-1 + n)*v))
162 !
163 ! SumB = SumB + COS(n*u)*((COSH(v) - COS(u))* &
164 ! & ((n + 1)*AN(n)*SINH((n + 1)*v) + &
165 ! & (n - 1)*Bn(n)*SINH((n - 1)*v) + &
166 ! & (n + 1)*Cn(n)*COSH((n + 1)*v) + &
167 ! & (n - 1)*Dn(n)*COSH((n - 1)*v)) - &
168 ! & SINH(v)*(AN(n)*COSH((n + 1)*v) + &
169 ! & Bn(n)*COSH((n - 1)*v) + &
170 ! & Cn(n)*SINH((n + 1)*v) + &
171 ! & Dn(n)*SINH((n - 1)*v)))
172 ! END DO
173 !
174 ! VeIX = -(SIN(u)*SINH(v)* &
175 ! & (-SIN(u)*(Kc*COS(u) + (A0 - Kc + Bn(1))*COSH(v) + &
176 ! & (C0 + D0*v)*SINH(v) + (An(1)*(COSH(v) + COSH(3*v)) + &
177 ! & Cn(1)*(SINH(v) + SINH(3*v))/2)) - &
178 ! & SumA)/(COSH(v) - COS(u))**2) + &
179 ! & ((-1 + COS(u)*COSH(v))* &
180 ! & (B0*(COS(u) - COSH(v)) - 2*Cn(1)*COS(u)* &
181 ! & COSH(2*v) - (A0 + D0)*SINH(v) - Kc*SINH(v) - &
182 ! & COSH(v)*(C0 + D0*v + 4*An(1)*COS(u)* &
183 ! & SINH(v)) + (SINH(v)*(A0*COSH(v) + &
184 ! & C0*SINH(v) + D0*v*SINH(v) + &
185 ! & COS(u)*(Bn(1) + An(1)*COSH(2*v) + &
186 ! & Cn(1)*SINH(2*v)))/(-COS(u) + COSH(v)) + &
187 ! & SumB/(COS(u) - COSH(v)))/(COSH(v) - COS(u));
188 !
189 ! VeIY = ((1 - COS(u)*COSH(v))* &
190 ! & (-SIN(u)*(Kc*COS(u) + (A0 - Kc + Bn(1))*COSH(v) + &
191 ! & (C0 + D0*v)*SINH(v) + (An(1)*(COSH(v) + COSH(3*v)) + &
192 ! & Cn(1)*(SINH(v) + SINH(3*v))/2)) - &
193 ! & SumA)/(COSH(v) - COS(u))**2) - &
194 ! & (SIN(u)*SINH(v)*(B0*(COS(u) - COSH(v)) - &
195 ! & 2*Cn(1)*COS(u)*COSH(2*v) - (A0 + D0)*SINH(v) - &
196 ! & Kc*SINH(v) - COSH(v)*(C0 + D0*v + &
197 ! & 4*An(1)*COS(u)*SINH(v)) + &
198 ! & (SINH(v)*(A0*COSH(v) + C0*SINH(v) + &
199 ! & D0*v*SINH(v) + COS(u)*(Bn(1) + &
200 ! & An(1)*COSH(2*v) + Cn(1)*SINH(2*v)))/ &
201 ! & (-COS(u) + COSH(v)) + &
202 ! & SumB/(COS(u) - COSH(v)))/(COSH(v) - COS(u));
203 !
204 !
205 ! ELSE IF (v >= vr(1)) THEN
206 ! VeIX = -(y - ORY(1))*w(1)
207 ! VeIY = x*w(1)
208 ! ELSE IF (v <= vr(2)) THEN
209 ! VeIX = -(y - ORY(2))*w(2)
210 ! VeIY = x*w(2)
211 ! END IF
212 !
213 ! VeIP = (/ VeIX, VeIY /)
214 ! END FUNCTION TRM111_veIx
215 !
216 !=====

```

```

217 END MODULE Mod_VeIFieldTRM

```

D:\Darth Vader\Escritorio\prtcl mk1\Mod\_SNEDOS.f90 1

```
1 MODULE Mod_SNEDOS
2 =====
3 |Version 1.0      07 / January / 2013
4 |
5 |----- Alfredo Sanjuan Sanjuan, in others words, me :
6 | This module have the subroutines to obtain the position of the dorp in a new time t1.
7 |-----
8 CONTAINS
9 SUBROUTINE Euler_Method(p, Ve1, h, npts, vna)
10 =====
11 | This subroutine computes the normal vector of every node on the surface
12 |-----
13 USE Mod_Nodal_Interp
14 USE Mod_SharedVars , ONLY:DBL
15 |-----
16 IMPLICIT NONE
17 |-----
18 | Variables
19 |-----
20 INTEGER, INTENT(IN) :: npts          !number of nodes
21 REAL (KIND = DBL), INTENT(IN) :: h    !step of time
22 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: p    !table of position vector of all points
23 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: ve1    !arrays of the components of
24                                     !velocity over all points
25 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: vna    !arrays of the components of normal
26                                     !vector over all points
27 |-----
28 | Variables inside the subroutine
29 |-----
30 INTEGER :: i, j                    !Counters
31 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), :: p1    !Dummy array
32 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), :: f     !Dummy array
33 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0      !Dummy array
34 |-----
35 |-----
36 !Initialize
37 ALLOCATE(f0(npts),f(npts,3), p1(npts,3))
38 f0 = 0.000
39 f = 0.000
40 p1 = 0.000
41 |-----
42 FORALL (i=1:npts)
43   f0(i,1) = (0.000*vna(i,1))+(0.000*vna(i,2))+(0.000*vna(i,3))
44   f0(i) = ((Vel(i,1)*vna(i,1))+(Vel(i,2)*vna(i,2))+(Vel(i,3)*vna(i,3)))
45 END FORALL
46 |-----
47 FORALL (i=1:npts)
48   f(i,:) = h*vna(i,:)*f0(i)
49   f(i,2) = h*vna(i,2)*f0(i)
50   f(i,3) = h*vna(i,3)*f0(i)
51 END FORALL
52 |-----
53 !f=h*Ve1
54 |-----
55 p1 = p + f
56 p = p1
57 |-----
58 |-----
59 END SUBROUTINE Euler_Method
60 SUBROUTINE Euler_Method2(x0, y0, z0, &
61 & Velix, Velly, Vellz, &
62 & h, nelm)
63 |-----
64 | This subroutine computes the normal vector of every node on the surface
65 |-----
66 USE Mod_Nodal_Interp
67 USE Mod_SharedVars , ONLY:DBL, vnx0, vny0, vnz0
68 |-----
69 IMPLICIT NONE
70 |-----
71 | Variables
72 |-----
```

D:\Darth Vader\Escritorio\prtcl mk1\Mod\_SNEDOS.f90 2

```
73 INTEGER, INTENT(IN) :: nelm          !number of elements
74 REAL (KIND = DBL), INTENT(IN) :: h    !step of time
75 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0, y0, z0 !arrays of the components of
76                                     !position vectors
77 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: Velix, Velly, Vellz !arrays of the components of
78                                     !velocity over collocation points
79 |-----
80 | Variables inside the subroutine
81 |-----
82 INTEGER :: i, j                    !Counters
83 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01    !Dummy array
84 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fx, fy, fz      !Dummy array
85 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0              !Dummy array
86 |-----
87 |-----
88 !Initialize
89 ALLOCATE(f0(nelm),fx(nelm),fy(nelm),fz(nelm), x01(nelm), y01(nelm), z01(nelm))
90 f0 = 0.000
91 fx = 0.000
92 fy = 0.000
93 fz = 0.000
94 x01 = 0.000
95 y01 = 0.000
96 z01 = 0.000
97 |-----
98 | EULER METHOD
99 |-----
100 FORALL (i=1:nelm)
101   f0(i) = (Velix(i)*vnx0(i) + Velly(i)*vny0(i) + Vellz(i)*vnz0(i))
102 END FORALL
103 |-----
104 FORALL (i=1:nelm)
105   fx(i) = h*vnx0(i)*f0(i)
106   fy(i) = h*vny0(i)*f0(i)
107   fz(i) = h*vnz0(i)*f0(i)
108 END FORALL
109 |-----
110 | LAGRANGE METHOD
111 |-----
112 !FORALL (i=1:nelm)
113 !   fx(i) = h*Velix(i)
114 !   fy(i) = h*Velly(i)
115 !   fz(i) = h*Vellz(i)
116 !END FORALL
117 |-----
118 x0 = x01
119 y0 = y01
120 z0 = z01
121 |-----
122 x0 = x01
123 y0 = y01
124 z0 = z01
125 |-----
126 END SUBROUTINE Euler_Method2
127 |-----
128 |-----
129 SUBROUTINE Euler_Method3(x0, y0, z0, &
130 & Velix, Velly, Vellz, &
131 & h, nelm, tr)
132 |-----
133 | This subroutine computes the normal vector of every node on the surface
134 |-----
135 USE Mod_Nodal_Interp
136 USE Mod_SharedVars , ONLY:DBL, vnx0, vny0, vnz0, farel
137 |-----
138 IMPLICIT NONE
139 |-----
140 | Variables
141 |-----
142 INTEGER, INTENT(IN) :: nelm          !number of elements
143 REAL (KIND = DBL), INTENT(IN) :: h    !step of time
144 REAL (KIND = DBL), INTENT(IN) :: tr  !factor to mix eulerian and lagrangian
```

D:\Darth Vader\Escritorio\prtcl mk1\Mod SHEDOS.f90 3

```
145 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0, y0, z0 !arrays of the components of
146 !position vectors
147 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: Vel1x, Vel1y, Vel1z !arrays of the components of
148 !velocity over collocation points
149 !-----
150 ! Variables inside the subroutine
151 !-----
152 INTEGER :: i, j !counters
153 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01 !Dummy array
154 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe, fye, fze !Dummy array
155 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fx1, fy1, fz1 !Dummy array
156 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0 !Dummy array
157 !-----
158 ! Initialize
159 ALLOCATE (f0(nelm), fxe(nelm), fye(nelm), fze(nelm), fx1(nelm), fy1(nelm), fz1(nelm), x01(nelm), y01(nelm), z01
160 (nelm))
161 f0 = 0.000
162 fxe = 0.000
163 fye = 0.000
164 fze = 0.000
165 fx1 = 0.000
166 fy1 = 0.000
167 fz1 = 0.000
168 x01 = 0.000
169 y01 = 0.000
170 z01 = 0.000
171 farel = 0.000
172 !-----
173 ! EULER METHOD
174 !-----
175 FORALL (i=1:nelm)
176 f0(i) = (Vel1x(i)*vnx0(i) + Vel1y(i)*vny0(i) + Vel1z(i)*vnz0(i))
177 END FORALL
178 farel = f0
179 !-----
180 FORALL (i=1:nelm)
181 fxe(i) = h*vnx0(i)*f0(i)
182 fye(i) = h*vny0(i)*f0(i)
183 fze(i) = h*vnz0(i)*f0(i)
184 END FORALL
185 !-----
186 ! LAGRANGE METHOD
187 !-----
188 FORALL (i=1:nelm)
189 fx1(i) = h*Vel1x(i)
190 fy1(i) = h*Vel1y(i)
191 fz1(i) = h*Vel1z(i)
192 END FORALL
193 !-----
194 x01 = x0 + (tr*fx1) + (1.000-tr)*fxe
195 y01 = y0 + (tr*fy1) + (1.000-tr)*fye
196 z01 = z0 + (tr*fz1) + (1.000-tr)*fze
197 !-----
198 x0 = x01
199 y0 = y01
200 z0 = z01
201 !-----
202 END SUBROUTINE Euler_Method3
203 !-----
204 SUBROUTINE Euler_Method4(p, h, npts, tr)
205 !-----
206 ! This subroutine computes the normal vector of every node on the surface
207 !-----
208 USE Mod_Nodal_Interp
209 USE Mod_SharedVars , ONLY:DBL, Vel, vna
210 !-----
211 IMPLICIT NONE
212 !-----
213 ! Variables
214 !-----
```

D:\Darth Vader\Escritorio\prtcl mk1\Mod SHEDOS.f90 4

```
215 INTEGER, INTENT(IN) :: npts !number of nodes
216 REAL (KIND = DBL), INTENT(IN) :: h !step of time
217 REAL (KIND = DBL), INTENT(IN) :: tr !factor to mix eulerian and lagrangian
218 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: p !arrays of the components of
219 !position vectors
220 ! Variables inside the subroutine
221 !-----
222 INTEGER :: i, j !counters
223 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: px1, py1, pz1 !Dummy array
224 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe, fye, fze !Dummy array
225 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fx1, fy1, fz1 !Dummy array
226 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0 !Dummy array
227 !-----
228 ! Initialize
229 ALLOCATE (f0(npts), fxe(npts), fye(npts), fze(npts), fx1(npts), fy1(npts), fz1(npts), px1(npts), py1(npts), pz1(npts))
230 f0 = 0.000
231 fxe = 0.000
232 fye = 0.000
233 fze = 0.000
234 fx1 = 0.000
235 fy1 = 0.000
236 fz1 = 0.000
237 px1 = 0.000
238 py1 = 0.000
239 pz1 = 0.000
240 !-----
241 ! EULER METHOD
242 !-----
243 FORALL (i=1:npts)
244 f0(i) = (vel(i,1)*vna(1,1) + Vel(i,2)*vna(1,2) + Vel(i,3)*vna(1,3))
245 END FORALL
246 !-----
247 fxe(:) = h*vna(:,1)*f0(:)
248 fye(:) = h*vna(:,2)*f0(:)
249 fze(:) = h*vna(:,3)*f0(:)
250 !-----
251 ! LAGRANGE METHOD
252 !-----
253 fx1(:) = h*Vel(:,1)
254 fy1(:) = h*Vel(:,2)
255 fz1(:) = h*Vel(:,3)
256 !-----
257 px1 = p(:,1) + (tr*fx1) + (1.000-tr)*fxe
258 py1 = p(:,2) + (tr*fy1) + (1.000-tr)*fye
259 pz1 = p(:,3) + (tr*fz1) + (1.000-tr)*fze
260 !-----
261 p(:,1) = px1
262 p(:,2) = py1
263 p(:,3) = pz1
264 !-----
265 END SUBROUTINE Euler_Method4
266 !-----
267 SUBROUTINE Euler_MethodM(x0, y0, z0, &
268 & Vel1x, Vel1y, Vel1z, &
269 & Vel1x, Vel1y, Vel1z, &
270 & h, nelM, tr)
271 !-----
272 ! This subroutine computes the normal vector of every node on the surface
273 !-----
274 USE Mod_Nodal_Interp
275 USE Mod_SharedVars , ONLY:DBL, vnx0, vny0, vnz0, vnx02, vny02, vnz02, farel
276 !-----
277 IMPLICIT NONE
278 !-----
279 ! Variables
280 !-----
281 INTEGER, INTENT(IN) :: nelM !number of elements
282 REAL (KIND = DBL), INTENT(IN) :: h !step of time
283 REAL (KIND = DBL), INTENT(IN) :: tr !factor to mix eulerian and lagrangian
```

D:\Darth Vader\Escritorio\prtcl mk1\Mod SHEDOS.f90 5

```
285 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0, y0, z0 !arrays of the components of
286 !position vectors
287 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: Vel1x, Vel1y, Vel1z !arrays of the components of
288 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: Vel2x, Vel2y, Vel2z !arrays of the components of
289 !velocity over collocation points
290 !-----
291 ! Variables inside the subroutine
292 !-----
293 INTEGER :: i, j !Counters
294 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01 !Dummy array
295 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe1, fye1, fze1 !Dummy array
296 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fx11, fy11, fz11 !Dummy array
297 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f01 !Dummy array
298 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe2, fye2, fze2 !Dummy array
299 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fx12, fy12, fz12 !Dummy array
300 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f02 !Dummy array
301 !-----
302 !Initialize
303 ALLOCATE(f01(nelm), f02(nelm), fxe1(nelm), fye1(nelm), fze1(nelm), fx11(nelm), fy11(nelm), fz11(nelm), fxe2(nelm),
304 fye2(nelm), fze2(nelm), fx12(nelm), fy12(nelm), fz12(nelm), x01(nelm), y01(nelm), z01(nelm))
305 f01 = 0.000
306 fxe1 = 0.000
307 fye1 = 0.000
308 fze1 = 0.000
309 fx11 = 0.000
310 fy11 = 0.000
311 fz11 = 0.000
312 f02 = 0.000
313 fxe2 = 0.000
314 fye2 = 0.000
315 fze2 = 0.000
316 fx12 = 0.000
317 fy12 = 0.000
318 fz12 = 0.000
319 x01 = 0.000
320 y01 = 0.000
321 z01 = 0.000
322 farel = 0.000
323 !-----
324 ! EULER METHOD
325 !-----
326 FORALL (i=1:nelm)
327 f01(i) = (Vel1x(i)*vnx0(i) + Vel1y(i)*vny0(i) + Vel1z(i)*vnz0(i))
328 f02(i) = (Vel2x(i)*vnx02(i) + Vel2y(i)*vny02(i) + Vel2z(i)*vnz02(i))
329 END FORALL
330 farel = 0.500*(f01+f02)
331 !-----
332 FORALL (i=1:nelm)
333 fxe1(i) = 0.500*h*vnx0(i)*f01(i)
334 fye1(i) = 0.500*h*vny0(i)*f01(i)
335 fze1(i) = 0.500*h*vnz0(i)*f01(i)
336 fxe2(i) = 0.500*h*vnx02(i)*f02(i)
337 fye2(i) = 0.500*h*vny02(i)*f02(i)
338 fze2(i) = 0.500*h*vnz02(i)*f02(i)
339 END FORALL
340 !-----
341 ! LAGRANGE METHOD
342 !-----
343 FORALL (i=1:nelm)
344 fx11(i) = 0.500*h*Vel1x(i)
345 fy11(i) = 0.500*h*Vel1y(i)
346 fz11(i) = 0.500*h*Vel1z(i)
347 fx12(i) = 0.500*h*Vel2x(i)
348 fy12(i) = 0.500*h*Vel2y(i)
349 fz12(i) = 0.500*h*Vel2z(i)
350 END FORALL
351 !-----
352 x01= x0 + (tr*(fx11+fx12)) + (1.000-tr)*(fxe1+fxe2)
353 y01= y0 + (tr*(fy11+fy12)) + (1.000-tr)*(fye1+fye2)
354 z01= z0 + (tr*(fz11+fz12)) + (1.000-tr)*(fze1+fze2)
355 !-----
```

D:\Darth Vader\Escritorio\prtcl mk1\Mod SHEDOS.f90 6

```
356 x0= x01
357 y0= y01
358 z0= z01
359 !-----
360 END SUBROUTINE Euler_MethodM
361 !-----
362 SUBROUTINE RK3(x0, y0, z0, h,
363 & Vel1x, Vel1y, Vel1z,
364 & h, nelm)
365 !-----
366 ! This subroutine computes the normal vector of every node on the surface using a Runge-Kutta of order 3
367 !-----
368 !-----
369 USE Mod_Nodal_Interp
370 USE Mod_SharedVars , ONLY:DBL, vnx0, vny0, vnz0
371 !-----
372 IMPLICIT NONE
373 !-----
374 ! Variables
375 !-----
376 INTEGER, INTENT(IN) :: nelm !number of elements
377 REAL (KIND = DBL), INTENT(IN) :: h !step of time
378 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0, y0, z0 !arrays of the components of
379 !position vectors
380 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: Vel1x, Vel1y, Vel1z !arrays of the components of
381 !velocity over collocation points
382 !-----
383 ! Variables inside the subroutine
384 !-----
385 INTEGER :: i, j !Counters
386 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01 !Dummy array
387 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fx, fy, fz !Dummy array
388 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: k1x, k2x, k3x !Dummy array
389 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: k1y, k2y, k3y !Dummy array
390 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: k1z, k2z, k3z !Dummy array
391 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0 !Dummy array
392 !-----
393 !Initialize
394 ALLOCATE(f0(nelm), fx(nelm), fy(nelm), fz(nelm), x01(nelm), y01(nelm), z01(nelm))
395 f0 = 0.000
396 fx = 0.000
397 fy = 0.000
398 fz = 0.000
399 x01 = 0.000
400 y01 = 0.000
401 z01 = 0.000
402 !-----
403 ALLOCATE(k1x(nelm), k2x(nelm), k3x(nelm), k1y(nelm), k2y(nelm), k3y(nelm), k1z(nelm), k2z(nelm), k3z(nelm))
404 k1x = 0.000
405 k2x = 0.000
406 k3x = 0.000
407 k1y = 0.000
408 k2y = 0.000
409 k3y = 0.000
410 k1z = 0.000
411 k2z = 0.000
412 k3z = 0.000
413 !-----
414 ! RUNGE-KUTTA OF THIRD ORDER RK3
415 ! EULERIAN METHOD
416 !-----
417 FORALL (i=1:nelm)
418 f0(i) = (Vel1x(i)*vnx0(i) + Vel1y(i)*vny0(i) + Vel1z(i)*vnz0(i))
419 END FORALL
420 !-----
421 FORALL (i=1:nelm)
422 fx(i) = h*vnx0(i)*f0(i)
423 fy(i) = h*vny0(i)*f0(i)
424 fz(i) = h*vnz0(i)*f0(i)
425 END FORALL
426 !-----
```

```

427 !=====
428 ! LAGRANGIAN METHOD
429 !=====
430 !First we calculate K1
431 !-----
432 FORALL (i=1:nelm)
433   k1x(i) = velli(i)
434   k1y(i) = velli(i)
435   k1z(i) = velli(i)
436 END FORALL
437 !=====
438 !Then we calculate K2
439 !-----
440 FORALL (i=1:nelm)
441   k2x(i) = (x0(i)) + ((h/3.000)*k1x(i))
442   k2y(i) = (y0(i)) + ((h/3.000)*k1y(i))
443   k2z(i) = (z0(i)) + ((h/3.000)*k1z(i))
444 END FORALL
445 !=====
446 !Finally we calculate K3
447 !-----
448 FORALL (i=1:nelm)
449   k3x(i) = (x0(i)) + ((h*2.000/3.000)*k2x(i))
450   k3y(i) = (y0(i)) + ((h*2.000/3.000)*k2y(i))
451   k3z(i) = (z0(i)) + ((h*2.000/3.000)*k2z(i))
452 END FORALL
453 !=====
454 x01= x0 + ((h/4.000)*(k1x+k3x))
455 y01= y0 + ((h/4.000)*(k1y+k3y))
456 z01= z0 + ((h/4.000)*(k1z+k3z))
457 !=====
458 x0= x01
459 y0= y01
460 z0= z01
461 !-----
462 END SUBROUTINE RK3
463 !=====
464 SUBROUTINE Euler_Met5(p, h, K1, npts,tr)
465 ! This subroutine computes the normal vector of every node on the surface
466 !
467 USE Mod_Nodal_Interp
468 USE Mod_SharedVars , ONLY:DBL, vna
469 ! IMPLICIT NONE
470 !
471 ! Variables
472 !-----
473 !
474 ! INTEGER, INTENT(IN) :: npts           !number of nodes
475 ! REAL (KIND = DBL), INTENT(IN) :: h    !step of time
476 ! REAL (KIND = DBL), INTENT(IN) :: tr   !factor to mix eulerian and lagrangian
477 !
478 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: K1 !arrays of the components of
479 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: p !arrays of the components of
480 !
481 ! Variables inside the subroutine
482 !-----
483 !
484 ! INTEGER :: i, j                       !Counters
485 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: px1, py1, pz1 !Dummy array
486 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe, fye, fze !Dummy array
487 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fx1, fy1, fz1 !Dummy array
488 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0 !Dummy array
489 !=====
490 !Initialize
491 ALLOCATE(f0(npts),fxe(npts),fye(npts),fze(npts),fx1(npts),fy1(npts),fz1(npts),px1(npts),py1(npts),pz1(npts))
492 f0 = 0.000
493 fxe = 0.000
494 fye = 0.000
495 fze = 0.000
496 fx1 = 0.000
497 fy1 = 0.000

```

```

498 fz1 = 0.000
499 px1 = 0.000
500 py1 = 0.000
501 pz1 = 0.000
502 !-----
503 ! EULER METHOD
504 !-----
505 FORALL (i=1:npts)
506   f0(i) = (K1(i,1)*vna(i,1) + K1(i,2)*vna(i,2) + K1(i,3)*vna(i,3))
507 END FORALL
508 !-----
509 fxe(:) = h*vna(:,1)*f0(:)
510 fye(:) = h*vna(:,2)*f0(:)
511 fze(:) = h*vna(:,3)*f0(:)
512 !-----
513 ! LAGRANGE METHOD
514 !-----
515 fx1(:) = h*K1(:,1)
516 fy1(:) = h*K1(:,2)
517 fz1(:) = h*K1(:,3)
518 !-----
519 px1= p(:,1) + (tr*fx1) + (1.000-tr)*fxe
520 py1= p(:,2) + (tr*fy1) + (1.000-tr)*fye
521 pz1= p(:,3) + (tr*fz1) + (1.000-tr)*fze
522 !-----
523 p(:,1)= px1
524 p(:,2)= py1
525 p(:,3)= pz1
526 !-----
527 END SUBROUTINE Euler_Met5
528 !=====
529 !
530 SUBROUTINE Euler_Met6(p, h, K1, K2, npts,tr)
531 ! This subroutine computes the normal vector of every node on the surface
532 !
533 USE Mod_Nodal_Interp
534 USE Mod_SharedVars , ONLY:DBL, vna
535 ! IMPLICIT NONE
536 !
537 ! Variables
538 !-----
539 !
540 ! INTEGER, INTENT(IN) :: npts           !number of nodes
541 ! REAL (KIND = DBL), INTENT(IN) :: h    !step of time
542 ! REAL (KIND = DBL), INTENT(IN) :: tr   !factor to mix eulerian and lagrangian
543 !
544 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: K1, K2 !arrays of the components of
545 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: p !arrays of the components of
546 !
547 ! Variables inside the subroutine
548 !-----
549 !
550 ! INTEGER :: i, j                       !Counters
551 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: px1, py1, pz1 !Dummy array
552 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe, fye, fze !Dummy array
553 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fx1, fy1, fz1 !Dummy array
554 ! REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0 !Dummy array
555 !
556 ! Initialize
557 ALLOCATE(f0(npts),fxe(npts),fye(npts),fze(npts),fx1(npts),fy1(npts),fz1(npts),px1(npts),py1(npts),pz1(npts))
558 f0 = 0.000
559 fxe = 0.000
560 fye = 0.000
561 fze = 0.000
562 fx1 = 0.000
563 fy1 = 0.000
564 fz1 = 0.000
565 px1 = 0.000
566 py1 = 0.000
567

```



```

568 pz1 = 0.000
569 Vel= 2*K2-K1
570 =====
571 ! EULER METHOD
572 =====
573 FORALL (i=1:npts)
574   f0(i)= (Vel(i,1)*vna(i,1) + Vel(i,2)*vna(i,2) + Vel(i,3)*vna(i,3))
575 END FORALL
576 =====
577 fxe(:) = h*vna(:,1)*f0(:)
578 fye(:) = h*vna(:,2)*f0(:)
579 fze(:) = h*vna(:,3)*f0(:)
580 =====
581 ! LAGRANGE METHOD
582 =====
583 fx1(:) = h*Vel(:,1)
584 fy1(:) = h*Vel(:,2)
585 fz1(:) = h*Vel(:,3)
586 =====
587 px1= p(:,1) + (tr*fx1) + (1.000-tr)*fxe
588 py1= p(:,2) + (tr*fy1) + (1.000-tr)*fye
589 pz1= p(:,3) + (tr*fz1) + (1.000-tr)*fze
590 =====
591 p(:,1)= px1
592 p(:,2)= py1
593 p(:,3)= pz1
594 =====
595 END SUBROUTINE Euler_Met6
596 =====
597 SUBROUTINE KMaker(xel, yel, zel, &
598   & K1, K2, K3, K4, nelM, h)
599 =====
600 ! This subroutine computes the normal vector of every node on the surface
601 =====
602 USE Mod_SharedVars , ONLY:DBL !, vnx0, vny0, vnz0, farel
603 =====
604 IMPLICIT NONE
605 =====
606 ! Variables
607 =====
608 INTEGER, INTENT(IN) :: nelM      !number of elements
609 REAL (KIND = DBL), INTENT(IN) :: h      !step of time
610 ! REAL (KIND = DBL), INTENT(IN) :: tr      !factor to mix eulerian and lagrangian ✓
611 =====
612 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: xel, yel, zel !arrays of the components of
613   !position vectors
614 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: K1, K2, K3, K4 !arrays of the components of
615   !position vectors
616 =====
617 ! Variables inside the subroutine
618 =====
619 INTEGER :: i, j      !Counters
620 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01 !Dummy array
621 =====
622 ALLOCATE(x01(nelM),y01(nelM),z01(nelM))
623 x01 = 0.000
624 y01 = 0.000
625 z01 = 0.000
626 =====
627 FORALL (i=1:nelM)
628   x01(i) = xel(i)+ (h/6.000)*(K1(i) +2.000*K2(i) +2.000*K3(i) +K4(i))
629   y01(i) = yel(i)+ (h/6.000)*(K1(i)+nelM +2.000*K2(i+nelM) +2.000*K3(i+nelM) +K4(i+nelM))
630   z01(i) = zel(i)+ (h/6.000)*(K1(i+nelM+nelM)+2.000*K2(i+nelM+nelM)+2.000*K3(i+nelM+nelM)+K4(i+nelM+nelM))
631 END FORALL
632 =====
633 xel = x01
634 yel = y01
635 zel = z01
636 =====
637 END SUBROUTINE KMaker
638 =====
639 SUBROUTINE KMK3(pe1, &

```

```

639   & K1, K2, K3, npts, h)
640 =====
641 ! This subroutine computes the normal vector of every node on the surface
642 =====
643 USE Mod_SharedVars , ONLY:DBL !, vnx0, vny0, vnz0, farel
644 =====
645 IMPLICIT NONE
646 =====
647 ! Variables
648 =====
649 INTEGER, INTENT(IN) :: npts      !number of elements
650 REAL (KIND = DBL), INTENT(IN) :: h      !step of time
651 ! REAL (KIND = DBL), INTENT(IN) :: tr      !factor to mix eulerian and lagrangian ✓
652 =====
653 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: pe1 !arrays of the components of
654   !position vectors
655 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: K1, K2, K3 !arrays of the components of
656   !position vectors
657 =====
658 ! Variables inside the subroutine
659 =====
660 INTEGER :: i, j      !Counters
661 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), :: p01 !Dummy array
662 =====
663 ALLOCATE(p01(npts,3))
664 p01 = 0.000
665 =====
666 FORALL (i=1:npts)
667   p01(i,:) = pe1(i,:) + (h/6.000)*(K1(i,:)+4.000*K2(i,:)+K3(i,:))
668 END FORALL
669 =====
670 pe1 = p01
671 END SUBROUTINE KMK3
672 =====
673 SUBROUTINE RK31(x0, y0, z0, h, K1, nelM, tr)
674 =====
675 ! This subroutine computes the normal vector of every node on the surface
676 =====
677 USE Mod_Nodal_Interp
678 USE Mod_SharedVars , ONLY:DBL, vnx0, vny0, vnz0
679 =====
680 IMPLICIT NONE
681 =====
682 ! Variables
683 =====
684 INTEGER, INTENT(IN) :: nelM      !number of nodes
685 REAL (KIND = DBL), INTENT(IN) :: h      !step of time
686 ! REAL (KIND = DBL), INTENT(IN) :: tr      !factor to mix eulerian and lagrangian ✓
687 =====
688 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: K1 !arrays of the components of
689 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0, y0, z0 !arrays of the components of
690 =====
691 ! Variables inside the subroutine
692 =====
693 INTEGER :: i, j      !Counters
694 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: px1, py1, pz1 !Dummy array
695 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe, fye, fze !Dummy array
696 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fx1, fy1, fz1 !Dummy array
697 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0 !Dummy array
698 =====
699 !Initialize
700 ALLOCATE(f0(nelM),fxe(nelM),fye(nelM),fze(nelM),fx1(nelM),fy1(nelM),fz1(nelM),px1(nelM),py1(nelM),pz1(nelM))
701 f0 = 0.000
702 fxe = 0.000
703 fye = 0.000
704 fze = 0.000
705 fx1 = 0.000
706 fy1 = 0.000
707 fz1 = 0.000
708 px1 = 0.000

```

```

709 py1 = 0.000
710 pz1 = 0.000
711 =====
712 ! EULER METHOD
713 !=====
714 FORALL (i=1:nelm)
715   f0(i) = (K1(i)*vnx0(i) + K1(i+nelm)*vny0(i) + K1(i+nelm+nelm)*vnz0(i))
716 END FORALL
717 !=====
718 fxe(:) = h*vnx0(:)*f0(:)
719 fye(:) = h*vny0(:)*f0(:)
720 fze(:) = h*vnz0(:)*f0(:)
721 !=====
722 ! LAGRANGE METHOD
723 !=====
724 FORALL (i=1:nelm)
725   fx1(i) = h*K1(i)
726   fy1(i) = h*K1(i+nelm)
727   fz1(i) = h*K1(i+nelm+nelm)
728 END FORALL
729 !=====
730 px1= x0 + (tr*fx1) + (1.000-tr)*fye
731 py1= y0 + (tr*fy1) + (1.000-tr)*fye
732 pz1= z0 + (tr*fz1) + (1.000-tr)*fze
733 !=====
734 x0= px1
735 y0= py1
736 z0= pz1
737 !=====
738 END SUBROUTINE RK31
739 !=====
740 !=====
741 SUBROUTINE RK32(x0, y0, z0, h, K1, K2, nelm, tr)
742 ! This subroutine computes the normal vector of every node on the surface
743 !=====
744 USE Mod_Nodal_interp
745 USE Mod_SharedVars , ONLY:DBL, vnx0, vny0, vnz0
746 !=====
747 IMPLICIT NONE
748 !=====
749 ! Variables
750 !=====
751 !=====
752 INTEGER, INTENT(IN) :: nelm !number of nodes
753 REAL (KIND = DBL), INTENT(IN) :: h !step of time
754 REAL (KIND = DBL), INTENT(IN) :: tr !factor to mix eulerian and lagrangian
755 !=====
756 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: K1, K2 !arrays of the components of
757 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0,y0,z0 !arrays of the components of
758 ! Variables inside the subroutine
759 !=====
760 INTEGER :: i, j !counters
761 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: px1, py1, pz1 !Dummy array
762 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe, fye, fze !Dummy array
763 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fx1, fy1, fz1 !Dummy array
764 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: f0 !Dummy array
765 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: vel !Dummy array
766 !=====
767 ! initialize
768 !=====
769 ALLOCATE(f0(nelm),fxe(nelm),fye(nelm),fze(nelm),fx1(nelm),fy1(nelm),fz1(nelm),px1(nelm),py1(nelm),pz1(nelm),
770 vel(3*nelm))
771 f0 = 0.000
772 fxe = 0.000
773 fye = 0.000
774 fze = 0.000
775 fx1 = 0.000
776 fy1 = 0.000
777 fz1 = 0.000
778 px1 = 0.000
779 py1 = 0.000

```

```

779 pz1 = 0.000
780 vel= 2*K2-K1
781 =====
782 ! EULER METHOD
783 !=====
784 FORALL (i=1:nelm)
785   f0(i) = (vel(i)*vnx0(i) + vel(i+nelm)*vny0(i) + vel(i+nelm+nelm)*vnz0(i))
786 END FORALL
787 !=====
788 fxe(:) = h*vnx0(:)*f0(:)
789 fye(:) = h*vny0(:)*f0(:)
790 fze(:) = h*vnz0(:)*f0(:)
791 !=====
792 ! LAGRANGE METHOD
793 !=====
794 FORALL (i=1:nelm)
795   fx1(i) = h*vel(i)
796   fy1(i) = h*vel(i+nelm)
797   fz1(i) = h*vel(i+nelm+nelm)
798 END FORALL
799 !=====
800 px1= x0 + (tr*fx1) + (1.000-tr)*fye
801 py1= y0 + (tr*fy1) + (1.000-tr)*fye
802 pz1= z0 + (tr*fz1) + (1.000-tr)*fze
803 !=====
804 x0= px1
805 y0= py1
806 z0= pz1
807 !=====
808 END SUBROUTINE RK32
809 !=====
810 !=====
811 SUBROUTINE RK3(x0, y0, z0, h, K1, K2, K3, nelm, h)
812 ! This subroutine computes the normal vector of every node on the surface
813 !=====
814 USE Mod_SharedVars , ONLY:DBL !, vnx0, vny0, vnz0, farel
815 !=====
816 IMPLICIT NONE
817 !=====
818 ! Variables
819 !=====
820 !=====
821 INTEGER, INTENT(IN) :: nelm !number of elements
822 REAL (KIND = DBL), INTENT(IN) :: h !step of time
823 REAL (KIND = DBL), INTENT(IN) :: tr !factor to mix eulerian and lagrangian
824 !=====
825 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0,y0,z0 !arrays of the components of
826 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: K1, K2, K3 !arrays of the components of
827 ! position vectors
828 ! Variables inside the subroutine
829 !=====
830 INTEGER :: i, j !counters
831 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01 !Dummy array
832 !=====
833 ALLOCATE(x01(nelm), y01(nelm), z01(nelm))
834 x01 = 0.000
835 y01 = 0.000
836 z01 = 0.000
837 !=====
838 FORALL (i=1:nelm)
839   x01(i) = x0(i) + (h/6.000)*(K1(i)+4.000*K2(i)+K3(i))
840   y01(i) = y0(i) + (h/6.000)*(K1(i+nelm)+4.000*K2(i+nelm)+K3(i+nelm))
841   z01(i) = z0(i) + (h/6.000)*(K1(i+nelm+nelm)+4.000*K2(i+nelm+nelm)+K3(i+nelm+nelm))
842 END FORALL
843 !=====
844 x0 = x01
845 y0 = y01
846 z0 = z01

```

```

850 !-----
851 END SUBROUTINE RMK3
852 !-----
853 !-----
854 SUBROUTINE ABM4tem(x0, y0, z0, &
855 & Vel1x1, Veliy1, Vel1z1, &
856 & Vel1x2, Veliy2, Vel1z2, &
857 & Vel1x3, Veliy3, Vel1z3, &
858 & Vel1x4, Veliy4, Vel1z4, nelm, h)
859 !-----
860 ! This subroutine computes the normal vector of every node on the surface
861 !-----
862 USE Mod_SharedVars , ONLY:DBL !, vnx0, vny0, vnz0, farel
863 !-----
864 IMPLICIT NONE
865 !-----
866 ! Variables
867 !-----
868 INTEGER, INTENT(IN) :: nelm !number of elements
869 REAL (KIND = DBL), INTENT(IN) :: h !step of time
870 ! REAL (KIND = DBL), INTENT(IN) :: tr !factor to mix eulerian and lagrangian
871 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0, y0, z0 !arrays of the components of
872 ! position vectors
873 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: Vel1x1, Veliy1, Vel1z1, &
874 & Vel1x2, Veliy2, Vel1z2, &
875 & Vel1x3, Veliy3, Vel1z3, &
876 & Vel1x4, Veliy4, Vel1z4
877 !arrays of the components of
878 ! position vectors
879 !-----
880 ! Variables inside the subroutine
881 !-----
882 INTEGER :: i, j !counters
883 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01 !Dummy array
884 !-----
885 !
886 ALLOCATE(x01(nelm),y01(nelm),z01(nelm))
887 x01 = 0.000
888 y01 = 0.000
889 z01 = 0.000
890 !-----
891 FORALL (i=1:nelm)
892 x01(i) = x0(i)+ (h/24.000)*(-9.000*Vel1x1(i) +37.000*Vel1x2(i) -59.000*Vel1x3(i) +55.000*Vel1x4(i))
893 y01(i) = y0(i)+ (h/24.000)*(-9.000*Veliy1(i) +37.000*Veliy2(i) -59.000*Veliy3(i) +55.000*Veliy4(i))
894 z01(i) = z0(i)+ (h/24.000)*(-9.000*Vel1z1(i) +37.000*Vel1z2(i) -59.000*Vel1z3(i) +55.000*Vel1z4(i))
895 END FORALL
896 !-----
897 x0 = x01
898 y0 = y01
899 z0 = z01
900 END SUBROUTINE ABM4tem
901 !-----
902 SUBROUTINE ABM4(x0, y0, z0, &
903 & Vel1x1, Veliy1, Vel1z1, &
904 & Vel1x2, Veliy2, Vel1z2, &
905 & Vel1x3, Veliy3, Vel1z3, &
906 & Vel1x4, Veliy4, Vel1z4, nelm, h)
907 !-----
908 ! This subroutine computes the normal vector of every node on the surface
909 !-----
910 USE Mod_SharedVars , ONLY:DBL !, vnx0, vny0, vnz0, farel
911 !-----
912 IMPLICIT NONE
913 !-----
914 ! Variables
915 !-----
916 INTEGER, INTENT(IN) :: nelm !number of elements
917 REAL (KIND = DBL), INTENT(IN) :: h !step of time
918 ! REAL (KIND = DBL), INTENT(IN) :: tr !factor to mix eulerian and lagrangian

```

```

919 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: x0, y0, z0 !arrays of the components of
920 ! position vectors
921 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: Vel1x1, Veliy1, Vel1z1, &
922 & Vel1x2, Veliy2, Vel1z2, &
923 & Vel1x3, Veliy3, Vel1z3, &
924 & Vel1x4, Veliy4, Vel1z4
925 !arrays of the components of
926 ! position vectors
927 !-----
928 ! Variables inside the subroutine
929 !-----
930 INTEGER :: i, j !counters
931 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01 !Dummy array
932 !-----
933 !
934 ALLOCATE(x01(nelm),y01(nelm),z01(nelm))
935 x01 = 0.000
936 y01 = 0.000
937 z01 = 0.000
938 !-----
939 FORALL (i=1:nelm)
940 x01(i) = x0(i)+ (h/24.000)*(Vel1x1(i) -5.000*Vel1x2(i) +19.000*Vel1x3(i) +9.000*Vel1x4(i))
941 y01(i) = y0(i)+ (h/24.000)*(Veliy1(i) -5.000*Veliy2(i) +19.000*Veliy3(i) +9.000*Veliy4(i))
942 z01(i) = z0(i)+ (h/24.000)*(Vel1z1(i) -5.000*Vel1z2(i) +19.000*Vel1z3(i) +9.000*Vel1z4(i))
943 END FORALL
944 !-----
945 x0 = x01
946 y0 = y01
947 z0 = z01
948 END SUBROUTINE ABM4
949 !-----
950 SUBROUTINE ABM4vel(Vel1x, Veliy, Vel1z, &
951 & nelm, h, tr)
952 !-----
953 ! This subroutine computes the normal vector of every node on the surface
954 !-----
955 USE Mod_SharedVars , ONLY:DBL !, vnx0, vny0, vnz0 !, farel
956 !-----
957 IMPLICIT NONE
958 !-----
959 ! Variables
960 !-----
961 !
962 INTEGER, INTENT(IN) :: nelm !number of elements
963 REAL (KIND = DBL), INTENT(IN) :: h !step of time
964 REAL (KIND = DBL), INTENT(IN) :: tr !factor to mix eulerian and lagrangian
965 !-----
966 ! Variables inside the subroutine
967 !-----
968 INTEGER :: i, j !counters
969 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: x01, y01, z01, f0 !Dummy array
970 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fxe, fye, fze !Dummy array
971 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: fx1, fyl, fz1 !Dummy array
972 !-----
973 ALLOCATE(x01(nelm),y01(nelm),z01(nelm))
974 x01 = 0.000
975 y01 = 0.000
976 z01 = 0.000
977 !-----
978 ! Initialize
979 ALLOCATE(f0(nelm),fxe(nelm),fye(nelm),fze(nelm),fx1(nelm),fyl(nelm),fz1(nelm))
980 f0 = 0.000
981 fxe = 0.000
982 fye = 0.000
983 fze = 0.000
984 fxe = 0.000
985 fx1 = 0.000
986 fyl = 0.000
987 fz1 = 0.000
988 !-----
989 ! EULER METHOD

```

```
990 !=====
991   FORALL (i = 1:nelm)
992     f0(i) = (Velix(i)*vnx0(i) + Velly(i)*vny0(i) + Veliz(i)*vnz0(i))
993   END FORALL
994 !=====
995   FORALL (i = 1:nelm)
996     fxe(i) = vnx0(i)*f0(i)
997     fye(i) = vny0(i)*f0(i)
998     fze(i) = vnz0(i)*f0(i)
999   END FORALL
1000 !=====
1001 ! LAGRANGE METHOD
1002 !=====
1003   FORALL (i = 1:nelm)
1004     fx1(i) = Velix(i)
1005     fy1(i) = Velly(i)
1006     fz1(i) = Veliz(i)
1007   END FORALL
1008 !=====
1009   x01 = (tr*fx1) + (1.000-tr)*fxe
1010   y01 = (tr*fy1) + (1.000-tr)*fye
1011   z01 = (tr*fz1) + (1.000-tr)*fze
1012 !=====
1013   Velix = x01
1014   Velly = y01
1015   Veliz = z01
1016 !=====
1017   END SUBROUTINE ABM4vel
1018
1019 END MODULE Mod_SNEDOS
```

```

1 MODULE Mod_Correction
2 =====
3 !Version 1.0      27 / November / 2013      Alfredo Sanjuan Sanjuan, in others words, me :)
4 !=====
5 ! The first action is to generate a correction of drop'e volume.
6 !=====
7 CONTAINS
8 =====
9 SUBROUTINE Corrector_vol(p, npts, prt_vlm_n, cx, cy, cz)
10 =====
11 ! This subroutine
12 !=====
13 USE Mod_SharedVars , ONLY:DBL
14 =====
15 IMPLICIT NONE
16 =====
17 ! Variables
18 !=====
19 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: p !The matrix with the information of
20 INTEGER, INTENT(IN) :: npts !number of nodes
21 REAL (KIND = DBL), INTENT(IN) :: prt_vlm_n
22 REAL (KIND = DBL), INTENT(IN) :: cx, cy, cz !drop's centroid coordinates
23 !=====
24 ! Variables inside the subroutine
25 !=====
26 INTEGER :: i !Counters
27 =====
28 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: p0, p1 !Dummies array to adjust the position vectors.
29 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: r0, r1 !Dummies array to adjust the position vectors.
30 !=====
31 ! The adjust begins here
32 ALLOCATE(p0(npts,3), p1(npts,3),r0(npts), r1(npts))
33 p0 = 0.000
34 p1 = 0.000
35 r0 = 0.000
36 r1 = 0.000
37 !=====
38 FORALL (i=1:npts)
39 p0(i,1)= (p(i,1) - cx)
40 p0(i,2)= (p(i,2) - cy)
41 p0(i,3)= (p(i,3) - cz)
42 END FORALL
43 FORALL (i=1:npts)
44 r0(i)= DSQRT( p0(i,1)**2 + p0(i,2)**2 + p0(i,3)**2 )
45 END FORALL
46 FORALL (i=1:npts)
47 r1(i)= r0(i) / ((prt_vlm_n)**(1.000/3.000))
48 p1(i,:) = p0(i,:) / ((prt_vlm_n)**(1.000/3.000))
49 END FORALL
50 !=====
51 ! p(i,1)= (p1(i,1))/((prt_vlm_n)**(1.000/3.000)) !+cx
52 ! p(i,2)= (p1(i,2))/((prt_vlm_n)**(1.000/3.000)) !+cy
53 ! p(i,3)= (p1(i,3))/((prt_vlm_n)**(1.000/3.000)) !+cz
54 !=====
55 !p0=0.000
56 FORALL (i=1:npts)
57 ! p0(i,1)= p(i,1) +cx
58 ! p0(i,2)= p(i,2) +cy
59 ! p0(i,3)= p(i,3) +cz
60 !=====
61 p=p1
62 END SUBROUTINE Corrector_vol
63 =====
64 SUBROUTINE Corrector_vol2(vel1, nelm)
65 =====
66 ! This subroutine
67 =====

```

```

73 USE Mod_SharedVars , ONLY:DBL, vnx0, vny0, vnz0, arel
74 =====
75 IMPLICIT NONE
76 =====
77 ! Variables
78 !=====
79 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(INOUT) :: vel1 !The matrix with the information of
80 INTEGER, INTENT(IN) :: nelm !number of nodes
81 REAL (KIND = DBL), INTENT(IN) :: prt_vlm_n
82 REAL (KIND = DBL), INTENT(IN) :: cx, cy, cz !drop's centroid coordinates
83 !=====
84 ! Variables inside the subroutine
85 !=====
86 INTEGER :: i !Counters
87 =====
88 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:) :: vel, val !Dummies array to adjust the position vectors.
89 REAL (KIND = DBL) :: areas, surface !Dummies array to adjust the position vectors.
90 !=====
91 ! The adjust begins here
92 ALLOCATE(val(nelm), vel(3*nelm))
93 vel = 0.000
94 val = 0.000
95 surface = 0.000
96 areas = 0.000
97 !=====
98 FORALL (i=1:nelm)
99 vel(i)= vel1(i)*vnx0(i) + Vel1(i+nelm)*vny0(i) + Vel1(i+nelm*2)*vnz0(i)
100 END FORALL
101 FORALL (i=1:nelm)
102 vel(i) = val(i)*vnx0(i)
103 vel(i+nelm) = val(i)*vny0(i)
104 vel(i+nelm*2) = val(i)*vnz0(i)
105 END FORALL
106 !=====
107 areas = SUM(arel)
108 surface = SUM(val)
109 !=====
110 WRITE(*,*) areas
111 WRITE(*,*) surface
112 WRITE(*,*) surface/areas
113 WRITE(*,*)
114 !=====
115 !FORALL (i=1:nelm)
116 ! Vel(i)= vel(i)
117 !=====
118 END SUBROUTINE Corrector_vol2
119 =====
120 SUBROUTINE D_taylor(p, npts, DT, DTm, DTmm, lmajor, bminor, wminor, angulo, cx, cy, cz, req)
121 =====
122 ! This subroutine
123 !=====
124 USE Mod_SharedVars , ONLY:DBL, pi
125 =====
126 ! Variables
127 !=====
128 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:), INTENT(IN) :: p !The matrix with the information of
129 INTEGER, INTENT(IN) :: npts !number of nodes
130 REAL (KIND = DBL), INTENT(IN) :: cx, cy, cz, req !drop's centroid coordinates and r0
131 REAL (KIND = DBL), INTENT(OUT) :: DT, DTm, DTmm, lmajor, bminor, wminor, angulo
132 !=====
133 ! Variables inside the subroutine
134 !=====
135 INTEGER :: i, j !Counters
136 =====
137 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:,:) :: xp !Dummies array to adjust the position vectors.
138 !=====
139 ! The adjust begins here
140 ALLOCATE(xp(npts,3))

```

```
145 xp = 0.000
146 lmajor = 0.000
147 bminor = 0.000
148 wminor = 0.000
149 angulo = 0.000
150 DT = 0.000
151 DTm = 0.000
152 -----
153 FORALL (i=1:npts)
154   xp(i,1) = DSQRT( (p(i,1)-cx)**2 + (p(i,2)-cy)**2 + (p(i,3)-cz)**2)
155   xp(i,2) = DSQRT(p(i,3)-cz)**2
156   xp(i,3) = DATAN((p(i,2)-cy)/(p(i,1)-cx))
157 END FORALL
158
159 lmajor = (MAXVAL(xp(:,1)))/req
160 bminor = (MINVAL(xp(:,1)))/req
161 wminor = (MAXVAL(xp(:,2)))/req
162
163 angulo = (xp(MAXLOC(xp(:,1),1),3)*180.000)/pi
164 DT = ((lmajor - bminor)/(lmajor + bminor))
165 DTm = ((lmajor - wminor)/(lmajor + wminor))
166 DTmm = ((wminor - bminor)/(wminor + bminor))
167 !-----
168 END SUBROUTINE D_taylor
169 !-----
170 END MODULE Mod_Correction
```

D:\Darth Vader\Escritorio\prtcl\_mkl\Mod Data Files.f90 1

```
1 MODULE Mod_Data_Files
2 |-----|
3 | Version: 0.9 created on 21 / 10 / 2013                               Alfredo Sanjuan Sanjuan
4 |-----|
5 | Version: 1.0 created on 14 / 01 / 2015                               Alfredo Sanjuan Sanjuan
6 |-----|
7 |-----|
8 CONTAINS
9 |-----|
10 SUBROUTINE write_datan(nelm, stride)
11 |-----|
12 | This subroutine write the data of nodes of the mesh in a file
13 | This subroutine applies to every step of time.
14 |-----|
15 USE Mod_SharedVars, ONLY: DBL, p, n
16 |-----|
17 IMPLICIT NONE
18 |-----|
19 | Variables
20 | Variables inside the subroutine
21 |-----|
22 INTEGER,          INTENT(IN) :: nelm          !number of points
23 INTEGER,          INTENT(IN) :: stride        !
24 |-----|
25 | Variables inside the subroutine
26 |-----|
27 CHARACTER (len=35) :: fname
28 CHARACTER (len=19) :: dname
29 CHARACTER (len= 4) :: ename
30 CHARACTER (len= 7) :: num
31 |-----|
32 INTEGER :: i, j                                !counters
33 INTEGER :: i1, i2, i3, i4, i5, i6             !
34 INTEGER :: select                             !selection
35 INTEGER :: fstatus                            !fstatus
36 |-----|
37 | First, the information of nelm and stride are transformed into character variables.
38 j = 1000000 + stride
39 WRITE(num,'(I7)') j
40 ename = '.dat'
41 |-----|
42 | Then, DOOM SUBROUTINE is called.
43 | CALL doom(dname)
44 dname = TRIM('Data\Position\nodes')
45 |-----|
46 | The name of file is made.
47 |-----|
48 fname = TRIM(TRIM(dname)//TRIM(num)//ename)
49 |-----|
50 OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
51 |-----|
52 IF (fstatus == 0) THEN
53 WRITE(500,*) nelm
54 WRITE(500,*) j
55 |-----|
56 DO i = 1, nelm
57 i1 = n(1,1)
58 i2 = n(1,2)
59 i3 = n(1,3)
60 i4 = n(1,4)
61 i5 = n(1,5)
62 i6 = n(1,6)
63 WRITE (500,103) p(i1,1),p(i1,2),p(i1,3)
64 WRITE (500,103) p(i2,1),p(i2,2),p(i2,3)
65 WRITE (500,103) p(i3,1),p(i3,2),p(i3,3)
66 WRITE (500,103) p(i4,1),p(i4,2),p(i4,3)
67 WRITE (500,103) p(i5,1),p(i5,2),p(i5,3)
68 WRITE (500,103) p(i6,1),p(i6,2),p(i6,3)
69 END DO
70 END IF
71 |-----|
72 |-----|
```

D:\Darth Vader\Escritorio\prtcl\_mkl\Mod Data Files.f90 2

```
73 CLOSE(UNIT = 500)
74 |-----|
75 103 FORMAT(10(1x,E24.16))
76 |-----|
77 END SUBROUTINE write_datan
78 |-----|
79 |-----|
80 SUBROUTINE write_data(nelm, stride)
81 |-----|
82 | This subroutine write the data of collocation points if the mesh in a file
83 | This subroutine applies to every step of time.
84 |-----|
85 USE Mod_SharedVars, ONLY: DBL, x0, y0, z0
86 |-----|
87 IMPLICIT NONE
88 |-----|
89 | Variables
90 | Variables inside the subroutine
91 |-----|
92 INTEGER,          INTENT(IN) :: nelm          !number of points
93 INTEGER,          INTENT(IN) :: stride        !counter
94 |-----|
95 | Variables inside the subroutine
96 |-----|
97 CHARACTER (len=35) :: fname
98 CHARACTER (len=19) :: dname
99 CHARACTER (len= 4) :: ename
100 CHARACTER (len= 7) :: num
101 |-----|
102 INTEGER :: i, j                                !counters
103 INTEGER :: i1, i2, i3, i4, i5, i6             !
104 INTEGER :: select                             !selection
105 INTEGER :: fstatus                            !fstatus
106 |-----|
107 | First, the information of nelm and stride are transformed into character variables.
108 j = 1000000 + stride
109 WRITE(num,'(I7)') j
110 ename = '.dat'
111 |-----|
112 dname = TRIM('Data\Positioc\collic')
113 |-----|
114 | The name of file is made.
115 |-----|
116 fname = TRIM(TRIM(dname)//TRIM(num)//ename)
117 |-----|
118 OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
119 |-----|
120 IF (fstatus == 0) THEN
121 WRITE(500,*) nelm
122 WRITE(500,*) j
123 |-----|
124 DO i = 1, nelm
125 WRITE (500,103) x0(i), y0(i), z0(i)
126 END DO
127 |-----|
128 CLOSE(UNIT = 500)
129 |-----|
130 103 FORMAT(10(1x,E24.16))
131 |-----|
132 END SUBROUTINE write_data
133 |-----|
134 |-----|
135 |-----|
136 SUBROUTINE write_data(nelm, stride)
137 |-----|
138 | This subroutine write the data of curvature in a file
139 | This subroutine applies to every step of time.
140 |-----|
141 USE Mod_SharedVars, ONLY: DBL, crvmel
142 |-----|
143 IMPLICIT NONE
144 |-----|
```



```

145 ! Variables
146 ! Variables inside the subroutine
147 -----
148 INTEGER, INTENT(IN) :: nelm           !number of elements
149 INTEGER, INTENT(IN) :: stride         !counter
150 -----
151 ! Variables inside the subroutine
152 -----
153 CHARACTER (len=35) :: fname
154 CHARACTER (len=15) :: dname
155 CHARACTER (len= 4) :: ename
156 CHARACTER (len= 7) :: num
157 -----
158 INTEGER :: i, j                       !counters
159 INTEGER :: select                      !selection
160 INTEGER :: fstatus                    !fstatus
161 -----
162 ! First, the information of nelm and stride are transformed into character variables.
163 j = 1000000 + stride
164 WRITE(num,'(I7)') j
165 ename = '.dat'
166 -----
167 ! Then, DOOM SUBROUTINE is called.
168 dname = TRIM('Data\Area\nelm')
169 -----
170 ! The name of file is made.
171 fname = TRIM(TRIM(dname)//TRIM(num)//ename)
172 -----
173 OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
174 -----
175 IF (fstatus == 0) THEN
176   WRITE(500,*) nelm
177   WRITE(500,*) j
178 -----
179   DO i = 1, nelm
180     WRITE (500,*) crvme1(i)
181   END DO
182 END IF
183 -----
184 CLOSE(UNIT = 500)
185 -----
186 !03 FORMAT(10(1x,ES24.16))
187 -----
188 END SUBROUTINE write_datak
189 -----
190 SUBROUTINE write_datar(nelm, stride)
191 -----
192 ! This subroutine write the data of element area in a file
193 ! This subroutine applies to every step of time.
194 -----
195 USE Mod_SharedVars, ONLY: DBL, are1
196 -----
197 IMPLICIT NONE
198 -----
199 ! Variables
200 ! Variables inside the subroutine
201 -----
202 INTEGER, INTENT(IN) :: nelm           !number of elements
203 INTEGER, INTENT(IN) :: stride         !counter
204 -----
205 ! Variables inside the subroutine
206 -----
207 CHARACTER (len=35) :: fname
208 CHARACTER (len=15) :: dname
209 CHARACTER (len= 4) :: ename
210 CHARACTER (len= 7) :: num
211 -----
212 INTEGER :: i, j                       !counters
213 INTEGER :: select                      !selection
214 INTEGER :: fstatus                    !fstatus

```

```

217 !-----
218 ! First, the information of nelm and stride are transformed into character variables.
219 j = 1000000 + stride
220 WRITE(num,'(I7)') j
221 ename = '.dat'
222 -----
223 ! Then, DOOM SUBROUTINE is called.
224 dname = TRIM('Data\Area\nelm')
225 -----
226 ! The name of file is made.
227 fname = TRIM(TRIM(dname)//TRIM(num)//ename)
228 -----
229 OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
230 -----
231 IF (fstatus == 0) THEN
232   WRITE(500,*) nelm
233   WRITE(500,*) j
234 -----
235   DO i = 1, nelm
236     WRITE (500,*) are1(i)
237   END DO
238 END IF
239 -----
240 CLOSE(UNIT = 500)
241 -----
242 !03 FORMAT(10(1x,ES24.16))
243 -----
244 END SUBROUTINE write_datar
245 -----
246 SUBROUTINE write_datavl(nelm, stride, Velix, Vely, Veliz)
247 -----
248 ! This subroutine write the data of velocity on collocation points in a file
249 ! This subroutine applies to every step of time.
250 -----
251 USE Mod_SharedVars, ONLY: DBL
252 -----
253 IMPLICIT NONE
254 -----
255 ! Variables
256 ! Variables inside the subroutine
257 -----
258 INTEGER, INTENT(IN) :: nelm           !number of elements
259 INTEGER, INTENT(IN) :: stride         !counter
260 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(IN) :: Velix, Vely, Veliz !arrays of the components
261                                     ! of normal vectors over collocation points
262 -----
263 ! Variables inside the subroutine
264 -----
265 CHARACTER (len=35) :: fname
266 CHARACTER (len=15) :: dname
267 CHARACTER (len= 4) :: ename
268 CHARACTER (len= 7) :: num
269 -----
270 INTEGER :: i, j                       !counters
271 INTEGER :: select                      !selection
272 INTEGER :: fstatus                    !fstatus
273 -----
274 ! First, the information of nelm and stride are transformed into character variables.
275 j = 1000000 + stride
276 WRITE(num,'(I7)') j
277 ename = '.dat'
278 -----
279 ! Then, DOOM SUBROUTINE is called.
280 dname = TRIM('Data\Velocity')
281 -----
282 ! The name of file is made.
283 fname = TRIM(TRIM(dname)//TRIM(num)//ename)
284 -----
285 OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
286 -----
287 -----
288 -----

```

```

289
290 IF (fstatus == 0) THEN
291   WRITE(500,*) nelm
292   WRITE(500,*) j
293 -----
294   DO i = 1, nelm
295     WRITE (500,103) Velix(i), Vely(i), Veliz(i)
296   END DO
297 END IF
298 -----
299 CLOSE(UNIT = 500)
300 -----
301 103 FORMAT(10(1X,E24.16))
302 -----
303 END SUBROUTINE write_datav1
304 -----
305 SUBROUTINE write_dataFG(nelm, stride, GM)
306 ! This subroutine write the data of curvature in a file
307 ! This subroutine applies to every step of time.
310 USE Mod_SharedVars, ONLY: DBL, crvmel
311 -----
312 IMPLICIT NONE
313 ! Variables
314 ! Variables inside the subroutine
317 -----
318 INTEGER,          INTENT(IN) :: nelm          !number of elements
319 INTEGER,          INTENT(IN) :: stride        !counter
320 REAL (KIND = DBL), ALLOCATABLE, DIMENSION(:), INTENT(INOUT) :: GM
321 -----
322 ! Variables inside the subroutine
323 -----
324 CHARACTER (len=35) :: fname
325 CHARACTER (len=15) :: dname
326 CHARACTER (len= 4) :: ename
327 CHARACTER (len= 7) :: num
328 -----
329 INTEGER :: i, j
330 INTEGER :: select
331 INTEGER :: fstatus
332 -----
333 ! First, the information of nelm and stride are transformed into character variables.
334 j = 1000000 + stride
335 WRITE(num, '(I7)') j
336 ename = '.dat'
337 -----
338 ! Then, DOOM SUBROUTINE is called.
339 dname = TRIM('Data\Curv\nelm')
340 -----
341 ! The name of file is made.
342 -----
343 fname = TRIM(TRIM(dname)//TRIM(num)//ename)
344 -----
345 OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
346 -----
347 IF (fstatus == 0) THEN
348   WRITE(500,*) nelm
349   WRITE(500,*) j
350 -----
351   DO i = 1, nelm
352     WRITE (500,103) GM(i), GM(i+nelm), GM(i+nelm*2), crvmel(i)
353   END DO
354 END IF
355 -----
356 CLOSE(UNIT = 500)
357 -----
358 103 FORMAT(10(1X,E24.16))
359 -----
360 END SUBROUTINE write_dataFG

```

```

361 !-----
362 ! SUBROUTINE doom(ddname)
363 !-----
364 ! This subroutine write the data in a file
365 ! This subroutine applies to velocity and position of collocation points.
367 !-----
368 ! USE Mod_SharedVars, ONLY: DBL, doomsday
369 !-----
370 ! IMPLICIT NONE
371 !-----
372 ! Variables
373 ! Variables inside the subroutine
374 !-----
375 ! CHARACTER (len=52),INTENT(INOUT) :: ddname          !semiroot
376 !-----
377 ! Select subroutine and there are three options
378 ! 1.- Alfredo PC
379 ! 2.- Cahn-Hilliard and Ladyzhenskaya Test_experiments
380 ! 3.- Cahn-Hilliard and Ladyzhenskaya second and third programs.
381 ! Default.- Alfredo PC
382 !-----
383 ! The choice is made here padawan.
384 !-----
385 ! SELECT CASE (doomsday)
386 !-----
387 ! CASE (1)
388 !-----
389 ! This option is for your PC
390 ! ddname = 'C:\Users\Alfredo\Desktop\prtcl_mkl\Test_experiments\'
391 !-----
392 ! CASE (2)
393 !-----
394 ! This option is for Ladyzhenskaya
395 ! ddname = 'C:\Users\Asanjuans\Desktop\Test_experiments\'
396 !-----
397 ! CASE (3)
398 !-----
399 ! This option is for Ladyzhenskaya II
400 ! ddname = 'C:\Users\Asanjuans\Desktop\Test_experimen_2\'
401 !-----
402 ! CASE (4)
403 !-----
404 ! This option is for Ladyzhenskaya III
405 ! ddname = 'C:\Users\Asanjuans\Desktop\Test_experimen_3\'
406 !-----
407 ! CASE DEFAULT
408 !-----
409 ! This expretion is for your PC
410 ! ddname = 'C:\Users\Asanjuans\Desktop\prtcl_mkl\Test_experiments\'
411 ! END SELECT
412 !-----
413 ! ddname = TRIM(ddname)
414 !-----
415 ! END SUBROUTINE doom
416 !-----
417 ! SUBROUTINE geometryprint(npts, stride)
418 !-----
419 ! This subroutine write the data of curvature in a file
421 ! This subroutine applies to every step of time.
422 !-----
423 ! USE Mod_SharedVars, ONLY: DBL, p
424 !-----
425 ! IMPLICIT NONE
426 !-----
427 ! Variables
428 ! Variables inside the subroutine
429 !-----
430 ! INTEGER,          INTENT(IN) :: npts          !number of points
431 ! INTEGER,          INTENT(IN) :: stride        !counter

```

```

432 !!-----
433 !! Variables inside the subroutine
434 !!-----
435 CHARACTER (len=25) :: fnum
436 CHARACTER (len=15) :: numch
437 CHARACTER (len=95) :: fname
438 CHARACTER (len=67) :: dname
439 CHARACTER (len=52) :: ddname
440 CHARACTER (len= 4) :: ename
441 CHARACTER (len= 5) :: num
442 !!-----
443 !! Counters
444 INTEGER :: i, j
445 INTEGER :: select
446 IFSTATUS
447 !!-----
448 !! First, the information of neIm and stride are transformed into character variables.
449 fnum = TRIM(' ' // TRIM(numch) // 'E525.17')
450 j = 10000 + stride
451 WRITE(num, '(I5)') j
452 ename = '.dat'
453 !!-----
454 !! Then, DOOM SUBROUTINE is called.
455 CALL doom(ddname)
456 dname = TRIM(TRIM(ddname)//'geometry.p')
457 dname = TRIM(dname)
458 !!-----
459 !! The name of file is made.
460 fname = TRIM(TRIM(dname)//TRIM(num)//ename)
461 !!-----
462 !! OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
463 IF (fstatus == 0) THEN
464   WRITE(500,'*) npts
465   WRITE(500,'*) j
466 !!-----
467 !! DO i = 1, npts
468   WRITE (500,103) p(1,1),p(1,2),p(1,3)
469 !!-----
470   END DO
471 END IF
472 !!-----
473 !! CLOSE(UNIT = 500)
474 !!-----
475 !! 103 FORMAT(10(Ix,ES24.16))
476 !!-----
477 !! END SUBROUTINE geometryprint
478 !!-----
479 !! SUBROUTINE uvaprint(neIm, stride)
480 !!-----
481 !! This subroutine write the data of curvature in a file
482 !! This subroutine applies to every step of time.
483 !!-----
484 !! USE Mod_SharedVars, ONLY: DBL, fanel
485 !!-----
486 !! IMPLICIT NONE
487 !!-----
488 !! Variables inside the subroutine
489 !!-----
490 !! Counters
491 !!-----
492 !! Number of elements
493 !!-----
494 !! Counters
495 !!-----
496 !! Variables inside the subroutine
497 !!-----
498 CHARACTER (len=25) :: fnum
499 CHARACTER (len=15) :: numch
500 CHARACTER (len=95) :: fname
501 CHARACTER (len=67) :: dname
502 CHARACTER (len=52) :: ddname
503 CHARACTER (len= 4) :: ename

```

```

504 !!-----
505 !! Counters
506 !!-----
507 INTEGER :: i, j
508 INTEGER :: select
509 INTEGER :: fstatus
510 !!-----
511 !! First, the information of neIm and stride are transformed into character variables.
512 fnum = TRIM(' ' // TRIM(numch) // 'E525.17')
513 j = 10000 + stride
514 WRITE(num, '(I5)') j
515 ename = '.dat'
516 !!-----
517 !! Then, DOOM SUBROUTINE is called.
518 CALL doom(ddname)
519 dname = TRIM(TRIM(ddname)//'neIm.uva')
520 dname = TRIM(dname)
521 !!-----
522 !! The name of file is made.
523 fname = TRIM(TRIM(dname)//TRIM(num)//ename)
524 !!-----
525 !! OPEN(UNIT = 500, FILE = fname ,STATUS = 'REPLACE', ACTION = 'WRITE', IOSTAT = fstatus)
526 IF (fstatus == 0) THEN
527   WRITE(500,'*) nelm
528   WRITE(500,'*) arel(1)
529   WRITE(500,'*) j
530 !!-----
531 !! DO i = 1, nelm
532   WRITE (500,*) fanel(i)
533   !!WRITE (8,103) arel(i)
534 !!-----
535   END DO
536 END IF
537 !!-----
538 !! CLOSE(UNIT = 500)
539 !!-----
540 !! 103 FORMAT(10(Ix,ES24.16))
541 !!-----
542 !! END SUBROUTINE uvaprint
543 !!-----
544 !! END MODULE Mod_Data_Files
545

```