UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

PROGRAMA DE POSGRADO EN CIENCIAS DE LA TIERRA

MODELACIÓN DE FLUJO CONVECTIVO EN UN MEDIO POROSO ANISOTRÓPICO Y HETEROGÉNEO

T E S I S

QUE COMO REQUISITO PARCIAL PARA OBTENER EL GRADO DE:

MAESTRO EN CIENCIAS DE LA TIERRA

P R E S E N T A

CONRADO MONTEALEGRE CÁZARES

JURADO EXAMINADOR
1) DRA. ELSA LETICIA FLORES MÁRQUEZ (DIRECTOR DE TESIS)
2) DR. GUILLERMO DE JESÚS HERNÁNDEZ GARCÍA (PRESIDENTE)
3) DRA. GRACIELA HERRERA ZAMARRÁN (VOCAL)
4) DR. ROBERT YATES (SUPLENTE)
5) DR. GABRIEL JIMÉNEZ SUÁREZ (SUPLENTE)

UNAM POSGRADO Ciencias de la Tierra

MÉXICO D.F. MAYO 2010

DEDICATORIA

A ti Isa que siempre has sido una fuente de inspiración y apoyo en mi vida. Nena, te amo.

A ti mi pequeño Tonatiuh que con tu alegría y espontaneidad logras siempre sacar lo mejor de mí. A ti mi pequeñita que con la proximidad de tu llegada has venido a darle un nuevo aliento a mi vida.

A mis padres y mi hermano por su apoyo incondicional, por la infinita fe que me tienen y por todo el amor que me han dado. Sé que siempre he contado y que siempre contaré con ustedes.

A mi familia y amigos porque lo que soy ahora es en gran medida resultado de los momentos que hemos compartido y las enseñanzas que hemos obtenido al cruzarse nuestros caminos.

AGRADECIMIENTOS

Quiero agradecer a mi esposa y a mi familia que siempre confiaron en que saldría avante en esta importante etapa de mi vida. Su apoyo fue fundamental en todo momento. Su sola presencia mi inspiración. Agradezco también a mis suegros quienes estuvieron ahí en todo momento dispuestos a ayudarnos en lo que hiciese falta.

Dra. Leticia, muchas gracias por brindarme su apoyo incondicional. Gracias por esa mezcla de firmeza y empatía con la cual logró guiarme a lo largo de mi estancia en el posgrado. Sin su ayuda no podría exclamar ahora: objetivo cumplido.

Agradezco de manera muy especial al Dr. Ismael Herrera y al Dr. Robert Yates que sembraron en mí la semilla del conocimiento y que amablemente me permitieron usar su material en mi trabajo. Espero ser un digno representante de su escuela.

Dra. Graciela Herrera, Dr. Guillermo Hernández y Dr. Gabriel Jiménez, les agradezco sinceramente el tiempo que dedicaron a la revisión de mi tesis, así como los comentarios y sugerencias con los cuales lograron enriquecer mi trabajo.

A la UNAM por brindarme la oportunidad de mejorar como profesionista y como persona. Siempre tendrás un lugar muy importante en mi vida mi querida alma máter.

Al CONACYT y al Instituto de Geofísica por brindarme los medios materiales que permitieron mi formación académica.

Al personal del Instituto de Geofísica y del Posgrado en Ciencias de la Tierra, en especial a Sandra Luz Morales y Araceli Chaman, siempre recordaré todo lo que me ayudaron y su trato amable.

A mis compañeros y amigos en el Instituto por todas esas ideas, anécdotas y tiempo que compartimos. En especial quiero agradecer a Juan, Albertico, Dr. Artrero, Toño, Carmen, Ingrid, Mauricio, Aixa, Esther y Miguel, por sus consejos y enriquecedoras charlas.

Gracias Dios por permitirme vivir esta maravillosa experiencia.

Índice

1 Introducción

La transferencia de calor mediante convección dentro de un medio poroso es de fundamental importancia en las aplicaciones tecnológicas relacionadas con: la extracción de hidrocarburos, manejo de acuíferos en geohidrología, explotación geotérmica, extracción de calor de rocas secas, manejo de residuos radioactivos y modelado de flujo subterráneo. Además es un área de estudio de interés en las Ciencias de la Tierra desde el punto de vista Geofísico. El estudio de este tipo de fenómenos se ha venido realizando con éxito mediante el uso de modelos matemáticos.

Los modelos son descripciones conceptuales o aproximaciones que describen sistemas físicos usando ecuaciones matemáticas, pero de ningún modo son descripciones exactas de los sistemas físicos o procesos. La aplicación y utilidad de los modelos dependen de cuan cerca las ecuaciones matemáticas aproximen a los sistemas físicos que están siendo modelados. Las ecuaciones usadas en los modelos, se basan en ciertas suposiciones simplificadas que típicamente involucran: dirección de flujo, geometría del yacimiento, heterogeneidad y anisotropía de los estratos geológicos, mecanismos de transporte de partículas y reacciones químicas. Es debido a estas suposiciones y la incertidumbre en los valores de los datos requeridos por el modelo, que el modelo debe ser visto como una aproximación y no como un duplicado exacto de las condiciones de campo.

Los modelos de flujo convectivo involucran el transporte de masa en conjunto con el transporte de calor. Este tipo de modelos abarcan el estudio de áreas extensas o de nivel regional, las cuales por naturaleza tienden a ser bastante heterogéneas y a presentar un comportamiento anisotrópico. Entonces, las simplificaciones a medios homogéneos e isotrópicos no son satisfactorias para representarlos de manera adecuada, por lo que requieren planteamientos analíticos más complejos.

El uso de modelos matemáticos ha mostrado ser muy útil en el área de las Ciencias de la Tierra, pues permite dar solución y coadyuvar al entendimiento de sistemas complejos como los antes mencionados. Las ecuaciones que describen los modelos pueden ser resueltas usando distintos métodos, como los métodos analíticos, que dan como resultado soluciones exactas a las ecuaciones que describen muy sencillas condiciones de flujo o de transporte, o bien los métodos numéricos que son aproximaciones discretas de las ecuaciones que describen los procesos físicos. Dentro de estos métodos numéricos se pueden mencionar los de Diferencias Finitas, Elementos Finitos y Volúmenes Finitos.

Los modelos matemáticos de gran complejidad normalmente no poseen una solución analítica, así que para su solución se recurre a alguno de los métodos numéricos que las aproximen. La elección del método que se use depende, entre otros factores, de la complejidad del dominio, la cantidad de información con que se cuente y los recursos computacionales disponibles.

El Método del Elemento Finito es una opción que está siendo cada vez más utilizada pues permite trabajar con dominios heterogéneos y que presentan una forma irregular y compleja. Si bien antes era considerado como un método que representa mayor carga en los cálculos a realizarse, actualmente, debido al gran avance en las computadoras, la implementación de métodos más sofisticados, que anteriormente estaban reservados a grandes supercomputadoras, ahora es accesible en pequeñas estaciones de trabajo.

En la actualidad, la solución de los métodos numéricos se hace impensable si no se implementa a través de un sistema computacional, debido a la gran cantidad de datos y cálculos que ésta involucra. La implementación computacional normalmente se ve reflejada en la elaboración de un programa que permita la solución numérica. Existen varios métodos o paradigmas para la elaboración de programas o software. Uno de estos paradigmas, que ha venido cobrando fuerza en los últimos años, es el de la Orientación a Objetos, el cual permite abordar de manera sencilla problemas complejos a partir de varias técnicas como lo son la herencia, modularidad, polimorfismo y encapsulamiento. Además la Orientación a Objetos esta soportada por varios lenguajes de programación de alto nivel, entre los cuales destaca JAVA, que ha venido cobrando gran popularidad debido a su facilidad de uso, la posibilidad de reutilizar su código y sobre todo por su portabilidad, es decir, el programa puede ser ejecutado en distintos sistemas operativos sin necesidad de volver a compilarse.

El objetivo de este trabajo está basado en todos los conceptos antes descritos y es realizar la modelación de flujo convectivo en un medio poroso anisotrópico y heterogéneo. Para tal efecto se eligió usar el Método del Elemento Finito, implementándolo computacionalmente con la metodología Orientada a Objetos, con un programa en JAVA, algo que es de interés puesto que los trabajos previos realizados con estos fines han sido desarrollados en Diferencias Finitas y programación estructurada (Combarnous y Bories, 1975; Royer y Flores, 1994). Por otro lado, si bien la metodología Orientada a Objetos se ha venido usando ampliamente en desarrollos de FEM, la mayoría de estos trabajos han sido enfocados a Ingeniería Civil o bien en programas que resuelven de manera general ecuaciones diferenciales parciales. Además, se observa que en la mayoría de los trabajos existentes se obvia la explicación de la implementación computacional, entonces se realizó un esfuerzo en la descripción detallada de esta, lo cual cubre el objetivo de brindar un punto de referencia o ejemplo concreto que ayude al entendimiento de la implementación de FEM usando la metodología Orientada a Objetos.

La presente tesis está estructurada en seis capítulos y seis apéndices. El primer y último capítulo corresponden a la presente introducción y las conclusiones, respectivamente. Los capítulos dos y tres, junto con los primeros cuatro apéndices, corresponden a conceptos teóricos necesarios para la comprensión de los procesos de transferencia de calor por fluidos en medios porosos y que son tomados de sus referencias originales.

El capítulo dos está dedicado a presentar los fundamentos teóricos de la modelación de sistemas terrestres usando el método sistemático planteado por el Dr. Ismael Herrera Revilla (Herrera Revilla 2006; Herrera y Pinder, 2011) y por ello se hace una transcripción

completa de esta metodología en el Apéndice I. También se abordan los fundamentos teóricos del fenómeno de convección natural, así como el desarrollo de una formulación matemática del flujo convectivo en un medio poroso heterogéneo y anisotrópico tomada de Flores Márquez (1992).

En el tercer capítulo se explica el proceso de solución numérica de una EDP por medio del Método del Elemento Finito estándar planteado por el Dr. Robert Yates (Yates, 2007), usado para resolver las EDP Advectivo-Difusivo-Reactivas con condiciones de frontera generales, el Método de Crank Nicholson que permite resolver problemas transitorios y la solución de sistemas de ecuaciones lineales por medio del procedimiento iterativo Gauss-Seidel sobrerrelajado (SOR). Con la finalidad de facilitar el entendimiento del método del elemento finito estándar, en el Apéndice III se presenta el planteamiento del FEM para problemas difusivos en una y dos dimensiones, presentado principalmente en el libro (Chen, Huan y Yuanle, 2006). También se incluye en el Apéndice IV una breve recopilación de los distintos métodos existentes para resolver sistemas de ecuaciones lineales.

El cuarto capítulo presenta la implementación computacional del modelo de flujo convectivo. Se comienza con algunas consideraciones generales en la implementación del FEM, para luego pasar a la presentación de algunos conceptos básicos del paradigma de Orientación a Objetos. Por último se explica detalladamente la estructura del programa en JAVA haciendo uso de los Diagramas de Clase del Lenguaje de Modelado Unificado (UML).

En el quinto capítulo se presentan los resultados de ensayos numéricos que permiten comprobar la efectividad en la implementación computacional, comparando los resultados con aquellos previamente publicados. De inicio se comprueba la convergencia del FEM en un problema teórico completo con condiciones de frontera mixtas (Dirichlet y Neumann) no homogéneas. Enseguida se realizan varios ejemplos de modelos de convección que se cotejan con los resultados en trabajos previos. Estos ejemplos abarcan desde dominios sencillos homogéneos, hasta dominios más complejos que presentan heterogeneidad y anisotropía.

Los tres apéndices restantes corresponden a la nomenclatura usada en la formulación matemática del modelo de convección (Apéndice II), una breve explicación del significado de los parámetros físicos involucrados en el modelo (Apéndice V), y el código fuente de las clases más importantes del programa (Apéndice VI).

2 Fundamentos de la modelación de los sistemas continuos y de la convección natural

La formulación de los modelos de los sistemas continuos que se utiliza en esta tesis, está tomada de las notas del Dr. Ismael Herrera Revilla (Herrera Revilla, 2006; Herrera y Pinder, 2011), que utiliza en su curso de Modelación Matemática y Computacional. A continuación se presentan las ecuaciones básicas de balance que posteriormente nos permiten plantear las ecuaciones diferenciales parciales que describen el transporte de masa y de calor. Dichas ecuaciones de balance son deducidas por el método sistemático que el Dr. Herrera introdujo, por ello se hace una transcripción de una parte de los apuntes de su curso en el Apéndice I (Modelación de los sistemas contínuos), con la finalidad de que las ecuaciones basadas en esta metodología puedan ser comprendidas dado que es una parte fundamental para la deducción de las ecuaciones que son programadas en esta tesis con la metodología de la orientación a objetos.

También se presentan los fundamentos teóricos de la convección natural, comenzando con una descripción de las formas de la transferencia de calor, tomado de Incropera y Dewitt (2002). Posteriormente se describirán las ecuaciones que caracterizan la transferencia de flujo y calor, tomadas de Combarnous (1978) y finalmente la formulación adimensional para simplificar las ecuaciones de calor y Darcy, así como su nomenclatura tomadas de Royer y Flores (1994).

2.1 Ecuaciones de balance de la Mecánica de Sólidos y Fluidos

El modelo de un sistema continuo estará constituido por una colección de propiedades extensivas y las intensivas asociadas que permiten representarlo. A su vez, las propiedades quedan representadas por un conjunto de ecuaciones de balance, las cuales pueden ser diferenciales y de salto, donde la velocidad en cada una de ellas corresponde a la fase de la cual se trate. Además se debe de contar con relaciones suficientes que permitan ligar todas las ecuaciones de balance y dónde todos sus parámetros queden definidos en términos de éstas. A estas relaciones se les conoce también como leyes constitutivas. Por último, se debe de contar con condiciones iniciales y de frontera que satisfagan las propiedades, para que la solución del problema sea única.

La mecánica de los medios continuos es, en cierto sentido, clásica puesto que fue la primera que se desarrolló. Ésta incluye los modelos del movimiento de los sólidos y los fluidos. La familia de propiedades extensivas en que se basa su modelo general del movimiento está constituida por las siguientes: *masa*¸ *momento lineal*, *momento angular* y *energía*. Si se toma en cuenta que tanto el modelo lineal como el momento angular son vectores, y que cada propiedad cuya expresión es vectorial es equivalente a tres propiedades escalares, se

ve que esta familia en realidad está constituida de ocho propiedades extensivas escalares. Así, asociada a ella hay una familia que consta de ocho propiedades intensivas escalares.

A continuación se presenta una tabla con un resumen de las ecuaciones básicas de la mecánica de sólidos y fluidos, la cual se obtiene a partir del método descrito en el **¡Error! No se encuentra el origen de la referencia.**). Para el detalle de cómo se obtuvo esta tabla también se puede consultar (Herrera Revilla, 2006; Herrera y Pinder, 2011).

No	Propiedad Extensiva	Propiedad Intensiva ψ	$\underline{\tau}$	g	Ecuación de Balance
1	Masa $M(t)$	ρ	0	0	$\frac{D\rho}{Dt}+\rho\nabla\cdot\underline{v}=0$
2	Momento Lineal $\mathcal{M}(t)$	$\rho\underline{v}$	$\underline{\underline{\sigma}}$	$\rho\underline{b}$	$\rho\frac{D\underline{v}}{Dt}-\nabla\cdot\underline{\underline{\sigma}}-\rho\underline{b}=0$
3	Momento Angular $\mathcal{M}_a(t)$	$\rho(\underline{x}\times\underline{v})$	$\underline{x}\times\underline{\underline{\sigma}}$	$\rho(\underline{x}\times\underline{b})$	$\underline{\underline{\sigma}}=\underline{\underline{\sigma}}^T$
4	Energía $\mathcal{E}(t)$	$\rho\left(E+\frac{1}{2}\lvert\underline{v}\rvert^2\right)$	$\underline{q}+\underline{\underline{\sigma}}\cdot\underline{v}$	$\rho(h+\underline{b}\cdot\underline{v})$	$\rho\frac{DE}{Dt}=\nabla\cdot\underline{q}+\rho h+\underline{\underline{\sigma}}:\nabla$

Tabla 1. Ecuaciones básicas de la mecánica de sólidos y fluidos. ***Tabla tomada de*** **(Herrera Revilla, 2006).**

Donde ρ representa la densidad, $\underline{v}$ la velocidad del medio, $\underline{b}$ la *fuerza de cuerpo* por unidad de masa, $\underline{\underline{\sigma}}$ el tensor de esfuerzos, E la energía interna por unidad de masa, h las fuentes de calor por unidad de masa y $\underline{q}$ el vector de flujo de calor.

2.2 Transferencia de calor

La transferencia de calor (o calor) es la energía térmica en tránsito ocasionada por una diferencia de temperatura. Donde quiera que se dé esta diferencia de temperatura, ya sea entre medios distintos o dentro de un mismo medio, la transferencia de calor se presentará.

La transferencia de calor se puede presentar a través de distintos procesos o modos:

i. Conducción
ii. Radiación
iii. Convección

La *conducción* se presenta cuando la transferencia de energía va desde las partículas más energéticas hacia las partículas menos energéticas de una substancia y se debe a la interacción entre las partículas.

La *radiación* es la energía emitida por la materia, todo esto a una temperatura finita. Normalmente se estudia la radiación que emiten superficies sólidas, sin embargo la emisión puede presentarse desde líquidos y gases. Sin importar la forma de la materia, la emisión se ha atribuido a los cambios en la configuración de los electrones de los átomos o moléculas que la constituyen. La energía del campo de radiación es transportada por ondas electromagnéticas (o alternativamente por fotones). Mientras la transferencia de energía a través de la conducción o la convección requiere la presencia de un medio material, la radiación no lo necesita. De hecho, la transferencia por radiación se lleva a cabo de manera más eficiente en el vacío (Incropera y Dewitt, 2002).

La transferencia de calor por *convección* consta de dos mecanismos. Además de la transferencia de energía debida al *movimiento aleatorio molecular* (*difusión*), la energía se transfiere por el movimiento macroscópico. Este movimiento del fluido se asocia con el hecho de que, en cualquier instante, un gran número de moléculas se mueven conjuntamente o como un agregado. Entonces, este movimiento en presencia de un gradiente de temperatura, contribuye a la transferencia de calor. Debido a que las moléculas en el agregado poseen aún movimiento aleatorio, el total de la transferencia de calor se debe a la superposición de la energía transportada por el movimiento aleatorio de las moléculas y el movimiento macroscópico del fluido. Comúnmente se usa el término *convección* cuando se habla del transporte acumulativo que se acaba de mencionar, mientras que se usa el término *advección* cuando se quiere referir únicamente al transporte debido al movimiento del conjunto del fluido.

El flujo por convección suele ser clasificado de acuerdo a la naturaleza del flujo. Se habla de *convección forzada* cuando el flujo se debe a causas externas, por ejemplo un ventilador, bombeo o el viento en la atmósfera. Por otro lado, para la *convección libre (o natural)* el flujo es inducido por las fuerzas de flotabilidad, las cuales surgen de las diferencias en la densidad del fluido ocasionadas por las variaciones de temperatura. Por último, si se presentan ambos tipos de convección: forzada y natural, y ambas son de magnitudes similares, se dice que se tiene una *convección mixta*.

Lo que hasta este momento se ha presentado en esta sección es válido de forma general para la transferencia de calor en distintos medios. Hablando particularmente de un medio poroso se tiene que, cuando la densidad de la fase correspondiente al fluido que satura el medio poroso no es uniforme, algunos movimientos ocasionados por los efectos de la fuerza de gravedad tienden a reducir las discrepancias generales en la densidad. Sin embargo, se presentan casos donde estos movimientos no son suficientes, y en el estado estacionario se alcanzan a distinguir movimiento de tipo convectivo, dependiendo de las diferencias de densidad del fluido y de las condiciones de frontera.

Desde el punto de vista fenomenológico se pueden presentar dos tipos de movimiento convectivo en un medio poroso: *convección natural* y *convección mixta*. La convección natural o libre se presenta cuando el medio está cerrado, es decir sin intercambio de masa con el exterior y con una velocidad media del fluido igual a cero. Por otro lado, la convección mixta se presenta cuando al flujo ocasionado por la flotabilidad se le superpone el flujo mismo del fluido, es decir cuando interviene el campo de velocidades en el transporte de calor.

2.3 Fundamentos teóricos de la modelación de la convección natural

La distribución de temperatura y velocidad de filtración obedecen las ecuaciones de balance de masa, momento lineal y energía que son ampliamente descritas por Combarnous y Bories (1975) y que para fines de claridad en esta tesis son nuevamente deducidas aquí. A partir de las ecuaciones de balance obtenidas según el método descrito en el Apéndice I, se suponen varias hipótesis que permiten adecuar y simplificar estas ecuaciones al estudio de la convección natural en medio poroso.

Una primera hipótesis consiste en que, si bien el medio puede ser heterogéneo, no se tomarán en cuenta condiciones de salto o de discontinuidad; es decir que, en las fronteras entre diferentes estratos (en donde se presentarían las condiciones de salto) se considerará que las propiedades intensivas a estudiarse no varían de una forma drástica. Por lo tanto, en la presentación de cada una de las condiciones de balance se omitirán las condiciones de salto.

2.3.1 Ecuación de balance de masa

De la Tabla 1 se tiene que la ecuación de balance de masa es

$$\frac{D\rho}{Dt} + \rho \nabla \cdot \underline{v} = 0 \tag{2.1}$$

o escrita de otro modo (ver Apéndice I)

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \left(\rho \underline{v} \right) = 0 \tag{2.2}$$

Como se supondrá que el fluido es incompresible, entonces el término $\frac{\partial \rho}{\partial t} = 0$, con lo que la ecuación de balance de masa, también conocida como *Ecuación de Continuidad*, para el caso de los fluidos incompresibles queda como:

$$\nabla \cdot \left(\rho \underline{v} \right) = 0 \tag{2.3}$$

2.3.2 Ecuación de balance de momento lineal

La Tabla 1 indica que la ecuación de balance de momento lineal es

$$\rho \frac{D\underline{v}}{Dt} - \nabla \cdot \underline{\underline{\sigma}} - \rho \underline{b} = 0 \tag{2.4}$$

donde $\underline{b}$ representa la *fuerza de cuerpo* por unidad de masa y $\underline{\underline{\sigma}}$ el tensor de esfuerzos.

Si ahora se toma en cuenta que la ecuación de balance de momento lineal se aplicará a la fase fluida del sistema poroso, exclusivamente. Se obtiene

$$\varepsilon\rho \frac{D\underline{v}}{Dt} - \nabla \cdot \underline{\underline{\sigma}} - g = 0 \tag{2.5}$$

Nótese que en esta última ecuación no se ha sustituido aún el valor correspondiente a g, debido a que este valor estará definido por las hipótesis que se presentarán más adelante.

A continuación se incorporará la Ley de Darcy a la ecuación de balance de momento lineal. La Ley de Darcy se estableció con una base empírica y por lo mismo constituye una *ecuación constitutiva* del modelo para el flujo de fluidos en medios porosos. Sin embargo, es importante analizar las hipótesis que ella implica en el marco de la ecuación de balance de momento lineal del fluido.

La Ley de Darcy relaciona a la velocidad de las partículas del fluido con su presión. Fue propuesta en el Siglo XIX por el ingeniero francés Henri Darcy. Cuando esta ley se aplica al flujo monofásico, en su forma más general tiene semejanza las *Leyes de Fourier y de Fick* pues establece que el campo vectorial correspondiente a la velocidad de *Darcy* es una función lineal del gradiente de la presión.

La *velocidad de Darcy* se define por la ecuación

$$\underline{U} \equiv \varepsilon \underline{v} \tag{2.6}$$

Es importante hacer la observación de que la velocidad de Darcy presenta la siguiente propiedad: cuando el vector $\underline{n}$, unitario, es normal a una superficie, $\underline{U} \cdot \underline{n}$ es el *gasto volumétrico* de fluido que pasa a través de esa superficie. En presencia de la gravedad, la *Ley de Darcy* está dada por la ecuación:

$$\underline{U} = -\frac{1}{\mu}\underline{\underline{k}} \cdot \left(\nabla p - \rho \underline{\hat{g}}\right) \tag{2.7}$$

Donde $\underline{\hat{g}}$ es la aceleración de la gravedad (como vector y dirigida hacia el centro de la Tierra), μ es la viscosidad dinámica del fluido, $\underline{\underline{k}}$ es el *tensor de permeabilidad* y p es la presión del fluido.

Las hipótesis con las cuales el balance de momento de la ecuación (2.5) implica a la Ley de Darcy, se describen a continuación:

i. Debido a la existencia de matriz sólida dispersa en el medio poroso con un pequeño diámetro de poro medio, tanto las velocidades de filtración como los gradientes de la velocidad de filtración son pequeños. En otras palabras, el movimiento del fluido es casi estático y la rapidez de cambio del momento es despreciable. Por lo tanto el término correspondiente a la derivada material en la ecuación (2.5) se anula

$$\varepsilon\rho \frac{D\underline{\underline{v}}}{Dt} = 0 \tag{2.8}$$

ii. La gravedad está presente, lo que da lugar a una fuerza de cuerpo igual a $\varepsilon\rho\underline{\hat{g}}$

iii. El intercambio de momento entre la fase sólida y la líquida tiene dos partes
   a. La primera parte es equivalente a una fuerza de cuerpo dada por el vector $p\nabla\varepsilon$
   b. Cuando hay flujo del fluido, las paredes de la matriz sólida ejercen sobre la fase fluida una fuerza, que se ejerce en todos los puntos del espacio ocupado por la fase fluida del sistema poroso y es equivalente a una fuerza de cuerpo, en el fluido, que se representará por $\underline{\upsilon}$. Una hipótesis básica es que esta fuerza de cuerpo es similar a la que ocurre en el flujo laminar en un tubo; específicamente, ella tiene el sentido opuesto al movimiento del fluido y es una función lineal de su velocidad. Tanto la fuerza de cuerpo como la velocidad son vectores, por lo que esta hipótesis implica que hay una transformación lineal, que se representa con el auxilio de la matriz positiva definida $\underline{\underline{\underline{M}}}$, dada por

$$\underline{\upsilon} = -\varepsilon \underline{\underline{\underline{M}}} \cdot \underline{\underline{v}} = -\underline{\underline{\underline{M}}} \cdot \underline{U} \tag{2.9}$$

iv. El tensor de esfuerzos, para el intercambio de momento lineal de la fase líquida consigo misma, es isotrópico y su expresión es $-p\underline{\underline{I}}$, donde $\underline{\underline{I}}$ es la matriz identidad. Esta hipótesis significa que $\underline{\underline{\sigma}} \equiv -\varepsilon p \underline{\underline{I}}$, entonces

$$\nabla \cdot \underline{\underline{\sigma}} = \nabla \cdot \varepsilon p \tag{2.10}$$

Debido entonces a las hipótesis ii) y iii), se tiene que

$$g \equiv \varepsilon\rho\underline{\hat{g}} + p\nabla\varepsilon - \varepsilon\underline{\underline{M}}\cdot\underline{v} \tag{2.11}$$

Incorporando las ecuaciones (2.8), (2.10) y (2.11) a la ecuación (2.5), se tiene que

$$-\nabla\cdot\varepsilon p - \varepsilon\rho\underline{\hat{g}} + p\nabla\varepsilon - \varepsilon\underline{\underline{M}}\cdot\underline{v} = 0 \tag{2.12}$$

En (Herrera Revilla, 2006; Herrera y Pinder, 2011) se muestra como, al realizarse la incorporación de la Ley de Darcy en la ecuación de balance de momento lineal, la matriz $\underline{\underline{M}}$ queda definida como

$$\underline{\underline{M}} = \frac{\mu}{\underline{\underline{k}}} \tag{2.13}$$

Por lo tanto, sustituyendo (2.13) en (2.12) , usando la igualdad $\varepsilon\nabla p = p\nabla\varepsilon - \nabla\cdot\varepsilon p$ y dividiendo entre ε se tiene finalmente:

$$\nabla p - \rho\underline{\hat{g}} - \frac{\mu}{\underline{\underline{k}}}\cdot\underline{v} = 0 \tag{2.14}$$

2.3.3 Ecuación de balance de energía

En la Tabla 1 se indica que la ecuación de balance de energía está dada por

$$\rho\frac{DE}{Dt} = \nabla\cdot\underline{q} + \rho h + \underline{\underline{\sigma}} : \nabla\underline{v} \tag{2.15}$$

donde $\underline{\underline{\sigma}}$ es el tensor de esfuerzos, E la energía interna por unidad de masa, h la generación de calor por unidad de masa y $\underline{q}$ el vector de flujo de calor.

Así como en la ecuación de balance de momento lineal se incorporó la Ley de Darcy, en este caso se incorporará la Ley de Fourier, la cual es una *ecuación constitutiva* que describe la transferencia de calor ocasionada por la conducción de calor. La Ley de Fourier para un caso general donde existe flujo de calor anisotrópico queda definida como

$$\underline{q} = \underline{\underline{\Lambda}}\cdot\nabla T \tag{2.16}$$

donde $\underline{q}$ representa el flujo de calor, $\underline{\underline{\Lambda}}$ el tensor de conductividad térmica anisotrópica y T la temperatura.

Además se tomarán en cuenta las siguientes hipótesis:

i. La temperatura de la matriz rocosa y del fluido que saturan los poros se considera igual. Esta consideración es válida para velocidades de filtración bajas.
ii. No se presenta disipación no viscosa de la energía cinética en calor, lo cual ocasiona que

$$\underline{\underline{\sigma}} : \nabla \underline{v} = -p \nabla \cdot \underline{v} \tag{2.17}$$

Entonces, incorporando la ecuación (2.17) en la ecuación de balance de energía (2.15) y dividiendo entre ρ se tiene

$$\frac{DE}{Dt} = \frac{1}{\rho} \nabla \cdot \underline{q} + h + - \frac{p}{\rho} \nabla \cdot \underline{v} \tag{2.18}$$

si se toma en cuenta la ecuación de balance de masa (2.1), entonces el último término de la ecuación anterior queda como $-\frac{p}{\rho^2}\frac{D\rho}{Dt}$ o bien $p\frac{D}{Dt}\left(\frac{1}{\rho}\right)$. Ahora, usando la regla de la cadena para expresar las derivadas materiales $\frac{DE}{Dt}$ y $\frac{D\rho}{Dt}$ en términos de la razón de cambio de temperatura $\frac{DT}{Dt}$, la ecuación (2.18) queda

$$\frac{\partial E}{\partial T}\frac{DT}{Dt} + p\frac{\partial}{\partial T}\left(\frac{1}{\rho}\right)\frac{DT}{Dt} = \frac{1}{\rho}\nabla \cdot \underline{q} + h \tag{2.19}$$

Se usará c_p para denotar el calor específico de un material medida bajo condiciones constantes de presión, la cual es igual a

$$c_p = \frac{\partial E}{\partial T} + p\frac{\partial}{\partial T}\left(\frac{1}{\rho}\right) \tag{2.20}$$

usando (2.20) en (2.19) se tiene

$$c_p \frac{DT}{Dt} = \frac{1}{\rho}\nabla \cdot \underline{q} + h \tag{2.21}$$

Sustituyendo el valor de $\underline{q}$ de la Ley de Fourier expresada en la ecuación (2.16), y desarrollando la derivada material se tiene

$$c_p \frac{\partial T}{\partial t} + c_p \underline{v} \cdot \nabla T = \frac{1}{\rho}\nabla \cdot \underline{\underline{\Lambda}} \cdot \nabla T + h \tag{2.22}$$

Por último, multiplicando la ecuación (2.22) por ρ y reagrupando términos se tiene que

$$\nabla \cdot \underline{\underline{\Lambda}} \cdot \nabla T = \left(\rho c_p\right) \underline{v} \cdot \nabla T + \left(\rho c_p\right) \frac{\partial T}{\partial t} - A \tag{2.23}$$

donde $\left(\rho c_p\right)$ es la capacidad calorífica a una presión y A es la generación de calor por unidad de volumen.

2.4 Formulación matemática

A continuación se presenta el estudio de la convección natural en un medio poroso heterogéneo y anisotrópico con dominio rectangular, saturado con un fluido de una sola fase desarrollado por Flores Márquez (1992). Para tal efecto se supone que la fase líquida se mueve a través del medio de acuerdo a la Ley de Darcy. La fase sólida es un medio poroso saturado, anisotrópico y heterogéneo, con generación interna de calor. Las características del sólido dependen de la temperatura.

Se usa una formulación adimensional para simplificar las ecuaciones de calor y Darcy. Para el caso particular de problemas bidimensionales las ecuaciones resultantes son similares y pueden resolverse numéricamente usando el mismo algoritmo.

Para esta sección se usa una nomenclatura que no necesariamente corresponde a la usada en las secciones anteriores, la cual se presenta en el **¡Error! No se encuentra el origen de la referencia.**), tomada de Royer y Flores (1994).

2.4.1 Análisis

Las ecuaciones fundamentales usadas en el estudio de la convección natural de fluidos incompresibles en un medio poroso rectangular (Figura 1) constituyen un problema clásico el cual incluye cuatro ecuaciones:

i. Ecuación de transferencia de calor (Ecuación de balance de energía)
ii. Ecuación de Darcy (Ecuación de balance de momento lineal)
iii. Ecuación de continuidad (Ecuación de balance de masa)
iv. Variación de las características del fluido con respecto a la temperatura

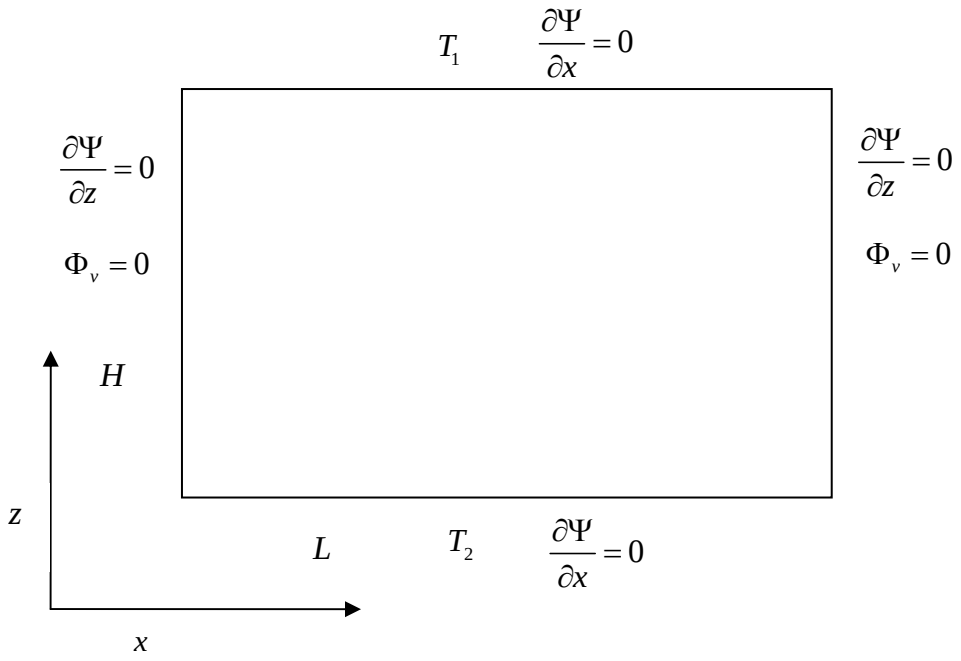


Figura 1. Medio poroso saturado confinado con bordes horizontales superior e inferior isotérmicamente definidos, bordes verticales adiabáticos, y sin intercambio de fluido en las fronteras

En investigaciones prácticas comúnmente se supone que la velocidad de filtración y su gradiente son muy pequeños, por lo cual se pueden despreciar las fuerzas inerciales. Siguiendo esta hipótesis y algunas otras, en la sección anterior se establecieron tres de las ecuaciones antes mencionadas. A continuación se presentan el conjunto de ecuaciones para la transferencia de calor en estado transitorio, siguiendo la nomenclatura propuesta por Royer y Flores (1994) y que también se reescribe en el **¡Error! No se encuentra el origen de la referencia.**):

$$\nabla \cdot \left(\underline{\underline{\mathbf{\Lambda}}}^* \nabla T^* \right) = \left(\rho c_p \right)_f \underline{\mathbf{v}} \cdot \nabla T^* + \left(\rho c_p \right)^* \frac{\partial T^*}{\partial t^*} - A^* \tag{2.24}$$

$$\nabla p - \rho \hat{\mathbf{g}} - \mu \underline{\underline{\mathbf{K}}}^{*-1} \underline{\mathbf{v}} = 0 \tag{2.25}$$

$$\nabla \cdot \left(\rho_f \underline{\mathbf{v}} \right) = 0 \tag{2.26}$$

$$\rho_f = \rho_r \left(1 - \beta_{th} \left(T^* - T_r^* \right) \right) \tag{2.27}$$

$$\eta = \eta_r \left(1 - \gamma^* \left(T^* - T_r^* \right) \right) \tag{2.28}$$

Comúnmente se simplifican las ecuaciones de balance de masa y de momento lineal suponiendo un medio homogéneo en el cual la variación de la densidad del fluido se descarta excepto en los términos de flotabilidad $\rho\hat{\mathbf{g}}$ (hipótesis de Boussinesq). Sin embargo, para sistemas hidrotermales de gran escala, algunos parámetros característicos del medio $\left(\eta, \underline{\underline{\mathbf{\Lambda}}}^*\right)$ pueden depender de la litología y del campo de temperaturas, mientras otros parámetros son menos sensibles $\left(\underline{\underline{\mathbf{K}}}^*, \beta_{th}, \rho_r\right)$. Debido a la heterogeneidad y la termo-dependencia del medio poroso, las ecuaciones clásicas para la convección obtenidas para un medio poroso homogéneo (Combarnous, 1978), no pueden ser aplicadas. Por lo tanto, un conjunto de ecuaciones, de la (2.24) a la (2.26) debe ser reescrito sin considerar homogeneidad o isotropía en las propiedades físicas del medio poroso.

2.4.2 Ecuación de transferencia de calor

La ecuación de transferencia de calor (2.24) es válida si se supone que la diferencia de temperaturas para la fase sólida T_s^* y para la fase líquida T_f^* se pueden descartar. Se supone también que la velocidad de filtración no es muy alta. Entonces, el medio puede ser equivalente a un único continuo con una temperatura promedio $T^* = T_f^* = T_s^*$. Esta aproximación es válida para los medios geológicos porosos saturados más comunes, como lo son las formaciones sedimentarias, pero su uso es limitado para los modelos de transferencia a través de rocas fracturadas.

Se supone que la capacidad calorífica del medio poroso saturado $\left(\rho c_p\right)^*$ está en función de la porosidad de acuerdo al siguiente modelo simple

$$\left(\rho c_p\right)^* = \left(1-\varepsilon\right)\left(\rho c_p\right)_s + \varepsilon\left(\rho c_p\right)_f \tag{2.29}$$

En el texto de Royer y Flores (1994) se puede encontrar como aproximar el tensor $\underline{\underline{\mathbf{\Lambda}}}^*$ en función de la temperatura, de varios parámetros de la roca (composición de la matriz rocosa, textura, tamaño de grano y de la composición mineral).

La termo-dependencia de la conductividad térmica hace que la ecuación de transferencia de calor sea no lineal. Esto representa una importante complicación comparada con la aproximación usual que considera que la conductividad térmica es constante para el medio poroso. Sin embargo, el efecto de la temperatura en las propiedades térmicas no puede descartarse en problemas donde se estudia un medio poroso de grandes dimensiones, como lo es el caso de los sistemas geotérmicos.

Para medios isotrópicos, la conductividad térmica equivalente puede relacionarse de forma experimental con la porosidad (Greenkorn, 1983):

$$\lambda_0 = \lambda_s^{(1-\varepsilon)} \lambda_f^{\varepsilon} \tag{2.30}$$

donde λ_s y λ_f son, respectivamente, la conductividad térmica de la matriz sólida y del fluido. Esta relación es válida únicamente cuando las fases sólida y líquida están equitativamente dispersas y el contraste entre las propiedades térmicas no es muy alto.

Para los medios anisotrópicos, el tensor $\underline{\underline{\mathbf{\Lambda}}}^*$ consiste en una complicada función de las propiedades térmicas de cada una de las fases constitutivas, las cuales se obtienen experimentalmente.

2.4.3 Ecuación de transferencia de calor adimensional

Si se introduce la notación de la conductividad térmica adimensional $\underline{\underline{\mathbf{\Lambda}}}$ y se toma en cuenta el hecho de que

$$\frac{2}{\text{tr}\ \underline{\underline{\mathbf{\Lambda}}}^*} \nabla \cdot \left(\underline{\underline{\mathbf{\Lambda}}} \nabla T^* \right) = \nabla \cdot \left(\underline{\underline{\mathbf{\Lambda}}} \nabla T^* \right) + \nabla \cdot \left(\ln\ \text{tr}\ \underline{\underline{\mathbf{\Lambda}}}^* \right) \cdot \underline{\underline{\mathbf{\Lambda}}} \nabla T^* \tag{2.31}$$

la ecuación (2.24) se puede reescribir como:

$$\nabla \cdot \left(\underline{\underline{\mathbf{\Lambda}}} \nabla T^* \right) = \frac{2\left(\rho c_p\right)_f}{\text{tr}\ \underline{\underline{\mathbf{\Lambda}}}^*} \underline{\mathbf{v}} \cdot \nabla T^* + \frac{2\left(\rho c_p\right)^*}{\text{tr}\ \underline{\underline{\mathbf{\Lambda}}}} \frac{\partial T^*}{\partial t^*} - \frac{2A^*}{\text{tr}\ \underline{\underline{\mathbf{\Lambda}}}^*} - \nabla \cdot \left(\ln\ \text{tr}\ \underline{\underline{\mathbf{\Lambda}}}^* \right) \cdot \underline{\underline{\mathbf{\Lambda}}} \nabla T^* \tag{2.32}$$

La ecuación (2.32) muestra que las heterogeneidades de la conductividad térmica a través del medio se toman en cuenta en el término fuente (similar a la generación interna de calor) en el segundo miembro de la ecuación de calor. Usando las variables adimensionales definidas en **¡Error! No se encuentra el origen de la referencia.**), la ecuación (2.32) se simplifica de la siguiente manera:

$$\begin{aligned} \nabla \cdot \left(\tilde{\underline{\underline{\mathbf{\Lambda}}}} \nabla T \right) &= \underline{v} \cdot \nabla T + \frac{\partial T}{\partial t} - A \\ \underline{v} &= \underline{v}_{Darcy} - \nabla \left(\ln\ \text{tr}\ \underline{\underline{\mathbf{\Lambda}}}^* \right) \cdot \tilde{\underline{\underline{\mathbf{\Lambda}}}} \end{aligned} \tag{2.33}$$

donde $\nabla\left(\ln \text{ tr } \underline{\underline{\mathbf{\Lambda}}}^{*}\right)$ denota la derivada de $\ln \text{ tr } \underline{\underline{\mathbf{\Lambda}}}^{*}$ con respecto al sistema coordenado adimensional (x, z).

2.4.4 Ecuación de Darcy

La ecuación (2.25), en la cual algunos términos inerciales han sido omitidos, es una forma generalizada de la ecuación de Darcy en estado estable. Ésta es válida para describir el movimiento de la fase líquida en sistemas convectivos en los cuales la velocidad de filtración no es muy importante (Combarnous, 1978) y para sistemas en los cuales la razón entre la permeabilidad media isotrópica y la altura del dominio al cuadrado $\left(\frac{k}{H^2}\right)$ es menor que 10^{-3}. Suponiendo que se mantiene constante la capacidad calorífica del fluido y el sólido y la termo-dependencia de las variaciones físicas características del fluido ρ y η, la ecuación (2.25) puede reescribirse (Quintard, 1983) como:

$$-(1-\gamma T)\underline{v}_{Darcy} = \frac{\text{tr } \underline{\underline{\mathbf{K}}}^{*}}{\text{tr } \underline{\underline{\mathbf{\Lambda}}}^{*}} \frac{\left(\rho c_p\right)_f}{\eta_r} \underline{\underline{\tilde{\mathbf{K}}}} \left(\nabla p\right)^t - Ra^{*}\left[T - \frac{1}{\beta}\right] \underline{\underline{\tilde{\mathbf{K}}}} \cdot \underline{e_3} \tag{2.34}$$

donde $\underline{v}_{Darcy}$, $\underline{\underline{\tilde{\mathbf{K}}}}$, T, β son las variables adimensionales definidas en **¡Error! No se encuentra el origen de la referencia.**). Ra^{*} es el número de Rayleigh de filtración local para un medio poroso anisotrópico y heterogéneo definido como:

$$Ra^{*} = \frac{\rho_r \hat{\mathbf{g}} \left(\rho c_p\right)_f \beta_{th} \Delta T^{*} H \text{ tr } \underline{\underline{\mathbf{K}}}^{*}}{\eta_r \text{ tr } \underline{\underline{\mathbf{\Lambda}}}^{*}} \tag{2.35}$$

La ecuación anterior describe explícitamente el criterio adimensional para la aparición de la convección hidrotermal libre, midiendo la influencia relativa de la fuerza “impulsora” de la convección sobre el efecto “estabilizador” ocasionado por la viscosidad del fluido η_r y la difusión térmica.

Medios porosos homogéneos han sido ya ampliamente estudiados por Combarnous y Bories (1975). En este tipo de medios, la condición para la ocurrencia de convección natural es que el número de filtración de Rayleigh sea mayor que un valor crítico $Ra_{cr} = 4\pi^2$. Para valores grandes de Ra, las celdas se vuelven inestables y la convección se torna turbulenta.

Sin embargo, el estudio de la convección natural en medios anisotrópicos, heterogéneos y con generación interna se ha enfocado principalmente en medios que presentan alta entalpía donde ya no es posible seguir aplicando la Ecuación de Darcy (Krishna, Basak, y Das, 2008; Nithiarasu, Sujatha, Ravindran, Sundararajan, y Seetharamu, 2000; Jue, 2003; Degan y Vasseur, 1995). Paras sistemas de baja entalpía, como el que es el objeto de estudio de este trabajo, se pueden encontrar trabajos recientes como (Luz Neto, Quaresma, y Cotta, 2006) donde se desarrolla un método híbrido analítico-numérico en 3 dimensiones, pero que solamente toma en cuenta un medio homogéneo isotrópico. También se pueden encontrar varios trabajos relativos a convección en sistemas de baja entalpía que estudian el fenómeno de doble difusividad y varios ejemplos en medios homogéneos isotrópicos con dominios distintos a una "cavidad rectangular". Desde el punto de vista de aplicaciones prácticas, en (Nagano, Mochida, y Ochifuji, 2002) se estudia el efecto que tiene la convección natural a gran escala en acuíferos donde se inyecta agua caliente (>50°C) con la finalidad almacenar la energía térmica, puesto que el fenómeno de convección genera un flujo superficial que afecta la efectividad en la recuperación de la energía térmica. En tal estudio se realizaron tanto simulaciones numéricas como experimentos de laboratorio, donde se comprobó que en sistemas de baja entalpía la ecuación de Darcy es adecuada y permite intentar hacer predicciones en acuíferos a gran escala. Sin embargo, ese trabajo tiene la limitante de que se enfoca solamente en medios homogéneos isotrópicos.

2.4.5 Formulación de la Ecuación de Darcy adimensional

Introduciendo las cantidades adimensionales $\underline{v}_{Darcy}$, f y g', la ecuación de Darcy puede reescribirse en forma de una función de corriente, ψ, de la siguiente manera:

$$\begin{aligned}
&\underline{v}_{Darcy} = \left(\mathbf{D}\psi\right)^t \\
&f = Ra^* \frac{\operatorname{tr} \underline{\underline{\underline{\mathbf{\Lambda}}}}^*}{\operatorname{tr} \underline{\underline{\underline{\mathbf{K}}}}^*} \frac{\eta_r}{\left(\rho c_p\right)_f}\left[T - \frac{1}{\beta}\right] \\
&g' = \frac{\operatorname{tr} \underline{\underline{\underline{\mathbf{\Lambda}}}}^*}{\operatorname{tr} \underline{\underline{\underline{\mathbf{K}}}}^*} \frac{\eta_r}{\left(\rho c_p\right)_f}\left[1 - \gamma T\right] \\
&\left(\nabla p\right)^t = f\,\underline{e}_3 - g'\underline{\underline{\underline{\tilde{\mathbf{K}}}}}^{-1}\left(\mathbf{D}\psi\right)^t
\end{aligned} \tag{2.36}$$

Mediante diferenciación cruzada y aplicando la relación de tensores 2D $\operatorname{tr}\left(\mathbf{D}^t\nabla p\right) = 0$, la ecuación (2.36) puede simplificarse en la siguiente ecuación diferencial no lineal que involucra la función de corriente ψ (los detalles de este proceso pueden observarse en Royer y Flores (1994)):

$$\operatorname{tr}\left(\mathbf{D}^t\left(f\,\underline{e}_3^t\right)\right) = \operatorname{tr}\left(\mathbf{D}^t\left(g'\left(\mathbf{D}\psi\right)\underline{\underline{\underline{\tilde{\mathbf{K}}}}}^{-1}\right)\right) \tag{2.37}$$

Realizando aún más simplificaciones con respecto a las propiedades de los operadores $\mathbf{D}$ y ∇ de los tensores 2D (Royer y Flores, 1994), se puede obtener una forma más práctica y simple de la ecuación de Darcy:

$$\nabla \cdot \left(\underline{\underline{\tilde{\mathbf{K}}}} \nabla \psi \right) = \underline{\mathbf{u}} \cdot \nabla \psi - S \tag{2.38}$$

con las siguiente cantidades adimensionales

$$\begin{aligned} \underline{\mathbf{u}} &= -\nabla \ln g \underline{\underline{\tilde{\mathbf{K}}}} \\ S &= \left(h \nabla \ln g + \nabla h \right) \cdot \underline{e}_1 \\ h &= Ra^* \det \underline{\underline{\tilde{\mathbf{K}}}} \frac{\left[T - \frac{1}{\beta} \right]}{\left[1 - \gamma T \right]} \\ g &= \frac{1}{\det \underline{\underline{\tilde{\mathbf{K}}}}} \frac{\operatorname{tr} \underline{\underline{\mathbf{\Lambda}}}^*}{\operatorname{tr} \underline{\underline{\mathbf{K}}}^*} \frac{\eta_r}{\left(\rho c_p \right)_f} \left[1 - \gamma T \right] \end{aligned} \tag{2.39}$$

La similitud entre la ecuación (2.38) y la ecuación de calor (2.33), que se encuentran en forma adimensional, es evidente. La cantidad vectorial $\underline{\mathbf{u}}$ sería el equivalente a la velocidad de filtración $\underline{\mathbf{v}}$ en la ecuación de calor, mientras que el término S sería similar a un término fuente. Esta similitud es de gran interés en la solución numérica de las ecuaciones acopladas de transferencia de masa y de calor en estado estable, puesto que se puede usar un procedimiento similar para resolver ambas ecuaciones.

Usando técnicas de derivación logarítmica, se puede evaluar el término S de la siguiente manera

$$\begin{aligned} \nabla \ln g &= \nabla \ln \left(\operatorname{tr} \underline{\underline{\mathbf{\Lambda}}}^* \right) - \nabla \ln \left(\operatorname{tr} \underline{\underline{\mathbf{K}}}^* \right) - \nabla \ln \left(\det \underline{\underline{\tilde{\mathbf{K}}}} \right) - \nabla \ln \left(\left(\rho c_p \right)_f \right) - \frac{\gamma \nabla T}{\left(1 - \gamma T \right)} \\ \nabla \ln h &= \nabla \ln \left(\det \underline{\underline{\tilde{\mathbf{K}}}} \right) + \nabla \ln \left(Ra^* \right) + \frac{\nabla T}{\left(T - \frac{1}{\beta} \right)} + \frac{\gamma \nabla T}{\left(1 - \gamma T \right)} \\ \nabla \ln \left(Ra^* \right) &= \nabla \ln \left(\left(\rho c_p \right)_f \right) - \nabla \ln \left(\operatorname{tr} \underline{\underline{\mathbf{\Lambda}}}^* \right) + \nabla \ln \left(\operatorname{tr} \underline{\underline{\mathbf{K}}}^* \right) \\ S &= \frac{\nabla T}{\left(T - \frac{1}{\beta} \right)} h \cdot \underline{e}_1 = \frac{Ra^* \det \underline{\underline{\tilde{\mathbf{K}}}}}{\left(1 - \gamma T \right)} \nabla T \cdot \underline{e}_1 \end{aligned} \tag{2.40}$$

En la ecuación (2.40) se observa claramente el significado físico del término S el cual depende de las propiedades anisotrópicas del medio poroso por causa del $\det \underline{\underline{\tilde{\mathbf{K}}}}$ y de la variación de la viscosidad del fluido ocasionada por la temperatura. Por otro lado, el término $\underline{\mathbf{u}}$ depende de la heterogeneidad del medio a través de los términos $\underline{\underline{\tilde{\mathbf{K}}}}$, $\nabla \ln\left(\mathrm{tr}\ \underline{\underline{\mathbf{K}}}^{*}\right)$, $\nabla \ln\left(\mathrm{tr}\ \underline{\underline{\boldsymbol{\Lambda}}}^{*}\right)$ y $\nabla \ln\left(\det \underline{\underline{\tilde{\mathbf{K}}}}\right)$.

Es de destacarse que, para el caso de un medio isotrópico y homogéneo saturado por un fluido Newtoniano con una viscosidad η constante e independiente de la temperatura, se tiene que $\underline{\underline{\tilde{\mathbf{K}}}} = 1$, $\nabla \ln\left(\mathrm{tr}\ \underline{\underline{\mathbf{K}}}^{*}\right) = \nabla \ln\left(\mathrm{tr}\ \underline{\underline{\boldsymbol{\Lambda}}}^{*}\right) = 0$. Entonces, el término fuente S en la ecuación de Darcy se reduce a su forma clásica $S = Ra^{*} \nabla T \cdot \underline{e}_1$.

2.4.6 Ecuación de continuidad

La ecuación de Darcy se ha derivado suponiendo que las variaciones de la densidad del fluido es descartable excepto en los términos de flotabilidad $\rho \hat{\mathbf{g}}$. Esta hipótesis también implica una simplificación para la ecuación de balance de masa (2.26), la cual puede reescribirse usando la velocidad de filtración adimensional $\underline{v}_{Darcy}$

$$\nabla \cdot \left(\mathrm{tr}\ \underline{\underline{\boldsymbol{\Lambda}}}^{*}\, \underline{v}_{Darcy}\right) = 0 \tag{2.41}$$

En la mayoría de las aplicaciones más comunes, la variación de la conductividad térmica con respecto al espacio, la cual se representa en la ecuación (2.41) con el término $\mathrm{tr}\ \underline{\underline{\boldsymbol{\Lambda}}}^{*}$, puede descartarse comparándola con la variación de la velocidad de filtración debida a la permeabilidad. Por ejemplo, la conductividad térmica de las rocas comúnmente varía entre los rangos de 1 y 3 $\left[\frac{\mathrm{W}}{\mathrm{mK}}\right]$ para un intervalo de temperatura de 25 a 300 $\left[{}^{\circ}\mathrm{C}\right]$, mientras que la permeabilidad varía entre los 10^{-20} y 10^{-12} $\left[\mathrm{m}^2\right]$. Siguiendo la anterior hipótesis, permite simplificar la ecuación (2.41) de la siguiente manera:

$$\nabla \cdot \underline{v}_{Darcy} = 0 \tag{2.42}$$

Cabe remarcar que la velocidad de Darcy $\underline{v}_{Darcy}$ en la ecuación (2.42) se obtiene de la derivación cruzada de la función de corriente $\left(\mathbf{D}\psi\right)^{t}$. En esta propiedad se asume

automáticamente la ecuación (2.42), porque $\nabla \cdot \left(\mathbf{D} \psi \right)^t$ es siempre igual a cero (Royer y Flores, 1994).

Cuando la variación de la conductividad térmica con respecto al espacio no puede descartarse, desarrollando (2.41) (Royer y Flores, 1994) e incorporando (2.42), la ecuación de continuidad queda como:

$$\nabla \text{tr} \, \underline{\underline{\mathbf{\Lambda}}}^* \cdot \underline{v}_{Darcy} = 0 \tag{2.43}$$

La ecuación (2.43) obliga a que, el producto escalar de la velocidad de Darcy adimensional por el gradiente de la conductividad térmica, sea igual a cero. En un medio homogéneo, esta ecuación se satisface automáticamente porque la conductividad térmica es constante. En el caso de dos capas homogéneas con diferentes conductividades térmicas, la ecuación (2.43) ocasiona que la velocidad de Darcy sea paralela a la interfaz entre las capas, lo cual ocurre comúnmente cuando las dos capaz tienen permeabilidades distintas. En otros casos, la ecuación (2.43) tiene que incluirse en las ecuaciones fundamentales, sin embargo en este trabajo no se tomarán en cuenta esta última opción.

2.4.7 El conjunto de ecuaciones completo

Finalmente, del desarrollo que se hizo en esta sección, se tiene que las ecuaciones fundamentales para el estudio de la convección libre se reducen a las ecuaciones adimensionales (2.33) y (2.38) acopladas, las cuales se resumen a continuación:

$$
\begin{aligned}
&\nabla\cdot\left(\underline{\underline{\tilde{\mathbf{\Lambda}}}}\nabla T\right)=\underline{v}\cdot\nabla T+\frac{\partial T}{\partial t}-A\\
&\nabla\cdot\left(\underline{\underline{\tilde{\mathbf{K}}}}\nabla\psi\right)=\underline{\mathbf{u}}\cdot\nabla\psi-S\\
&\underline{v}=\underline{v}_{Darcy}-\nabla\left(\ln\ \mathrm{tr}\ \underline{\underline{\mathbf{\Lambda}}}^{*}\right)\cdot\underline{\underline{\tilde{\mathbf{\Lambda}}}}\\
&\underline{v}_{Darcy}=\left(\mathbf{D}\psi\right)^{t}=\left(\frac{\partial\psi}{\partial z},-\frac{\partial\psi}{\partial x}\right)^{t}\\
&\underline{\mathbf{u}}=-\nabla\ln g\underline{\underline{\tilde{\mathbf{K}}}}\\
&S=\frac{Ra^{*}\det\underline{\underline{\tilde{\mathbf{K}}}}}{\left(1-\gamma T\right)}\nabla T\cdot\underline{e}_{1}\\
&g=\frac{1}{\det\underline{\underline{\tilde{\mathbf{K}}}}}\frac{\mathrm{tr}\ \underline{\underline{\mathbf{\Lambda}}}^{*}}{\mathrm{tr}\ \underline{\underline{\mathbf{K}}}^{*}}\frac{\eta_{r}}{\left(\rho c_{p}\right)_{f}}\left[1-\gamma T\right]\\
&Ra^{*}=\frac{\rho_{r}\hat{\mathbf{g}}\left(\rho c_{p}\right)_{f}\beta_{th}\Delta T^{*}H\ \mathrm{tr}\ \underline{\underline{\mathbf{K}}}^{*}}{\eta_{r}\ \mathrm{tr}\ \underline{\underline{\mathbf{\Lambda}}}^{*}}
\end{aligned}
\tag{2.44}
$$

Este conjunto de ecuaciones es válido para fluidos incompresibles en un medio poroso anisotrópico y heterogéneo. En cualquier punto del dominio estudiado, el campo de presiones puede derivarse de la integración, dando las condiciones de frontera apropiadas, de la siguiente ecuación que involucra la función de corriente:

$$
\left(\nabla p\right)^{t}=f\underline{e}_{3}-g'\underline{\underline{\tilde{\mathbf{K}}}}^{-1}\left(\mathbf{D}\psi\right)^{t}
\tag{2.45}
$$

En resumen, el análisis presentado en esta sección concluye con dos ecuaciones diferenciales reportadas en (2.44) que relacionan dos funciones escalares desconocidas $\left(T,\psi\right)$, las cuales se resolverán simultáneamente de forma numérica, asignándoles las condiciones de frontera adecuadas.

3 Solución numérica

Un método numérico para la solución de un problema con ecuaciones diferenciales parciales involucra la discretización del problema, el cual tiene originalmente un número infinito de grados de libertad, para producir de esta manera un *problema discreto*, el cual tiene un número finito de grados de libertad y puede resolverse usando una computadora.

El método del elemento finito (FEM), obtiene soluciones aproximadas de las ecuaciones diferenciales parciales, a partir de subdividir el dominio (espacio geométrico en el cual se halla definido el problema) en un número finito de regiones más pequeñas llamadas elementos finitos. A esta partición (comúnmente llamada discretización) se le asocia un espacio vectorial de dimensión finita, el cual permite aproximar una solución numérica a partir de la combinación lineal de dicho espacio vectorial. La proyección del problema original (reformulado en forma variacional o débil) sobre el espacio vectorial obtenido de la discretización, da lugar a un sistema de ecuaciones finito. Finalmente, se resuelve el sistema de ecuaciones para obtener la solución numérica aproximada del problema original.

Las ventajas de los elementos finitos sobre las diferencias finitas son que: las condiciones de frontera generales, las geometrías complejas, y propiedades materiales variables pueden ser manejadas de una manera más sencilla.

También, la estructura clara y versátil de los métodos del elemento finito hace posible el desarrollo de software para aplicaciones de propósito general. Más aún, los fundamentos teóricos sólidos, añaden confiabilidad y, en muchos casos, es posible obtener estimaciones del error concretas para las soluciones usando el elemento finito.

Comparado con los métodos de diferencias finitas, la introducción de los métodos de elemento finito es relativamente reciente. Los métodos del elemento finito fueron presentados por primera vez por Courant en 1943. Desde 1950 hasta 1970 éstos fueron desarrollados por ingenieros y matemáticos hasta llegar a una metodología general para la solución de ecuaciones diferenciales parciales.

3.1 Introducción a los Métodos del Elemento Finito (FEM)

De forma general, el método se basa en dividir el cuerpo, estructura o dominio (medio continuo) —sobre el que están definidas ciertas ecuaciones integrales que caracterizan el comportamiento físico del problema— en una serie de subdominios no intersectantes entre sí denominados "*elementos finitos*". El conjunto de elementos finitos forma una partición del dominio también denominada discretización. Dentro de cada elemento se distinguen una serie de puntos representativos llamados *"nodos"*. Dos nodos son adyacentes si

pertenecen al mismo elemento finito; además, un nodo sobre la frontera de un elemento finito puede pertenecer a varios elementos. El conjunto de nodos considerando sus relaciones de adyacencia se llama *"malla"*.

Los cálculos se realizan sobre una malla o discretización creada a partir del dominio con programas especiales llamados generadores de mallas, en una etapa previa a los cálculos que se denomina generalmente "pre-proceso". De acuerdo con estas relaciones de adyacencia o conectividad se relaciona el valor de un conjunto de variables incógnitas definidas en cada nodo y denominadas grados de libertad. El conjunto de relaciones entre el valor de una determinada variable entre los nodos se puede escribir en forma de sistema de ecuaciones lineales. La matriz de dicho sistema de ecuaciones se llama matriz de rigidez del sistema. El número de ecuaciones de dicho sistema es proporcional al número de nodos.

Se puede decir que existen seis clases principales de métodos del elemento finito, las cuales son las más comúnmente utilizadas en la actualidad (Chen, 2005):

i. Estándar
ii. Control del volumen
iii. Discontinuos
iv. Mixtos
v. Característicos
vi. Adaptativos

Este trabajo se enfoca solamente en el método estándar (para una revisión del resto de los métodos pueden consultarse los trabajos de Chen (2005), Chen, Huan, y Yuanle (2006)), presentándose una variación de éste que permite resolver problemas en su forma más general, es decir reactivos-difusivos-advectivos. Para fines de claridad, en el **¡Error! No se encuentra el origen de la referencia.**) se presentan los fundamentos del método del elemento finito estándar unidimensional y bidimensional en el caso de problemas difusivos. Este apéndice fue tose basa principalmente en los libros del Dr. Chen consultados en la realización de este trabajo.

El uso del método del elemento finito estándar se justifica ya que, aunque se estudiarán problemas en donde el término advectivo es importante y el dominio es heterogéneo (es decir, compuesto por varios estratos), se supone que las discontinuidades o condiciones de salto al interior del dominio pueden ser despreciadas por no ser de varios órdenes de magnitud. Por lo tanto, se espera que las soluciones del modelo sean lo suficientemente *suaves* como para justificar el uso del método estándar, que es relativamente más fácil de implementar y de uso general, y no optar por métodos más elaborados como lo serían los discontinuos o bien los característicos, los cuales se usan cuando el problema es principalmente advectivo y donde la gran heterogeneidad del medio hace esperar que la solución tenga cambios o saltos bruscos (Chen, 2005).

3.2 Método del Elemento Finito Estándar para ecuaciones Advectivo-Difusivo-Reactivas

Esta sección se basa principalmente en una recopilación y adaptación de las notas del curso de Modelación Computacional de Sistemas Terrestres impartido por el Dr. Robert Yates (2007), donde se presenta el planteamiento del método del elemento finito estándar para ecuaciones advectivo-difusivo-reactivas. Estas notas del curso pueden encontrarse también en http://www.mmc.igeofcu.unam.mx/. Es importante mencionar que el método descrito por el Dr. Yates es el que se toma como base para implementar las ecuaciones de transferencia de calor y de transferencia de masa involucradas en la convección natural.

3.2.1 Forma de la Divergencia

La forma general de una Ecuación Diferencial Parcial lineal de segundo orden bidimensional (donde $x_1 = x$ y $x_2 = y$) es:

$$\mathrm{L}u = Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu \tag{1.1}$$

Donde $u_x = \frac{\partial u}{\partial x}$ $u_{xx} = \frac{\partial u}{\partial x \partial x}$ $u_{yy} = \frac{\partial u}{\partial y \partial y}$ $u_{xy} = \frac{\partial u}{\partial x \partial y}$

En forma de la divergencia, el operador $\mathrm{L}u$ se escribe como:

$$\mathrm{L}u = -\nabla \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u + \nabla \cdot \left(\underline{\mathbf{b}} u\right) + \mathbf{c}u \tag{1.2}$$

Donde $\underline{\underline{\mathbf{a}}} = \begin{pmatrix} \mathbf{a_{11}} & \mathbf{a_{12}} \\ \mathbf{a_{21}} & \mathbf{a_{22}} \end{pmatrix}$ es una matriz simétrica, $\underline{\mathbf{b}} = \begin{pmatrix} \mathbf{b_1} \\ \mathbf{b_2} \end{pmatrix}$ es un vector y c un escalar.

Entonces se tiene que:

$$\mathrm{L}u = -\nabla \cdot \begin{pmatrix} \mathbf{a_{11}} & \mathbf{a_{12}} \\ \mathbf{a_{21}} & \mathbf{a_{22}} \end{pmatrix} \cdot \begin{pmatrix} u_x \\ u_y \end{pmatrix} + \nabla \cdot \begin{pmatrix} \mathbf{b_1} u \\ \mathbf{b_2} u \end{pmatrix} + \mathbf{c}u \tag{1.3}$$

Desarrollando (1.3) e igualando a la ecuación (1.1) se tiene que:

$$\begin{aligned} \mathrm{L}u &= -\mathbf{a}_{11}u_{xx} - \mathbf{a}_{11x}u_x - \mathbf{a}_{12x}u_y - \mathbf{a}_{12}u_{xy} - \mathbf{a}_{12y}u_x - \mathbf{a}_{12}u_{xy} - \mathbf{a}_{22}u_{yy} - \mathbf{a}_{22y}u_y + \mathbf{b}_{1x}u + \mathbf{b}_1u_x + \mathbf{b}_{2y}u + \mathbf{b}_2u_y + \mathbf{c}u \\ &= Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu \end{aligned}$$

Por lo tanto:

$$\begin{aligned} \underline{\underline{\mathbf{a}}} &= -\begin{pmatrix} A & B/2 \\ B/2 & C \end{pmatrix} \\ \underline{\mathbf{b}} &= \begin{pmatrix} D - A_x - \frac{1}{2}B_y \\ E - C_y - \frac{1}{2}B_x \end{pmatrix} \\ \mathbf{c} &= F - \left(D_x - A_{xx} - \frac{1}{2}B_{xy}\right) - \left(E_y - C_{yy} - \frac{1}{2}B_{xy}\right) = F + A_{xx} + C_{yy} + B_{xy} - D_x - E_y \end{aligned} \tag{1.4}$$

Haciendo una analogía con el discriminante de las ecuaciones cuadráticas, la ecuación será elíptica si la matriz $\underline{\underline{\mathbf{a}}}$ es positiva definida, en cuyo caso:

$$\det \underline{\underline{\mathbf{a}}} = \mathbf{a}_{11}\ \mathbf{a}_{22} - \mathbf{a}_{12}^2 > 0 \tag{1.5}$$

La ecuación es parabólica si

$$\det \underline{\underline{\mathbf{a}}} = \mathbf{a}_{11}\ \mathbf{a}_{22} - \mathbf{a}_{12}^2 = 0 \tag{1.6}$$

Y será hiperbólica si

$$\det \underline{\underline{\mathbf{a}}} = \mathbf{a}_{11}\ \mathbf{a}_{22} - \mathbf{a}_{12}^2 < 0 \tag{1.7}$$

El adjunto del operador es:

$$\begin{aligned} \mathrm{L}^*u &= \partial_{xx}(Au) + \partial_{xy}(Bu) + \partial_{yy}(Cu) - \partial_x(Du) - \partial_y(Eu) + Fu \\ \mathrm{L}^*u &= -\nabla \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u - \underline{\mathbf{b}} \cdot \nabla u + \mathbf{c}u \end{aligned} \tag{1.8}$$

3.2.2 Discretización espacial

Se tiene un problema del tipo

$$\begin{aligned} \mathrm{L}u &= f \qquad \text{en } \Omega \\ u(\underline{\mathbf{x}}) &= g(\underline{\mathbf{x}}) \qquad \text{en } \Gamma \end{aligned} \tag{1.9}$$

Donde Ω representa el dominio espacial, Γ las fronteras del dominio, $\mathrm{L}u$ la forma de la divergencia (1.2) y $\underline{\underline{\mathbf{a}}}(\underline{x},t)$ es simétrica y positiva definida.

Multiplicando por una función $w(\underline{x})$ resulta:

$$\begin{aligned} w\mathrm{L}u &= -w\nabla\cdot\underline{\underline{\mathbf{a}}}\cdot\nabla u + w\nabla\cdot(\underline{\mathbf{b}}u) + \mathbf{c}wu \\ &= -\nabla\cdot(w\underline{\underline{\mathbf{a}}}\nabla u) + \nabla w\cdot\underline{\underline{\mathbf{a}}}\nabla u + \nabla.(w\underline{\mathbf{b}}u) - u\underline{\mathbf{b}}\cdot\nabla\mathrm{w} + \mathbf{c}wu \\ &= fw \end{aligned} \tag{1.10}$$

Integrando sobre Ω y aplicando el Teorema de Gauss o de la divergencia **¡Error! No se encuentra el origen de la referencia.**:

$$\int_\Omega\left(\nabla w\cdot\underline{\underline{\mathbf{a}}}\cdot\nabla u - u\underline{\mathbf{b}}\cdot\nabla w + \mathbf{c}wu\right)d\underline{\mathbf{x}} = \int_\Omega fw\,d\underline{\mathbf{x}} + \int_\Gamma w\left(\underline{\underline{\mathbf{a}}}\cdot\nabla u - \underline{\mathbf{b}}u\right)\cdot\underline{\hat{\mathbf{n}}}\,ds \tag{1.11}$$

donde $\underline{\hat{\mathbf{n}}}$ es el vector unitario normal a Γ apuntando hacia afuera del dominio.

Haciendo que

$$u = u_0(\underline{\mathbf{x}}) + v(\underline{\mathbf{x}}) \tag{1.12}$$

Donde $u_0 = g\,\mathrm{en}\,\Gamma$ y haciendo $w = 0\,\mathrm{en}\,\Gamma$, de forma que el término de la integral referente a la frontera desaparezca, resulta:

$$\begin{aligned} \int_\Omega\left(\nabla w\cdot\underline{\underline{\mathbf{a}}}\cdot\nabla v - v\underline{\mathbf{b}}\cdot\nabla w + \mathbf{c}wv\right)d\underline{\mathbf{x}} &= \int_\Omega fw\,d\underline{\mathbf{x}} \\ -\int_\Omega\left(\nabla w\cdot\underline{\underline{\mathbf{a}}}\cdot\nabla u_0 - u_0\underline{\mathbf{b}}\cdot\nabla w + \mathbf{c}wu_0\right)d\underline{\mathbf{x}} & \end{aligned} \tag{1.13}$$

Escogiendo un subespacio de dimensión finita V_N conformado por una base $\{w_1,\ldots,w_N\}$ y haciendo que:

$$\hat{v}(x) = \sum_{j=1}^{N} v_j w_j(x) \approx v(x) \tag{1.14}$$

Se tiene que:

$$\sum_{j=1}^{N} v_j \int_\Omega \left(\nabla w_i \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla w_j - w_j \underline{\mathbf{b}} \cdot \nabla w_i + \mathbf{c} w_i w_j \right) d\underline{\mathbf{x}} = \int_\Omega f w_i \, d\underline{\mathbf{x}} - \int_\Omega \left(\nabla w_i \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u_0 - u_0 \underline{\mathbf{b}} \cdot \nabla w_i + \mathbf{c} w_i u_0 \right) d\underline{\mathbf{x}} \tag{1.15}$$

Haciendo

$$\begin{aligned} \mathrm{A}_{ij} &= \int_\Omega \left(\nabla w_i \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla w_j - w_j \underline{\mathbf{b}} \cdot \nabla w_i + \mathbf{c} w_i w_j \right) d\underline{\mathbf{x}} \\ r_i &= \int_\Omega f w_i \, d\underline{\mathbf{x}} - \int_\Omega \left(\nabla w_i \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u_0 - u_0 \underline{\mathbf{b}} \cdot \nabla w_i + \mathbf{c} w_i u_0 \right) d\underline{\mathbf{x}} \end{aligned} \tag{1.16}$$

Lo cual resulta en el sistema lineal de $N \times N$:

$$\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{v}} = \underline{\mathbf{r}} \tag{1.17}$$

De forma que la solución del problema está en:

$$\underline{\mathbf{v}} = \underline{\underline{\mathbf{A}}}^{-1} \cdot \underline{\mathbf{r}} \tag{1.18}$$

3.2.3 Extensión a condiciones de frontera generales

Se tiene el problema

$$\begin{aligned} \mathrm{L}u &= f && \text{en } \Omega \\ u(\underline{\mathbf{x}}) &= g(\underline{\mathbf{x}}) && \text{en } \Gamma_{\mathrm{D}} \\ \mathbf{a}_n \frac{\partial u}{\partial \hat{n}}(\underline{\mathbf{x}}) &= h(\underline{\mathbf{x}}) - e(\underline{\mathbf{x}}) u(\underline{\mathbf{x}}) && \text{en } \Gamma_{\mathrm{N}} \end{aligned} \tag{1.19}$$

Donde Ω representa el dominio espacial, Γ_{D} y Γ_{N} las fronteras del dominio, $\mathrm{L}u$ la forma de la divergencia (1.2) y $\underline{\underline{\mathbf{a}}}(\underline{\mathbf{x}},t)$ es simétrica y positiva definida (Figura 1).

Además, las fronteras del dominio presentan las siguientes características

$$\begin{aligned} \Gamma_N \bigcap \Gamma_D &= \varnothing \\ \Gamma_N \bigcup \Gamma_D &= \Gamma \end{aligned} \tag{1.20}$$

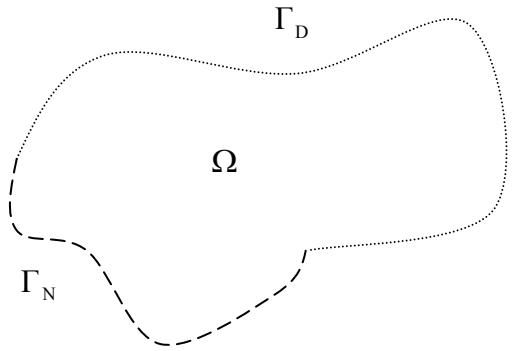


Figura 1. Representación de un dominio bidimensional donde $\Omega \in \mathbb{R}^2$, Γ_D representa las fronteras Dirichlet y Γ_N las fronteras Neumann

Tomando en cuenta la última ecuación de (1.19), se tiene que

$$\begin{aligned} &\text{si } e(x) = 0 \text{ entonces la frontera es Neumann} \\ &\text{si } e(x) \neq 0 \text{ entonces la frontera es Robin} \end{aligned} \tag{1.21}$$

Se tiene que

$$\begin{aligned} u &= u_0(\underline{\mathbf{x}}) + v(\underline{\mathbf{x}}) \qquad \text{en } \Gamma_D \\ u &= v(\underline{\mathbf{x}}) \qquad \text{en } \Gamma_N \end{aligned} \tag{1.22}$$

donde se busca que $v(\underline{\mathbf{x}})$ sea cero en la frontera Dirichlet y no importando que valor tome en la frontera Neumann.

Multiplicando por w la primera ecuación de (1.19), integrando sobre Ω, aplicando el Teorema de Gauss o de la divergencia **¡Error! No se encuentra el origen de la referencia.** y tomando en cuenta la derivada direccional **¡Error! No se encuentra el origen de la referencia.**, se tiene que

$$\begin{aligned}\int_\Omega wLu\, d\underline{\mathbf{x}} &= \int_\Omega \left(\nabla w \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u - u\underline{\mathbf{b}} \cdot \nabla w + \mathbf{c}wu\right) d\underline{\mathbf{x}} - \int_\Gamma w\left(\mathbf{a}_n \frac{\partial u}{\partial \hat{n}} - \mathbf{b}_n u\right) ds \\ &= \int_\Omega fw\, d\underline{\mathbf{x}}\end{aligned} \tag{1.23}$$

donde $\mathbf{a}_n = \underline{\hat{\mathbf{n}}} \cdot \underline{\underline{\mathbf{a}}} \cdot \underline{\hat{\mathbf{n}}}$ y $\mathbf{b}_n = \underline{\mathbf{b}} \cdot \underline{\hat{\mathbf{n}}}$

Se puede tomar que $w = 0$ en la parte Dirichlet de la frontera Γ y que $w \neq 0$ en la parte Neumann, se tiene entonces:

$$\begin{aligned}\int_\Omega wLu\, d\underline{\mathbf{x}} &= \int_\Omega \left(\nabla w \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u - u\underline{\mathbf{b}} \cdot \nabla w + \mathbf{c}wu\right) d\underline{\mathbf{x}} - \int_{\Gamma_N} w\left(\mathbf{a}_n \frac{\partial u}{\partial \hat{n}} - \mathbf{b}_n u\right) ds \\ &= \int_\Omega fw\, d\underline{\mathbf{x}}\end{aligned} \tag{1.24}$$

Sustituyendo en la ecuación anterior las condiciones de frontera Neumann tomadas de (1.19) se tiene

$$\int_\Omega \left(\nabla w \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u - u\underline{\mathbf{b}} \cdot \nabla w + \mathbf{c}wu\right) d\underline{\mathbf{x}} + \int_{\Gamma_N} wu\left(e - \mathbf{b}_n\right) ds = \int_\Omega fw\, d\underline{\mathbf{x}} + \int_{\Gamma_N} hw\, ds \tag{1.25}$$

Ahora, escogiendo un subespacio de dimensión finita V_N que contiene una base $\{w_1, \ldots, w_N\}$, tal que genera a $\hat{v}(x)$ como en (1.14), y tomando en cuenta (1.22), se tiene

$$\begin{aligned}&\sum_{j=1}^{N} v_j \int_\Omega \left(\nabla w_i \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla w_j - w_j \underline{\mathbf{b}} \cdot \nabla w_i + \mathbf{c}w_i w_j\right) d\underline{\mathbf{x}} + \int_{\Gamma_N} w_i w_j \left(e - \mathbf{b}_n\right) ds \\ &= \int_\Omega fw_i\, d\underline{\mathbf{x}} + \int_{\Gamma_N} hw_i\, ds - \int_\Omega \left(\nabla w_i \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u_0 - u_0 \underline{\mathbf{b}} \cdot \nabla w_i + \mathbf{c}w_i u_0\right) d\underline{\mathbf{x}}\end{aligned} \tag{1.26}$$

Haciendo

$$\begin{aligned}\mathrm{A}_{ij} &= \int_\Omega \left(\nabla w_i \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla w_j - w_j \underline{\mathbf{b}} \cdot \nabla w_i + \mathbf{c}w_i w_j\right) d\underline{\mathbf{x}} + \int_{\Gamma_N} w_i w_j \left(e - \mathbf{b}_n\right) ds \\ r_{ij} &= \int_\Omega fw_i\, d\underline{\mathbf{x}} + \int_{\Gamma_N} hw_i\, ds - \int_\Omega \left(\nabla w_i \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u_0 - u_0 \underline{\mathbf{b}} \cdot \nabla w_i + \mathbf{c}w_i u_0\right) d\underline{\mathbf{x}}\end{aligned} \tag{1.27}$$

Lo cual resulta en el sistema lineal de $N \times N$:

$$\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{v}} = \underline{\mathbf{r}} \tag{1.28}$$

De forma que la solución del problema está en:

$$\underline{\mathbf{v}} = \underline{\underline{\mathbf{A}}}^{-1} \cdot \underline{\mathbf{r}} \tag{1.29}$$

3.3 Procedimiento de Crank-Nicholson

En las secciones anteriores se presentó el método del elemento finito para distintos tipos de problemas en estado estacionario, es decir sin dependencia del tiempo. En esta sección se presenta la forma de resolver problemas transitorios (parabólicos), es decir dependientes del tiempo.

Se considera el problema

$$\begin{aligned} &\alpha \frac{\partial u}{\partial t} + \mathrm{L}u = f\left(\underline{\mathbf{x}}, t\right) && \text{en } \Omega \times \left[t_0, t_f\right] \\ &u\left(x, t\right) && \text{en } \Gamma_{\mathrm{D}} \times \left[t_0, t_f\right] \\ &a_n \frac{\partial u}{\partial n} = h\left(\underline{\mathbf{x}}, t\right) - e\left(\underline{\mathbf{x}}, t\right) u && \text{en } \Gamma_{\mathrm{N}} \times \left[t_0, t_f\right] \end{aligned} \tag{1.30}$$

Donde $\mathrm{L}u$ (el símbolo) es el mismo que se definió en (1.2), Γ_{D} representa la frontera Dirichlet y Γ_{N} la frontera Neumann. Además, se toma en cuenta la condición inicial:

$$u\left(\underline{\mathbf{x}}, t_0\right) = u_0(\underline{\mathbf{x}}) \tag{1.31}$$

El procedimiento de semi-discretización de Crank-Nicholson hace uso de la siguiente aproximación:

$$u\left(t + \frac{1}{2}\Delta t\right) = \left(1 - \theta\right) u\left(t\right) + \theta u\left(t + \Delta t\right) + O\left(\Delta t^2\right) \tag{1.32}$$

$$u_t\left(t + \frac{1}{2}\Delta t\right) = \frac{u\left(t + \Delta t\right) - u\left(t\right)}{\Delta t} + O(\Delta t^2) \tag{1.33}$$

Donde $0<\theta\leq 1$, O es el orden del error y $u_t=\frac{\partial u}{\partial t}$. Ahora, usando la notación $u^k=u\left(x,t_k\right)=u(x,t_0+k\Delta t)$, y sustituyendo (1.32) en (1.33) se tiene:

$$u_t^{k+\frac{1}{2}}=\frac{1}{\theta}\left(u^{k+\frac{1}{2}}-(1-\theta)u^k-\theta u^k\right)+O(\Delta t^2) \tag{1.34}$$

Si se supone que se avanza hasta un tiempo $k+\frac{1}{2}$, entonces la primera ecuación del problema (1.30), puede reescribirse como:

$$\alpha u_t^{k+\frac{1}{2}}+\mathrm{L}u^{k+\frac{1}{2}}=f^{k+\frac{1}{2}} \tag{1.35}$$

donde, incorporando (1.34) a la anterior ecuación se tiene:

$$\mathrm{L}u^{k+\frac{1}{2}}+\frac{\alpha u^{k+\frac{1}{2}}}{\theta\Delta t}=f^{k+\frac{1}{2}}+\frac{\alpha u^k}{\theta\Delta t} \tag{1.36}$$

La cual es una ecuación diferencial parcial elíptica para $u^{k+\frac{1}{2}}(x)$, puesto que $f^{k+\frac{1}{2}}+\frac{\alpha u^k}{\theta\Delta t}$ son valores conocidos. Una vez que se le da solución a la ecuación (1.36), se puede obtener de forma iterativa el valor para el siguiente paso en el tiempo usando

$$u^{k+1}\left(x\right)=\frac{u^{k+\frac{1}{2}}-\left(1-\theta\right)u^k}{\theta} \tag{1.37}$$

Cabe destacar que el problema (1.35), puede reescribirse de la siguiente manera:

$$
\begin{aligned}
&\mathrm{L}'v = f'(\underline{\mathbf{x}}) \qquad \text{donde} \\
&\mathrm{L}'v = -\nabla \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla v + \nabla(\underline{\mathbf{b}} v) + \left(\mathbf{c} + \frac{\alpha}{\theta \Delta t}\right) v \\
&f'(\underline{\mathbf{x}}) = f^{k+\frac{1}{2}} + \frac{\alpha u^k}{\theta \Delta t} \\
&v(\underline{\mathbf{x}}) = u\left(\underline{\mathbf{x}}, t^{k+\frac{1}{2}}\right)
\end{aligned}
\tag{1.38}
$$

En (1.38) se hace más evidente que el problema es del tipo elíptico con $\left(\mathbf{c} + \frac{\alpha}{\theta \Delta t}\right)$ en lugar de c, y $\left(f^{k+\frac{1}{2}} + \frac{\alpha u^k}{\theta \Delta t}\right)$ como el término derecho de la ecuación. Las condiciones de frontera quedan definidas como

$$
\begin{aligned}
&v(\underline{\mathbf{x}}) = g\left(\underline{\mathbf{x}}, t^{k+\frac{1}{2}}\right) \qquad \forall \underline{\mathbf{x}} \in \Gamma_{\mathrm{D}} \\
&\mathbf{a}_n \frac{\partial v(\underline{\mathbf{x}})}{\partial n} = h\left(x, t^{k+\frac{1}{2}}\right) - e\left(x, t^{k+\frac{1}{2}}\right) v \qquad \forall \underline{\mathbf{x}} \in \Gamma_{\mathrm{N}}
\end{aligned}
\tag{1.39}
$$

El proceso iterativo de resolución se expresa entonces de la siguiente manera

$$
u^{k+1}(x) = \frac{v - (1-\theta) u^k}{\theta} \tag{1.40}
$$

Se observa que, para la resolución de (1.38) por el método del elemento finito, se requiere resolver la integral

$$
\int_{\Omega} u^k w_i \, d\underline{\mathbf{x}} \tag{1.41}
$$

la cual tiene la siguiente solución

$$
\begin{aligned}
&\int_{\Omega} u^k w_i \, d\underline{\mathbf{x}} = \sum_j M_{ij} u_j^k \qquad \text{donde} \\
&\mathrm{M}_{ij} = \int_{\Omega} w_i w_j \, d\underline{\mathbf{x}}
\end{aligned}
\tag{1.42}
$$

3.4 Solución de Sistemas de Ecuaciones Lineales

A lo largo de este capítulo, hemos visto como proceder para transformar un problema de ecuaciones diferenciales parciales, con valores en la frontera, en un sistema algebraico de ecuaciones lineales, que se puede expresar en la forma matricial

$$\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}} \tag{1.43}$$

La elección del método específico para resolver el sistema de ecuaciones depende de las propiedades particulares de la matriz $\underline{\underline{\mathbf{A}}}$. En términos generales, si el problema de ecuaciones diferenciales parciales, con valores en la frontera y en dos dimensiones, se discretiza usando una malla de $n \times m$ nodos, el sistema algebraico de ecuaciones asociado es del orden de $(n \times m)^2$, pero en general la matriz $\underline{\underline{\mathbf{A}}}$ resultante para el tipo de problemas de interés en el presente trabajo es bandada, por ello es posible hacer uso de estas características para resolver el sistema algebraico de ecuaciones de forma óptima.

Los métodos de resolución del sistema algebraico de ecuaciones $\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}}$ se clasifican en dos grandes grupos (Nakamura, 1995):

- métodos directos
- métodos iterativos

En los métodos directos la solución $\underline{\mathbf{u}}$ se obtiene en un número fijo de pasos y sólo están sujetos a los errores de redondeo. En los métodos iterativos, se realizan iteraciones para aproximarse a la solución $\underline{\mathbf{u}}$ aprovechando las características propias de la matriz $\underline{\underline{\mathbf{A}}}$, tratando de usar un menor número de pasos que en un método directo.

Los métodos iterativos rara vez se usan para resolver sistemas lineales de dimensión pequeña (aunque el concepto de "dimensión pequeña" puede ser muy relativo), ya que el tiempo necesario para conseguir una exactitud satisfactoria rebasa el que requieren los métodos directos. Sin embargo, en el caso de sistemas grandes con un alto porcentaje de elementos cero, son eficientes, tanto en el almacenamiento en la computadora, como en el tiempo que se invierte en su solución. Por esta razón, al resolver éstos sistemas algebraicos de ecuaciones es preferible aplicar los métodos iterativos tales como son; Jacobi, Gauss-Seidel, sobrerrelajación sucesiva (SOR), etc.

Además, es importante mencionar que la mayoría del tiempo de cómputo necesario para resolver el problema de ecuaciones diferenciales parciales, es consumido en la solución del sistema algebraico de ecuaciones asociado a la discretización, por ello es determinante elegir aquel método numérico que minimice el tiempo invertido en este proceso.

En el **¡Error! No se encuentra el origen de la referencia.**) se presenta una breve recopilación de los métodos más comunes.

3.5 Procedimiento iterativo Gauss-Seidel sobrerrelajado (SOR) para matrices bandadas

En el **¡Error! No se encuentra el origen de la referencia.**) se observa que el método SOR, parte del método Gauss-Seidel sobrerrelajando ese esquema, con lo cual se obtiene que:

$$u_i^{k+1} = (1-\omega)u_i^k + \omega\left[\sum_{j=1}^{i-1} a_{ij}u_j^{k+1} + \sum_{j=i+1}^{N} a_{ij}u_j^k + b_i\right] \tag{1.44}$$

donde $\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}}$ y $\omega \in (0,2)$. En este método se supone que la matriz $\underline{\underline{\mathbf{A}}}$ es densa, sin embargo la adaptación a matrices bandadas es muy sencilla, tomando en cuenta solamente los valores "no vacíos" o correspondientes a las bandas.

A continuación se presenta la adaptación del algoritmo del SOR a matrices bandadas:

Entradas: A, b, ω
Salida: φ

Se elige una solución inicial $\varphi^{(0)}$
repeat hasta la convergencia
    for *i* **from** 1 **until** *n* **do**
        $\sigma \leftarrow 0$
        for *c* **from** 1 **until** *NB* **do** (donde *NB* representa el número de bandas)
            $j \leftarrow$ columna correspondiente a la banda c en el renglón i
            if *j*>*i* **then** $\sigma \leftarrow \sigma + a_{ij}\phi_j^{(k)}$
            else if *j*<*i* **then** $\sigma \leftarrow \sigma + a_{ij}\phi_j^{(k+1)}$
        end (*c*)
    $\phi_i^{(k+1)} \leftarrow (1-\omega)\phi_i^{(k)} + \dfrac{\omega}{a_{ii}}(b_i - \sigma)$
    end (*i*)
    verificar si se alcanzó el criterio de convergencia
end (repeat)

El anterior algoritmo se usó en la implementación del programa computacional y resultó ser muy eficiente, tanto en el tiempo que tardaba en resolver los sistemas lineales, como en el uso de memoria, puesto que usa matrices bandadas.

4 Implementación computacional

En este capítulo se presentan algunas consideraciones de carácter general que permiten la programación computacional del Método del Elemento Finito en dos dimensiones. Enseguida, se plantean conceptos básicos de la Metodología de Programación Orientada a Objetos y su subsecuente utilización en la implementación del Método del Elemento Finito Estándar para ecuaciones Advectivo-Difusivo-Reactivas, el cual se presentó en el capítulo anterior. Por último se describe cómo se implementó un programa en JAVA que da solución numérica a un modelo de flujo convectivo en medio poroso.

4.1 Consideraciones de programación

Si bien en el capítulo anterior se planteó el Método del Elemento Finito en algunas de sus distintas variantes, hay algunas consideraciones particulares que deben tomarse en cuenta cuando se implementan éstos en un sistema computacional; consideraciones válidas de manera general, independientemente de la plataforma o lenguaje de programación elegido.

Los aspectos esenciales de un programa computacional que implemente el método del elemento finito cubren los siguientes tópicos:

1. Entrada de datos referentes al dominio Ω, el lado derecho de la función f, las condiciones de frontera y los coeficientes que definen el operador L (Ecuación **¡Error! No se encuentra el origen de la referencia.**) los cuales dependen del problema en cuestión.
2. Construcción de la triangulación K_h.
3. Cómputo y ensamble de la matriz de rigidez $\underline{\underline{\mathbf{A}}}$ y el vector $\underline{\mathbf{f}}$ correspondiente al lado derecho de la ecuación.
4. Solución del sistema lineal de la ecuación $\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{v}} = \underline{\mathbf{r}}$ (Ecuación **¡Error! No se encuentra el origen de la referencia.**).
5. Salida de los resultados computacionales

De los puntos enumerados anteriormente, el primero y el último, normalmente se tratan de manera separada al programa principal que resuelve el FEM, puesto que éstos dependen en gran medida de la plataforma de hardware y software usados. Cuando se utiliza la metodología de programación orientada a objetos se hace más evidente el tratamiento por "separado" del manejo de las entradas y salidas. En relación al cuarto punto, en el capítulo anterior ya se mencionaron los métodos más comúnmente utilizados, así que a continuación se hará un breve tratamiento de los puntos restantes.

4.1.1 Construcción de la triangulación K_h

En esta sección se tratará únicamente la creación de un mallado conformado por triángulos o *triangulación,* puesto que se trata de un mallado fácil de implementar y lo suficientemente flexible para adaptarse a distintas formas de dominios. En los trabajos de Chen (2005); Chen, Huan, y Yuanle (2006) se pueden revisar otras técnicas de mallado.

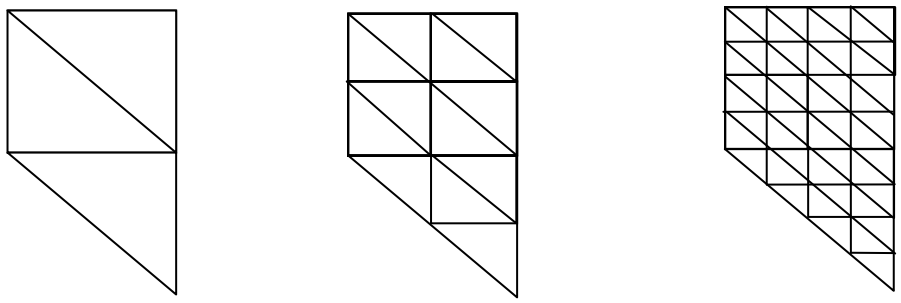


Figura 1. Proceso de refinamiento uniforme de una malla

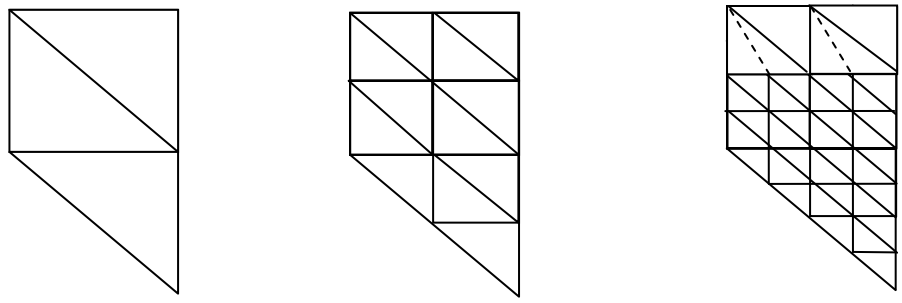


Figura 2. Proceso de refinamiento no-uniforme de una malla, donde una estrategia de refinamiento local se lleva a cabo en el tercer ejemplo

La triangulación K_h puede construirse a partir de refinar sucesivamente una partición más amplia del dominio Ω; donde triángulos más finos se obtienen conectando los puntos medios de los lados del triángulo, por poner un ejemplo. Una secuencia de diferentes etapas de refinación uniforme, conlleva a mallas *cuasi-uniformes*, donde los triángulos en K_h esencialmente tienen el mismo tamaño en todas las regiones de Ω (Figura 1). Si la frontera Γ de Ω es una curva, se debe realizar un tratamiento especial en los elementos cercanos a la frontera, estos casos están fuera del alcance del presente trabajo, pero si se requiere se puede revisar Chen, Huan, y Yuanle (2006) para más referencia.

En aplicaciones prácticas, comúnmente es necesario utilizar triángulos K_h que varíen considerablemente en tamaño a través de diferentes regiones de Ω. Por ejemplo, se pueden utilizar triángulos pequeños en las regiones donde se sabe que la solución exacta tiene una variación rápida o bien donde ciertos términos del operador diferencial son grandes, a esta estrategia se le llama *Refinamiento Local* (Figura 2). En esta estrategia, se tiene que tener especial cuidado en las zonas de transición entre regiones con triángulos de diferente tamaño. Por ejemplo en la Figura 2 la línea punteada en el tercer dibujo, representa un detalle que tiene cuidarse en el refinamiento local. Los métodos que automáticamente refinan la malla cuando es necesario se llaman *métodos adaptativos*, los cuales están fuera del alcance de este trabajo, pero se pueden revisar en Chen, Huan, y Yuanle (2006).

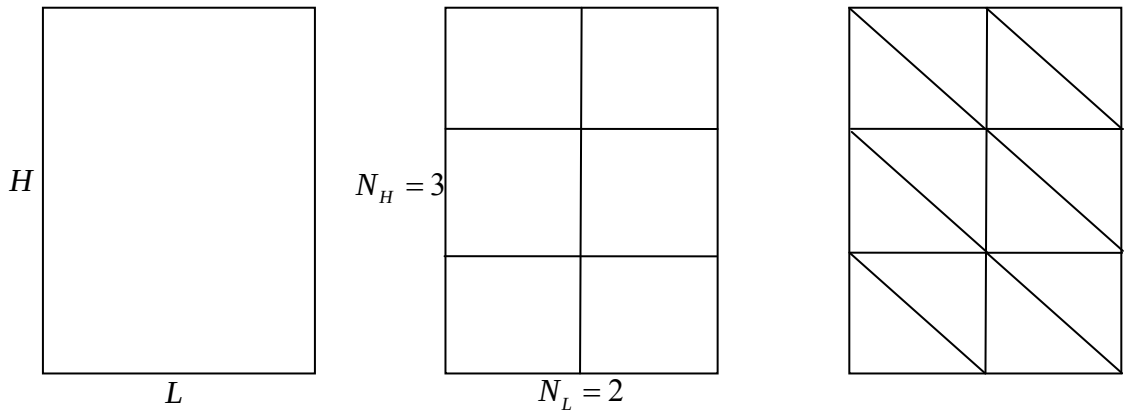


Figura 3. Proceso de creación de un mallado triangular sencillo a partir de un dominio rectangular

Un método de creación del mallado que es práctico y sencillo, siempre y cuando el dominio sea de forma rectangular, consiste en dividir el dominio original en rectángulos regulares. Esto se logra dividiendo el ancho L entre un número entero N_L, y el alto H entre un

entero N_H. Enseguida, se obtiene la triangulación "partiendo" en dos cada rectángulo uniendo de forma regular dos de los vértices opuestos del mismo (Figura 3).

Haciendo que una triangulación K_h tenga M nodos y M triángulos. La triangulación puede representarse mediante dos arreglos $\mathbf{Z}(2,M)$ y $Z(3,\mathrm{M})$, donde $\mathbf{Z}(i,j)$ $(i=1,2)$ indica las coordenadas del j-ésimo nodo, $j=1,2,\ldots,M$, y $Z(i,k)$ $(i=1,2,3)$ enumera los nodos del k-ésimo triángulo, $k=1,2,\ldots,\mathrm{M}$.

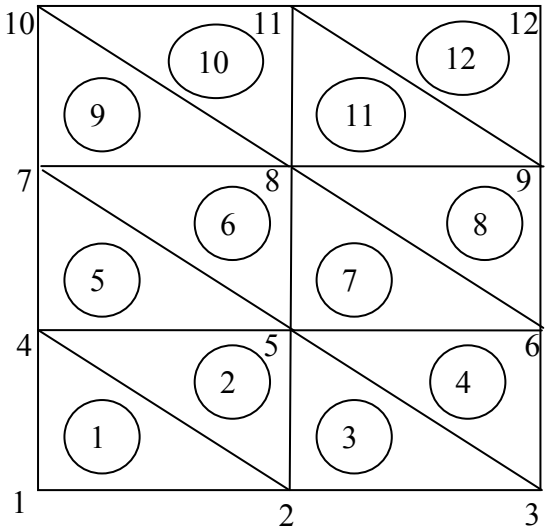


Figura 4. Numeración de nodos y triángulos en una triangulación

En la Figura 4 se muestra un ejemplo de numeración, donde la numeración de los triángulos es la que se marca con números encerrados en círculos, mientras que la numeración de los nodos se representa solamente con números. Para este ejemplo, el arreglo $Z(3,\mathrm{M})$ tiene la siguiente forma (donde $M=\mathrm{M}=12$),

$$Z=\begin{pmatrix} 1 & 2 & 2 & 3 & 4 & 5 & 5 & 6 & 7 & 8 & 8 & 9 \\ 2 & 5 & 3 & 6 & 5 & 8 & 6 & 9 & 8 & 11 & 9 & 12 \\ 4 & 4 & 5 & 5 & 7 & 7 & 8 & 8 & 10 & 10 & 11 & 11 \end{pmatrix} \qquad (1.1)$$

4.1.2 Ensamble de la matriz de rigidez

Una vez que la triangulación K_h se ha construido, se procede al cómputo de la matriz de rigidez con las entradas a_{ij}^k, donde $\underline{\underline{\mathbf{A}}}^k = \left(a_{ij}^k\right)$ dadas por la ecuación **¡Error! No se encuentra el origen de la referencia.** (en el caso más general de ecuación diferencial parcial) para cada triángulo k de la triangulación. Es importante hacer énfasis en el hecho de que $a_{ij}^k = 0$ a menos que los nodos $\mathbf{x}_i$ y $\mathbf{x}_j$ sean ambos vértices de $K \in K_h$.

Para el k-ésimo triángulo K_k, $Z\left(m,k\right)\left(m=1,2,3\right)$ son los número de vértices de K_k y el elemento de la matriz de rigidez $\underline{\underline{\mathbf{A}}}^{(k)} = \left(a_{mn}^k\right)_{m,n=1}^3$ y se calcula (basándose en el caso más general de la ecuación **¡Error! No se encuentra el origen de la referencia.**)

$$\begin{aligned} &a_{mn}^k = \int_{K_k} \left(\nabla w_m \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla w_n - w_n \underline{\mathbf{b}} \cdot \nabla w_m + \mathbf{c} w_m w_n\right) d\underline{\mathbf{x}} + \int_{\Gamma_N} w_m w_n \left(e - \mathbf{b}_{normal}\right) ds \\ &m,\ n\ =\ 1,\ 2,\ 3 \end{aligned} \tag{1.2}$$

donde las funciones de la base (lineal) w_m sobre K_k satisface

$$w_m\left(x_{Z(n,k)}\right) = \begin{cases} 1 & \text{si } m = n \\ 0 & \text{si } m \neq n \end{cases} \tag{1.3}$$

El lado derecho de la ecuación **¡Error! No se encuentra el origen de la referencia.** , $\underline{\mathbf{r}}^{(k)} = \left(r_m^k\right)_{m=1}^3$ sobre K_k se calcula

$$\begin{aligned} &r_{mn}^k = \int_{K_k} f w_m \, d\underline{\mathbf{x}} + \int_{\Gamma_N} h w_m \, ds - \int_{K_k} \left(\nabla w_m \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u_0 - u_0 \underline{\mathbf{b}} \cdot \nabla w_m + \mathbf{c} w_m u_0\right) d\underline{\mathbf{x}} \\ &m\ =\ 1,\ 2,\ 3 \end{aligned} \tag{1.4}$$

Es importante resaltar que m y n son los números locales de los tres vértices de K_k, mientras que i y j usados en **¡Error! No se encuentra el origen de la referencia.** son los números globales de los vértices en K_h.

Para ensamblar la matriz global $\underline{\underline{\mathbf{A}}} = \left(a_{ij}\right)$ y el vector del lado derecho $\underline{\mathbf{r}} = \left(r_j\right)$, se realiza un bucle sobre los triángulos K_k y sucesivamente se va adicionando las contribuciones de los diferentes K_k's:

$$\text{For } k = 1, 2 \ldots, \text{M} \text{ , calcula}$$

$$a_{Z(m,k),Z(n,k)} = a_{Z(m,k),Z(n,k)} + a_{mn}^{k}, \qquad (1.5)$$

$$r_{Z(m,k)} = r_{Z(m,k)} + r_{m}^{k} \qquad m, n = 1, 2, 3$$

En este caso se usa la aproximación *orientada a elementos*, es decir, el bucle se realiza sobre los elementos (por ejemplo, triángulos). Esta orientación es más eficiente que la orientación a nodos, puesto que esta última desperdicia mucho tiempo realizando cálculos repetidos de $\underline{\underline{\mathbf{A}}}$ y $\underline{\mathbf{r}}$.

4.2 Programación Orientada a Objetos

La Programación Orientada a Objetos (POO u OOP según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo, y encapsulamiento. Su uso se popularizó a principios de la década de 1990. Actualmente varios lenguajes de programación soportan la orientación a objetos.

Los objetos son entidades que combinan estado, comportamiento e identidad. El estado está compuesto de datos, y el comportamiento por procedimientos o métodos. La identidad es una propiedad de un objeto que lo diferencia del resto. La programación orientada a objetos expresa un programa como un conjunto de estos objetos, que colaboran entre ellos para realizar tareas. Esto permite hacer los programas y módulos más fáciles de escribir, mantener y reutilizar.

De esta forma, un objeto contiene toda la información que permite definirlo e identificarlo frente a otros objetos pertenecientes a otras clases e incluso frente a objetos de una misma clase, al poder tener valores bien diferenciados en sus atributos. A su vez, los objetos disponen de mecanismos de interacción llamados métodos que favorecen la comunicación entre ellos. Esta comunicación favorece a su vez el cambio de estado en los propios objetos. Esta característica lleva a tratarlos como unidades indivisibles, en las que no se separan ni deben separarse el estado y el comportamiento.

Los métodos y atributos están estrechamente relacionados por la propiedad de conjunto. Esta propiedad destaca que una clase requiere de métodos para poder tratar los atributos con los que cuenta. El programador debe pensar indistintamente en ambos conceptos, sin separar ni darle mayor importancia a ninguno de ellos. Hacerlo podría resultar en seguir el hábito erróneo de crear clases contenedoras de información por un lado y clases con métodos que la manejen por el otro. De esta manera se estaría llegando a una programación estructurada camuflada en un lenguaje de programación orientado a objetos.

Esto difiere de la programación estructurada tradicional, en la que los datos y los procedimientos están separados y sin relación, ya que lo único que se busca es el procesamiento de unos datos de entrada para obtener otros de salida. La programación estructurada anima al programador a pensar sobre todo en términos de procedimientos o funciones, y en segundo lugar en las estructuras de datos que esos procedimientos manejan. En la programación estructurada sólo se escriben funciones que procesan datos. Los programadores que emplean éste nuevo paradigma, en cambio, primero definen objetos para luego enviarles mensajes solicitándoles que realicen sus métodos por sí mismos.

4.2.1 Conceptos Básicos en la POO

La programación orientada a objetos es una forma de programar que trata de encontrar una solución a estos problemas. Introduce nuevos conceptos, que superan y amplían conceptos antiguos ya conocidos. Entre ellos destacan los siguientes:

Objeto: entidad provista de un conjunto de propiedades o atributos (datos) y de comportamiento o funcionalidad (métodos). Corresponden a los objetos reales del mundo que nos rodea, o a objetos internos del sistema (del programa). Es una instancia a una clase.

Clase: definiciones de las propiedades y comportamiento de un tipo de objeto concreto. La instanciación es la lectura de estas definiciones y la creación de un objeto a partir de ellas.

Método: algoritmo asociado a un objeto (o a una clase de objetos), cuya ejecución se desencadena tras la recepción de un "mensaje". Desde el punto de vista del comportamiento, es lo que el objeto puede hacer. Un método puede producir un cambio en las propiedades del objeto, o la generación de un "evento" con un nuevo mensaje para otro objeto del sistema.

Evento: un suceso en el sistema (tal como una interacción del usuario con la máquina, o un mensaje enviado por un objeto). El sistema maneja el evento enviando el mensaje adecuado al objeto pertinente. También se puede definir como evento, a la reacción que puede desencadenar un objeto, es decir la acción que genera.

Mensaje: una comunicación dirigida a un objeto, que le ordena que ejecute uno de sus métodos con ciertos parámetros asociados al evento que lo generó.

Propiedad o atributo: contenedor de un tipo de datos asociados a un objeto (o a una clase de objetos), que hace los datos visibles desde fuera del objeto y esto se define como sus características predeterminadas, y cuyo valor puede ser alterado por la ejecución de algún método.

Estado interno: es una propiedad invisible de los objetos, que puede ser únicamente accedida y alterada por un método del objeto, y que se utiliza para indicar distintas situaciones posibles para el objeto (o clase de objetos).

Componentes de un objeto: atributos, identidad, relaciones y métodos.

Representación de un objeto: un objeto se representa por medio de una tabla o entidad que esté compuesta por sus atributos y funciones correspondientes.

4.2.2 Características particulares de la POO

Hay un cierto desacuerdo sobre exactamente qué características de un método de programación o lenguaje le definen como "orientado a objetos", pero hay un consenso general en que las características siguientes son las más importantes:

Abstracción: cada objeto en el sistema sirve como modelo de un "agente" abstracto que puede realizar trabajo, informar y cambiar su estado, y "comunicarse" con otros objetos en el sistema sin revelar cómo se implementan estas características. Los procesos, las funciones o los métodos pueden también ser abstraídos y cuando lo están, una variedad de técnicas son requeridas para ampliar una abstracción.

Encapsulamiento: significa reunir a todos los elementos que pueden considerarse pertenecientes a una misma entidad, al mismo nivel de abstracción. Esto permite aumentar la cohesión de los componentes del sistema. Algunos autores confunden este concepto con el principio de ocultación, principalmente porque se suelen emplear conjuntamente.

Principio de ocultación: cada objeto está aislado del exterior, es un módulo natural y cada tipo de objeto expone una interfaz a otros objetos que específica cómo pueden interactuar con los objetos de la clase. El aislamiento protege a las propiedades de un objeto contra su modificación por quien no tenga derecho a acceder a ellas, solamente los propios métodos internos del objeto pueden acceder a su estado. Esto asegura que otros objetos no pueden cambiar el estado interno de un objeto de maneras inesperadas, eliminando efectos secundarios e interacciones inesperadas. Algunos lenguajes relajan esto, permitiendo un acceso directo a los datos internos del objeto de una manera controlada y limitando el grado de abstracción. La aplicación entera se reduce a un agregado o rompecabezas de objetos.

Polimorfismo: comportamientos diferentes, asociados a objetos distintos, pueden compartir el mismo nombre, al llamarlos por ese nombre se utilizará el comportamiento correspondiente al objeto que se esté usando. O dicho de otro modo, las referencias y las colecciones de objetos pueden contener objetos de diferentes tipos, y la invocación de un comportamiento en una referencia producirá el comportamiento correcto para el tipo real del objeto referenciado. Cuando esto ocurre en *tiempo de ejecución*, es decir mientras se ejecuta el programa, esta última característica se llama asignación tardía o asignación

dinámica. Algunos lenguajes proporcionan medios más estáticos (en *tiempo de compilación*) de polimorfismo, tales como las plantillas y la sobrecarga de operadores de C++.

Herencia: las clases no están aisladas, sino que se relacionan entre sí, formando una jerarquía de clasificación. Los objetos heredan las propiedades y el comportamiento de todas las clases a las que pertenecen. La herencia organiza y facilita el polimorfismo y el encapsulamiento permitiendo a los objetos ser definidos y creados como tipos especializados de objetos preexistentes. Estos pueden compartir y extender su comportamiento sin tener que re implementar su comportamiento. Esto suele hacerse habitualmente agrupando los objetos en clases y estas en árboles o enrejados que reflejan un comportamiento común. Cuando un objeto hereda de más de una clase se dice que hay herencia múltiple; esta característica no está soportada por algunos lenguajes (como Java).

4.2.3 UML

El Lenguaje Unificado de Modelado (UML, *Unified Modeling Language*) es un lenguaje de modelado visual de propósito general que se utiliza para especificar, visualizar, construir y documentar los artefactos de un sistema de software (Booch, Jacobson, y Rumbaugh, 2007). UML proporciona una forma estándar de representar los planos de un sistema, y comprende tanto elementos conceptuales, como los procesos del negocio y las funciones del sistema, cuanto elementos concretos, como las clases escritas en un lenguaje de programación específico, esquemas de bases de datos y componentes software reutilizables (Booch, Jacobson, y Rumbaugh, 2006). UML es sólo un lenguaje y, por tanto, es tan sólo una parte de un método de desarrollo de software, así que UML es independiente del proceso, así como también es independiente a un lenguaje de programación en particular.

Para comprender UML, se necesita adquirir un modelo conceptual del lenguaje, y esto requiere aprender tres elementos principales:

- los bloques básicos de construcción de UML
- las reglas que dictan cómo se pueden combinar estos bloque básicos
- y algunos mecanismos comunes que se aplican a través de UML

Una vez comprendidas estas ideas, se pueden leer modelos UML y crear algunos modelos básicos.

El vocabulario de UML incluye tres clases de bloques básicos:

1. Elementos: abstracciones que constituyen los ciudadanos de primera clase en un modelo
2. Relaciones: ligan a los elementos entre sí
3. Diagramas: agrupan colecciones interesantes de elementos

En este trabajo, con la finalidad de facilitar el entendimiento de la implementación computacional, se hará uso principalmente de los diagramas de clases, de componentes y de paquetes.

Un *diagrama de clases* muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Estos diagramas son los diagramas más comunes en el modelado de sistemas orientados a objetos. Los diagramas de clase abarcan la vista de diseño estática de un sistema. Los *diagramas de componentes* son una variante de los diagramas de clases. Representan la encapsulación de una clase, junto con sus interfaces, puertos y estructura interna, la cual está formada por otros componentes anidados y conectores. Los diagramas de componentes cubren la vista de implementación estática del diseño de un sistema. Un *diagrama de paquetes* muestra la descomposición del propio modelo en unidades organizativas y sus dependencias.

4.3 Programa en JAVA

Para realizar la implementación computacional se eligió el lenguaje de programación JAVA. JAVA es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma muchas de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros o memoria.

Cabe destacarse el uso de JAVA puesto que este lenguaje de programación es multiplataforma, es decir, el programa puede ejecutarse, sin necesidad de compilarse nuevamente, en diversos sistemas operativos, como podrían ser Windows, UNIX, LINUX, etc. Otra ventaja que ofrece JAVA, al ser un lenguaje orientado a objetos, es que hace que el código sea fácilmente reutilizable, puesto que el programa puede modificarse o bien irse escalando de manera muy sencilla.

La programación se realizó usando el entorno de desarrollo integrado (IDE, *Integrated Development Environment*) NetBeans (NetBeans, 2009), el cual es un conjunto de programas empaquetado como un programa de aplicación que permite editar código, compilar, depurar y realizar otras funciones a partir de una interfaz gráfica (GUI). NetBeans tiene además la ventaja de ser un producto libre y gratuito sin restricciones de uso.

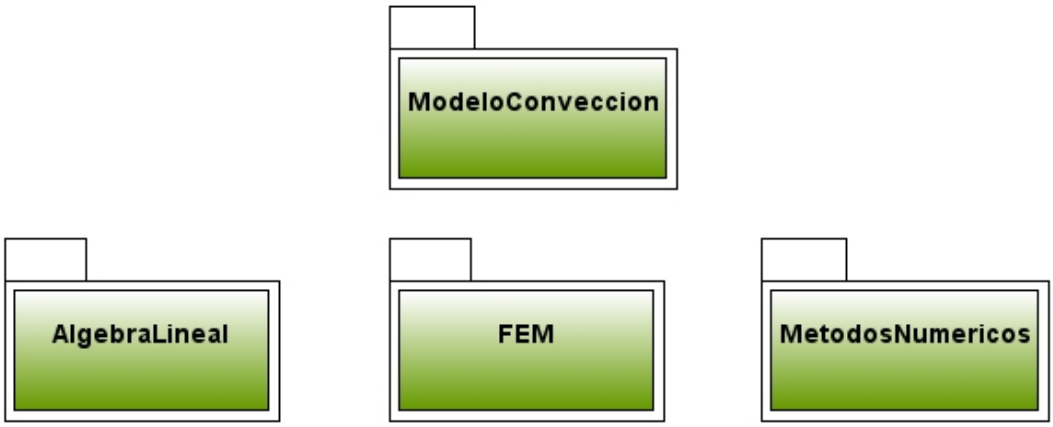


Figura 5. Paquetes que componen el programa en JAVA

4.3.1 Estructura básica del programa

El programa está conformado por cuatro paquetes (Figura 5):

1. **AlgebraLineal**: incluye las clases usadas para manejar matrices y vectores
2. **MetodosNumericos**: contiene las clases que permiten realizar interpolaciones, solucionar sistemas lineales y hacer integraciones y diferenciaciones numéricas
3. **FEM**: contiene las clases que implementan el FEM para ecuaciones Advectivo-Difusivo-Reactivas
4. **ModeloConveccion**: clases que permiten controlar la definición y el funcionamiento general del modelo de convección. También incluye la interfaz que permite "conectar" el programa con medios externos.

El paquete **AlgebraLineal** incluye las siguientes clases:

- **Matriz:** Es una clase abstracta donde se definen las operaciones comunes a todos los tipos de matrices.
- **MatrizBandada:** Esta clase es una extensión de la clase Matriz, en donde se definen las operaciones particulares de una matriz bandada.
- **MatrizDensa:** Esta clase es una extensión de la clase Matriz, en donde se definen las operaciones particulares de una matriz densa.
- **MatrizInt:** Esta clase sirve para definir y operar Matrices con valores enteros.
- **MatrizTridiagonal:** Esta clase es una extensión de la clase Matriz, en donde se definen las operaciones particulares de una matriz tridiagonal.
- **Vector.** Es una clase donde queda definido un vector y su funcionalidad.
- **VectorInt.** Esta clase maneja vectores con valores enteros.

El paquete **MetodosNumericos** se compone de las clases:

- **Diferenciador:** Contiene métodos utilizados para realizar diferenciaciones numéricas.
- **Integrador:** Métodos usados para realizar integraciones numéricas
- **Interpolador:** Métodos numéricos utilizados para realizar interpolaciones
- **LinearSolve:** Métodos usados para resolver un sistema lineal. Incluye tanto métodos directos como iterativos.

El paquete **FEM** contiene las clases:

- **Base:** Clase abstracta usada para definir una Base que será usada por el FEM.
- **BaseLinealTriangulo:** Base que usa funciones lineales (Chapeau), definidas sobre un elemento bidimensional triangular formado por 3 nodos correspondientes a los vértices del triángulo.
- **Discretización:** Clase abstracta de las discretizaciones espaciales.
- **Dominio:** Clase abstracta usada para definir un dominio.
- **DominioRectangular:** Dominio bidimensional de forma rectangular.
- **Elemento:** Clase abstracta de un Elemento.
- **FemEstandar:** Clase que representa el Método del Elemento Finito (FEM) estándar para ecuaciones diferenciales parciales (EDP) advectivo-difusivo-reactivas.
- **Frontera:** Clase usada para definir las fronteras de un problema de una EDP.
- **Funcion:** Interfaz usada para valuar una función en un nodo.
- **FuncionExpresion:** Función que está definida por una expresión matemática.
- **FuncionValor:** Función definida por un valor correspondiente a un nodo de la discretización.
- **Nodo:** Clase que representa un Nodo en una discretización.
- **NodoBorde:** Subclase de Nodo que pertenece o forma parte del borde de un domino.
- **Problema:** Clase abstracta usada para definir un problema completo de una EDP.
- **ProblemaPruebaCompleta:** Clase que define un problema teórico completo en 2D (es decir, un problema en el cual todos los coeficientes de la EDP son no homogéneos y que además contiene condiciones de frontera Neumann y Dirichlet) el cual sirve para probar que el método de solución implementado funcione correctamente.
- **ProblemaPruebaCompletaB:** Esta clase es idéntica a ProblemaPruebaCompleta, con la única diferencia de que en ésta las "funciones" están definidas por valores definidos en los nodos (lo cual es lo más común en la práctica), mientras que la anterior clase las "funciones" quedan definidas por expresiones.
- **ProblemaStream:** Clase usada para definir la ecuación de flujo de masa en el modelo de flujo convectivo sobre un dominio rectangular.

- **ProblemaTemperatura:** Clase usada para definir la ecuación de flujo de calor en el modelo de flujo convectivo sobre un dominio rectangular.
- **PruebaFEM:** Clase usada para probar el FEM.
- **TriangulacionDominoRectangular:** Discretización espacial de un dominio rectangular, mediante la triangulación usando elementos compuestos por tres nodos correspondientes a los vértices del triángulo.
- **Triangulo3Nodos:** Elemento bidimensional triangular formado por tres nodos correspondientes a los vértices del triángulo.

El paquete **ModeloConveccion** contiene las clases:

- **DefinicionModelo:** Clase abstracta usada para definir modelos o problemas.
- **DefinicionModeloConveccion:** Interfaz usada para definir un problema de Convección. Esta interfaz contiene los métodos que leerán de un medio externo los parámetros que definen un modelo de flujo convectivo.
- **Main:** Clase usada para probar la solución de un modelo de flujo de calor convectivo.
- **ModeloFlujoConvectivo:** Clase que controla la definición y solución de un modelo de flujo convectivo en medio poroso.

En la Figura 6 puede observarse el diagrama de clases del paquete FEM, que es donde se centra la implementación computacional de la solución del modelo. La única clase que se ha excluido del diagrama es la clase **PruebaFEM**, puesto que esta clase implementa las corridas de prueba de convergencia en el espacio y en el tiempo del ejemplo "completo", siendo en realidad prescindible su explicación en el marco del funcionamiento del FEM.

Para el diseño del programa, se procuró en todo momento seguir los lineamientos del paradigma de orientación a objetos. De esta manera, como puede observarse en el diagrama de clases, el diseño resultante es bastante claro e incluso intuitivo para entender de forma gráfica tanto el método de FEM como la estructura básica del programa. Es muy importante mencionar que el resultado final en la elección de clases-objetos, no responde únicamente a un solo criterio, ya sea éste matemático, físico, numérico o de paradigma de implementación del programa. Por lo tanto, podrá pensarse que en algunos casos la elección de las clases y/o su jerarquía resulta ilógica o carente de sentido si se analiza ésta exclusivamente desde alguno de los criterios antes mencionados. Sin embargo debe recordarse que es una práctica muy común que esto suceda, puesto que muchas veces este tipo de decisiones de diseño van enfocadas a seguir los lineamientos del paradigma de orientación a objetos (Booch, 1996)

Si bien es cierto que el seguir este lineamiento implicó decisiones de diseño que, en algunos casos, pareciesen sacrificar el desempeño del programa, la claridad y consistencia del diseño elegido posibilitan el relativamente fácil mantenimiento y escalamiento de éste, lo

cual cumple con creces una de las finalidades principales de usar la metodología orientada a objetos. Por otro lado, siempre queda abierta la posibilidad de mejorar el desempeño en sucesivas "iteraciones" en el proceso de desarrollo del programa, lo cual podría facilitarse al contarse con un modelo de diseño base consistente y sencillo. También se intentó dejar sentados fundamentos que permitiesen en un futuro incrementar la funcionalidad del programa, extendiendo éste a distintas discretizaciones, uso de otras bases, variaciones del FEM o incluso a la solución de problemas en más o menos dimensiones.

4.3.2 Descripción del programa

Puesto que en secciones anteriores se ha hecho ya una descripción detallada de la forma en que se resuelven EDP usando FEM, así como también se ha tratado ya los detalles más importantes de su implementación, resta solamente describir la forma particular en la cual se diseñó el programa usando la metodología orientada a objetos. Para tal descripción se hará uso de los diagramas de clases, realizando una labor de síntesis que permita construir el sistema completo a partir de sus partes componentes. Además, si se requiere una comprensión más detallada del programa computacional, en el **¡Error! No se encuentra el origen de la referencia.**) se presenta un listado de las clases más importantes del programa.

Teniendo en cuenta el enfoque antes mencionado, se tiene que uno de los componentes básicos es el *dominio* sobre el cual quedará definido el modelo. En la Figura 7 puede observarse como se parte de una clase abstracta **Dominio**, la cual permite referirse a un dominio de manera general en las otras clases. En este caso particular se extiende la clase abstracta **Dominio** con una subclase llamada **DominioRectangular**, la cual representa un dominio rectangular definido por un par de vértices ubicados en un sistema coordenado bidimensional (x,z).

En la Figura 8 se observa cómo la clase abstracta **Base** se usa para definir de manera general una *base*, la cual es utilizada por el FEM para interpolar. En esta clase abstracta se define la estructura que tendrán los métodos usados para valuar la base y su gradiente. Enseguida se observa como la clase **BaseLinealTriangulo** extiende la clase abstracta correspondiente, con lo cual se logra definir una *base lineal* o Chapeau, sobre un triángulo.

Otro componente básico es una *función*, la cual permite valuar expresiones o bien asignar valores constantes a los nodos. La representación de ésta se logra a partir de la creación de la interfaz **Funcion** (Figura 9), la cual contiene el método que permite valuar una función en un nodo. La implementación de esta interfaz se lleva a cabo en las clases **FuncionValor** y **FuncionExpresion**, las cuales permiten definir el valor dado en nodo y valuar una expresión matemática respectivamente.

La definición de una *frontera* (Figura 10) se lleva a cabo en la clase **Frontera**, la cual contiene un identificador del tipo de función (Dirichlet o Neumann) y una **FuncionExpresion** la cual permite asignar un valor a la frontera de la cual se trata.

La clase **Nodo** (Figura 11) representa un *nodo* en el espacio n-dimensional, a la vez que permite guardar información referente a su ubicación en una discretización (a partir de los nodos vecinos a éste) y su identificador dentro de la numeración de incógnitas de un problema. Además existe un subtipo de *nodo*, **NodoBorde**, el cual representa un nodo que es parte del borde del dominio. Un **NodoBorde** permite asignar una lista de **Frontera** que representa las distintas fronteras a las cuales puede pertenecer el nodo borde.

Un *elemento* se representa con la clase abstracta **Elemento** (Figura 12), en la cual queda especificado que un elemento estará compuesto por un identificador, objetos **Nodo** y la estructura de los métodos que comparten todos los distintos tipos de elementos (como lo es el calcular una integral sobre el elemento). Es importante enfatizar que esta clase abstracta engloba todos los tipos de elementos que pueden utilizarse en el FEM, lo cual permite referirse a un *elemento* de forma genérica en otras partes del programa. La clase **Triangulo3Nodos** extiende la clase abstracta **Elemento**, con lo cual se representa a los elementos de forma triangular, compuestos por 3 nodos correspondientes a los vértices del triángulo. Además de implementar los métodos heredados de la superclase **Elemento**, esta clase contiene otros métodos que permiten trabajar sobre éste de forma particular.

En el diseño del programa se supuso que una *discretización espacial* se realiza sobre un *dominio*. La discretización genera la "división" del dominio en un conjunto de *elementos* que, a su vez, están conformados por *nodos*. Además, una discretización tiene asociada una *base* la cual será usada por FEM. La anterior conceptualización se plasmó en el programa creando una clase abstracta **Discretizacion** (Figura 13), la cual representa de manera general los distintos tipos de driscretizaciones espaciales que pueden llegar a presentarse. Se asocia a **Discretizacion** las clases **Dominio**, **Base**, **Elemento**, **Nodo**, y también se define la estructura que tendrán los métodos que permiten establecer las características comunes a las distintas discretizaciones y las operaciones que el FEM requiere de una discretización (por ejemplo la integración sobre una frontera). De forma particular se creó la clase **TriangulacionDominioRectangular**, que extiende la clase abstracta **Discretizacion** y permite representar la discretización espacial realizada a un dominio rectangular, de manera que se construyen elementos triangulares constituidos por tres nodos coincidentes a los vértices de cada triángulo. Esta triangulación corresponde al sencillo método descrito al final de la sección (4.1.1). En esta clase también se indica que *base* se usará, siendo ésta la **BaseLinealTriangulo** descrita con anterioridad.

La clase abstracta **Problema** (Figura 14) representa un *problema* descrito por una EDP de segundo orden (que puede ser advectivo-difusivo-reactiva) definida sobre un dominio, con sus respectivos valores en la frontera (que pueden ser Dirichlet o Neumann, tanto homogéneos como no homogéneos) y sus condiciones iniciales (en el caso de que se trate de un problema transitorio). Se observa como la clase **Problema** está asociada a la clase **Discretizacion**, la cual recibe como argumento en su constructor, por lo tanto para poder

crear un objeto **Problema**, se requiere tener previamente creado un objeto que represente una discretización espacial sobre un dominio en particular.

Además, **Problema** usa un arreglo de objetos **Funcion**, los cuales permiten establecer los "coeficientes" de la EDP en su forma de la divergencia (Ecuación**¡Error! No se encuentra el origen de la referencia.**) y el coeficiente α de los problemas transitorios (Ecuación**¡Error! No se encuentra el origen de la referencia.**). El hecho de que los "coeficientes" queden definidos por **Funcion**, permite que cada uno de éstos pueda ser determinado por una expresión matemática o bien mediante la definición de un valor en cada uno de los nodos de la discretización. El uso de un arreglo de **Frontera**, permite representar las fronteras del problema. En la clase abstracta, se define la estructura de los métodos comunes a los distintos tipos de *problemas*, y los métodos que servirán para definir de manera correcto los componentes de éstos.

En la Figura 14 se muestra también una de las clases que extienden la clase abstracta **Problema**. La clase **ProblemaPruebaCompleta** corresponde a un problema teórico completo que se utilizó para comprobar que el FEM estuviese funcionando de forma correcta. En esta clase se expresa explícitamente la manera en que se conforman los "coeficientes" de la EDP y las fronteras, correspondientes al problema usado para probar el FEM.

La resolución del *problema*, llevada a cabo mediante el FEM Estándar (descrito en la Sección **¡Error! No se encuentra el origen de la referencia.**), queda implementada en la clase **FemEstandar** (Figura 15), la cual usa una clase del tipo **Problema** (que recibe como argumento en su constructor) para representar los problemas a resolverse. Cabe destacarse que esta clase es capaz de resolver *problemas* de manera general, sin importar sobre qué tipo de dominio esté definido, o que discretización se halla utilizado, incluso no importa la dimensión sobre la cual esté definido el problema. Esta versatilidad se logra gracias a que se trabaja con componentes básicos cuyo comportamiento útil a este nivel, fue definido de manera general. Por ejemplo, cuando se procesa cada elemento de la discretización para obtener la matriz de rigidez local que permita ensamblar la matriz de rigidez global (Ecuación(1.2)), se le "pide" al elemento que se integre (pasándole los parámetros adecuados) sin importar si el elemento es una línea, un triángulo o un cubo, puesto que los detalles de cómo realizar esta integración se resuelven en la clase correspondiente al elemento en particular del cual se trate.

La definición de un modelo de flujo convectivo se muestra en la Figura 16. En este diagrama se presentan todas las clases del paquete **ModeloConveccion** (excepto la clase **Main**) y algunas clases del paquete **FEM** que permiten hacer más claro la forma en que se resuelve un modelo de convección en el programa.

En la Figura 17 se observa como la interfaz **DefinicionModeloConveccion2D** se ocupa para definir los métodos que permiten obtener los parámetros del modelo de flujo convectivo. Por otro lado se usa la clase abstracta **DefinicionModelo**, para posibilitar el uso de una clase que obtenga de un medio externo información referente a un *problema* (en la Figura 16 se observa como la clase **Problema** se asocia a **DefinicionModelo**, mientras que en la Figura 14 se ve como la clase **Problema** tiene un constructor que incluye en sus argumentos el uso de un objeto del tipo **DefinicionModelo**). La clase **PruebaDefinicionModeloConveccion** implementa la interfaz **DefinicionModeloConveccion2D** y además extiende la clase abstracta **DefinicionModelo**, con lo cual puede obtener información usada para definir los problemas de flujo de masa y de flujo de calor.

La **¡Error! No se encuentra el origen de la referencia.** muestra como la clase **ModeloFlujoConvectivo** controla la solución y acoplamiento de los problemas que conforman el modelo completo (siguiendo el procedimiento descrito en el Capítulo 2). Es importante resaltar que el acoplamiento se logra gracias a que las clases **ProblemaTemperatura** y **ProblemaStream** tienen un método que permite incorporar los valores de *Stream* y *Temperatura*, respectivamente, a los datos que usan para calcular sus soluciones. En este último diagrama de clases se observa con mayor claridad como la interfaz **DefinicionModeloConveccion**, la cual se asocia a **ModeloFlujoConvectivo**, **ProblemaTemperatura** y **ProblemaStream**, es la que aporta la información que define el modelo. De esta manera, la información proporcionada por la interfaz sirve para definir las EDP que modelan el flujo de calor y flujo de masa.

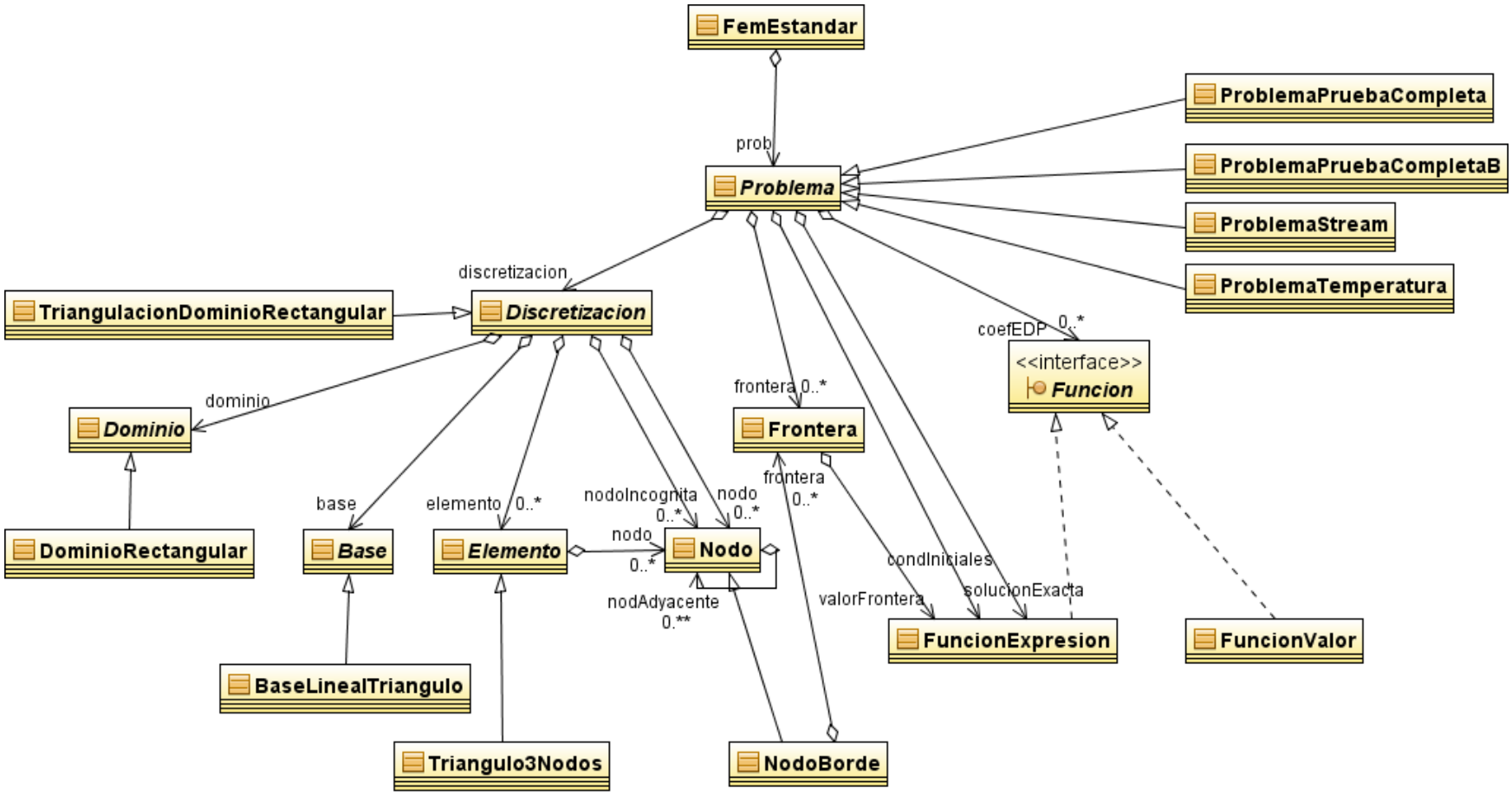


Figura 6. Diagrama de clases del paquete FEM

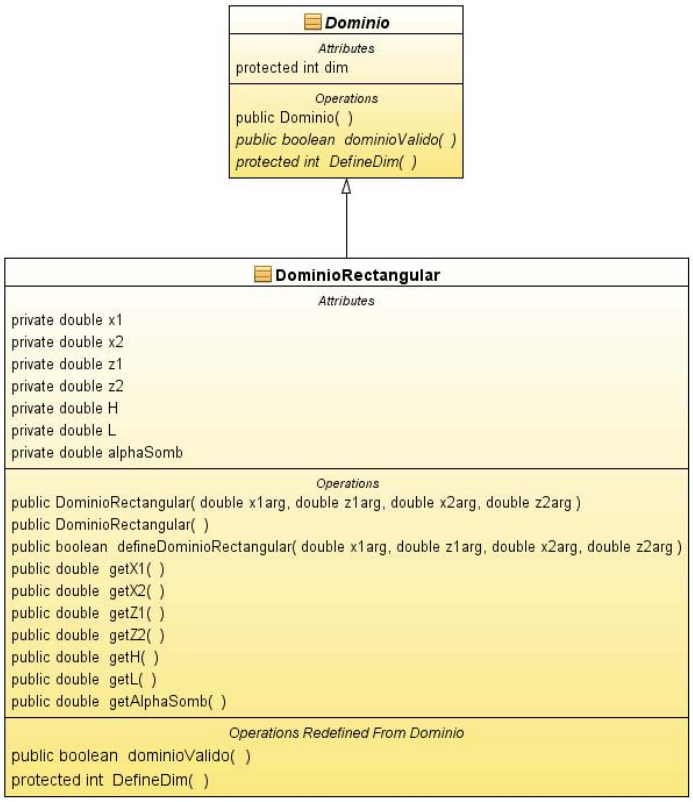


Figura 7. Definición de un Dominio

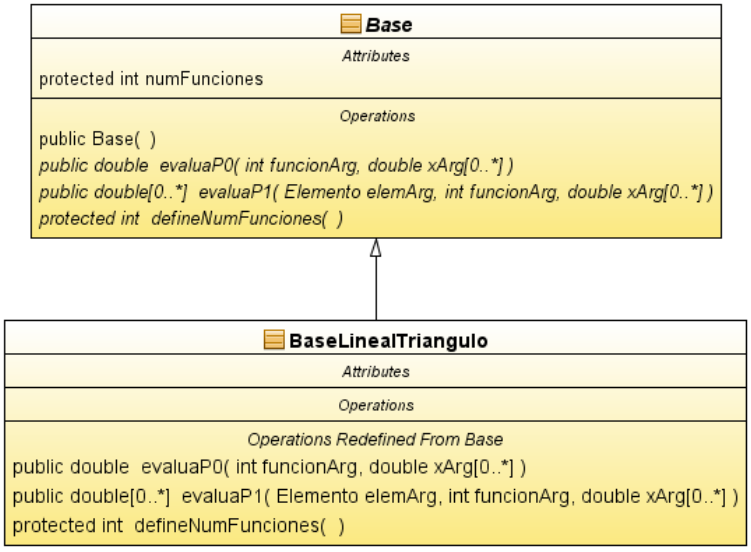


Figura 8. Definición de una Base

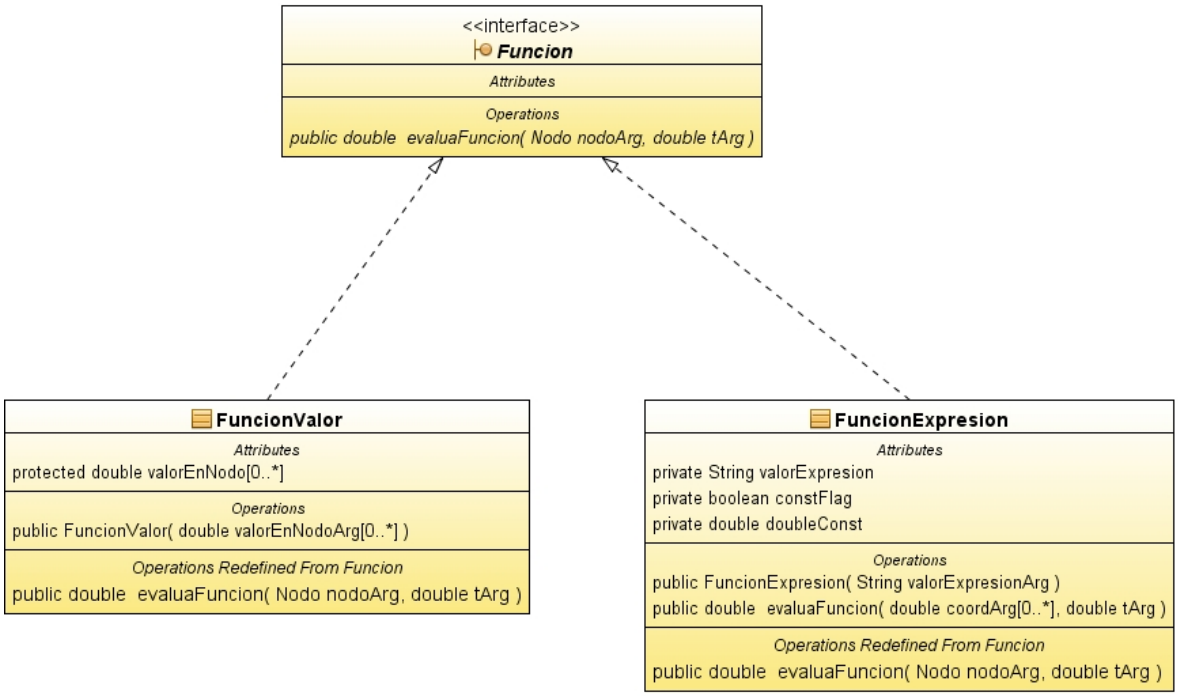


Figura 9. Definición de una Función

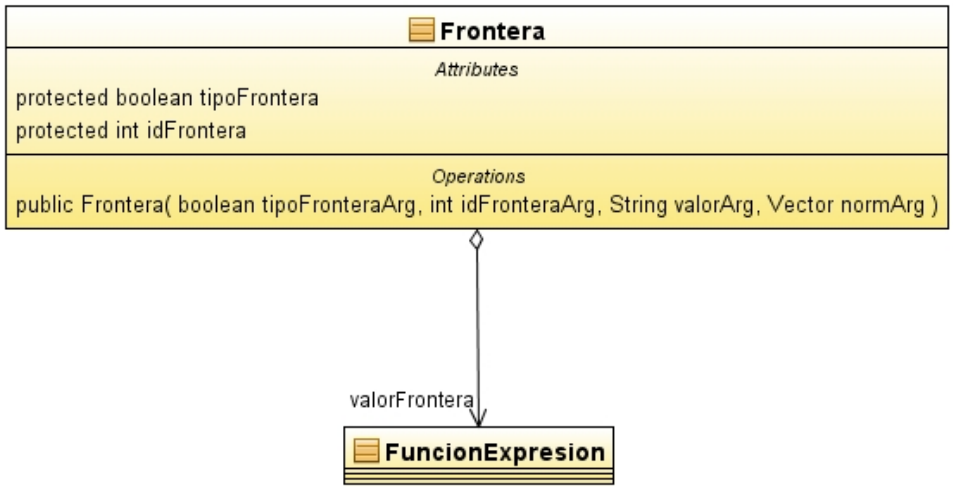


Figura 10. Definición de una Frontera

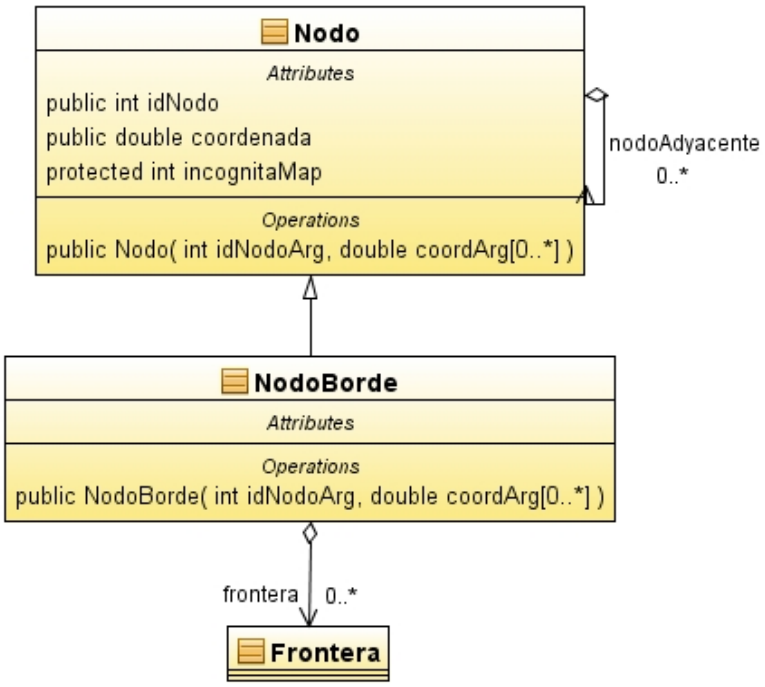


Figura 11. Definición de un Nodo

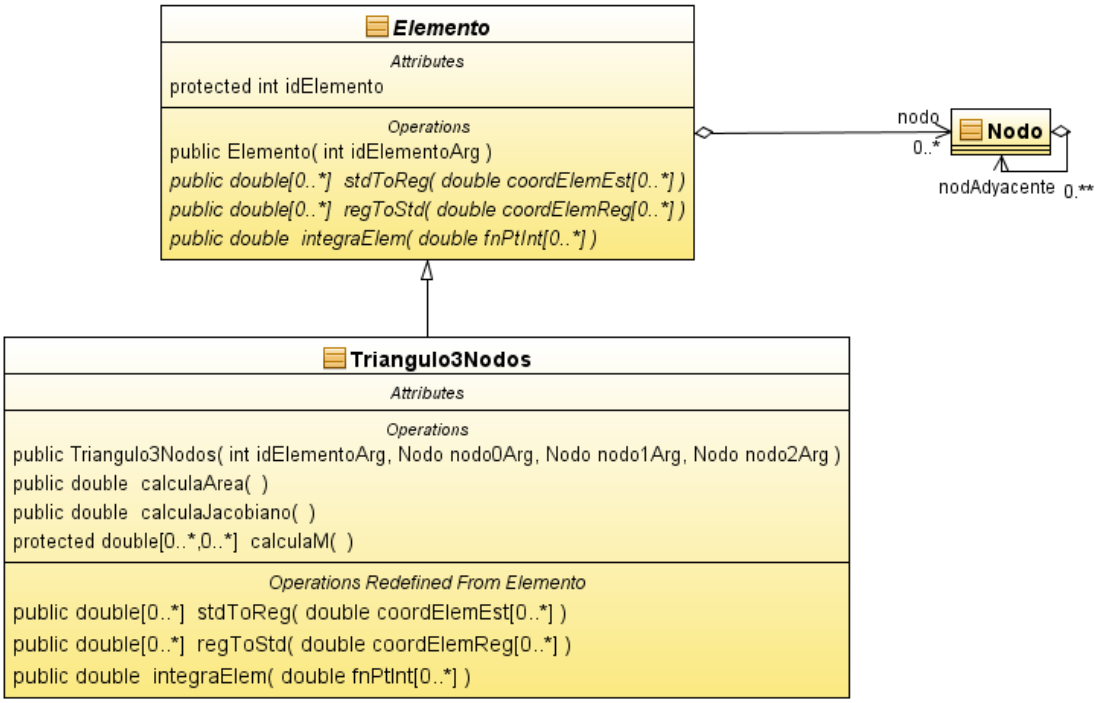


Figura 12. Definición de un Elemento

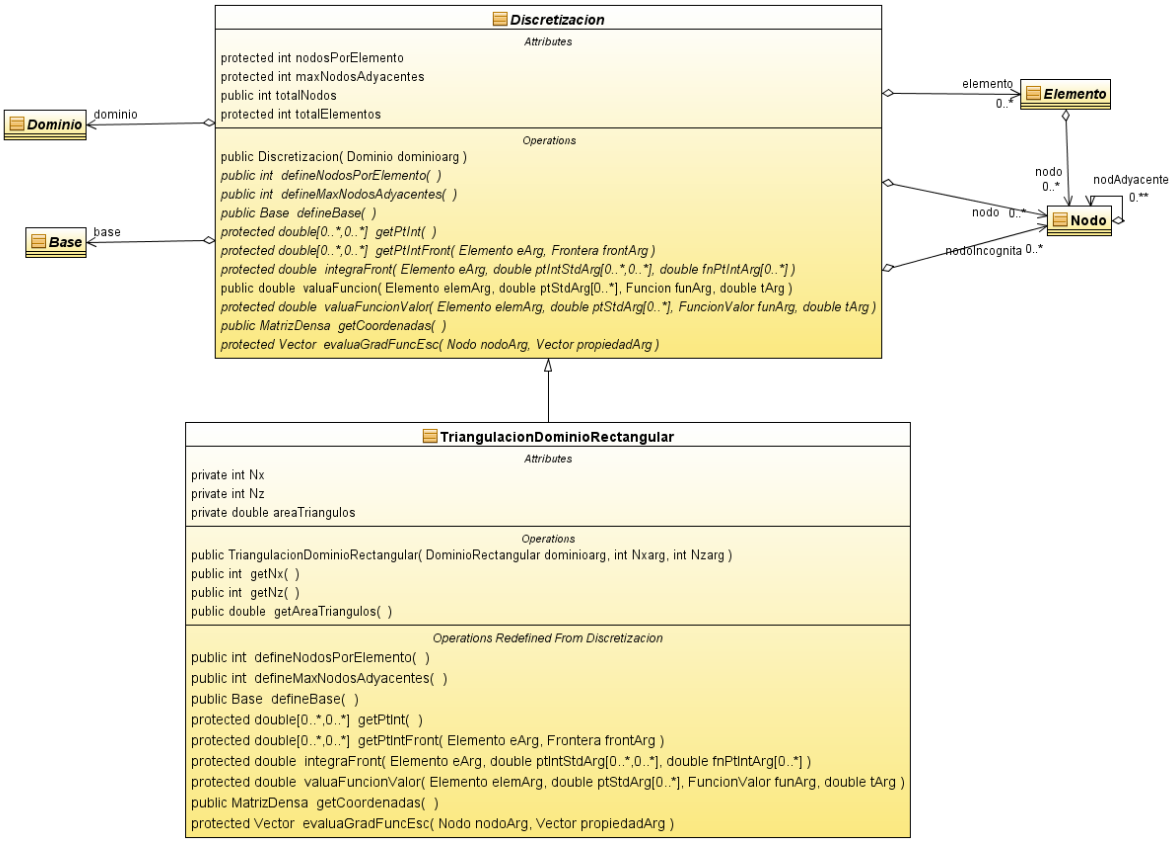


Figura 13. Definición de una Discretización

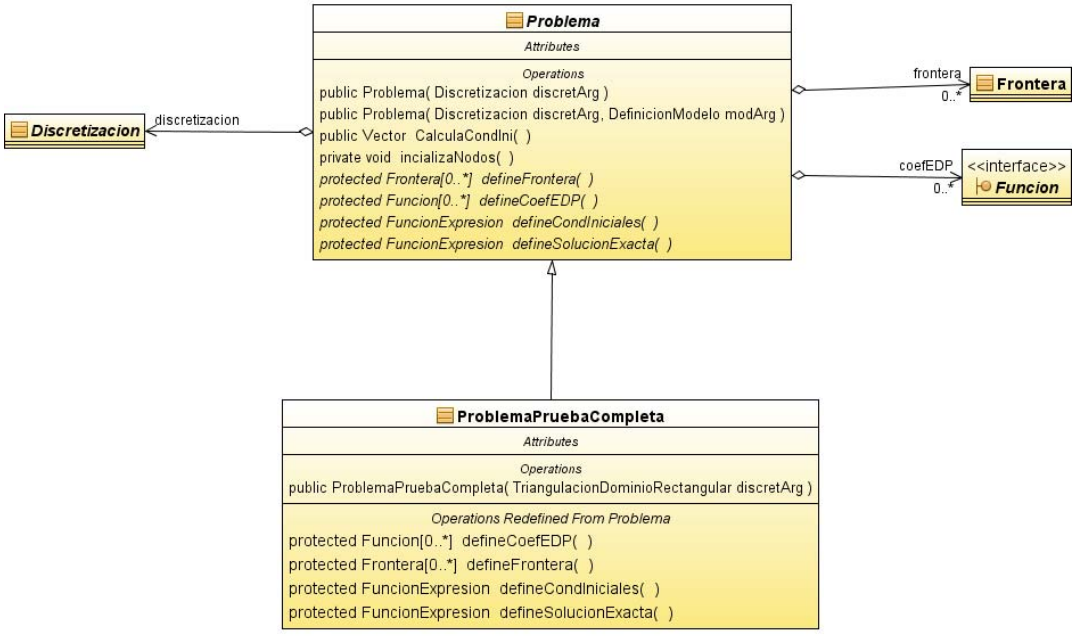


Figura 14. Definición de un Problema

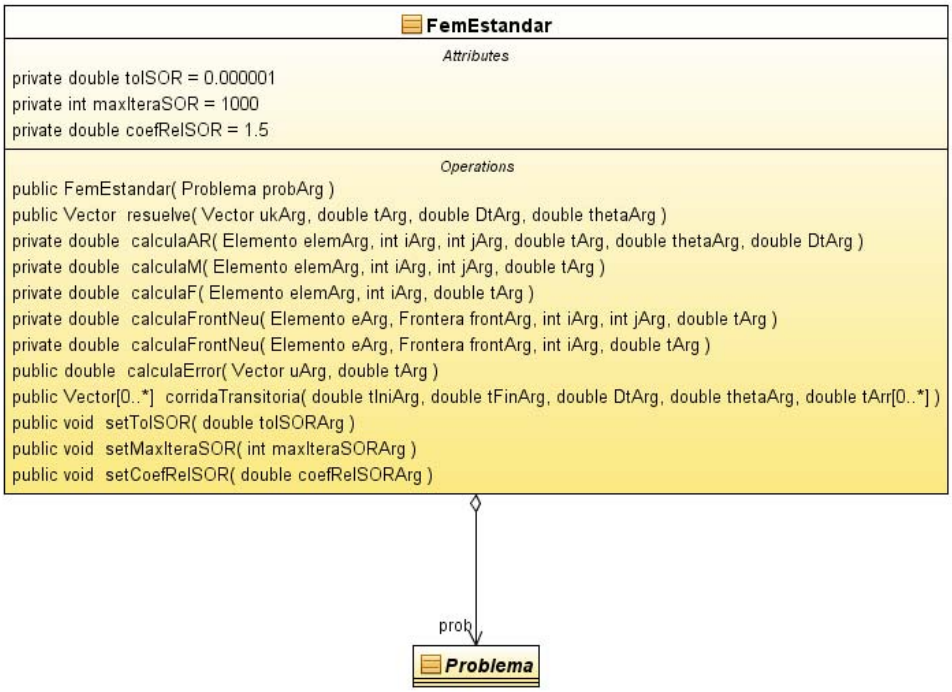


Figura 15. Definición de FEM Estándar

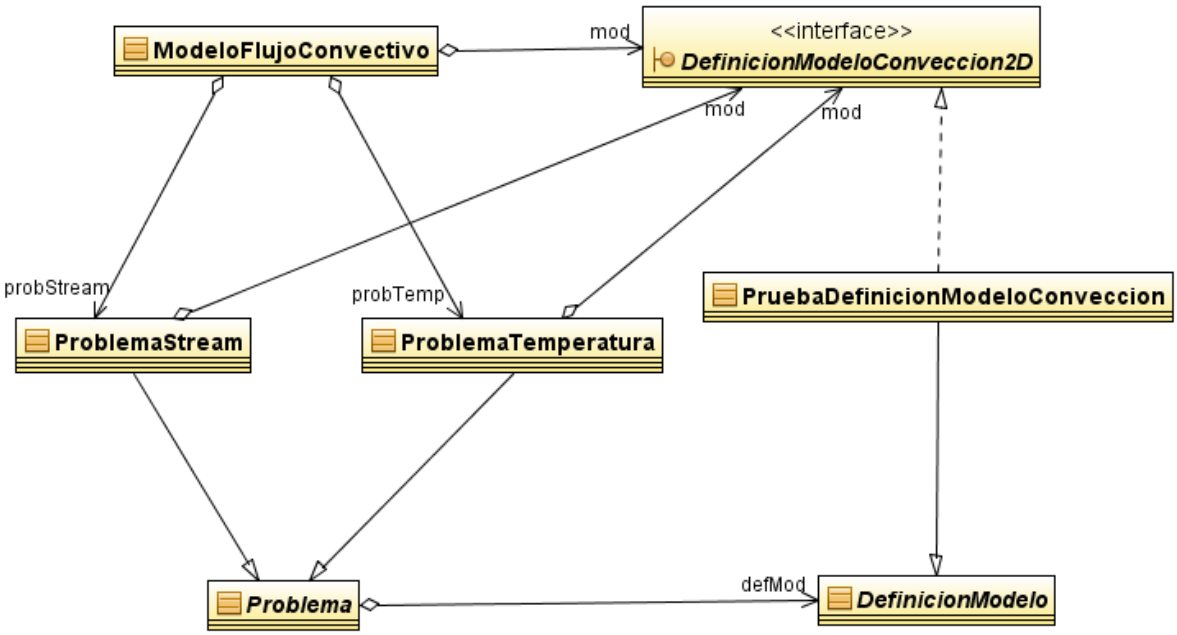


Figura 16. Clases involucradas en la definición de un Modelo de Flujo Convectivo

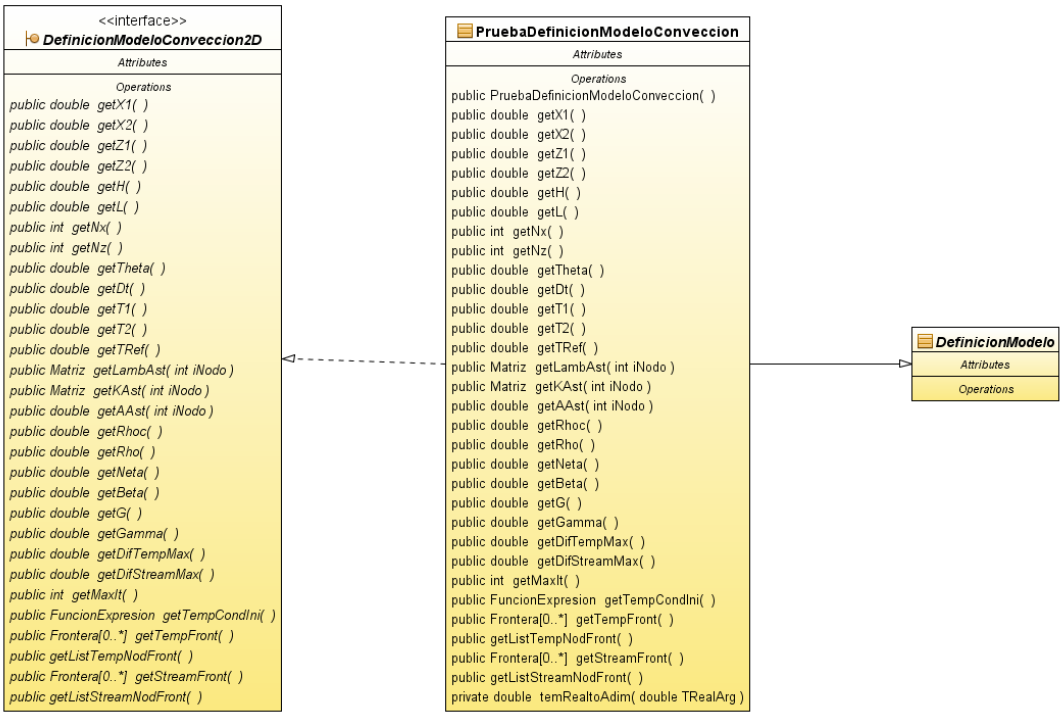
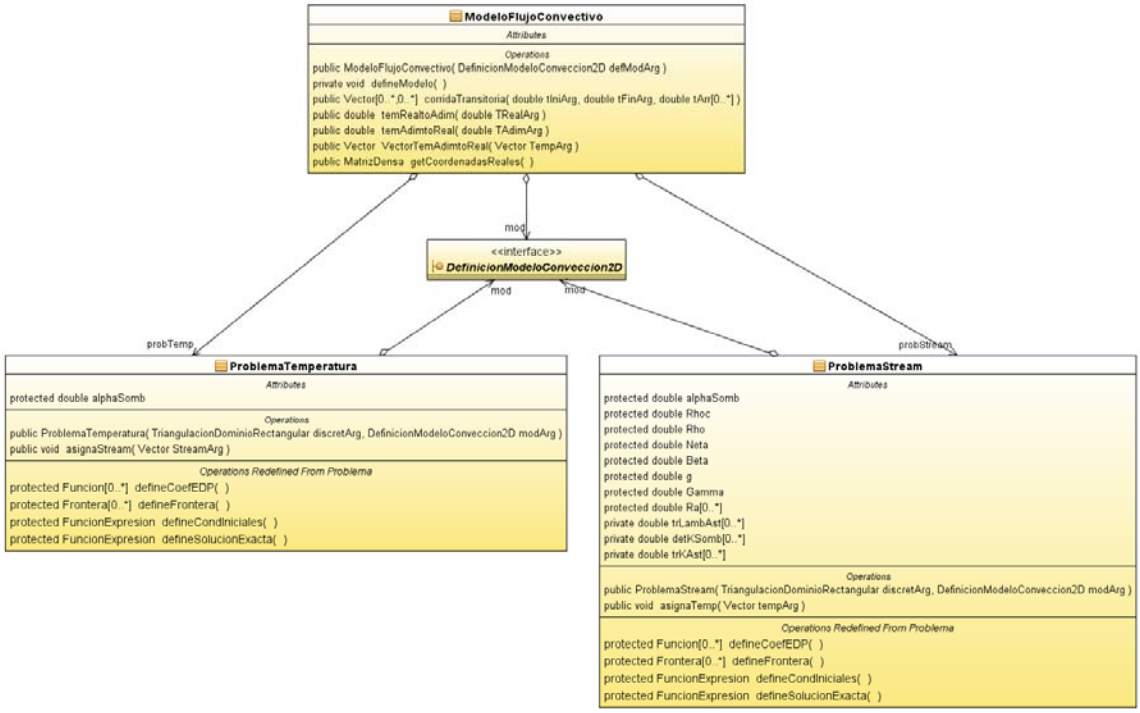


Figura 17. Interfaz y Clase Abstracta que permiten la definición de un Modelo de Convección



5 Pruebas numéricas

Este capítulo tiene como objetivo comparar las salidas numéricas del programa en JAVA, desarrollado en esta tesis, calculadas para problemas sintéticos que previamente han sido estudiados por otros autores (Flores, 1992; Combarnous, 1978; Danis, 1988) tanto con soluciones analíticas como con estudios experimentales en laboratorio. Para tal efecto se comienza probando el desempeño del FEM y luego se continúa con pruebas realizadas a modelos de flujo convectivo reportados en la literatura.

5.1 FEM Estándar

Para realizar pruebas al programa en su parte que resuelve EDP mediante el método del elemento finito estándar, se planteó un "problema completo", es decir una EDP de segundo orden con todos sus "coeficientes" distintos de cero (una ecuación advectivo-difusivo-reactiva, tal cual se describe en la sección **¡Error! No se encuentra el origen de la referencia.**) y que además tuviese fronteras del tipo Dirichlet y Neumann no homogéneas.

Uno de los criterios que existen para probar la eficiencia en la solución numérica de una EDP, es a partir de la convergencia del método, tanto en el espacio como en el tiempo (Yates, 2007). Para poder probar la convergencia de la solución numérica, se necesita conocer la solución analítica. Una forma fácil de lograr esto es creando un problema teórico construido a partir de definir una función a la cual se le aplica el operador diferencial, cuyo resultado corresponde a la parte derecha de la ecuación, mientras que la función original es la solución.

Para obtener la EDP que se usa para probar la convergencia, se parte de la función:

$$u(x,y,t) = e^{-xy-t} \tag{1.1}$$

a la cual se le aplica el operador diferencial

$$-u_{xx} - u_{yy} + u_x + u_y + u + u_t \tag{1.2}$$

Donde $u_x = \frac{\partial u}{\partial x}$ $u_{xx} = \frac{\partial u}{\partial x \partial x}$ $u_{yy} = \frac{\partial u}{\partial y \partial y}$ $u_t = \frac{\partial u}{\partial t}$

de tal manera que la EDP queda

$$-u_{xx} - u_{yy} + u_x + u_y + u + u_t = -\left(x^2 + y^2 + x + y\right)e^{-xy-t} \tag{1.3}$$

Si se supone que la EDP está definida en el dominio

$$(x,y)\in[0,1]\times[0,1] \tag{1.4}$$

y que las bordes superiores e inferiores son fronteras Dirichlet, mientras que los bordes derecho e izquierdo son fronteras Neumann, las fronteras quedarían definidas de la siguiente manera:

$$\begin{aligned}
\Gamma_{D1} &= u(x,0,t) = e^{-t} \\
\Gamma_{N2} &= \frac{\partial u(1,y,t)}{\partial \hat{\underline{\mathbf{n}}}} = -ye^{-y-t} \\
\Gamma_{D3} &= u(x,1,t) = e^{-x-t} \\
\Gamma_{N4} &= \frac{\partial u(0,y,t)}{\partial \hat{\underline{\mathbf{n}}}} = ye^{-t}
\end{aligned} \tag{1.5}$$

Además, si $t\in[0,1]$, las condiciones iniciales son

$$u(x,y,0) = e^{-xy} \tag{1.6}$$

Por lo tanto el problema completo queda definido por la EDP (1.3), en el dominio (1.4), con condiciones en la frontera (1.5), condiciones iniciales (1.6) y solución analítica (1.1).

El FEM estándar implementado, requiere que la EDP se exprese en su forma de la divergencia (sección **¡Error! No se encuentra el origen de la referencia.**). Entonces, haciendo uso de la ecuación **¡Error! No se encuentra el origen de la referencia.** y tomando en cuenta el coeficiente α correspondiente al termino que varía en función del tiempo **¡Error! No se encuentra el origen de la referencia.**, se tiene la EDP definida por

$$\begin{aligned}
&\alpha\frac{\partial u}{\partial t} + \mathrm{L}u = f\left(\underline{\mathbf{x}},t\right) \\
&\textit{donde} \\
&\mathrm{L}u = -\nabla\cdot\underline{\underline{\mathbf{a}}}\cdot\nabla u + \nabla\cdot\left(\underline{\mathbf{b}}u\right) + \mathbf{c}u \\
&\underline{\underline{\mathbf{a}}} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
&\underline{\mathbf{b}} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} \\
&\mathbf{c} = 1 \\
&f = -\left(x^2+y^2+x+y\right)e^{-xy-t} \\
&\alpha = 1
\end{aligned} \tag{1.7}$$

Para probar la convergencia en el espacio y en el tiempo, se grafica la variación del logaritmo del error máximo (el cual corresponde a la diferencia máxima entre la solución

obtenida numéricamente con la solución analítica en todos los nodos) con respecto al logaritmo de N (número de particiones en el espacio) o el logaritmo de Dt (tamaño de partición en el tiempo), respectivamente. En la gráfica obtenida se obtiene la pendiente de la recta y se comprueba si ésta es igual a 2, cómo se espera teóricamente al ocupar el FEM una base lineal (Yates, 2007).

N	MaxErr	Log(N)	-LOG(MaxErr)
2	0.005909098	0.301029996	2.228478799
3	0.004102523	0.477121255	2.386948976
4	0.002850843	0.602059991	2.545026673
5	0.002043277	0.698970004	2.68967283
6	0.001509561	0.77815125	2.821149342
7	0.001145065	0.84509804	2.941169785
8	8.89E-04	0.903089987	3.051288019
9	7.34E-04	0.954242509	3.134558912
10	6.14E-04	1	3.212099972
11	5.18E-04	1.041392685	3.285507071
12	4.41E-04	1.079181246	3.355193253
13	3.79E-04	1.113943352	3.421590493
14	3.27E-04	1.146128036	3.484872901
15	2.85E-04	1.176091259	3.545682416
16	2.52E-04	1.204119983	3.59828681
17	2.25E-04	1.230448921	3.647993614
18	2.01E-04	1.255272505	3.69591197
19	1.81E-04	1.278753601	3.742716933
20	1.63E-04	1.301029996	3.788345328
21	1.47E-04	1.322219295	3.832706616
22	1.34E-04	1.342422681	3.872180867
23	1.23E-04	1.361727836	3.910363107
24	1.13E-04	1.380211242	3.947743502
25	1.04E-04	1.397940009	3.984553588
26	9.54E-05	1.414973348	4.020620771
27	8.85E-05	1.431363764	4.053185341
28	8.22E-05	1.447158031	4.084976501
29	7.65E-05	1.462397998	4.116540375
30	7.12E-05	1.477121255	4.147308226
31	6.66E-05	1.491361694	4.176210591
32	6.25E-05	1.505149978	4.204149929
33	5.88E-05	1.51851394	4.230975871
34	5.52E-05	1.531478917	4.258260577
35	5.21E-05	1.544068044	4.283141042
36	4.92E-05	1.556302501	4.307727676
37	4.67E-05	1.568201724	4.330407907
38	4.43E-05	1.579783597	4.353110403
39	4.22E-05	1.591064607	4.374803492
40	4.02E-05	1.602059991	4.396176752
41	3.83E-05	1.612783857	4.416299324

Tabla 1. Prueba de convergencia en el espacio

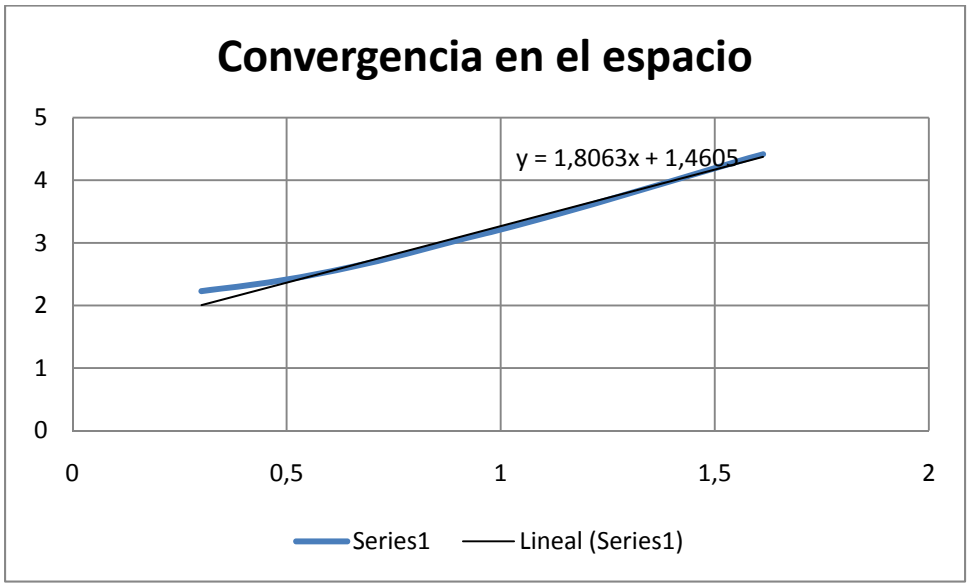


Figura 1. Gráfico de convergencia en el espacio usando todos los valores de la tabulación

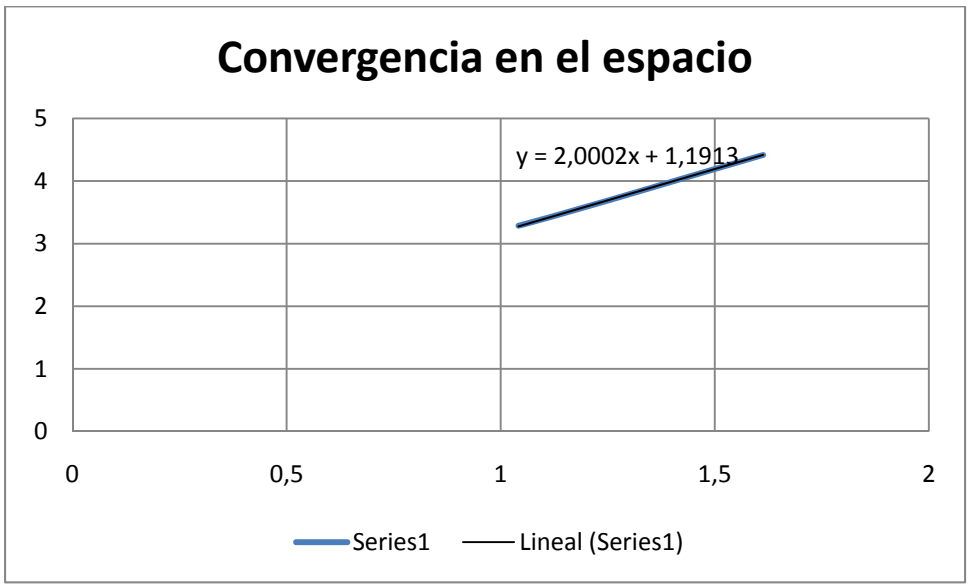


Figura 2. Gráfico de convergencia en el espacio eliminando algunos de los primeros valores de la tabulación

En el caso de la convergencia en el espacio, se dejó un valor constante de $Dt = 0.0125$ y se procedió a variar el número de particiones N, calculando en cada uno de estos pasos el máximo error. Las soluciones se calcularon usando el primer paso en el tiempo del método

de Crank-Nicholson, es decir la solución en $u(x, y, 0.0125)$. Se fijó un valor del coeficiente de relajación $\theta = 0.5$. Los resultados numéricos obtenidos se presentan en la Tabla 1. En la Figura 1 se observa la gráfica que se obtuvo tomando todos los valores de la tabulación. Se observa como el valor de la pendiente es de 1.8, el cual se aproxima mucho al valor esperado de 2. De hecho, si se eliminan los primeros valores de la tabla, como se muestra en la Figura 2, se observa como el valor de la pendiente se hace 2.

N	MaxErr	-LOG(Dt)	-LOG(MaxErr)
0.5	0.046936884	0.301029996	1.328485743
0.25	0.012891608	0.602059991	1.889692916
0.166666667	0.005843729	0.77815125	2.233309941
0.125	0.003297104	0.903089987	2.481867372
0.1	0.002104679	1	2.676814075
0.083333333	0.001454664	1.079181246	2.837237339
0.071428571	1.06E-03	1.146128036	2.973531219
0.0625	8.09E-04	1.204119983	3.092092975
0.055555556	6.36E-04	1.255272505	3.196803741
0.05	5.12E-04	1.301029996	3.290696239
0.045454545	4.21E-04	1.342422681	3.375779243
0.041666667	3.52E-04	1.380211242	3.453147126
0.038461538	2.99E-04	1.414973348	3.52449639
0.035714286	2.57E-04	1.447158031	3.590814181
0.033333333	2.23E-04	1.477121255	3.652317743
0.03125	1.95E-04	1.505149978	3.709379678
0.029411765	1.73E-04	1.531478917	3.763056834
0.027777778	1.54E-04	1.556302501	3.813792824
0.026315789	1.37E-04	1.579783597	3.862003532
0.025	1.24E-04	1.602059991	3.906808164

Tabla 2. Prueba de convergencia en el tiempo

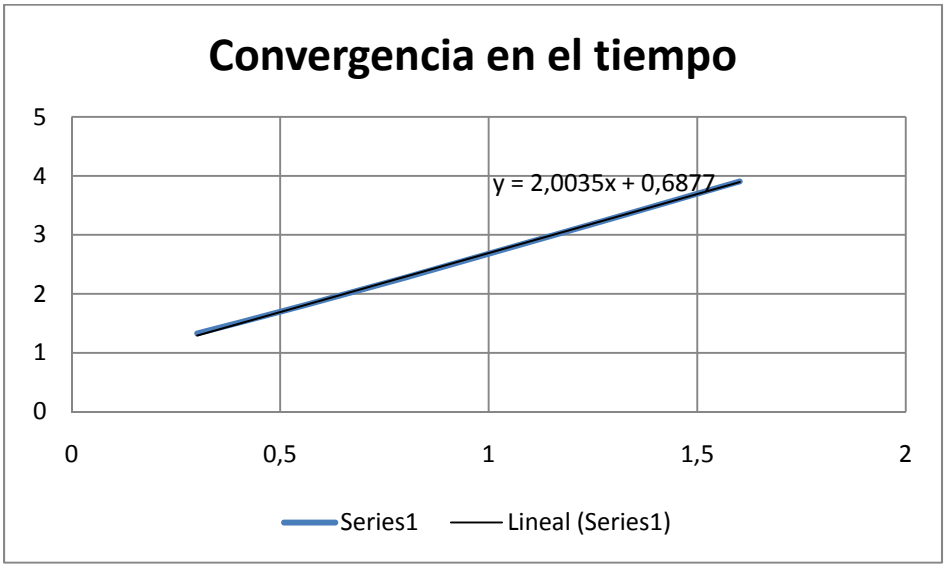


Figura 3. Gráfico de convergencia en el tiempo usando todos los valores de la tabulación

Para probar la convergencia en el tiempo, se fijó el valor de $\mathrm{N}=30$ y $\theta=0.5$. En este caso, con el tamaño de partición en el espacio fijo, lo que se varía es el valor de Dt haciéndolo cada vez más pequeño, calculando en cada corrida completa (en $t\in[0,1]$) el valor del error máximo para cada valor de Dt. En la Tabla 2 se presentan los valores obtenidos de esta prueba, mientras que en la Figura 3 se muestra el gráfico, en el cual se observa que la pendiente obtenida es de 2, que corresponde a lo que se esperaba teóricamente.

5.2 Modelo de flujo convectivo

Para probar el programa en cuanto a la solución de modelos de flujo convectivo, se realizaron algunas corridas correspondientes a resultados numéricos y experimentales publicados previamente (Flores Márquez, 1992).

Las ecuaciones de flujo de calor y de flujo de masa (ecuación **¡Error! No se encuentra el origen de la referencia.**), se presentan en la forma de la divergencia (sección **¡Error! No se encuentra el origen de la referencia.**):

$$\alpha\frac{\partial u}{\partial t}+\mathrm{L}u=f\left(\underline{\mathbf{x}},t\right)$$

donde

$$\mathrm{L}u=-\nabla\cdot\underline{\underline{\mathbf{a}}}\cdot\nabla u+\nabla\cdot\left(\underline{\mathbf{b}}u\right)+\mathbf{c}u$$

quedando los “coeficientes” de las EDP correspondientes a la ecuación de calor y flujo de masa de la siguiente manera:

- Coeficientes de la ecuación de Flujo de Calor

$$\underline{\underline{a}} = \begin{bmatrix} \tilde{\lambda}_{11} & \tilde{\lambda}_{12} \\ \tilde{\lambda}_{21} & \tilde{\lambda}_{22} \end{bmatrix}, \quad \underline{\underline{\tilde{\mathbf{\Lambda}}}} = \left(\tilde{\lambda}_{ij}\right), \quad \tilde{\lambda}_{12} = \tilde{\lambda}_{21}$$
$$\underline{b} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}, \quad \underline{v} = \left(v_i\right)$$
$$c = 0 \tag{1.8}$$
$$f = \frac{2A^* LH}{\text{tr } \underline{\underline{\mathbf{\Lambda}}}^* \Delta T^*}$$
$$\alpha = 1$$

- Coeficientes de la ecuación de Flujo de Masa

$$\underline{\underline{a}} = \begin{bmatrix} \tilde{k}_{11} & \tilde{k}_{12} \\ \tilde{k}_{21} & \tilde{k}_{22} \end{bmatrix}, \quad \underline{\underline{\tilde{\mathbf{K}}}} = \left(\tilde{k}_{11}\right), \quad \tilde{k}_{12} = \tilde{k}_{21}$$
$$\underline{b} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \quad \underline{\mathbf{u}} = \left(u_i\right)$$
$$c = 0 \tag{1.9}$$
$$f = S$$
$$\alpha = 0$$

donde se usa la nomenclatura presentada en el **¡Error! No se encuentra el origen de la referencia.**), y los parámetros que incluyen los coeficientes de la EDP son los presentados en la sección **¡Error! No se encuentra el origen de la referencia.** y que se reescriben a continuación;

$$\underline{v} = \underline{v}_{Darcy} - \nabla\left(\ln \text{ tr } \underline{\underline{\mathbf{\Lambda}}}^*\right) \cdot \underline{\underline{\tilde{\mathbf{\Lambda}}}}$$
$$\underline{v}_{Darcy} = \left(\mathbf{D}\psi\right)^t = \left(\frac{\partial \psi}{\partial z}, -\frac{\partial \psi}{\partial x}\right)^t$$
$$\underline{\mathbf{u}} = -\nabla \ln g \underline{\underline{\tilde{\mathbf{K}}}} \qquad S = \frac{Ra^* \det \underline{\underline{\tilde{\mathbf{K}}}}}{\left(1-\gamma T\right)} \nabla T \cdot \underline{e}_1 \tag{1.10}$$
$$g = \frac{1}{\det \underline{\underline{\tilde{\mathbf{K}}}}} \frac{\text{tr } \underline{\underline{\mathbf{\Lambda}}}^*}{\text{tr } \underline{\underline{\mathbf{K}}}^*} \frac{\eta_r}{\left(\rho c_p\right)_f} \left[1-\gamma T\right]$$
$$Ra^* = \frac{\rho_r \hat{\mathbf{g}} \left(\rho c_p\right)_f \beta_{th} \Delta T^* H \text{ tr } \underline{\underline{\mathbf{K}}}^*}{\eta_r \text{ tr } \underline{\underline{\mathbf{\Lambda}}}^*}$$

La modelización de la convección en un medio poroso permeable, permite el mejor entendimiento de la hidrodinámica a través de un sistema hidrotermal. Sin embargo, estos modelos no dependen del cálculo de todos los fenómenos hidrodinámicos existentes in situ.

Los principales parámetros **¡Error! No se encuentra el origen de la referencia.**) que entran en juego en el modelo son, por una parte, los parámetros del medio (principalmente la permeabilidad y conductividad térmica) y por otra las condiciones en las fronteras (impuestas al sistema) en particular la diferencia de temperaturas entre los limites superior e inferior del dominio.

Por otro lado se considera que el medio está saturado por un único fluido en una sola fase. Para los modelos numéricos es fundamental conocer, por lo menos en una buena aproximación, las propiedades físicas del fluido **¡Error! No se encuentra el origen de la referencia.**):

- Capacidad calorífica ρc_p $\left[\frac{J}{m^3\ \mathrm{K}}\right]$
- Viscosidad dinámica η $\left[\frac{kg}{m\,s}\right]$
- Densidad media ρ $\left[\frac{kg}{m^3}\right]$
- Calor específico c_p $\left[\frac{J}{kg\ \mathrm{K}}\right]$
- Coeficiente de expansión volumétrica β_{th} $\left[\mathrm{K}^{-1}\right]$

Para los ensayos numéricos que se presentan a continuación, se consideró que el fluido saturante era agua pura (Tabla 3).

Propiedad media del agua pura	Valor
Capacidad calorífica	$4180000\ \left[\frac{J}{m^3\ \mathrm{K}}\right]$
Viscosidad dinámica	$0.00018\ \left[\frac{kg}{m\,s}\right]$
Densidad media	$968\ \left[\frac{kg}{m^3}\right]$
Coeficiente de expansión volumétrica	$0.0008275\ \left[\mathrm{K}^{-1}\right]$

Tabla 3. Propiedades del agua pura usadas en los ensayos numéricos

5.2.1 Dominio de una sola capa con permeabilidad homogénea

Para el primer ensayo numérico se consideró un dominio de 40 x 40 [m], compuesto por una sola capa con permeabilidad y conductividad térmica homogénea. Las fronteras superior e inferior son isotérmicamente fijadas, las fronteras verticales son adiabáticas y no se presenta intercambio de fluidos en las fronteras. Las características del medio poroso se presentan en la Tabla 4.

Los resultados obtenidos en esta prueba son muy similares al comportamiento reportado anteriormente en un medio poroso homogéneo por Combarnous y Bories (1975). Estos resultados se presentan en la Figura 4, donde se puede observar cómo la variación en la permeabilidad, mientras el resto de los parámetros se conservan constantes, se refleja en un estrechamiento de las líneas de corriente (Stream) y en la deformación de la celda de convección. El estrechamiento de las líneas de corriente se puede interpretar como un aumento en la velocidad del fluido, lo que provoca a su vez un aumento en la diferencia de temperaturas (con respecto a una distribución homogénea de temperaturas), sobre todo hacia los extremos de las celdas de convección.

Característica del medio	Valor
Permeabilidad	a) $9.87^{-13}\left[m^2\right]$ b) $1.97^{-12}\left[m^2\right]$
Conductividad térmica	$2.7\ \left[\frac{W}{m\,\mathrm{K}}\right]$
Razón geométrica H/L	1.0
Temperatura en la base	$130\ \left[^{\circ}C\right]$
Temperatura en la superficie	$100\ \left[^{\circ}C\right]$

Tabla 4. Características del medio poroso usadas en el primer ensayo numérico sobre un dominio cuadrado compuesto por una sola capa con permeabilidades y conductividades homogéneas, donde se varía la permeabilidad

En Flores Márquez (1992) se puede observar que en anteriores trabajos se ha reportado que los valores de permeabilidad pueden variar de $10^{-21}\left[m^2\right]$ (en el caso de rocas ígneas

fracturadas) a $10^{-8}\left[m^2\right]$ (para corteza oceánica muy fracturada). Esto implica contrastes que pueden ser incluso de varios órdenes de magnitud. Por otro lado, la variación de la conductividad térmica varían de $1.36\left[\frac{W}{m\,\mathrm{K}}\right]$ (para el sílice) a $10.39\left[\frac{W}{m\,\mathrm{K}}\right]$ (para anhidritas con alto contenido de cuarzo), lo cual representa cambios sobre un mismo orden de magnitud.

En la Tabla 6 se presenta como varía el número de Rayleigh (calculándolo a partir de la ecuación **¡Error! No se encuentra el origen de la referencia.**) cuando se varía la permeabilidad en un dominio similar al mostrado con anterioridad. De igual manera en la Figura 6 se grafica esta relación, restringiendo el rango de valores de *Ra* a los valores en los cuales se presenta la convección natural. Tanto en la tabla como en el gráfico se observa como el valor de *Ra* es muy sensible a los cambios en la permeabilidad. Es importante mencionar que, en la práctica, la simulación de la convección natural usando el modelo descrito en este trabajo, se restringe a valores de *Ra* menores de 600, puesto que fuera de este rango el flujo tiende a perder sus características de flujo laminar (Combarnous y Bories, 1975; Flores Márquez, 1992). Este hecho se comprobó graficando la variación de la velocidad máxima, obtenida de la ecuación

$\underline{v}_{Darcy} = \left(\mathbf{D}\,\psi\right)^t = \left(\frac{\partial \psi}{\partial z}, -\frac{\partial \psi}{\partial x}\right)^t$, en función del número de Rayleigh, en un dominio cuadrado (H/L=1) y homogéneo. En la Figura 7 se observa como efectivamente como la velocidad máxima en el estado estable convectivo depende del número de Rayleigh. Cabe resaltar que en números de Rayleigh por debajo de 40, la velocidad es igual a cero.

Por otro lado, en la Tabla 7 y en la Figura 8, se muestra la variación de *Ra* con respecto a la conductividad térmica. En este caso se dejó la permeabilidad fija en un valor de $9.87^{-13}\left[m^2\right]$ en un dominio con las mismas características antes mostradas. Se puede observar como el número de Rayleigh es bastante menos sensible a los cambios en la conductividad térmica, comparado con la sensibilidad a los valores de permeabilidad. Si además se toma en cuenta que el rango de variación de conductividad térmica es mucho más restringido que el de la permeabilidad, queda claro que el factor del medio que más influirá en la variación de *Ra*, será la permeabilidad.

Otro factor que influye en la convección natural, es la geometría del medio. Para mostrar este hecho se toma un dominio similar al que se ha venido trabajando, con los cambios presentados en la Tabla 5.

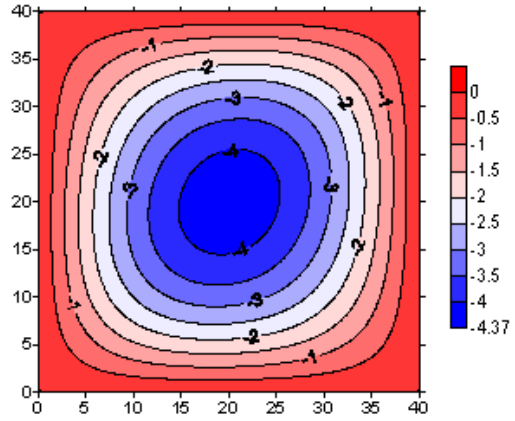
Característica del medio	Valor
Permeabilidad	a) $2.5^{-12}\left[m^2\right]$ b) $1.32^{-12}\left[m^2\right]$
Conductividad térmica	2.7 $\left[\frac{W}{m\,\mathrm{K}}\right]$
Razón geométrica H/L	a) 1.8 b) 0.33

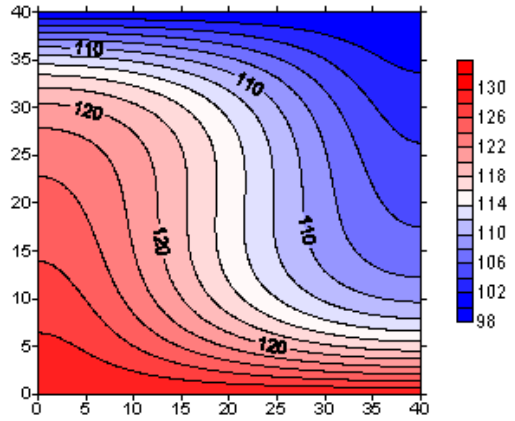
Tabla 5. Características del medio poroso usadas en el primer ensayo numérico sobre un dominio cuadrado compuesto por una sola capa con permeabilidades y conductividades homogéneas, donde se varía la geometría del dominio.

En la Figura 5.a) se muestra como una razón geométrica H/L = 1.8 ocasiona una respuesta similar a la del dominio cuadrado, con la diferencia en la elongación de la celda convectiva y su consecuente efecto en el campo de temperaturas. En el caso de que H/L = 0.33 (Figura 5.b), se observa como las celdas convectivas se multiplican en un número igual a L/H presentando cada una de estas celdas un comportamiento similar al de las celdas en un dominio cuadrado. Es importante observar como el signo de las líneas de Stream, en las celdas de este último ejemplo, cambia entre celdas contiguas. También es interesante observar el efecto de las celdas convectivas en el campo de temperaturas, el cual tiende a deformarlo sobre todo en los bordes verticales de las celdas y entre dos celdas contiguas. En este último caso se hace evidente que el signo de Stream puede interpretarse como una dirección en el sentido del "giro" en las celdas convectivas. En este caso se puede decir que el signo negativo indica un sentido de giro horario, mientras que el signo positivo representa un sentido de giro antihorario.

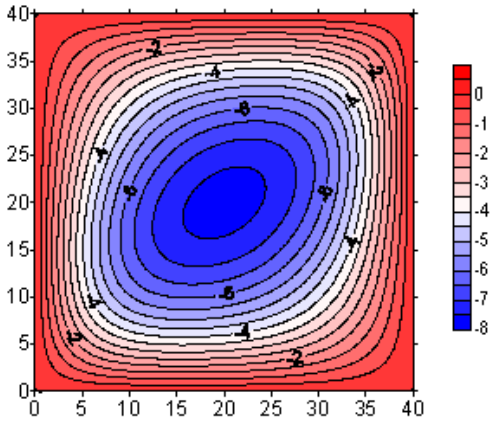
Las figuras 24 y 25 son salidas del programa JAVA que reproducen fielmente las configuraciones obtenidas por Flores (1992), Combarnous (1984), y Danis (1988) para medios homogéneos isotrópicos, esta comparación nos permite concluir que el programa produce una solución adecuada de las ecuaciones diferenciales advectivo-difusivas para estos medios.

a) **Ra = 80**
H/L = 1





b) **Ra = 160**
H/L = 1



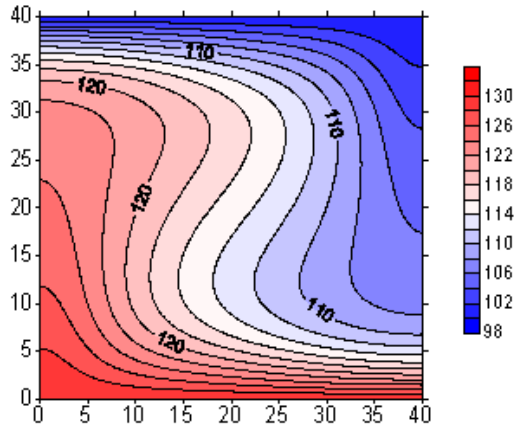


Figura 4. Estado convectivo estable en un medio homogéneo isotrópico. a) Dominio cuadrado (H/L=1) y Ra = 80. b) Dominio cuadrado (H/L=1) y Ra = 160.

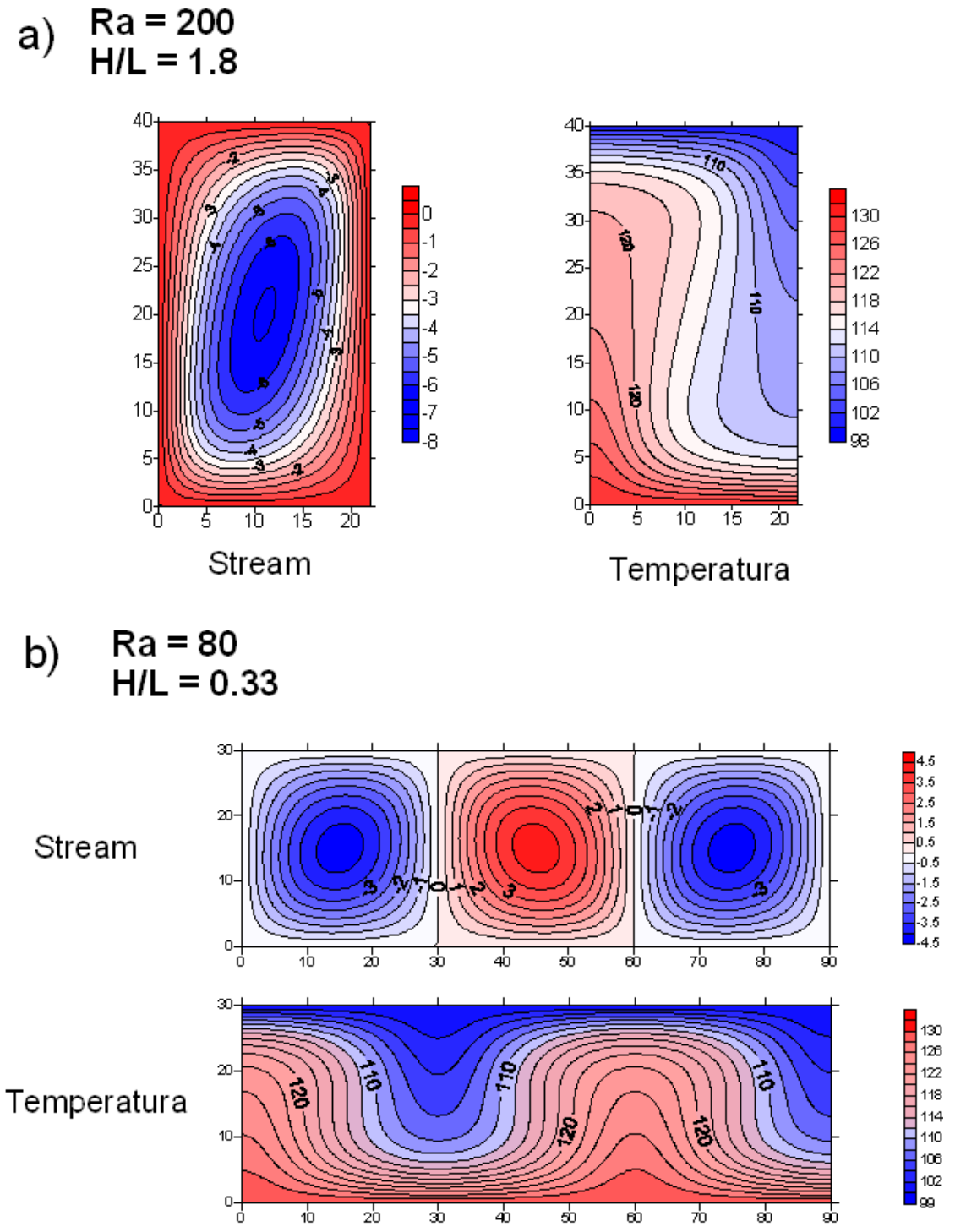


Figura 5. Estado convectivo estable en un medio homogéneo isotrópico. a) Dominio rectangular (H/L=1.8) y Ra = 200. b) Dominio cuadrado (H/L= 0.33) y Ra = 80.

K	Ra
1.0E-20	8.11024E-07
1.0E-19	8.11024E-06
1.0E-18	8.11024E-05
1.0E-17	0.000811024
1.0E-16	0.008110238
1.0E-15	0.081102385
1.0E-14	0.81102385
1.0E-13	8.110238498
1.0E-12	81.10238498
1.0E-11	811.0238498
1.0E-10	8110.238498
1.0E-09	81102.38498
1.0E-08	811023.8498

Tabla 6. Número de Rayleigh en función de la permeabilidad en un medio homogéneo

Lamb	Ra
1	216.020258
2	108.010129
3	72.0067525
4	54.0050644
5	43.2040515
6	36.0033763
7	30.8600368
8	27.0025322
9	24.0022508
10	21.6020258
11	19.6382052

Tabla 7. Número de Rayleigh en función de la conductividad térmica en un medio homogéneo

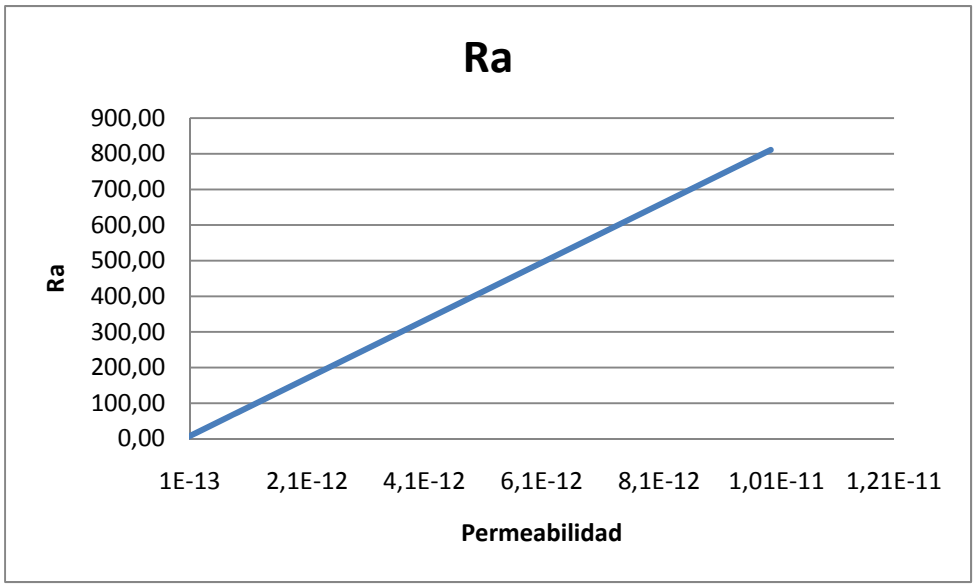


Figura 6. Número de Rayleigh en función de la permeabilidad en un medio homogéneo

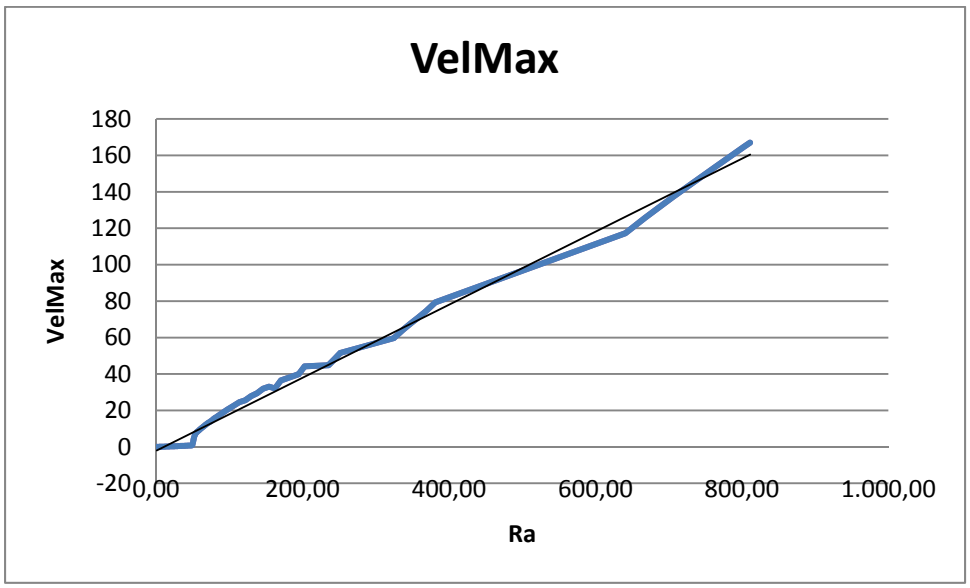


Figura 7. Variación de la Velocidad Máxima de filtración con respecto al número de Rayleigh.

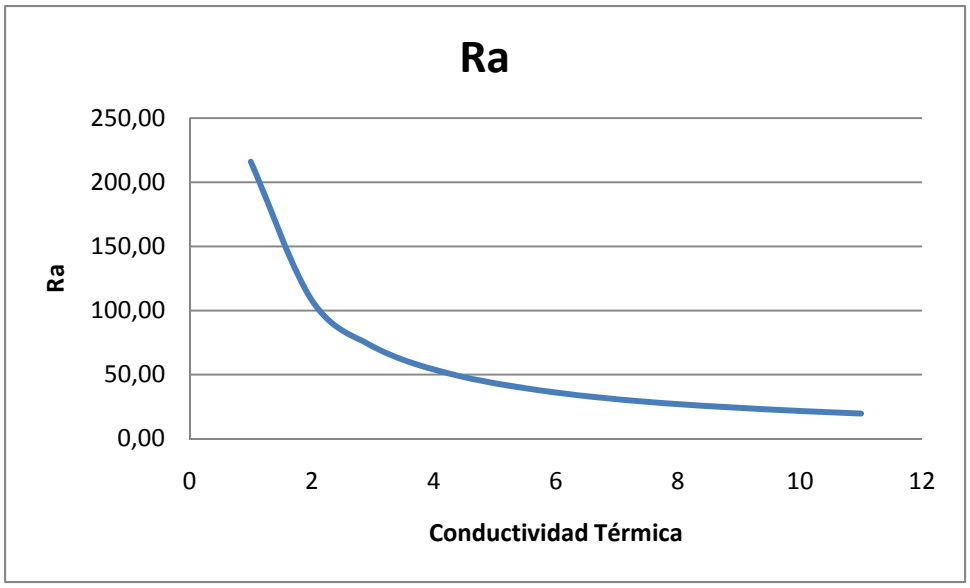


Figura 8. Número de Rayleigh en función de la conductividad térmica en un medio homogéneo

5.2.2 Dominio de varias capas horizontales con características físicas distintas

En esta sección se considera primero un domino cuadrado dividido en dos capas horizontales que presentan un fuerte contraste de permeabilidades, teniendo la capa superior un valor más alto que el de la capa inferior, lo cual ocasiona que el número de Rayleigh de la capa superior sea 100 veces más grande que el de la capa inferior. El valor de conductividad térmica se mantiene constante en ambas capas. Las fronteras superior e inferior son isotérmicamente fijadas, las fronteras verticales son adiabáticas y no se presenta intercambio de fluidos en las fronteras. Las características del medio poroso se presentan en la Tabla 8.

En la Figura 9.a) se observa el resultado de esta prueba. Como se esperaba, en la capa superior es dónde se presenta la convección, mientras que en la capa inferior el régimen de transferencia es conductivo. Se observa también como en la capa superior se forman dos celdas convectivas, lo cual corresponde a lo reportado en trabajos anteriores (Combarnous y Bories, 1975).

Se realiza otro ensayo donde se define un dominio rectangular (H/L = 0.33) compuesto por tres capas horizontales de iguales dimensiones, pero con permeabilidades y conductividades térmicas distintas. Las fronteras superior e inferior son isotérmicamente fijadas, mientras que las fronteras verticales son adiabáticas. El dominio se supone sin intercambio de fluidos con el exterior. Las características del medio poroso se resumen en la Tabla 9.

Característica del medio	Valor
Permeabilidad	Capa superior: $1.1^{-12}\left[m^2\right]$ Capa inferior: $1.1^{-16}\left[m^2\right]$
Conductividad térmica	2.7 $\left[\frac{W}{m\,\mathrm{K}}\right]$
Razón geométrica H/L	1.0
Temperatura en la base	130 $\left[^{\circ}C\right]$
Temperatura en la superficie	100 $\left[^{\circ}C\right]$

Tabla 8. Características del medio poroso usadas en el ensayo numérico sobre un dominio cuadrado compuesto por dos capas con permeabilidades distintas y conductividades térmicas homogéneas

Característica del medio	Valor
Permeabilidad	Capa superior: $1.1^{-15}\left[m^2\right]$ Capa media: $1.1^{-12}\left[m^2\right]$ Capa inferior: $1.1^{-15}\left[m^2\right]$
Conductividad térmica	Capa superior: 5.2 $\left[\frac{W}{m\,\mathrm{K}}\right]$ Capa media: 2.5 $\left[\frac{W}{m\,\mathrm{K}}\right]$ Capa inferior: 5.2 $\left[\frac{W}{m\,\mathrm{K}}\right]$
Razón geométrica H/L	0.33
Temperatura en la base	130 $\left[^{\circ}C\right]$
Temperatura en la superficie	100 $\left[^{\circ}C\right]$

Tabla 9. Características del medio poroso usadas en el ensayo numérico sobre un dominio rectangular compuesto por tres capas con permeabilidades y conductividades térmicas distintas

En los resultados de esta prueba (Figura 9.b) se puede observar cómo en la capa central, que es la que presenta características físicas que generan un número de Rayleigh mayor, se acentúa la presencia de las celdas convectivas. Es de resaltar que el número de celdas se

corresponde a lo que se esperaba de acuerdo a los trabajos anteriores. De igual manera se observa cómo estas tres celdas convectivas dentro de la capa central, tienden a producir en ésta una ligera deformación en la distribución de temperaturas. El comportamiento en las capas superior e inferior, presenta características de transferencia de calor conductivo.

Se realizó otra prueba enfocada en mostrar el efecto de variar la conductividad térmica mientras se mantiene constante el valor de la permeabilidad. Se supuso un dominio cuadrado (H/L=1) dividido en 5 capas horizontales iguales, fronteras superior e inferior isotérmicamente fijadas, fronteras verticales adiabáticas, y cero intercambio de fluidos con el exterior. Las características del medio poroso se resumen en la Tabla 10.

En la Figura 10.c) se muestran los resultados, en los cuales se corrobora que las variaciones en la conductividad térmica, aunque éstas sean significativas, no afectan en gran medida al fenómeno de convección. De esta manera queda verificado que en un modelo de convección natural representado sobre un medio natural, la permeabilidad es el parámetro físico que juega el rol más importante, mientras que la conductividad térmica no afecta a éste en casi nada.

Característica del medio	Valor
Permeabilidad	$3.0^{-13}\left[m^2\right]$
Conductividad térmica	Capa 1 (capa superior): $5.2\ \left[\frac{W}{m\,\mathrm{K}}\right]$ Capa 2: $4.7\ \left[\frac{W}{m\,\mathrm{K}}\right]$ Capa 3: $3.7\ \left[\frac{W}{m\,\mathrm{K}}\right]$ Capa 4: $2.2\ \left[\frac{W}{m\,\mathrm{K}}\right]$ Capa 5 (capa inferior): $1.0\ \left[\frac{W}{m\,\mathrm{K}}\right]$
Razón geométrica H/L	1.0
Temperatura en la base	$130\ \left[^{\circ}C\right]$
Temperatura en la superficie	$100\ \left[^{\circ}C\right]$

Tabla 10. Características del medio poroso usadas en el ensayo numérico sobre un dominio cuadrado compuesto por cinco capas con permeabilidades iguales y conductividades térmicas distintas

5.2.3 Dominio con una capa inclinada porosa permeable intercalada en un medio poco permeable

En esta sección se realizan un par de ensayos sobre un dominio rectangular (H/L = 0.33) donde se presenta una capa inclinada permeable, intercalada dentro de un medio impermeable.

En el primer ejemplo se considera que el dominio tiene fronteras superior e inferior isotérmicamente fijadas, mientras que las fronteras verticales son adiabáticas. El dominio se supone sin intercambio de fluidos con el exterior. Las características del medio poroso se presentan en la Tabla 11. La capa porosa inclinada se supone homogénea y además la conductividad térmica de todo el dominio es homogénea.

En la Figura 11.a) se observa cómo se presentan celdas convectivas a través de la capa inclinada, las cuales a su vez generan perturbaciones oscilantes ligeras en la temperatura. Además, de manera general, se observa como el campo de temperaturas se ve afectado, presentándose un ligero incremento en las temperaturas en un extremo del dominio, mientras que en el extremo opuesto las temperaturas decrecen.

En una segunda prueba se considera que la capa inclinada tiene una permeabilidad anisotrópica, con la componente horizontal diez veces más grande que la componente vertical, mientras que el medio que la rodea es impermeable. La conductividad térmica de la capa inclinada es menor que la conductividad térmica del medio que la envuelve. La frontera superior está isotérmicamente fijada, mientras que la inferior queda definida por un flujo de calor que entra al dominio. En las fronteras no se presenta intercambio de fluidos. En la Tabla 12 se presentan las características del medio.

En este último ejemplo, en la Figura 11.b) se observa como a través de la capa inclinada se forma una gran celda convectiva, la cual provoca que la temperatura se incremente en la misma dirección que la capa inclinada.

Es importante mencionar que en algunos ejemplos, sobre todo en aquellos con dominios heterogéneos, se observó que la solución dependía en gran medida de las condiciones iniciales de temperatura (por la naturaleza no lineal de algunos términos de las EDP acopladas). Por tal motivo se procuró que las soluciones iniciales dadas (compuestas por señales senoidales) se correspondiesen en amplitud y periodo a los resultados que se esperaban.

La comparación visual y numérica de los archivos correspondientes a las pruebas numéricas realizadas para medios heterogéneos por estratos, con aquellas reportadas por los autores antes citados (Flores, 1992; Combarnous, 1978; Danis, 1988) permite comprobar la efectividad del programa computacional que resuelve las ecuaciones advectivo-difusivas.

Característica del medio	Valor
Permeabilidad	Capa inclinada: $1.3^{-12}\left[m^2\right]$ Resto del medio: $1.2^{-16}\left[m^2\right]$
Conductividad térmica	$2.7\ \left[\frac{W}{m\,\mathrm{K}}\right]$
Razón geométrica H/L	0.33
Temperatura en la base	$130\ \left[^{\circ}C\right]$
Temperatura en la superficie	$100\ \left[^{\circ}C\right]$

Tabla 11. Características del medio poroso usadas en el ensayo numérico sobre un dominio rectangular compuesto por una capa inclinada homogénea isotrópica permeable, rodeada por un medio homogéneo impermeable. La conductividad térmica es homogénea en todo el dominio.

Característica del medio	Valor
Permeabilidad	Capa inclinada: $k_x = 1.0^{-12}\left[m^2\right]$ $k_x = 1.0^{-13}\left[m^2\right]$ Resto del medio: $1.2^{-16}\left[m^2\right]$
Conductividad térmica	Capa inclinada: $2.7\ \left[\frac{W}{m\,\mathrm{K}}\right]$ Resto del medio: $5.4\ \left[\frac{W}{m\,\mathrm{K}}\right]$
Razón geométrica H/L	0.33
Flujo de calor en la base	$300\left[\frac{\mathrm{m}W}{m^2}\right]$
Temperatura en la superficie	$10\ \left[^{\circ}C\right]$

Tabla 12. Características del medio poroso usadas en el ensayo numérico sobre un dominio rectangular compuesto por una capa inclinada homogénea anisotrópica permeable, rodeada por un medio homogéneo impermeable. La

conductividad térmica es menor en la capa inclinada que en el resto del dominio.

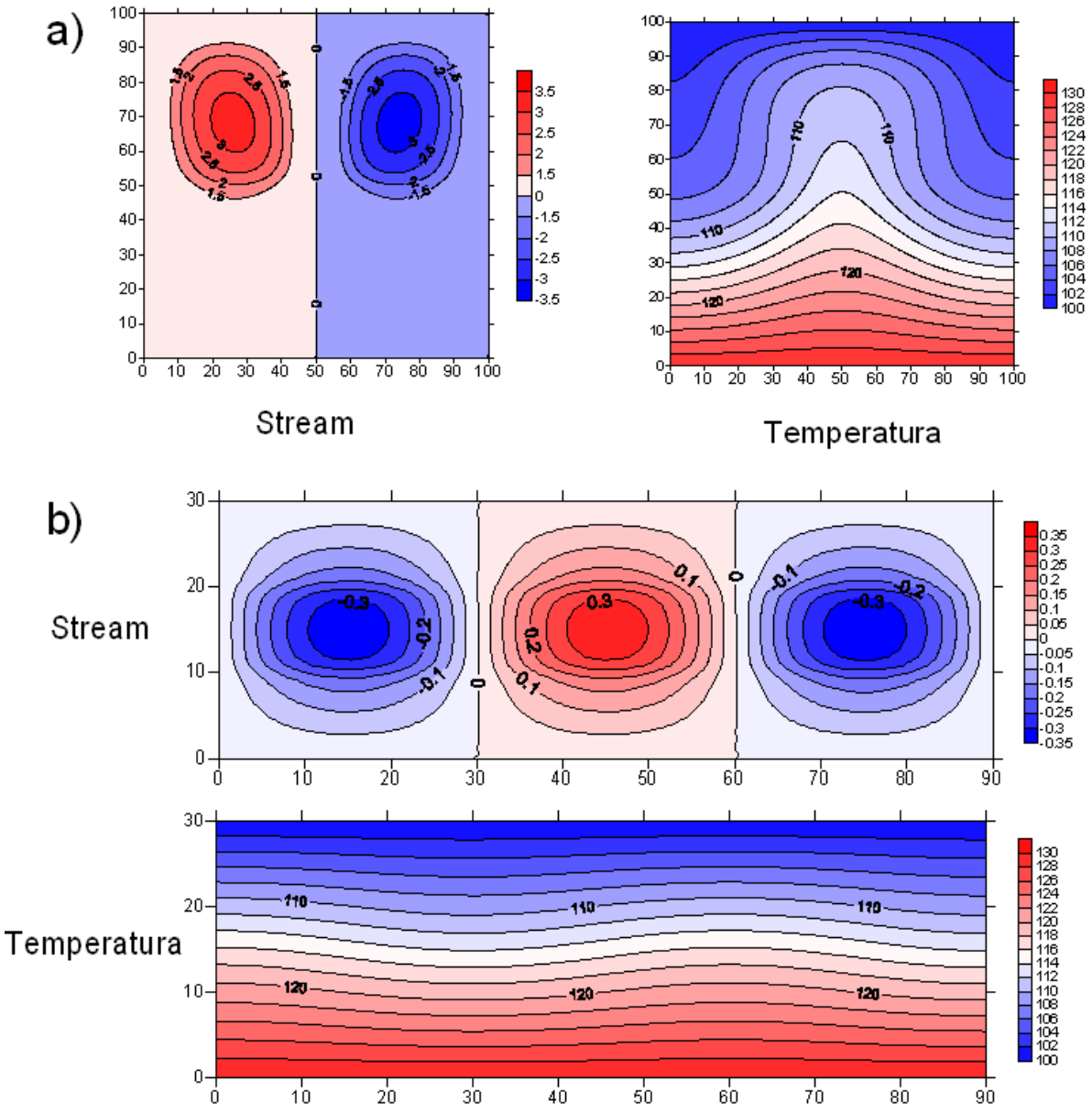


Figura 9. Estado convectivo estable en un medio poroso heterogéneo estratificado formado por capas homogéneas isotrópicas. a) Dominio cuadrado (H/L=1) compuesto por dos capas horizontales con distintos valores de permeabilidad y una misma conductividad térmica. b) Dominio rectangular (H/L=0.33) compuesto por 3 capas del mismo tamaño, siendo las capas superior e inferior

iguales (baja permeabilidad y alta conductividad térmica) mientras que la capa de en medio contrasta con las otras dos (permeabilidad más alta y baja conductividad térmica).

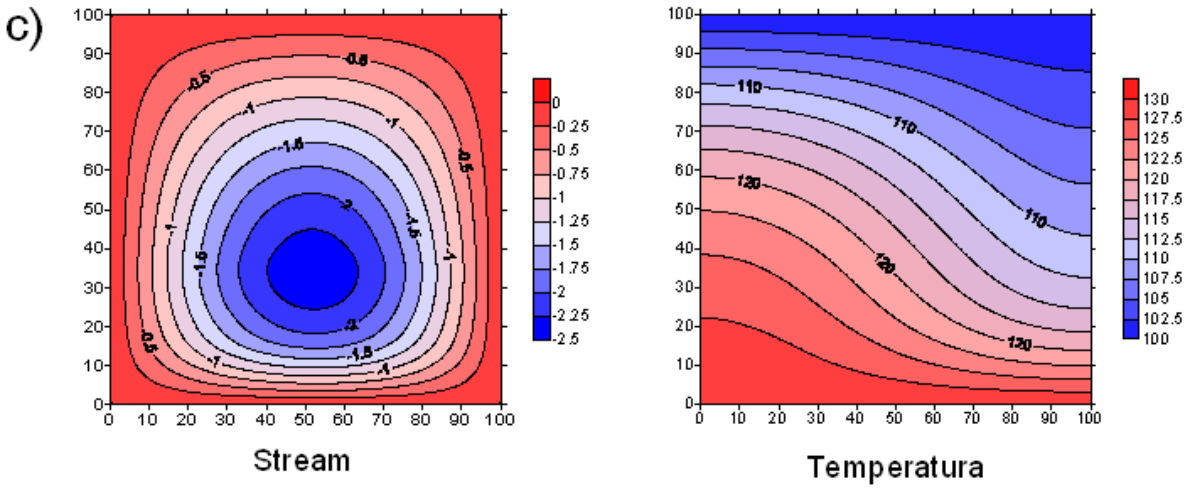


Figura 10. Estado convectivo estable en un medio poroso heterogéneo estratificado formado por capas homogéneas isotrópicas. c) Dominio cuadrado (H/L=1) compuesto por cinco capas horizontales con distintos valores de conductividad térmica y una misma permeabilidad.

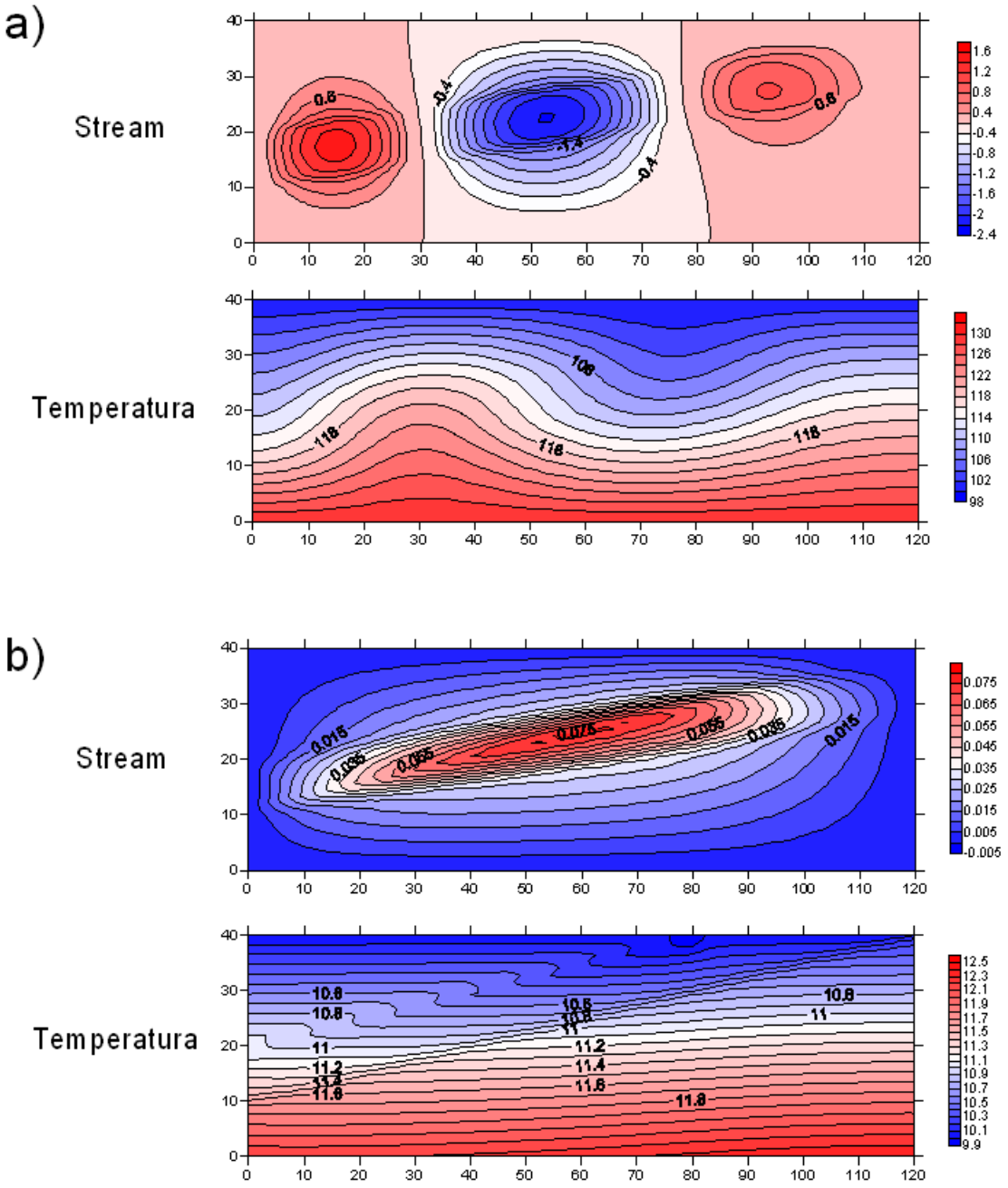


Figura 11. Estado convectivo estable en un medio poroso heterogéneo formado por una capa inclinada permeable rodeada de un medio impermeable. a) Dominio rectangular (H/L=0.33) donde la capa inclinada es homogénea isotrópica permeable, mientras que el medio que la rodea es homogéneo impermeable. La conductividad térmica es homogénea en todo el dominio. b) Dominio rectangular (H/L=0.33) donde la capa inclinada es homogénea anisotrópica permeable, mientras que el medio que la rodea es homogéneo impermeable. La conductividad térmica de la capa inclinada es menor que en el resto del dominio. La frontera inferior presenta flujo de calor.

6 Conclusiones

El objetivo de la presente tesis fue realizar la modelación de flujo convectivo en un medio poroso anisotrópico y heterogéneo mediante la programación de las ecuaciones de flujo y transporte de energía con el Método del Elemento Finito (FEM), usando el paradigma de la Orientación a Objetos con una implementación en el lenguaje JAVA. Este objetivo fue completamente alcanzado en el ámbito de la modelación de un fenómeno terrestre, desde su planteamiento teórico hasta la implementación concreta en un sistema computacional, haciendo especial énfasis en este último.

Las conclusiones a las que se llegó al realizar este trabajo son:

1. En el planteamiento teórico del modelo se buscó que éste fuese capaz de representar la complejidad inherente del subsuelo, sobre todo en cuanto a dominios heterogéneos y anisotrópicos se refiere. Las ecuaciones consideradas son la ley generalizada de Darcy y la de Fourier o del Calor; estas además de tomar en cuenta las características anteriores, permiten modelar medios con generación interna de calor, con las siguientes hipótesis: fluido incompresible; bajas velocidades de filtración y las variaciones en la velocidad son despreciables; el medio está completamente saturado por un solo fluido sin que se presenten cambios de fase.
2. La efectividad de FEM en la solución de Ecuaciones Diferenciales Parciales (EDP), quedo manifiesta en este trabajo; en primera instancia, el programa diseñado resuelve de manera general EDP Advectivo-Difusivas-Reactivas con condiciones de fronteras generales, en problemas variantes en el tiempo, el cual posteriormente se utilizó para resolver las ecuaciones de calor y de flujo involucradas en el modelo de convección. También se observó la conveniencia del FEM en el hecho de que no se presentaron problemas de convergencia derivados del contraste o salto de los parámetros físicos de capas, puesto que el método tiende de manera natural a “atenuar” estas diferencias debido a que se basa en el cálculo de integrales sobre los elementos y no sólo en diferencias entre los valores de nodos. Además mostró ser bastante robusto en las pruebas de convergencia tanto en el espacio como en el tiempo.
3. El uso del paradigma de Orientación a Objetos resultó adecuado en la implementación del programa. En el presente trabajo se observa cómo un análisis enfocado a objetos resulta en el diseño de un programa sumamente claro, sencillo e incluso intuitivo; cualidades que no deben despreciarse si se tiene en mente que éstas permiten el fácil mantenimiento del programa y su escalabilidad. El uso de la Orientación a Objetos permitió dividir un problema complejo en partes mucho más sencillas y manejables, facilitando en gran medida el proceso de implementación computacional. De igual manera, esta característica facilita la modificación del programa cuando se pretende mejorar su desempeño o bien ampliar su funcionalidad.

4. La elección del lenguaje de programación JAVA resultó adecuada debido a su facilidad de uso, las herramientas de desarrollo con las que cuenta y sobre todo por su portabilidad. Aunque se considera que JAVA es un lenguaje lento, los avances tecnológicos que existen en el hardware, permiten tener gran velocidad en una simple computadora de escritorio, con lo cual se puede considerar "sacrificar" la eficiencia en aras de la flexibilidad y facilidad de uso. Por otro lado, se justifica el esfuerzo invertido en el desarrollo de un programa nuevo, puesto que comercialmente no existen programas que resuelvan este tipo de modelos de convección natural de baja entalpía en medios poroso heterogéneos e isotrópicos. Además, las clases usadas, sobre todo las referentes a la implementación del FEM, pueden ser fácilmente reutilizadas en otros problemas bidimensionales, o bien servir de base para desarrollar variaciones del FEM o implementaciones en 3 dimensiones. El hecho de tener una representación gráfica del modelo computacional facilita en gran medida la reutilización de los componentes de software creados en este trabajo.
5. El proceso de solución numérica fue probado en numerosos ejemplos teóricos y comparado con resultados previamente publicados (Flores, 1992; Combarnous, 1978, Danis, 1988), dichos ejemplos contemplan procesos de laboratorio y de soluciones analíticas. Los resultados obtenidos son consistentes con los mostrados en trabajos previos, tanto para casos homogéneos, como en casos de medios heterogéneos anisotrópicos y con geometrías más complejas. Por lo que este programa puede ser utilizado para la modelación de transferencia de calor y de masa en medios porosos en casos que suceden en la naturaleza. En los ensayos del modelo de convección se observó claramente como la ocurrencia de celdas convectivas depende principalmente de la permeabilidad, mientras que la conductividad térmica casi no afecta.

Es importante mencionar que debido a los términos no lineales involucrados en el modelo de convección, se tuvo que tener cuidado en la elección del coeficiente de relajación usado en el SOR. Si bien no se tomó en cuenta ninguna consideración especial en el tratamiento de los términos no lineales (fundamentando esta decisión de estudios anteriores como el de Flores, 1992), se puede decir que el estudio más detallado de los efectos de los términos no lineales involucrados en el fenómeno de convección natural representa un área de oportunidad en estudios posteriores.

Por último, el hecho de que la implementación computacional este basada en el desarrollo de clases en JAVA que resuelven de manera general ecuaciones diferenciales parciales de segundo orden con términos advectivos-difusivos-reactivos, y de que además el diseño de estas clases se hizo tomando en cuenta el paradigma de orientación a objetos, nos permite pensar que estas clases pueden ser reutilizadas o servir como base en la solución de otros problemas. También podría tomarse la estructura completa del programa como base de desarrollo para un software con alcances más ambiciosos.

7 Bibliografía

Allen, M. B., Herrera, I., & Pinder, G. P. (1988). *Numerical modeling in science and engineering.* New York: John Wiley & Sons.

Booch, G. (1996). *Análisis y diseño orientado a objetos con aplicaciones* (2 ed.). Madrid: Pearson Educación, S.A.

Booch, G., Jacobson, I., & Rumbaugh, J. (2006). *El lenguaje unificado de modelado* (2 ed.). Madrid: Pearson Educación, S.A.

Booch, G., Jacobson, I., & Rumbaugh, J. (2007). *El lenguaje unificado de modelado. Manual de referencia.* (2 ed.). Madrid: Pearson educación, S.A.

Carrillo Ledesma, A. (2006). *Aplicaciones del Cómputo de Paralelo a la Modelación de Sistemas Terrestres.* Ciudad de México, D.F., México: TESIS - Maestro en Ciencias de la Tierra. Instituto de Geofísica.

Chen, Z. (2005). *Finite element methods and their applications.* New York: Springer-Verlag, Heidelberg.

Chen, Z., Huan, G., & Yuanle, M. (2006). *Computational methos for multiphase flows in porous media.* Philadelphia: SIAM.

Combarnous, M. A. (1978). Natural convection in porous media and geothermal systems. *6th Int. Heat Transfer Conf. , 6,* 45-59.

Combarnous, M. A., & Bories, S. A. (1975). Hydrothermal convection in saturated porous media. *Advances in Hydroscience , 10,* 231-307.

Danis, M. (1988). *Hydrogéologie et transferts thermiques pour un aquifer en Bassin Sédimentaire: Approche numérique. Thèse de I'NLP.*

Degan, G., & Vasseur, E. B. (1995). Convective heat transfer in a vertical anisotropic porous layer. *International Journal of Heat and Mass Transfer ,* 1975-1987.

Flores Márquez, L. (1992). *Transferts de chaleur et de masse en milieu sédimentaire et fracturé. Modélisation numérique de la convection naturelle autour du site géothermique de Soultz (Graben du Rhin).* These présentée devant l'Institut National Polytechnique de Lorraine por l'obtention du titre de Docteur de L'I.N.P.L.

Greenkorn, R. A. (1983). *Flow phenomena in porous media. Fundamentals and aplications in petroleum, water and food production.* New York: Dekker.

Herrera Revilla, I. (2006). *Modelación matemática de sistemas terrestres.* México: Apuntes curso de modelación matemática y computacional I.

Herrera, I., & Allen, M. B. (1986). Modelación Computacional de Sistemas en Ciencias e Ingeniería. *Comunicaciones Técnicas, Serie Docencia y Divulgación* (9), D17.

Herrera, I., & Pinder, G. (2011). *Mathematical models of science and engineering.* John Willey.

Incropera, F. P., & Dewitt, D. P. (2002). *Fundamentals of Heat and Mass Transfer* (Fifth Edition ed.). John Wiley & Sons.

Jue, T. (2003). Analysis of thermal convection in a fluid-saturated porous cavity with internal heat generation. *Heat and Mass Transfer* , 83-89.

Krishna, J., Basak, T., & Das, S. (2008). Natural convection in a heat generating hydrodynamically and thermally anisotropic non-Darcy porous medium. *International Journal of Heat and Mass Transfer* , 4691-4703.

Larman, C. (2003). *UML y patrones. Una introducción al análisis y diseño orientado a objetos y al proceso unificado.* Madrid: Pearson Educación, S.A.

Luz Neto, H., Quaresma, J., & Cotta, R. (2006). Integral transform solution for natural convection in three-dimensional porous cavities: Aspecto ratio effects. *International Journal of Heat and Mass Transfer* , 4687-4695.

Nagano, K., Mochida, T., & Ochifuji, K. (2002). Influence of natural convection on forced horizontal flow in saturated porous media for aquifer thermal energy storage. *Applied thermal engineering* , 1299-1311.

Nakamura. (1995). *Métodos numéricos con software.* Prentice Hall.

NetBeans. (2009). Obtenido de http://netbeans.org/features/index.html

Nithiarasu, P., Sujatha, K., Ravindran, K., Sundararajan, T., & Seetharamu, K. (2000). Non-Dacry natural convection in a hydrodynamically and thermally anisotropic porous medium. *Computer methods in applied Mechanics and Engineering* , 413-430.

Press, W. H., Teukolsky, S. A., Vetterlin, W. T., & Flannery, B. P. (2007). *Numerical Recipes: The Art of Scientific Computing.* Cambridge University Press.

Quintard, M. (1983). *Stabilité des déplacements miscibles en milieu poreux homogène: injection d'un fluide chaud dans un massif poreux saturé par ce même fluide froid. Thèse Doc. d'Etat Es-sciences.* Univ. Bordeaux I.

Royer, J. J., & Flores, L. (1994). Two-dimensional natural convection in an anisotropic and heterogeneous porous medium with internal heat generation. *Ing. J. Heat Mass Transfer* , *37* (9), 1387-1399.

Saad, Y. (2000). *Iterative Methods for Sparse Linear Systems* (2 ed. ed.). SIAM.

Skiba, Y. (2005). *Métodos y equemas numéricos, un análisis computacional.* Ciudad de México: UNAM.

Yates, R. (2007). *Apuntes de la clase de modelación computacional de sistemas terrestres.* México D.F.: Posgrado en Ciencias de la Tierra. Instituto de Geofísica. UNAM.

Zienkiewicz, O. C., & Taylor, R. L. (200). *The Finite Element Method* (Fifth ed., Vol. 1: The Basis). London: Butterworth-Heinemann.

Apéndice I. Modelación de los sistemas continuos

En este apéndice se transcribe, con el permiso del Dr. Ismael Herrera Revilla, parte de sus apuntes del curso de Modelación Matemática de Sistemas Terrestres. El resto de los apuntes puede también consultarse en (Herrera Revilla, 2006; Herrera y Pinder, 2011; http://www.mmc.geofisica.unam.mx).

Los modelos

Un modelo de un sistema es un sustituto de cuyo comportamiento es posible derivar el correspondiente al sistema original. Los modelos matemáticos, en la actualidad, son los utilizados con mayor frecuencia y también los más versátiles. En las aplicaciones específicas, están constituidos por programas de cómputo cuya aplicación y adaptación a cambios de las propiedades de los sistemas es relativamente fácil. También, sus bases y las metodologías que utilizan son de gran generalidad, por lo que es posible construirlos para situaciones y sistemas muy diversos.

Los modelos matemáticos son entes en los que se integran los conocimientos científicos y tecnológicos y con ellos se construyen programas de cómputo que se implementan con medios computacionales. Hoy en día, la simulación numérica permite estudiar sistemas complejos y fenómenos naturales que sería muy costoso, peligroso o incluso imposible estudiar por experimentación directa. En esta perspectiva la significación de los modelos matemáticos en Ciencias e Ingeniería es clara, porque la modelación matemática constituye el método más efectivo de predecir el comportamiento de los diversos sistemas de interés. En nuestro país, éstos son usados ampliamente en la industria petrolera, en las ciencias y la ingeniería del agua, en la industria automotriz y en muchas otras.

Física microscópica y física macroscópica

El estudio de la materia y su movimiento puede abordarse a partir de dos enfoques:

> Enfoque microscópico: el de las moléculas, los átomos y las partículas elementales. La predicción del comportamiento de estas partículas es el objeto de estudio de la Mecánica Cuántica y la Física Nuclear.
> Enfoque macroscópico: el correspondiente a la llamada *Física Macroscópica* cuyas bases teóricas las proporciona la Mecánica de los Medios Continuos.

Cuando se desea predecir el comportamiento de sistemas tan grandes como la atmósfera o un acuífero, los cuales están formados por un número extraordinariamente grande de

moléculas y átomos, su estudio resulta inaccesible a partir del enfoque microscópico, por lo tanto el enfoque macroscópico es el apropiado.

El estudio de la materia bajo el enfoque macroscópico considera que los cuerpos llenan el espacio que ocupan, es decir que no tienen huecos, que es la forma en que los vemos sin el auxilio de un microscopio. Por ejemplo, el agua llena todo el espacio del recipiente donde está contenida. Este enfoque macroscópico está presente en la física clásica. Sin embargo, la ciencia ha avanzado y ahora se sabe que la materia está llena de huecos, que nuestros sentidos no perciben y que la energía también está cuantificada.

A pesar de que estos dos enfoques para el análisis de los sistemas físicos parecen a primera vista conceptualmente contradictorios, ambos son no tan sólo compatibles sino además complementarios, y es posible establecer la relación entre ellos utilizando a la Mecánica Estadística. La Teoría de los Sistemas Macroscópicos es aplicable no solamente a sistemas físicos, sino también a sistemas químicos y a algunos sistemas formados por seres vivos. Así, utilizándola en algunos casos es posible predecir el movimiento y evolución de poblaciones formadas por seres vivos microscópicos. Para ello, se ignora a los individuos microscópicos y se les considera distribuidos en todo el espacio de estudio.

Cinemática de los sistemas continuos

La Teoría de los Sistemas Continuos indica que los *cuerpos* llenan todo el espacio que ocupan y que en cada punto del espacio físico hay una y solamente una partícula. Así se define a un *sistema continuo* como un conjunto de partículas.

Un cuerpo es un subconjunto de partículas que en cualquier instante dado ocupan un *dominio* (en el sentido matemático) del espacio físico; es decir, del espacio Euclidiano tridimensional. Se denota por $B(t)$ a la región ocupada por el cuerpo $\mathcal{B}$, en el tiempo t. En general, t puede ser cualquier número real; es decir $-\infty < t < \infty$, pero en cada caso de estudio hay un intervalo finito de tiempo en el cual se centra el interés. Dado un cuerpo $\mathcal{B}$, todo subdominio $\tilde{\mathcal{B}} \subset \mathcal{B}$, constituye a su vez otro cuerpo; en tal caso, se dice que $\tilde{\mathcal{B}} \subset \mathcal{B}$ es un subcuerpo de $\mathcal{B}$.

De acuerdo con lo mencionado antes, una hipótesis básica de la teoría de los sistemas continuos es que en cualquier tiempo $t \in (-\infty, \infty)$ y en cada punto $\underline{x} \in B(t)$ de la región ocupada por el cuerpo, hay una y sólo una partícula del cuerpo $\mathcal{B}$.

Como el estudio de los sistemas continuos incluye no solamente la estática (es decir, los cuerpo en reposo), sino también la dinámica (es decir, los cuerpo en movimiento), un primer problema de la cinemática de los sistemas continuos consiste en establecer un procedimiento para identificar a las partículas cuando están en movimiento.

Sea $\underline{X}$ una partícula y $\underline{p}(\underline{X}, t)$ el vector de la posición que ocupa en el espacio físico dicha partícula en el tiempo t. Una forma, aunque no la única, de identificar a la partícula

$\underline{X}$, es asociándole la posición que ocupa en un instante determinado al cual se le llama *tiempo inicial*. Si se toma en particular el tiempo $t = 0$, se tiene que

$$\underline{p}(\underline{X}, 0) \equiv \underline{X} \tag{I.1}$$

A las coordenadas del vector $\underline{X} \equiv (X_1, X_2, X_3)$, se les llama las *coordenadas materiales* de la partícula. En este caso, las coordenadas materiales de una partícula son las coordenadas del punto del espacio físico que ocupaba la partícula en el tiempo inicial $t = 0$. El tiempo inicial puede ser cualquier otro, si así se conviene. Frecuentemente se utiliza la notación $\underline{x}$ para las coordenadas del punto que ocupa la partícula en el tiempo t y en tal caso $\underline{x} = \underline{p}(\underline{X}, t)$. Además, en general $\underline{x} \neq \underline{X}$.

Sea $\mathcal{B}$ el dominio ocupado por un cuerpo en el tiempo inicial, entonces $\underline{X} \in \mathcal{B}$ sí y solamente si la partícula $\underline{X}$ pertenece al cuerpo. Es decir, $\mathcal{B}$ caracteriza al cuerpo. Sin embargo, debido al movimiento, la región ocupada por $\mathcal{B}$ cambia con el tiempo y entonces se denota por $B(t)$ (ver Figura 1). Formalmente, para cualquier $t \in (-\infty, \infty)$, $B(t)$ se define por

$$B(t) \equiv \left\{ \underline{x} \in R^3 \,\middle|\, \exists \underline{X} \in \mathcal{B} \ni \underline{x} = \underline{p}(\underline{X}, t) \right\} \tag{I.2}$$

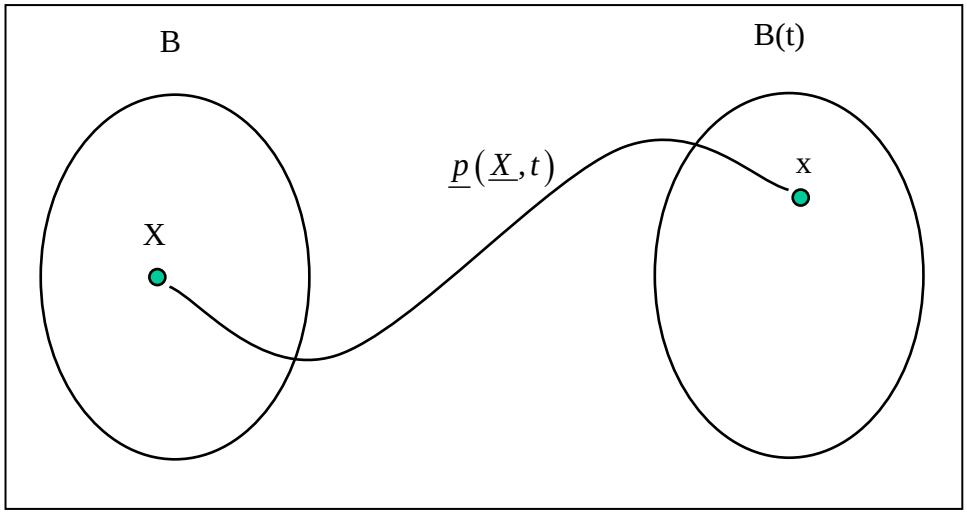


Figura 1. Movimiento de partículas

El vector de posición $\underline{p}(\underline{X}, t)$ es función del vector (tridimensional) $\underline{X}$ y del tiempo t. Fijando t, $\underline{p}(\underline{X}, t)$ define una transformación del espacio euclidiano $\mathbb{R}^3$ en sí mismo y la ecuación (I.2) es equivalente a $B(t) = \underline{p}(\mathcal{B}, t)$. Una notación utilizada para representar esta

familia de mapeos es $\underline{p}(\square, t)$. De acuerdo a la hipótesis de los sistemas continuos: *En cualquier tiempo* $t \in (-\infty, \infty)$ *y en cada punto* $\underline{x} \in B(t)$ *de la región ocupada por el cuerpo hay una, y solo una, partícula del cuerpo* $\mathcal{B}$ *para cada* t *fijo.* Es decir, $\underline{p}(\square, t)$ es un mapeo biunívoco, por lo que existe el mapeo inverso $\underline{p}^{-1}(\square, t)$.

Si se fija la partícula $\underline{X}$ en la función $\underline{p}(\underline{X}, t)$ y se varía el tiempo t, se obtiene su trayectoria. Esto permite obtener la velocidad de cualquier partícula, la cual es un concepto central en la descripción del movimiento. Ella se define como la derivada con respecto al tiempo de la posición cuando la partícula se mantiene fija. Es decir, es la derivada parcial con respecto al tiempo de la función de posición $\underline{p}(\underline{X}, t)$. Por lo mismo, la velocidad como función de las coordenadas materiales de las partículas, está dada por

$$\underline{V}(\underline{X}, t) \equiv \frac{\partial \underline{p}}{\partial t}(\underline{X}, t) \tag{I.3}$$

Propiedades intensivas y sus representaciones

A las funciones definidas, para cada tiempo, en cada una de las partículas de un sistema continuo se les llaman *propiedades intensivas*. Las *propiedades intensivas* pueden ser funciones escalares o funciones vectoriales. Por ejemplo, la velocidad definida por la ecuación (I.3) es una función vectorial que depende de la partícula $\underline{X}$ y del tiempo t. Una *propiedad intensiva* con valores vectoriales es equivalente a tres escalares, correspondientes a cada una de sus tres componentes. Hay dos formas de representar a las propiedades intensivas: la representación Euleriana y la representación Lagrangiana. Los nombres son en honor a los matemáticos Leonard Euler (1707-1783) y Joseph Louis Lagrange (1736-1813), respectivamente. Frecuentemente, el punto de vista Lagrangiano es ampliamente utilizado en el estudio de los sólidos, mientras que el Euleriano se usa más en el estudio de los fluidos.

Considerando una propiedad intensiva escalar, la cual en el tiempo t toma el valor $\phi(\underline{X}, t)$ en la partícula $\underline{X}$. Entonces, de esta manera se define una función $\phi : \mathcal{B} \to \square^1$, para cada $t \in (-\infty, +\infty)$, a la que se denomina representación Lagrangiana de la propiedad intensiva considerada. Ahora, sea $\psi(\underline{x}, t)$ el valor que toma esa propiedad en la partícula que ocupa la posición $\underline{x}$ en el tiempo t. En este caso, para cada $t \in (-\infty, +\infty)$ se define una función $\psi : B(t) \to \square^1$, a la cual se denomina representación Euleriana de la función considerada. Estas dos representaciones de una misma propiedad están relacionadas por la siguiente identidad:

$$\phi(\underline{X},t) \equiv \psi(\underline{p}(\underline{X},t),t) \tag{I.4}$$

Cabe destacar que, aunque ambas representaciones satisfacen la ecuación (I.4), las funciones $\phi(\underline{X},t)$ y $\psi(\underline{x},t)$ no son idénticas. Sus argumentos $\underline{X}$ y $\underline{x}$ son vectores tridimensionales (es decir, puntos de $\mathbb{R}^3$), sin embargo si se tiene que $\underline{x} = \underline{X}$, en general

$$\phi\left(\underline{X},t\right) \neq \psi\left(\underline{X},t\right) \tag{I.5}$$

La expresión de la velocidad de una partícula dada por la ecuación (I.3), define a su representación Lagrangiana, por lo que utilizando la ecuación (I.4) es claro que

$$\frac{\partial \underline{p}}{\partial t}\left(\underline{X},t\right) = \underline{V}(\underline{X},t) \equiv \underline{v}(\underline{p}(\underline{X},t),t) \tag{I.6}$$

donde $\underline{v}(\underline{x},t)$ es la representación Euleriana de la velocidad. Por lo mismo

$$\underline{v}(\underline{x},t) \equiv \underline{V}(\underline{p}^{-1}(\underline{x},t),t); \tag{I.7}$$

De la ecuación anterior se interpreta que la velocidad en el punto $\underline{x}$ del espacio físico, es igual a la velocidad de la partícula que pasa por dicho punto en el instante t. La ecuación (I.7) es un caso particular de la relación:

$$\psi(\underline{x},t) \equiv \phi(\underline{p}^{-1}(\underline{x},t),t) \tag{I.8}$$

la cual es de validez general y se trata de otra forma de expresar la ecuación (I.4), que indica la relación entre las dos representaciones de una misma propiedad intensiva.

La derivada parcial con respecto al tiempo de la representación Lagrangiana $\phi\left(\underline{X},t\right)$ de una propiedad intensiva, de acuerdo a la definición de la derivada parcial de una función, es la tasa de cambio con respecto al tiempo que ocurre en una partícula fija. Es decir, suponiendo que el observador se *monta* en una partícula, se mide la propiedad intensiva y luego se derivan con respecto al tiempo los valores así obtenidos, el resultado final es $\partial\phi(\underline{X},t)/\partial t$.

En cambio, si $\psi(\underline{x},t)$ es la representación Euleriana de esa misma propiedad, entonces $\partial\psi(\underline{x},t)/\partial t$ es simplemente la tasa de cambio con respecto al tiempo que ocurre en un punto fijo en el espacio. Tiene interés evaluar la tasa de cambio con respecto al tiempo que ocurre en una partícula fija cuando se usa la representación Euleriana. Derivando con

respecto al tiempo a la identidad de la ecuación (I.4) y aplicando *la regla de la cadena*, se obtiene

$$\frac{\partial \phi(\underline{X},t)}{\partial t} = \frac{\partial \psi}{\partial t}(\underline{p}(\underline{X},t),t) + \sum_{i=1}^{3} \frac{\partial \psi}{\partial x_i}(\underline{p}(\underline{X},t),t)\frac{\partial p_i}{\partial t}(\underline{X},t) \tag{I.9}$$

La derivada material se acostumbra definir por el símbolo $D\psi / Dt$, donde

$$D\psi / Dt = \partial \psi / \partial t + \sum_{i=1}^{3} v_i \partial \psi / \partial x_i \tag{I.10}$$

o, más brevemente

$$D\psi / Dt = \partial \psi / \partial t + \underline{v} \cdot \nabla \psi \tag{I.11}$$

Utilizando esta notación, se puede escribir

$$\frac{\partial \phi(\underline{X},t)}{\partial t} = \left(D\psi / Dt\right)\left(\underline{p}(\underline{X},t),t\right) \equiv \left(\partial \psi / \partial t + \underline{v} \cdot \nabla \psi\right)\left(\underline{p}(\underline{X},t),t\right) \tag{I.12}$$

Por ejemplo, la aceleración de una partícula se define como la derivada de la velocidad cuando se mantiene a la partícula fija. Así, la representación Euleriana de la aceleración se obtiene aplicando la ecuación (I.11) a la representación Euleriana de velocidad:

$$D\underline{v} / Dt = \partial \underline{v} / \partial t + \underline{v} \cdot \nabla \underline{v} \tag{I.13}$$

Una expresión tal vez más transparente se obtiene aplicando la Ec.(I.12) a cada una de las componentes de la velocidad. Así:

$$Dv_i / Dt = \partial v_i / \partial t + \underline{v} \cdot \nabla v_i; \quad i = 1,...,3 \tag{I.14}$$

Desde luego, la aceleración, en representación Lagrangiana es simplemente

$$\frac{\partial}{\partial t}\underline{V}(\underline{X},t) \equiv \frac{\partial^2}{\partial t^2}\underline{p}(\underline{X},t) \tag{I.15}$$

Propiedades extensivas

En la sección anterior se consideraron funciones definidas en las partículas de un cuerpo; más precisamente, funciones que hacen corresponder a cada partícula y cada tiempo un número real, o un vector del espacio Euclidiano tridimensional $\mathbb{R}^3$.

En ésta sección, en cambio, se empezará por considerar funciones que, a cada cuerpo $\mathcal{B}$ de un sistema continuo y a cada tiempo t, le asocia un número real o un vector de $\mathbb{R}^3$. A una función de este tipo $E(\mathcal{B},t)$ se le llama *propiedad extensiva* cuando está dada por una integral:

$$E(\mathcal{B},t) \equiv \int_{B(t)} \psi(\underline{x},t)d\underline{x}; \tag{I.16}$$

Se observa que en tal caso el integrando es una función $\psi(\underline{x},t)$ necesariamente integrable, y por lo mismo, una *propiedad intensiva* en su representación Euleriana. Además, la ecuación (I.16) establece una correspondencia biunívoca entre las propiedades extensivas y las intensivas, porque dada la representación Euleriana $\psi(\underline{x},t)$ de cualquier *propiedad intensiva*, su integral sobre el dominio ocupado por cualquier cuerpo define una *propiedad extensiva* e inversamente, dada una *propiedad extensiva* el integrando, cuando se le expresa en la forma de la ecuación (I.16), define una *propiedad intensiva*.

Además, la notación empleada en la ecuación (I.16) es muy explícita, pues ahí se ha escrito $E(\mathcal{B},t)$ para enfatizar que el valor de la propiedad extensiva corresponde al cuerpo $\mathcal{B}$. Sin embargo en lo que sucesivo, siempre que sea posible, sin caer en ambigüedad, se simplificará la notación omitiendo el símbolo $\mathcal{B}$; es decir, se escribirá $E(t)$ en vez de $E(\mathcal{B},t)$.

Hasta el momento se ha definido a las propiedades intensivas por unidad de volumen. Sin embargo, es frecuente que se les defina por unidad de masa (Allen, Herrera, y Pinder, 1988). Es fácil ver que la *propiedad intensiva por unidad de volumen* es igual a la *propiedad intensiva por unidad de masa* multiplicada por la densidad de masa (es decir, masa por unidad de volumen), por lo que utilizando la densidad de masa, se puede pasar de un concepto al otro sin dificultad. Una ventaja de utilizar las propiedades intensivas por unidad de volumen, en lugar de las propiedades intensivas por unidad de masa, es que la correspondencia entre las *propiedades extensivas* y las *propiedades intensivas por unidad de volumen* es más directa; dada una *propiedad extensiva*, la *propiedad intensiva* correspondiente es la función que aparece como integrando, cuando aquella se expresa como una integral de volumen. Además, del cálculo se sabe que

$$\psi(\underline{x},t) \equiv \lim_{Vol\to 0} \frac{E(t)}{Vol} = \lim_{Vol\to 0} \frac{\int_{B(t)} \psi(\underline{\xi},t)d\underline{\xi}}{Vol} \tag{I.17}$$

La ecuación (I.17) proporciona un procedimiento efectivo para determinar las propiedades extensivas experimentalmente: se mide la *propiedad extensiva* en un volumen pequeño del sistema continuo de que se trate, se le divide entre el volumen, y el cociente que se obtiene

es una buena aproximación de la *propiedad intensiva* si el volumen es suficientemente pequeño.

Por último, se puede decir que una propiedad es extensiva, cuando ésta satisfaga la definición de *propiedad extensiva* que se estableció en esta sección. La importancia del concepto de propiedad extensiva radica en que el modelo general de la teoría de los sistemas continuos se formula en términos de ecuaciones de balance de propiedades extensivas.

Balance de propiedades extensivas e intensivas

Los modelos matemáticos de los sistemas continuos están constituidos por balances de propiedades extensivas. Por ejemplo, los modelos de transporte de solutos (los contaminantes transportados por corrientes superficiales o subterráneas, son un caso particular de estos procesos de transporte) se construyen haciendo el balance de la masa de soluto que hay en cualquier dominio del espacio físico. Aquí, el término balance se usa, esencialmente, en un sentido contable. En la contabilidad que se realiza para fines financieros o fiscales, la diferencia de las entradas menos las salidas nos da el aumento, o cambio, de capital. En forma similar, en la mecánica de los medios continuos se realiza, en cada cuerpo del sistema continuo, un balance de las propiedades extensivas en que se basa el modelo.

Ecuación de balance global

Para realizar tales balances es necesario, en primer lugar, identificar las causas por las que las propiedades extensivas pueden cambiar. Se toma como ejemplo el de una propiedad extensiva a las existencias de maíz que hay en el país. El primer cuestionamiento radica en saber qué causas pueden motivar la variación, o cambio, de esas existencias. Un análisis sencillo muestra que la variación de las cantidades de maíz se debe a la producción o consumo de éste. Otra causa de variación puede deberse a la importación o exportación a través de los límites del país (fronteras o litorales). Con esto se agotan las causas posibles; es decir, esta lista es exhaustiva. Producción y consumo son términos similares que fácilmente se engloban en uno sólo de esos conceptos, pues la diferencia entre ellos es que sus efectos tienen signos opuestos. Así, si por convención se dice que la producción puede ser negativa, entonces el consumo es simplemente una producción negativa. Una vez que se adopta esta convención, ya no es necesario ocuparse por separado del consumo. En forma similar, la exportación puede tratarse como una importación negativa. Entonces, el incremento en las existencias ΔE en un período Δt queda dado por la ecuación

$$\Delta E = P + I \tag{I.18}$$

donde a la producción y a la importación, ambas con signo, se les ha representado por P e I, respectivamente.

De forma similar, en la mecánica de los medios continuos, la lista exhaustiva de las causas por las que una propiedad extensiva de cualquier cuerpo puede cambiar, contiene solamente dos motivos:

Por producción en el interior del cuerpo
Por importación (es decir, transporte) a través de la frontera.

Esto conduce a la siguiente ecuación de *balance global*, de gran generalidad, para las propiedades extensivas:

$$\frac{dE(t)}{dt} = \int_{B(t)} g(\underline{x},t)\, d\underline{x} + \int_{\partial B(t)} q(\underline{x},t)\, d\underline{x} + \int_{\Sigma(t)} g_{\Sigma}(\underline{x},t)\, d\underline{x} \qquad \text{(I.19)}$$

Donde $g(\underline{x},t)$ es la generación en el interior del cuerpo, con signo, de la propiedad extensiva correspondiente, por unidad de volumen, por unidad de tiempo. Además, en la ecuación (I.19) se ha tomado en cuenta la posibilidad de que haya producción concentrada en la superficie $\Sigma(t)$, la cual está dada en esa ecuación por la última integral, donde $g_{\Sigma}(\underline{x},t)$ es la producción por unidad de área. Por otra parte, $q(\underline{x},t)$ es lo que se importa, o transporta, hacia el interior del cuerpo a través de la *frontera del cuerpo* $\partial B(t)$; en otras palabras, es el *flujo* de la propiedad extensiva a través de la frontera del cuerpo, por unidad de área, por unidad de tiempo. Puede demostrarse, con base en hipótesis válidas en condiciones muy generales, que para cada tiempo t existe un campo vectorial $\underline{\tau}(\underline{x},t)$ tal que

$$q(\underline{x},t) \equiv \underline{\tau}(\underline{x},t)\cdot \underline{n}(\underline{x},t) \qquad \text{(I.20)}$$

donde $\underline{n}(\underline{x},t)$ es normal exterior a $\partial B(t)$. En vista de esta relación, la ecuación de balance se puede escribir como

$$\frac{dE}{dt}(t) = \int_{B(t)} g(\underline{x},t)\, d\underline{x} + \int_{\partial B(t)} \underline{\tau}(\underline{x},t)\cdot \underline{n}(\underline{x},t)\, d\underline{x} + \int_{\Sigma(t)} g_{\Sigma}(\underline{x},t) d\underline{x} \qquad \text{(I.21)}$$

A la función $g(\underline{x},t)$ se le denomina *generación interna* y al campo vectorial $\underline{\tau}(\underline{x},t)$ *campo de flujo* (Allen, Herrera, y Pinder, 1988). La ecuación (I.21) se conoce con el nombre de *Ecuación General de Balance Global* y es la ecuación básica de los balances de los sistemas continuos.

Condiciones de balance local

Los modelos de los sistemas continuos están constituidos por las ecuaciones de balance correspondientes a una colección de propiedades extensivas. Así, a cada sistema continuo le corresponde una familia de propiedades extensivas, tal que el modelo matemático del

sistema está constituido por las condiciones de balance de cada una de las propiedades extensivas de dicha familia. Sin embargo, las propiedades extensivas mismas no se utilizan directamente en la formulación del modelo, en su lugar se usan las propiedades intensivas asociadas a cada una de ellas. Esto es posible porque las Ecuaciones de Balance Global son equivalentes a las llamadas *Condiciones de Balance Local*, las cuales se expresan en términos de las propiedades intensivas correspondientes. Las Condiciones de Balance Local son de dos clases:

Ecuaciones Diferenciales de Balance Local
Condiciones de Salto

Las primeras son ecuaciones diferenciales parciales que se deben satisfacer en cada punto del espacio ocupado por el sistema continuo. Las segundas son ecuaciones algebraicas que las discontinuidades deben satisfacer donde ocurren; es decir, en cada punto de Σ.

Cabe mencionar que las Ecuaciones Diferenciales de Balance Local son de uso mucho más amplio que las Condiciones de Salto, pues estas últimas solamente se aplican cuando y donde hay discontinuidades, mientras que las primeras en todo punto del espacio ocupado por el sistema continuo. Además, solamente en problemas de carácter especial es necesario introducir modelos en que las propiedades intensivas son discontinuas. Los llamados *choques*, ampliamente conocidos en el flujo supersónico de fluidos compresibles no viscosos, son cambios muy rápidos tanto en la presión como en otras propiedades del fluido, los cuales en los modelos en que se desprecia la viscosidad, se simulan como discontinuidades de dichas propiedades del fluido.

Una vez establecidas las ecuaciones diferenciales y de salto del balance local, e incorporada la información científica y tecnológica necesaria para completar el modelo (la cual se introduce a través de las llamadas *ecuaciones constitutivas*), el problema matemático de desarrollar el modelo y derivar sus predicciones se transforma en uno correspondiente a la Teoría de la Ecuaciones Diferenciales, generalmente parciales, y sus Métodos Numéricos.

Las ecuaciones de Balance Local

En esta sección se supondrá que las propiedades intensivas pueden tener discontinuidades de *salto*, exclusivamente a través de la superficie $\Sigma(t)$. Además, se entiende por *discontinuidad de salto*, cuando el límite por ambos lados de $\Sigma(t)$ existe, pero son diferentes.

La demostración de los resultados matemáticos que se dan a continuación se pueden encontrar en (Herrera Revilla, 2006; Herrera y Pinder, 2011).

Teorema 1 *Para cada t se supondrá* $t \geq 0$, *sea* $B(t) \subset \mathbb{R}^3$ *el dominio ocupado por un cuerpo, como se le definió en este capítulo. Se supone también que la propiedad intensiva,* $\psi(\underline{x},t)$ *es* C^1, *excepto a través de la superficie* $\Sigma(t)$. *Además, sean las funciones* $\underline{v}(\underline{x},t)$ *y* $\underline{v}_\Sigma(\underline{x},t)$, *esta última definida para* $\underline{x} \in \Sigma(t)$ *solamente, las velocidades de las partículas y la de* $\Sigma(t)$, *respectivamente. Entonces*

$$\frac{d}{dt}\int_{B(t)} \psi \, dx \equiv \int_{B(t)} \left\{ \frac{\partial \psi}{\partial t} + \nabla \cdot (\underline{v}\psi) \right\} dx + \int_\Sigma \left[(\underline{v} - \underline{v}_\Sigma)\psi \right] \cdot \underline{n} dx \qquad \text{(I.22)}$$

Teorema 2 *Si se considera un sistema continuo, entonces la Ecuación de Balance Global (I.21) se satisface para todo cuerpo del sistema continuo si y solamente si se cumplen simultáneamente las siguientes condiciones:*

La ecuación diferencial

$$\frac{\partial \psi}{\partial t} + \nabla \cdot (\underline{v}\psi) = \nabla \cdot \underline{\tau} + g \qquad \textit{(I.23)}$$

vale en todo punto $\underline{x} \in R^3$, *excepto en* Σ, *de la región ocupada por el sistema.*

La ecuación

$$\left[\psi (\underline{v} - \underline{v}_\Sigma) - \underline{\tau} \right] \cdot \underline{n} = \underline{g}_\Sigma \qquad \textit{(I.24)}$$

vale en todo punto, $\underline{x} \in \Sigma$.

A las ecuaciones (I.23) y (I.24) se les llama *Ecuación Diferencial de Balance Local* y *Condición de Salto*, respectivamente.

Desde luego, el caso más general que se estudiará se refiere a situaciones dinámicas; es decir, aquéllas en que las propiedades intensivas cambian con el tiempo. Sin embargo, los estados estacionarios de los sistemas continuos son de sumo interés. Por estado estacionario se entiende uno en que las propiedades intensivas son independientes del tiempo. En los estados estacionarios, además, las superficies de discontinuidad $\Sigma(t)$ se mantienen fijas (no se mueven). En este caso $\partial\psi / \partial t \equiv 0$ y $\underline{v}_\Sigma \equiv 0$. Por lo mismo, para los estados estacionarios, la Ecuación de Balance Local y la Condición de Salto se reducen a

$$\nabla \cdot (\underline{v}\psi) = \nabla \cdot \underline{\tau} + g \qquad \text{(I.25)}$$

que vale en todo punto $\underline{x} \in R^3$, y

$$\left[\psi \underline{v} - \underline{\tau} \right] \cdot \underline{n} = \underline{g}_\Sigma \qquad \text{(I.26)}$$

que se satisface en todo punto de la discontinuidad $\Sigma(t)$, respectivamente. Merece la pena resaltar que solamente en algunos problemas de carácter bastante especial $\underline{g}_{\Sigma} \neq 0$, como en el tratamiento de problemas de Ingeniería de Yacimientos donde el punto de burbuja es variable (Allen, Herrera, y Pinder, 1988).

Ejemplos de Condiciones de Balance Local

Una de las aplicaciones más sencillas de las Condiciones de Balance Local, es para formular restricciones en el movimiento. En esta sección se ilustran este tipo de aplicaciones formulando condiciones que se deben cumplir localmente cuando un fluido es incompresible. La afirmación de que un fluido es *incompresible* significa que todo cuerpo conserva el volumen de fluido en su movimiento. Entonces se considerarán dos casos: el de un *fluido libre* y el de un *fluido en medio poroso*. En el primer caso, el fluido llena completamente el espacio físico que ocupa el cuerpo, por lo que el volumen del fluido es igual al volumen del dominio que ocupa el cuerpo. Así

$$V_f(t) = \int_{B(t)} d\underline{x} \tag{I.27}$$

aquí $V_f(t)$ es el volumen del fluido contenido en el cuerpo y $B(t)$ es el dominio del espacio físico (es decir, de $\mathbb{R}^3$) ocupado por el cuerpo. De forma más explícita, la ecuación (I.27) se puede escribir como

$$V_f(t) = \int_{B(t)} 1\, d\underline{x} \tag{I.28}$$

Comparando esta ecuación con la ecuación (I.16), se observa que el volumen del fluido es una *propiedad extensiva* y que la *propiedad intensiva* que le corresponde es $\psi \equiv 1$. Además, la hipótesis de incompresibilidad implica

$$\frac{dV_f}{dt}(t) = 0 \tag{I.29}$$

esta es el *Balance Global* de la ecuación (I.21), con $g = \underline{g}_{\Sigma} \equiv 0$ y $\underline{\tau} \equiv 0$, el cual a su vez es equivalente a las ecuaciones (I.23) y (I.24) con $\psi \equiv 1$. Estas ecuaciones se reducen a

$$\left.\begin{array}{c} \nabla \cdot \underline{v} = 0 \\ [\underline{v}] \cdot \underline{n} = 0, \, en \, \Sigma \end{array}\right\} \tag{I.30}$$

La primera es la bien conocida condición de incompresibilidad para un *fluido libre* y la segunda implica, dado que $\underline{n}$ es cualquier dirección, que si un fluido libre es incompresible la velocidad de sus partículas es necesariamente continua.

El caso en que el fluido se encuentra en un *'medio poroso'*, es diferente. Un medio poroso es un material sólido que tiene huecos distribuidos en toda su extensión (Figura 2). Cuando los poros están llenos de un fluido, se dice que el medio poroso está *saturado*. Esta

situación es la de mayor interés en la práctica y es también la más estudiada. A la fracción del volumen del sistema, constituido por la *matriz sólida* y los huecos, se le llama *porosidad* y en este capítulo se le representa con ε . Así

$$\varepsilon\left(\underline{x},t\right)=\lim_{Vol\to 0}\frac{Volumen\ de\ huecos}{Volumen\ total} \tag{I.31}$$

Aquí se ha escrito $\varepsilon\left(\underline{x},t\right)$ para enfatizar que la porosidad generalmente es función tanto de la posición como del tiempo. Las variaciones con la posición pueden ser debidas, por ejemplo, a heterogeneidad del medio y los cambios con el tiempo a su elasticidad; es decir, los cambios de presión del fluido originan esfuerzos en los porosos que los dilatan o los encogen.

Cuando el medio está *saturado*, el volumen del fluido (V_f) es igual al volumen de los huecos del dominio del espacio físico que ocupa. Así,

$$V_f\left(t\right)=\int_{B(t)}\varepsilon\left(\underline{x},t\right)dx \tag{I.32}$$

en vista de esta ecuación, la propiedad intensiva asociada al volumen de fluido es la porosidad, $\varepsilon\left(\underline{x},t\right)$, por lo que la *condición de incomprensibilidad de un fluido contenido en un medio poroso*, está dada por las ecuaciones (I.23) y (I.24), con $\psi\equiv\varepsilon\left(\underline{x},t\right)$, $g=0$ y $\underline{\tau}=0$, es decir

$$\left.\begin{aligned}&\frac{\partial\varepsilon}{\partial t}+\nabla\cdot\left(\varepsilon\underline{v}\right)=0\\&\left[\varepsilon\left(\underline{v}-\underline{v}_{\Sigma}\right)\right]\cdot\underline{n}=0,\ en\ \Sigma\end{aligned}\right\} \tag{I.33}$$

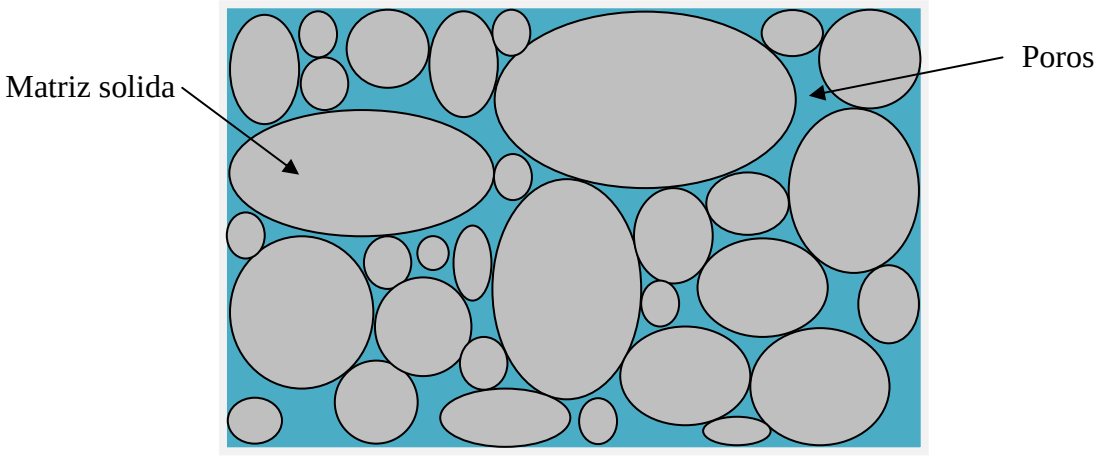


Figura 2. Medio poroso

Forma General de los Modelos de los Sistemas Continuos

En esta sección se explica cómo se construyen los modelos de los sistemas continuos. En realidad es más apropiado llamarlos modelos de los procesos que tienen lugar en dichos sistemas, pues más que los sistemas mismos, se modelan sus procesos; por ejemplo, el transporte de solutos contenidos en un fluido.

Primero se identifica a un conjunto finito, o familia, de propiedades extensivas. En el ejemplo planteado de transporte de solutos, la única propiedad extensiva sería la masa de dicho soluto.

A cada una de las propiedades extensivas de esa familia se les impone la condición de satisfacer la ecuación general de Balance Global (I.21), en cada cuerpo del sistema continuo, lo cual es equivalente a la ecuación diferencial de Balance Local y a la condición de salto. Es interesante mencionar que esta forma de proceder se aplica no solamente a sistemas continuos de una sola fase, sino también a sistemas de varias fases.

Se dice que el modelo de un sistema continuo es *completo* se define un problema *bien planteado*. Se dice que un problema de valores iniciales y de frontera es bien planteado si se cumple que:

i. Existe una y sólo una solución
ii. La solución depende de manera continua de las condiciones iniciales y de frontera del problema

Para obtener *modelos completos*, además de las ecuaciones básicas (es decir las ecuaciones de balance (I.23) y condiciones de salto (I.24) correspondientes a cada propiedad extensiva y a cada fase), es necesario evaluar la generación interna, determinada por las funciones $g(\underline{x},t)$ y $g_{\Sigma}(\underline{x},t)$, y el *campo de flujo* $\underline{\tau}(\underline{x},t)$, en términos de funciones conocidas de las propiedades intensivas asociadas.

A través de estas funciones, llamadas *ecuaciones constitutivas*, se integra el conocimiento científico y tecnológico en los modelos matemáticos, el cual es proporcionado por las ciencias y las tecnologías. Ellas pueden ser la Física, Química, Bilogía, Ingenierías, etc.; todo depende de la clase de procesos involucrados. Por ejemplo, al estudiar el transporte de un soluto producido por una reacción química, la rapidez con que se produce el soluto depende de la concentración de él y de las demás sustancias que participen en la reacción, por lo que *suministro externo* $g(\underline{x},t)$ sería una función de esos parámetros. En este caso, a través de esta función se integraría el conocimiento químico del modelo. Es de destacarse que, si bien en este caso el *suministro externo* proviene del interior del sistema en estudio, éste se origina en alguna otra componente que es ajena o *externa* a aquella a la que se le hace balance.

En resumen, los modelos de los sistemas continuos están constituidos por:

i. Una colección de propiedades intensivas o, lo que es lo mismo, extensivas
ii. Un conjunto de ecuaciones de balance correspondientes (diferenciales y de salto), en cada una de las cuales la velocidad de las partículas es la de la fase correspondiente
iii. Suficientes relaciones que ligan a las propiedades intensivas entre sí y que definen a g, $\underline{\tau}$ y $\underline{V}$ en términos de éstas, las cuales se conocen como leyes constitutivas
iv. Condiciones iniciales y de frontera que deben satisfacer las propiedades intensivas

5 Apéndice II. Nomenclatura usada en la formulación matemática

La nomenclatura presentada en este apéndice se tomó de (Royer & Flores, 1994).

Parámetros dimensionales

a_t Coeficiente de variación de la conductividad térmica con respecto a la temperatura $\left[\frac{1}{\mathrm{K}}\right]$

A^* Producción de calor $\left[\frac{\mathrm{J}}{\mathrm{m}^3 \cdot \mathrm{s}}\right]$

$\underline{\mathbf{e}_1}$ Vector unitario para el sistema coordenado horizontal $[\mathrm{m}]$

$\underline{\mathbf{e}_3}$ Vector unitario para el sistema coordenado vertical $[\mathrm{m}]$

$\hat{\mathbf{g}}$ Aceleración de la gravedad $\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right]$

H Altura del rectángulo $[\mathrm{m}]$

k Permeabilidad media isotrópica $\left[\mathrm{m}^2\right]$

$\underline{\underline{\mathbf{K}}}^*$ Tensor simétrico de permeabilidad anisotrópica $\left[\mathrm{m}^2\right]$

L Ancho del rectángulo $[\mathrm{m}]$

p Presión $[\mathrm{Pa}]$

$\underline{\underline{\mathbf{S}}}$ Matriz diagonal asociada al sistema coordenado $\left(x^*, y^*\right), \begin{pmatrix} 1/H & 0 \\ 0 & 1/L \end{pmatrix}$

t^* Tiempo $[\mathrm{s}]$

T^* Temperatura $[\mathrm{K}]$

T_r^* Temperatura al nivel de referencia $[\mathrm{K}]$. Por ejemplo $T_r^* = \frac{\left(T_1^* + T_2^*\right)}{2}$

$\underline{\mathbf{v}}$ Velocidad de filtración $\left[\frac{\mathrm{m}}{\mathrm{s}}\right]$

x^* Coordenadas horizontales paralelas a L $[\mathrm{m}]$

z^* Coordenadas verticales paralelas a H $[\mathrm{m}]$

β_{th} Coeficiente de expansión volumétrica del fluido saturante $\left[\frac{1}{\mathrm{K}}\right]$

γ^* Coeficiente de variación de la viscosidad dinámica con relación a la temperatura del fluido saturante $\left[\frac{1}{\mathrm{K}}\right]$

ΔT^* Diferencia de temperaturas $[\mathrm{K}]$, $T_2^* - T_1^*$

η Viscosidad dinámica del fluido $\left[\frac{\mathrm{kg}}{\mathrm{m}\cdot\mathrm{s}}\right]$

η_r Viscosidad dinámica del fluido a la T_r^* $\left[\frac{\mathrm{kg}}{\mathrm{m}\cdot\mathrm{s}}\right]$

$\underline{\underline{\mathbf{\Lambda}}}^*$ Tensor simétrico de la conductividad térmica anisotrópica $\left[\frac{\mathrm{W}}{\mathrm{m}\cdot\mathrm{K}}\right]$

ρ_r Densidad de masa del fluido a la T_r^* $\left[\frac{\mathrm{kg}}{\mathrm{m}^3}\right]$

$\left(\rho c_p\right)^*$ Capacidad calorífica (fluido + sólido) a una presión constante $\left[\frac{\mathrm{J}}{\mathrm{m}^3\cdot\mathrm{K}}\right]$

$\left(\rho c_p\right)_f$ Capacidad calorífica del fluido a una presión constante $\left[\frac{\mathrm{J}}{\mathrm{m}^3\cdot\mathrm{K}}\right]$

Φ_h^* Densidad de flujo de calor vertical a través de los límites horizontales o paralelos a los isotérmicos

Φ_v^* Densidad de flujo de calor horizontal a través de las superficies verticales o perpendiculares a las isotérmicas

Parámetros adimensionales

A Producción de calor adimensional, $\dfrac{2A^{*}LH}{\text{tr}\,\underline{\underline{\boldsymbol{\Lambda}}}^{*}\Delta T^{*}}$

$\underline{e}_1$ Vector unitario adimensional para el sistema coordenado horizontal

$\underline{e}_3$ Vector unitario adimensional para el sistema coordenado vertical

$\underline{\underline{\mathbf{K}}}$ Tensor de permeabilidad adimensional en *2D*, $\left(\dfrac{2}{\text{tr}\,\underline{\underline{\mathbf{K}}}^{*}}\right)\cdot\underline{\underline{\mathbf{K}}}^{*}$

$\underline{\underline{\mathbf{K}}}$ Tensor de permeabilidad adimensional en el sistema coordenado adimensional (x,z), $\left(\underline{\underline{\alpha}}\cdot\underline{\underline{\mathbf{K}}}\cdot\underline{\underline{\alpha}}\right)$

Ra^{*} Número de Rayleigh modificado para filtración de Darcy en un medio heterogéneo, $Ra^{*}=\dfrac{\rho_r\hat{\mathbf{g}}\left(\rho c_p\right)_f\beta_{th}\Delta T^{*}H\ \text{tr}\,\underline{\underline{\mathbf{K}}}^{*}}{\eta_r\ \text{tr}\,\underline{\underline{\boldsymbol{\Lambda}}}^{*}}$

t Tiempo adimensional, $\dfrac{t^{*}\ \text{tr}\,\underline{\underline{\boldsymbol{\Lambda}}}}{2LH\left(\rho c_p\right)^{*}}$

T Temperatura adimensional, $\dfrac{T^{*}-T_r^{*}}{\Delta T^{*}}$

$\underline{v}$ Velocidad de flujo adimensional, $\underline{v}_{Darcy}-\nabla\left(\ln\text{tr}\,\underline{\underline{\boldsymbol{\Lambda}}}^{*}\right)\cdot\underline{\underline{\boldsymbol{\Lambda}}}$

$\underline{v}_{Darcy}$ Velocidad de filtración adimensional, componentes $\left[v_x,v_y\right]$

v_x Velocidad de filtración adimensional horizontal, $\dfrac{2H\left(\rho c_p\right)_f\mathbf{v}_x}{\text{tr}\,\underline{\underline{\boldsymbol{\Lambda}}}^{*}}$

v_z Velocidad de filtración adimensional horizontal, $\dfrac{2L\left(\rho c_p\right)_f\mathbf{v}_z}{\text{tr}\,\underline{\underline{\boldsymbol{\Lambda}}}^{*}}$

x Coordenada horizontal *x* adimensional, $\dfrac{x^{*}}{L}$

z Coordenada vertical *z* adimensional, $\dfrac{z^{*}}{H}$

α Razón geométrica adimensional, $\frac{H}{L}$

β Coeficiente adimensional de expansión volumétrica del fluido saturante, $\beta_{th}\Delta T^*$

γ Coeficiente adimensional de variación de la viscosidad dinámica del fluido saturante a una temperatura dada, $\gamma^*\Delta T^*$

ε Porosidad adimensional

$\underline{\underline{\Lambda}}$ Tensor de conductividad térmica adimensional en *2D*, $\left(\frac{2}{\text{tr}\ \underline{\underline{\Lambda}}^*}\right)\cdot\underline{\underline{\Lambda}}^*$

$\underline{\underline{\Lambda}}$ Tensor de conductividad térmica adimensional en el sistema coordenado adimensional (x,z), $\left(\underline{\underline{\alpha}}\cdot\underline{\underline{\Lambda}}\cdot\underline{\underline{\alpha}}\right)$

Φ_h Densidad de flujo de calor vertical adimensional a través de los límites horizontales o paralelos a los isotérmicos, $\frac{2H\Phi_h^*}{\text{tr}\ \underline{\underline{\Lambda}}^*\Delta T^*}$

Φ_v Densidad de flujo de calor horizontal adimensional a través de las superficies verticales o perpendiculares a las isotérmicas, $\frac{2L\Phi_v^*}{\text{tr}\ \underline{\underline{\Lambda}}^*\Delta T^*}$

Ψ Función de corriente en condiciones estacionarias

Superíndices

t Operador matricial transpuesta

-1 Operador matricial inversa

Subíndices

f Fluido

h Horizontal o paralelo a los límites isotérmicos

r Nivel de referencia

s Sólido

v Vertical o perpendicular a la superficie isotérmica

x Componente horizontal

z Componente vertical

Operadores

$\underline{\underline{\mathbf{j}}}$ Matriz de rotación $\begin{pmatrix} 0 & -1 \\ 1 & 0 \end{pmatrix}$, se nota que $\underline{\underline{\mathbf{j}}}^2 = -\underline{\underline{\mathbf{I}}}$, donde $\underline{\underline{\mathbf{I}}}$ es el operador de identidad

$\underline{\underline{\alpha}}$ Tensor métrico asociado al sistema coordenada adimensional (x, z), $\begin{pmatrix} \sqrt{\alpha} & 0 \\ 0 & 1/\sqrt{\alpha} \end{pmatrix}$

tr Operador traza

det Determinante

$\llcorner$ Producto escalar

$\otimes$ Producto tensorial

$\{a_i\}$ Sumatoria de términos a_i con respecto a i, $\sum_i a_i$

∇ Operador nabla 2D, $\left(\frac{\partial}{\partial x^*}, \frac{\partial}{\partial z^*}\right) = \mathbf{grad}$

J(f) Jacobiano de la función vectorial $\underline{\mathbf{f}}$

$\mathbf{H}(f)$ Hessiano de la función escalar f

D Operador rotacional 2D, $\left(\frac{\partial}{\partial z^*}, -\frac{\partial}{\partial x^*}\right) = \mathbf{curl}$

Δf Laplaciano de una función escalar f

6 Apéndice III. Fundamentos del Método del elemento finito estándar

Este apéndice se basa principalmente en el libro “Computational methos for multiphase flows in porous media” (Chen, Huan, y Yuanle , 2006), sin embargo en algunos casos se incorporó información del libro “Finite element methods and their applications” (Chen, 2005).

Problema de un modelo en una dimensión

Como introducción, se considera el problema estacionario de una dimensión tomado de Chen, Huan, y Yuanle (2006)

$$\begin{aligned} -\frac{d^2 p}{dx^2} &= f(x) \qquad 0 < x < 1 \\ p(0) &= p(1) = 0 \end{aligned} \tag{III.1}$$

donde f es una función real acotada y continua por partes (“*piecewise*”).

Para poder reescribir la discretización de la ecuación (III.1) en su forma variacional equivalente, se introduce el *producto escalar*

$$(v, w) = \int_0^1 v(x)\, w(x)\, dx \tag{III.2}$$

con v y w funciones reales acotadas y continuas por partes. Además, se define el espacio lineal

$$\begin{aligned} V = \Big\{ & v : v \text{ es una función continua en } [0,1], \\ & \frac{dv}{dx} \text{ es continua por tramos y acotada en } (0,1), \text{ y } v(0) = v(1) = 0 \Big\} \end{aligned} \tag{III.3}$$

Definiendo el funcional $F : V \to \mathbb{R}$

$$F(v) = \frac{1}{2}\left(\frac{dv}{dx}, \frac{dv}{dx}\right) - (f, v) \qquad v \in V \tag{III.4}$$

donde $\mathbb{R}$ es el conjunto de los números reales.

En Chen, Huan, y Yuanle (2006) se puede observar como la búsqueda de p en (III.1) equivale a un *problema de minimización*

$$\text{Encuentre } p \in V \text{ tal que } F(p) \le F(v) \qquad \forall v \in V \tag{III.5}$$

Donde el problema (III.5) representa la forma variacional de Ritz de la ecuación (III.1).

Si se multiplica la primera ecuación de (III.1) por cualquier $v \in V$, llamada *función de prueba*, e integrando sobre el intervalo $(0,1)$

$$-\left(\frac{d^2p}{dx^2}, v\right) = (f, v) \tag{III.6}$$

obteniendo así una forma más útil y directa desde el punto de vista computacional. Aplicando integración por partes a la anterior ecuación se tiene que

$$\left(\frac{dp}{dx}, \frac{dv}{dx}\right) = (f, v) \tag{III.7}$$

donde se ha usado el hecho de que $v(0) = v(1) = 0$ tomado de la definición de V. La ecuación (III.7) es comúnmente llamada *variacional de Galerkin* o *forma débil* de (III.1).

Se tiene entonces que si p es una solución de (III.1), entonces también satisface (III.7). De forma recíproca, si $\frac{d^2p}{dx^2}$ existe, entonces se puede afirmar que es continua por partes y acotada en $(0,1)$. De lo anterior se observa que (III.1) y (III.7) son equivalentes.

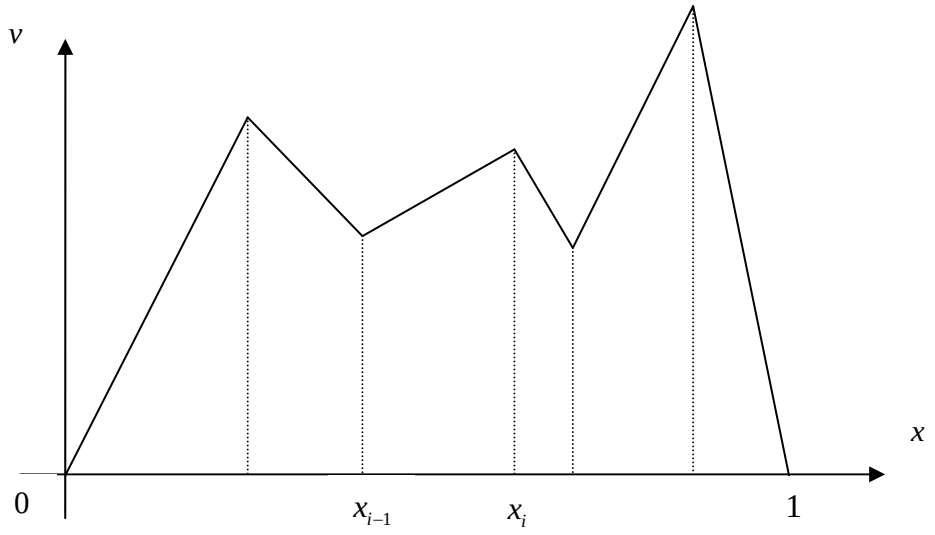


Figura 3. Ilustración de la función $v \in V_h$

Ahora se construye el método del elemento finito para resolver (III.1). Con tal fin, para un M entero positivo, se tiene que $0 = x_0 < x_1 < ... < x_M < x_{M+1} = 1$ es una partición del intervalo $(0,1)$ en un conjunto de subintervalos $I_i = (x_{i-1}, x_i)$ de longitud $h_i = x_i - x_{i-1}$, donde $i = 1, 2, ..., M+1$. Estableciendo $h = \max\{h_i : i = 1, 2, ..., M+1\}$. El *tamaño de la partición* h sirve para determinar cuán fina la partición es.

Se define entonces el espacio del elemento finito

$$V_h = \{v : v \text{ es una función continua en } [0,1], \\ v \text{ es lineal en cada subintervalo } I_i, \text{ y } v(0) = v(1) = 0\} \tag{III.8}$$

En la Figura 3 se observa una ilustración de la función $v \in V_h$. Nótese que $V_h \subset V$, es decir V_h es un subconjunto de V.

La versión discreta de (III.5) es

$$\text{Encuentre } p_h \in V_h \text{ tal que } F(p_h) \le F(v) \qquad \forall v \in V_h \tag{III.9}$$

(III.9) es conocido como el *Método del Elemento Finito de Ritz*. De la misma manera que para (III.7), (III.9) es equivalente al problema

$$\text{Encuentre } p_h \in V_h \text{ tal que } \left(\frac{dp_h}{dx}, \frac{dv}{dx}\right) = (f, v) \qquad \forall v \in V_h \tag{III.10}$$

al cual se le llama comúnmente *Método del Elemento Finito de Galerkin*.

Es fácil observar que (III.10) tiene una solución única. De hecho, si se hace $f = 0$ y $v = p_h$ en (III.10), se tiene que

$$\left(\frac{dp_h}{dx}, \frac{dp_h}{dx}\right) = 0 \tag{III.11}$$

entonces p_h es una constante. Tomando en cuenta las condiciones de frontera en V_h, se concluye que $p_h = 0$.

Si se introducen las funciones base $\varphi_i \in V_h$, $i = 1, 2, \ldots, M$ (comúnmente llamada función sombrero o *función chapeau*)

$$\varphi_i(x_j) = \begin{cases} 1 & i = j \\ 0 & i \neq j \end{cases} \qquad \text{(III.12)}$$

Se observa que φ_i es una función lineal continua por partes en $[0,1]$, tomando el valor unitario en el *nodo* x_i y cero en los otros nodos. En 1D la función chapeau (Figura 4) queda definida como

$$\varphi_i(x) = \begin{cases} \dfrac{x - x_{i-1}}{x_i - x_{i-1}} & x_{i-1} \leq x \leq x_i \\ \dfrac{x_{i+1} - x}{x_{i+1} - x_i} & x_i \leq x \leq x_{i+1} \end{cases} \qquad \text{(III.13)}$$

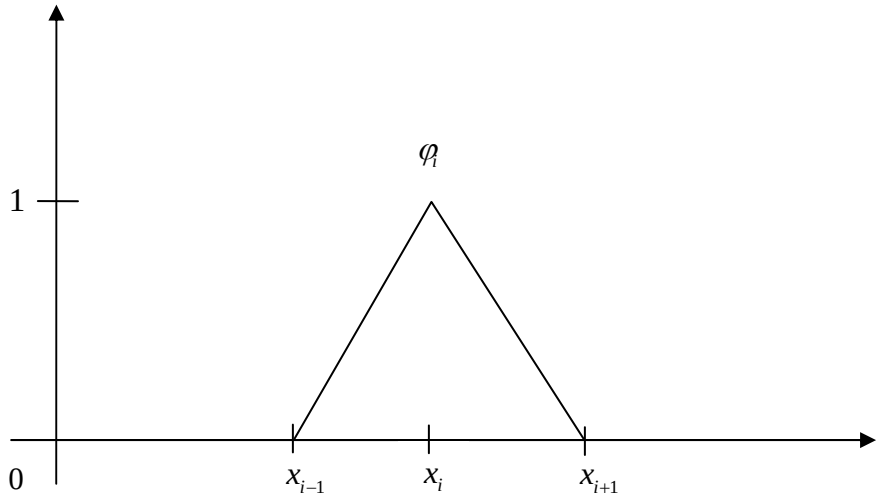


Figura 4. Funciones base en 1D (Función Chapeau)

Cualquier función $v \in V_h$ tiene una única representación

$$v(x) = \sum_{i=1}^{M} v_i \varphi_i(x) \qquad 0 \leq x \leq 1 \qquad \text{(III.14)}$$

dónde $v_i = v(x_i)$. Para cada j, tomando $v = \varphi_j$ en (III.10) se observa que

$$\left(\frac{dp_h}{dx}, \frac{d\varphi_j}{dx} \right) = (f, \varphi_j) \qquad j = 1, 2, \ldots, M \qquad \text{(III.15)}$$

Definiendo

$$p_h(x) = \sum_{i=1}^{M} p_i \varphi_i(x) \qquad p_i = p_h(x_i) \tag{III.16}$$

y sustituyendo la ecuación anterior en (III.15) se obtiene

$$\sum_{i=1}^{M} \left(\frac{dp_h}{dx}, \frac{d\varphi_j}{dx} \right) p_i = \left(f, \varphi_j \right) \qquad j = 1, 2, \ldots, M \tag{III.17}$$

el cual es un sistema lineal de M ecuaciones algebraicas en M incógnitas $p_1, p_2, \ldots p_M$, que puede además escribirse de forma matricial como

$$\underline{\underline{\mathbf{A}}}\,\underline{\mathbf{p}} = \underline{\mathbf{f}} \tag{III.18}$$

donde la matriz $\underline{\underline{\mathbf{A}}}$ y los vectores $\underline{\mathbf{p}}$ y $\underline{\mathbf{f}}$ están dados por

$$\underline{\underline{\mathbf{A}}} = \begin{pmatrix} a_{11} & a_{12} & \cdots & a_{1M} \\ a_{21} & a_{22} & \cdots & a_{2M} \\ \vdots & \vdots & \ddots & \vdots \\ a_{M1} & a_{M1} & \cdots & a_{MM} \end{pmatrix} \qquad p = \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_M \end{pmatrix} \qquad f = \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_M \end{pmatrix}$$

$$a_{ij} = \left(\frac{d\varphi_i}{dx}, \frac{d\varphi_j}{dx} \right) \qquad f_j = \left(f, \varphi_j \right) \qquad i, j = 1, 2, \ldots, M \tag{III.19}$$

Donde $\underline{\underline{\mathbf{A}}}$ se le conoce como *matriz de rigidez,* y $\underline{\mathbf{f}}$ como *vector fuente*.

De las funciones que componen la base se tiene que

$$\left(\frac{d\varphi_i}{dx}, \frac{d\varphi_j}{dx} \right) = 0 \qquad \text{si } |i - j| \geq 2 \tag{III.20}$$

entonces $\underline{\underline{\mathbf{A}}}$ es *tridiagonal*; es decir, solamente las entradas en la diagonal principal y en ambas diagonales adyacentes son diferentes de cero. De hecho, cada entrada a_{ij} se puede calcular de la siguiente manera

$$a_{ii} = \frac{1}{h_i} + \frac{1}{h_{i+1}} \qquad a_{i-1,i} = -\frac{1}{h_i} \qquad a_{i,i+1} = -\frac{1}{h_{i+1}} \tag{III.21}$$

Se observa también que $\underline{\underline{\mathbf{A}}}$ es simétrica y positiva definida,

$$\underline{\boldsymbol{\eta}}^{\mathrm{T}}\,\underline{\underline{\mathbf{A}}}\,\underline{\boldsymbol{\eta}} = \sum_{i,j=1}^{M} \eta_i a_{ij} \eta_j > 0 \qquad \text{para todo } \underline{\boldsymbol{\eta}} \in \mathbb{R}^M \text{ diferente de cero} \qquad \text{(III.22)}$$

donde $\underline{\boldsymbol{\eta}}^{\mathrm{T}}$ denota la transpuesta de $\underline{\boldsymbol{\eta}}$. Debido a que una matriz positiva definida es no singular, el sistema lineal (III.18) tiene una solución única. Consecuentemente, esto implica que (III.10) tiene una solución única $p_h \in V_h$.

La simetría de $\underline{\underline{\mathbf{A}}}$ se observa de la definición de a_{ij}. El hecho de que sea positiva definida se puede comprobar de la siguiente manera: con

$$\underline{\boldsymbol{\eta}} = \sum_{i=1}^{M} \eta_i \varphi_i \in V_h \qquad \underline{\boldsymbol{\eta}}^{\mathrm{T}} = (\eta_1, \eta_2, \ldots, \eta_M) \qquad \text{(III.23)}$$

se observa que

$$\sum_{i,j=1}^{M} \eta_i a_{ij} \eta_j = \sum_{i,j=1}^{M} \eta_i \left(\frac{d\varphi_i}{dx}, \frac{d\varphi_j}{dx} \right) \eta_j = \left(\sum_{i=1}^{M} \eta_i \frac{d\varphi_i}{dx}, \sum_{j=1}^{M} \eta_j \frac{d\varphi_j}{dx} \right) = \left(\frac{d\eta}{dx}, \frac{d\eta}{dx} \right) \geq 0 \qquad \text{(III.24)}$$

En lo que respecta a (III.10), la igualdad se cumple solamente para $\underline{\boldsymbol{\eta}} \equiv 0$ puesto que la función constante $\underline{\boldsymbol{\eta}}$ debe de ser cero tomando en cuenta las condiciones de frontera.

Es importante señalar que $\underline{\underline{\mathbf{A}}}$ es *poco densa* o *rala* (“sparse”); es decir, solamente algunas entradas de cada renglón de $\underline{\underline{\mathbf{A}}}$ son distintas de cero (en 1D la matriz es tridiagonal). Esto se debe a que la base V_h es diferente de cero solamente en algunos intervalos. La elección de bases con esta naturaleza representa una de las ventajas del método del elemento finito.

En el caso particular de una partición del dominio unidimensional uniforme $h = h_i, \quad i = 1, 2, \ldots, M+1$, la matriz de rigidez $\underline{\underline{\mathbf{A}}}$ toma la siguiente forma

$$\underline{\underline{\mathbf{A}}} = \frac{1}{h} \begin{pmatrix} 2 & -1 & 0 & \cdots & 0 & 0 \\ -1 & 2 & -1 & \cdots & 0 & 0 \\ 0 & -1 & 2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 2 & -1 \\ 0 & 0 & 0 & \cdots & -1 & 2 \end{pmatrix} \qquad \text{(III.25)}$$

Con la división entre h en $\underline{\underline{\mathbf{A}}}$, (III.10) se puede ver el FEM como una variante del *esquema en diferencias centrales*, mientras que el lado derecho consiste en lo valores medios de $f\varphi_j$ sobre el intervalo $\left(x_{j-1}, x_{j+1}\right)$.

En general, el cálculo del *error estimado* para los métodos del elemento finito es teórico y queda fuera del alcance del presente trabajo, pero si se desea, se puede consultar en Chen, Huan, y Yuanle (2006), en donde también se encuentra una demostración de la convergencia de este método en particular.

Problema de un modelo en dos dimensiones

Considerando ahora el problema estacionario en dos dimensiones:

$$\begin{aligned} -\Delta p &= f \qquad \text{en } \Omega \\ p &= 0 \qquad \text{en } \Gamma \end{aligned} \tag{III.26}$$

donde Ω es un dominio acotado en un plano con frontera Γ y $f \in \Omega$, es una función real acotada continua por partes. Además se recuerda que Δ en este contexto se toma como el *Operador Laplaciano* definido por

$$\Delta p = \frac{\partial^2 p}{\partial x_1^2} + \frac{\partial^2 p}{\partial x_2^2} \tag{III.27}$$

Se define el espacio lineal

$$\begin{aligned} V = \{ v : & \, v \text{ es una función continua en } \Omega, \\ & \frac{\partial v}{\partial x_1} \text{ y } \frac{\partial v}{\partial x_2} \text{son continuas por tramos y acotadas en } \Omega, \text{ y } v = 0 \text{ en } \Gamma \} \end{aligned} \tag{III.28}$$

A continuación se presenta la *fórmula de Green*. Para una función vectorial $\underline{\mathbf{b}} = \left(b_1, b_2\right)$, el *teorema de la divergencia* indica que

$$\int_\Omega \nabla \cdot \underline{\mathbf{b}} \, d\underline{x} = \int_\Gamma \underline{\mathbf{b}} \cdot \underline{\hat{\mathbf{n}}} \, ds \tag{III.29}$$

donde el operador de la divergencia, en este caso particular 2D, esta dado por

$$\nabla \cdot \underline{\mathbf{b}} = \frac{\partial b_1}{\partial x_1} + \frac{\partial b_2}{\partial x_2} \tag{III.30}$$

$\hat{\underline{\mathbf{n}}}$ es el vector unitario normal a Γ apuntando hacia afuera del dominio, y el producto punto $\underline{\mathbf{b}} \cdot \hat{\underline{\mathbf{n}}}$ es

$$\underline{\mathbf{b}} \cdot \hat{\underline{\mathbf{n}}} = b_1 \hat{n}_1 + b_2 \hat{n}_2 \tag{III.31}$$

Con $v, w \in V$, se sustituye $\underline{\mathbf{b}} = \left(\frac{\partial v}{\partial x_1} w, 0 \right)$ y $\underline{\mathbf{b}} = \left(0, \frac{\partial v}{\partial x_2} w \right)$ en (III.29), respectivamente, entonces se observa que

$$\int_\Omega \frac{\partial^2 v}{\partial x_i^2} w \, d\underline{x} + \int_\Omega \frac{\partial v}{\partial x_i} \frac{\partial w}{\partial x_i} \, d\underline{x} = \int_\Gamma \frac{\partial v}{\partial x_i} w v_i \, ds \qquad i = 1, 2 \tag{III.32}$$

Usando la definición del *operador gradiente*

$$\nabla v = \left(\frac{\partial v}{\partial x_1}, \frac{\partial v}{\partial x_2} \right) \tag{III.33}$$

y sumando sobre $i = 1, 2$ en (III.32) se obtiene

$$\int_\Omega \Delta v \; w \, d\underline{\mathbf{x}} = \int_\Gamma \frac{\partial v}{\partial \hat{\underline{\mathbf{n}}}} \, w \, ds - \int_\Omega \nabla v \, \nabla w \, d\underline{\mathbf{x}} \tag{III.34}$$

donde la *derivada normal* es la derivada direccional

$$\frac{\partial v}{\partial \hat{\underline{\mathbf{n}}}} = \nabla v \cdot \hat{\underline{\mathbf{n}}} = \frac{\partial v}{\partial x_1} \hat{n}_1 + \frac{\partial v}{\partial x_2} \hat{n}_2 \tag{III.35}$$

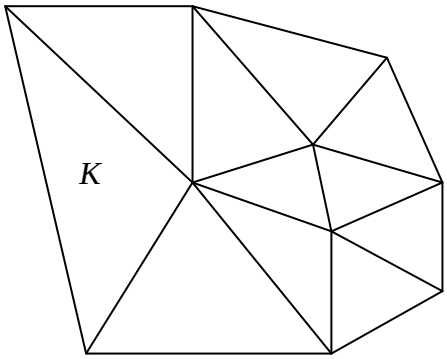
La relación (III.34) es la *fórmula de Green*, y también puede aplicarse en tres dimensiones.

Introduciendo la notación

$$\begin{aligned} a(p, v) &= \int_\Omega \nabla p \cdot \nabla v \, d\underline{\mathbf{x}} \\ (f, v) &= \int_\Omega f v \, d\underline{\mathbf{x}} \end{aligned} \tag{III.36}$$

La forma $a(\cdot, \cdot)$ es la *forma bilineal* en $V \times V$; esto es

$$a(u,\alpha v+\beta w)=\alpha a(u,v)+\beta a(u,w)$$
$$a(\alpha u+\beta v,w)=\alpha a(u,w)+\beta a(v,w) \qquad \text{(III.37)}$$



para $\alpha,\beta\in\mathbb{R}$ y $u,v,w\in V$. También se define el funcional $F:V\to\mathbb{R}$ como

$$F(v)=\frac{1}{2}a(v,v)-(f,v) \qquad v\in V \qquad \text{(III.38)}$$

Tal como se hizo en el problema en una dimensión, (III.26) puede formularse como un problema de minimización

$$\text{Encuentre } p\in V \text{ tal que } F(p)\le F(v) \qquad \forall v\in V \qquad \text{(III.39)}$$

Este problema es también equivalente al problema variacional (III.42), cuya demostración es similar a la de los problemas (III.5) y (III.7) (Chen, Huan, y Yuanle, 2006).

Multiplicando la primera ecuación de (III.26) por $v\in V$ e integrando sobre Ω, se observa que

$$-\int_\Omega \nabla p\, v\, d\underline{\mathbf{x}}=\int_\Omega f\, v\, d\underline{\mathbf{x}} \qquad \text{(III.40)}$$

Aplicando (III.34) a la ecuación anterior y usando las condiciones de frontera homogéneas, se tiene que

$$\int_\Omega \nabla p\,\nabla v\, d\underline{\mathbf{x}}=\int_\Omega f\, v\, d\underline{\mathbf{x}} \qquad \forall v\in V \qquad \text{(III.41)}$$

Figura 5. Una partición del elemento finito en dos dimensiones

Entonces, utilizando la formulación variacional, se tiene que el problema se convierte en:

$$\text{Encuentre } p \in V \text{ tal que } a(p,v) = (f,v) \qquad \forall v \in V \tag{III.42}$$

Para plantear el FEM que resuelve la ecuación (III.26) se asume que Ω es un dominio poligonal (el tratamiento para un dominio curvo puede observarse en Chen, 2005).

Haciendo que K_h sea una partición, llamada *triangulación*, de Ω en triángulos K_i no superpuestos (abiertos):

$$\bar{\Omega} = \bar{K}_1 \cup \bar{K}_2 \cup \cdots \cup \bar{K}_{\bar{M}} \tag{III.43}$$

tal que ningún vértice de un triángulo cae dentro del interior del borde de otro triángulo (Figura 5), donde $\bar{\Omega}$ representa la cerradura de Ω (es decir $\bar{\Omega} = \Omega \cup \Gamma$).

Para los triángulos $K \in K_h$, se definen los siguientes *parámetros de malla*

$$\begin{aligned} &\text{diam}\,(K) = \text{ el borde más largo de } \bar{K} \\ &h = \max_{K \in K_h} \text{diam}(K) \end{aligned} \tag{III.44}$$

Ahora, se introduce el espacio del elemento finito

$$\begin{aligned} V_h = \{ &v : v \text{ es una función continua en } \Omega, \\ &v \text{ es lineal en cada triángulo } K \in K_h, \text{ y } v = 0 \text{ en } \Gamma \} \end{aligned} \tag{III.45}$$

Cabe notar que $V_h \subset V$. El método del elemento finito para (III.26) se formula como

$$\text{Encuentre } p_h \in V_h \text{ tal que } a(p_h,v) = (f,v) \qquad \forall v \in V_h \tag{III.46}$$

La existencia y unicidad de la solución de (III.46) se obtiene de la misma manera que para (III.10) (Chen, Huan, y Yuanle, 2006). También, de la misma manera en que (III.5) y (III.7) son equivalentes, se puede observar que (III.46) es equivalente a un problema de minimización discreto:

$$\text{Encuentre } p_h \in V_h \text{ tal que } F(p_h) \le F(v) \qquad \forall v \in V_h \tag{III.47}$$

Se nombran los vértices (*nodos*) de los triángulos en K_h cómo $\underline{\mathbf{x}}_1, \underline{\mathbf{x}}_2, \ldots, \underline{\mathbf{x}}_{\tilde{M}}$. Las funciones base φ_i en V_h, $i = 1, 2, \ldots, \tilde{M}$, están definidas por

$$\varphi_i\left(x_j\right) = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{si } i \neq j \end{cases} \tag{III.48}$$

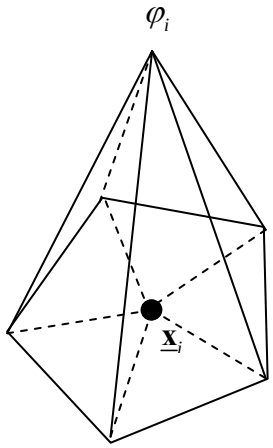


Figura 6. Función base en dos dimensiones

El soporte de φ_i, por el ejemplo el conjunto de $\underline{\mathbf{x}}$ donde $\varphi_i\left(\underline{\mathbf{x}}\right) \neq 0$, está formado por los triángulos con el nodo común $\underline{\mathbf{x}}_i$ (Figura 6). La función φ_i representa la función sombrero o *función chapeau* bidimensional. Se define entonces M que corresponde al número de vértices interiores en K_h, donde los primeros M vértices sean los interiores. Se tiene entonces que cualquier función $v \in V_h$ tiene una única representación

$$v\left(\underline{\mathbf{x}}\right) = \sum_{i=1}^{M} v_i \varphi_i\left(\underline{\mathbf{x}}\right) \qquad \underline{\mathbf{x}} \in \Omega \tag{III.49}$$

dónde $v_i = v\left(\underline{\mathbf{x}}_i\right)$. Debido a que en este ejemplo particular se tienen condiciones de frontera Dirichlet homogéneas, se pueden excluir los vértices de la frontera de Ω.

De la misma manera que en (III.10), la ecuación (III.46) puede escribirse en forma matricial

$$\underline{\underline{\mathbf{A}}}\,\underline{\mathbf{p}} = \underline{\mathbf{f}} \tag{III.50}$$

donde, igual que anteriormente, la matriz $\underline{\underline{\mathbf{A}}}$ y los vectores $\underline{\mathbf{p}}$ y $\underline{\mathbf{f}}$ son

$$\underline{\underline{\mathbf{A}}} = \left(a_{ij}\right) \qquad \underline{\mathbf{p}} = \left(p_j\right) \qquad \underline{\mathbf{f}} = \left(f_j\right)$$

con (III.51)

$$a_{ij} = a\left(\varphi_i, \varphi_j\right) \qquad f_i = \left(f, \varphi_j\right) \qquad i, j = 1, 2, \ldots M$$

En la implementación computacional, las entradas a_{ij} en $\underline{\underline{\mathbf{A}}}$ se obtienen sumando las contribuciones de los diferentes triángulos $K \in K_h$

$$a_{ij} = a\left(\varphi_i, \varphi_j\right) = \sum_{K \in K_h} a^K\left(\varphi_i, \varphi_j\right)$$

donde (III.52)

$$a_{ij}^K \equiv a^K\left(\varphi_i, \varphi_j\right) = \int_K \nabla\varphi_i \cdot \nabla\varphi_j \, d\underline{\mathbf{x}}$$

Al igual que en el caso unidimensional, puede verificarse que la matriz de rigidez $\underline{\underline{\mathbf{A}}}$ es simétrica positiva definida. De forma particular, es no singular. Consecuentemente, (III.50) y por consiguiente (III.46) tienen una solución única.

Como se mencionó anteriormente, la estimación del error es algo delicada. Algunas aproximaciones de éste pueden revisarse en (Chen, 2005) y (Chen, Huan, y Yuanle, 2006).

7 Apéndice IV. Métodos de solución de sistemas de ecuaciones lineales

En este apéndice se hace una breve descripción de los métodos de solución de sistemas de ecuaciones lineales más comunes.

Métodos Directos

En estos métodos, la solución $\underline{\mathbf{u}}$ se obtiene en un número fijo de pasos y sólo están sujetos a los errores de redondeo. Entre los métodos más importantes podemos encontrar: eliminación Gaussiana, descomposición LU, eliminación bandada y descomposición de Cholesky.

Los métodos que se han mencionado se colocaron en orden descendente en cuanto al consumo de recursos computacionales, y ascendente en cuanto al aumento en su eficiencia (Carrillo Ledesma, 2006). A continuación se hace una breve descripción de éstos.

Eliminación Gaussiana

Tal vez es el método más utilizado para encontrar la solución usando métodos directos. Este algoritmo, sin embargo, no es muy eficiente, ya que en general un sistema de N ecuaciones requiere para su almacenaje en memoria de N^2 entradas para la matriz $\underline{\underline{\mathbf{A}}}$, pero cerca de $\frac{N^3}{3}+O\left(N^2\right)$ multiplicaciones y $\frac{N^3}{3}+O\left(N^2\right)$ adiciones para encontrar la solución siendo muy costoso computacionalmente.

La eliminación Gaussiana se basa en la aplicación de operaciones elementales a renglones o columnas de tal forma que es posible obtener matrices equivalentes.

Escribiendo el sistema de N ecuaciones lineales con N incógnita como

$$\sum_{j=1}^{N} a_{ij}^{(0)} x_j = a_{i,n+1}^{(0)}, \qquad i=1,2,\ldots,N \tag{IV.1}$$

y si $a_{ij}^{(0)} \neq 0$ y los pivotes $a_{ij}^{(i-1)}, i=2,3,\ldots,N$ de las demás filas, que se obtienen en el curso de los cálculos, son distintos de cero, entonces, el sistema lineal anterior se reduce a la forma triangular superior (eliminación hacia adelante)

$$x_i + \sum_{j=1+1}^{N} a_{ij}^{(i)} x_j = a_{i,n+1}^{(i)}, \qquad i=1,2,\ldots,N \tag{IV.2}$$

donde

$$\begin{aligned}
&k=1,2,\ldots,N; \quad \{j=k+1,\ldots.N\{ \\
&a_{kj}^{(k)} = \frac{a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}}; \\
&i=k+1,\ldots,N+1\{ \\
&a_{ij}^{(k)} = a_{ij}^{(k-1)} - a_{kj}^{(k)} a_{ik}^{(k-1)} \}\}\}
\end{aligned}$$

y las incógnitas se calculan por sustitución hacia atrás, usando las fórmulas

$$\begin{aligned}
&x_N = a_{N,N+1}^{(N)}; \\
&i=N-1,N-2,\ldots,1 \\
&x_i = a_{i,N+1}^{(i)} - \sum_{j=i+1}^{N} a_{ij}^{(i)} x_j
\end{aligned} \tag{IV.3}$$

En algunos casos interesa conocer $\underline{\underline{\mathbf{A}}}^{-1}$, por ello si la eliminación se aplica a la matriz aumentada $\underline{\underline{\mathbf{A}}} \mid \underline{\underline{\mathbf{I}}}$ entonces la matriz $\underline{\underline{\mathbf{A}}}$ de la matriz aumentada se convierte en la matriz $\underline{\underline{\mathbf{I}}}$ y la matriz $\underline{\underline{\mathbf{I}}}$ de la matriz aumentada será $\underline{\underline{\mathbf{A}}}^{-1}$. Así el sistema $\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}}$ se transformará en $\underline{\mathbf{u}} = \underline{\underline{\mathbf{A}}}^{-1} \cdot \underline{\mathbf{b}}$ obteniendo la solución de $\underline{\mathbf{u}}$.

Descomposición LU

Sea $\underline{\underline{\mathbf{U}}}$ una matriz triangular superior obtenida de $\underline{\underline{\mathbf{A}}}$ por eliminación bandada. Entonces $\underline{\underline{\mathbf{U}}} = \underline{\underline{\mathbf{L}}}^{-1} \underline{\underline{\mathbf{A}}}$, donde $\underline{\underline{\mathbf{L}}}$ es una matriz triangular inferior con unos en la diagonal. Las entradas de $\underline{\underline{\mathbf{L}}}^{-1}$ pueden obtenerse de los coeficientes m_{ij} definidos en el método anterior y pueden ser almacenados estrictamente en las entradas de la diagonal inferior de $\underline{\underline{\mathbf{A}}}$ ya que estas ya fueron eliminadas. Esto proporciona una factorización $\underline{\underline{\mathbf{L}}}\underline{\underline{\mathbf{U}}}$ de $\underline{\underline{\mathbf{A}}}$ en la misma matriz $\underline{\underline{\mathbf{A}}}$ ahorrando espacio de memoria.

El problema original $\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}}$ se escribe como $\underline{\underline{\mathbf{L}}}\underline{\underline{\mathbf{U}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}}$ y se reduce a la solución sucesiva de los sistemas lineales triangulares

$$\underline{\underline{\mathbf{L}}} \cdot \underline{\mathbf{y}} = \underline{\mathbf{b}} \qquad y \qquad \underline{\underline{\mathbf{U}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{y}} \tag{IV.4}$$

La descomposición $\underline{\underline{\mathbf{L}}}\underline{\underline{\mathbf{U}}}$ requiere también $\frac{N^3}{3}$ operaciones aritméticas para la matriz llena, pero sólo Nb^2 operaciones aritméticas para la matriz con un ancho de banda de b siendo esto más económico computacionalmente.

Se nota que para una matriz no singular $\underline{\underline{\mathbf{A}}}$, la eliminación Gaussiana (sin redondear filas y columnas) es equivalente a la factorización $\underline{\underline{\mathbf{L}}}\underline{\underline{\mathbf{U}}}$.

Eliminación Bandada

Cuando se usa la ordenación natural de los nodos, la matriz $\underline{\underline{\mathbf{A}}}$ que se genera es bandada, por ello se puede ahorrar considerable espacio de almacenamiento en ella. Este algoritmo consiste en triangular a la matriz $\underline{\underline{\mathbf{A}}}$ por eliminación hacia adelante operando sólo sobre las entradas dentro de la banda central no cero. Así el renglón j es multiplicado por $m_{ij} = \frac{a_{ij}}{a_{jj}}$ y el resultado es restado al renglón i para $i = j+1, j+2, \ldots$

El resultado es una matriz triangular superior $\underline{\underline{\mathbf{U}}}$ que tiene ceros debajo de la diagonal en cada columna. Así, es posible resolver el sistema resultante al sustituir en forma inversa las incógnitas.

Descomposición de Cholesky

Cuando la matriz es simétrica y definida positiva, se obtiene la descomposición $\underline{\underline{\mathbf{L}}}\underline{\underline{\mathbf{U}}}$ de la matriz $\underline{\underline{\mathbf{A}}}$, así $\underline{\underline{\mathbf{A}}} = \underline{\underline{\mathbf{L}}}\underline{\underline{\mathbf{D}}}\underline{\underline{\mathbf{U}}} = \underline{\underline{\mathbf{L}}}\underline{\underline{\mathbf{D}}}\underline{\underline{\mathbf{L}}}^T$ donde $\underline{\underline{\mathbf{D}}} = diag\left(\underline{\underline{\mathbf{U}}}\right)$ es la diagonal con entradas positivas. La mayor ventaja de esta descomposición es que, en el caso en que es aplicable, el costo de cómputo es sustancialmente reducido, ya que requiere de $\frac{N^3}{6}$ multiplicaciones y $\frac{N^3}{6}$ adiciones.

Métodos Iterativos

En estos métodos se realizan iteraciones para aproximarse a la solución $\underline{\mathbf{u}}$ aprovechando las características propias de la matriz $\underline{\underline{\mathbf{A}}}$, tratando de usar un menor número de pasos que en un método directo, para mayor información de estos y otros métodos se pueden revisar Allen, Herrera, & Pinder (1988) y Saad (2000). También existen libros de texto en métodos numéricos que contienen amplias explicaciones de estos métodos, como los son Nakamura (1995); Press, Teukolsky, Vetterlin, y Flannery (2007)

Un método iterativo en el cual se resuelve el sistema lineal

$$\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}} \tag{IV.5}$$

comienza con una aproximación inicial $\underline{\mathbf{u}}^0$ a la solución $\underline{\mathbf{u}}$ y genera una sucesión de vectores $\left\{\mathbf{u}^k\right\}_{k=1}^{\infty}$ que converge a $\underline{\mathbf{u}}$. Los métodos iterativos traen consigo un proceso que convierte el sistema $\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}}$ en otro equivalente de la forma $\underline{\mathbf{u}} = \underline{\underline{\mathbf{T}}} \cdot \underline{\mathbf{u}} + \underline{\mathbf{c}}$ para alguna matriz fija $\underline{\underline{\mathbf{T}}}$ y un vector $\underline{\mathbf{c}}$. Luego de seleccionar el vector inicial $\underline{\mathbf{u}}^0$ la sucesión de los vectores de la solución aproximada se genera calculando

$$\underline{\mathbf{u}}^k = \underline{\underline{\mathbf{T}}} \cdot \underline{\mathbf{u}}^{k-1} + \underline{\mathbf{c}} \qquad \forall k = 1, 2, 3, \ldots \tag{IV.6}$$

La convergencia a la solución la garantiza el siguiente teorema, cuya solución puede revisarse en (Skiba, 2005)

Teorema *Si* $\left\|\underline{\underline{\mathbf{T}}}\right\|<1$, *entonces el sistema lineal* $\underline{\mathbf{u}}=\underline{\underline{\mathbf{T}}}\cdot\underline{\mathbf{u}}+\underline{\mathbf{c}}$ *tiene una solución única* $\underline{\mathbf{u}}^{*}$ *y las iteraciones* $\underline{\mathbf{u}}^{k}$ *definidas por la fórmula* $\underline{\mathbf{u}}^{k}=\underline{\underline{\mathbf{T}}}\cdot\underline{\mathbf{u}}^{k-1}+\underline{\mathbf{c}}\quad\forall k=1,2,3,\ldots$ *convergen hacia la solución exacta* $\underline{\mathbf{u}}^{*}$ *para cualquier aproximación lineal* $\underline{\mathbf{u}}^{0}$.

Se observa que mientras menor sea la norma de la matriz $\underline{\underline{\mathbf{T}}}$, más rápida es la convergencia, en el caso cuando $\left\|\underline{\underline{\mathbf{T}}}\right\|$ es menor que uno, pero cercano a uno, la convergencia es muy lenta y el número de iteraciones necesario para disminuir el error depende significativamente del error inicial. En este caso, es deseable proponer al vector inicial $\underline{\mathbf{u}}^{0}$ de forma tal que sea mínimo el error inicial. Sin embargo, la elección de dicho vector no tiene importancia si la $\left\|\underline{\underline{\mathbf{T}}}\right\|$ es pequeña ya que la convergencia es rápida.

La velocidad de convergencia de los métodos iterativos depende de las propiedades espectrales de la matriz de coeficientes del sistema de ecuaciones, cuando el operador diferencial L de la ecuación del problema a resolver es auto-adjunto, se obtiene una matriz simétrica y positiva definida, y el número de condicionamiento de la matriz $\underline{\underline{\mathbf{A}}}$, es por definición (Carrillo Ledesma, 2006)

$$\mathrm{cond}\left(\underline{\underline{\mathbf{A}}}\right)=\frac{\lambda_{\max}}{\lambda_{\min}}\geq 1 \qquad \text{(IV.7)}$$

donde $\lambda_{\max}$ y $\lambda_{\min}$ son el máximo y mínimo de los eigenvalores de la matriz $\underline{\underline{\mathbf{A}}}$ respectivamente. Si el número de condicionamiento es cercano a 1, los métodos numéricos a solucionar el problema convergerán en pocas iteraciones; en caso contrario requerirán muchas iteraciones. Frecuentemente al usar el método del elemento finito, se tiene una velocidad de convergencia de $O\left(\frac{1}{h^{2}}\right)$, donde *h* es la máxima distancia de separación entre nodos continuos de la partición, es decir, posee una pobre velocidad de convergencia cuando $h\to 0$ (Carrillo Ledesma, 2006).

Entre los métodos iterativos más usados se pueden encontrar los de: Jacobi, Gauss-Seidel, Richardson, relajación sucesiva (SOR), gradiente conjugado (CG) y gradiente conjugado precondicionado.

Los métodos antes mencionados se colocaron en orden descendente en cuando al consumo de recursos computacionales y ascendente en cuanto al aumento en la eficiencia en su desempeño, describiéndose a continuación los cuatro primeros (el resto pueden revisarse en Carrillo Ledesma, 2006).

Jacobi

Si todos los elementos de la diagonal principal de la matriz $\underline{\underline{\mathbf{A}}}$ son diferentes de cero $a_{ii} \neq 0$ para $i = 1, 2, \ldots, n$, se puede dividir la *i*-ésima ecuación del sistema lineal (IV.5) por a_{ii} para $i = 1, 2, \ldots, n$ y después se trasladan todas las incógnitas, excepto x_i a la derecha, se obtiene el sistema equivalente

$$\underline{\mathbf{u}} = \underline{\underline{\mathbf{B}}} \cdot \underline{\mathbf{u}} + \underline{\mathbf{d}}$$

$$\text{donde} \qquad \mathrm{d}_i = \frac{b_i}{a_{ii}} \quad y \quad \underline{\underline{\mathbf{B}}} = \left\{ b_{ij} \right\} = \begin{cases} -\dfrac{a_{ij}}{a_{ii}} & si\ j \neq i \\ 0 & si\ j = 1 \end{cases} \tag{IV.8}$$

Las iteraciones del método del Jacobi están definidas por la fórmula

$$x_i = \sum_{j=1}^{n} b_{ij} x_j^{(k-1)} + d_i \tag{IV.9}$$

donde $x_i^{(0)}$ son arbitrarias $\left(i = 1, 2, \ldots, n; k = 1, 2, \ldots, n \right)$.

También el método de Jacobi se puede expresar en términos de matrices. Si se supone por un momento que la matriz $\underline{\underline{\mathbf{A}}}$ tiene la diagonal unitaria, esto es $diag\left(\underline{\underline{\mathbf{A}}}\right) = \underline{\underline{\mathbf{I}}}$. Si se descompone $\underline{\underline{\mathbf{A}}} = \underline{\underline{\mathbf{I}}} - \underline{\underline{\mathbf{B}}}$, entonces el sistema dado por la ecuación (IV.5) se puede escribir como

$$\left(\underline{\underline{\mathbf{I}}} - \underline{\underline{\mathbf{B}}} \right) \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}} \tag{IV.10}$$

Para la primera iteración se asume que $\underline{\mathbf{k}} = \underline{\mathbf{b}}$; entonces la última ecuación se escribe como $\underline{\mathbf{u}} = \underline{\underline{\mathbf{B}}} \cdot \underline{\mathbf{u}} + \underline{\mathbf{k}}$. Si se toma una aproximación inicial $\underline{\mathbf{u}}^0$, se puede obtener una mejor aproximación si se reemplaza $\underline{\mathbf{u}}$ por la más reciente aproximación de ese $\underline{\mathbf{u}}^m$. Ésta es la idea que subyace en el método de Jacobi. El proceso iterativo queda entonces como

$$\underline{\mathbf{u}}^{m+1} = \underline{\underline{\mathbf{B}}} \cdot \underline{\mathbf{u}}^m + \underline{\mathbf{k}} \tag{IV.11}$$

La aplicación del método a la ecuación de la forma $\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}}$, con la matriz $\underline{\underline{\mathbf{A}}}$ distinta de cero en los elementos de la diagonal, se obtiene multiplicando la ecuación (IV.5) por $\underline{\underline{\mathbf{D}}}^{-1} = \left[diag\left(\underline{\underline{\mathbf{A}}}\right) \right]^{-1}$, con lo cual se obtiene finalmente

$$\underline{\underline{\mathbf{B}}} = \underline{\underline{\mathbf{I}}} - \underline{\underline{\mathbf{D}}}^{-1}\,\underline{\underline{\mathbf{A}}}, \qquad \underline{\mathbf{k}} = \underline{\underline{\mathbf{D}}}^{-1} \cdot \underline{\mathbf{b}} \tag{IV.12}$$

Gauss-Seidel

Este método es una modificación del método Jacobi, en el cual, una vez obtenido algún valor de $\underline{\mathbf{u}}^{m+1}$, éste es usado para obtener el resto de los valores utilizando los valores más actualizados de $\underline{\mathbf{u}}^{m+1}$. Así, la ecuación (IV.11) se puede escribir como

$$u_i^{m+1} = \sum_{j<i} b_{ij} u_j^{m+1} + \sum_{j>i} b_{ij} u_j^m + k_i \tag{IV.13}$$

Cabe destacar que el método de Gauss-Seidel requiere el mismo número de operaciones aritméticas por iteración que el método de Jacobi. Este método se escribe en forma matricial como

$$\underline{\mathbf{u}}^{m+1} = \underline{\underline{\mathbf{E}}} \cdot \underline{\mathbf{u}}^{m+1} + \underline{\underline{\mathbf{F}}} \cdot \underline{\mathbf{u}}^{m} + \underline{\mathbf{k}} \tag{IV.14}$$

donde $\underline{\underline{\mathbf{E}}}$ y $\underline{\underline{\mathbf{F}}}$ son las matrices triangular superior y triangular inferior, respectivamente. Éste método mejora la convergencia con respecto al método de Jacobi en un factor aproximado de 2.

Richardson

Se escribe el método de Jacobi como

$$\underline{\mathbf{u}}^{m+1} - \underline{\mathbf{u}}^{m} = \underline{\mathbf{b}} - \underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}}^{m} \tag{IV.15}$$

entonces el método Richardson se genera al incorporar una estrategia de sobrerrelajación de la forma siguiente

$$\underline{\mathbf{u}}^{m+1} = \underline{\mathbf{u}}^{m} + \omega\left(\underline{\mathbf{b}} - \underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}}^{m}\right) \tag{IV.16}$$

donde ω es el coeficiente de relajación. Finalmente, el método de Richardson se define como

$$\underline{\mathbf{u}}^{m+1} = \left(\underline{\underline{\mathbf{I}}} - \omega\underline{\underline{\mathbf{A}}}\right) \cdot \underline{\mathbf{u}}^{m} + \omega\underline{\mathbf{b}} \tag{IV.17}$$

En la práctica, encontrar el valor de ω puede resultar muy costoso computacionalmente y las diversas estrategias para encontrar ω dependen de las características propias del problema. Sin embargo, este método usando un valor óptimo de ω resulta mejor que el método de Gauss-Seidel.

Relajación Sucesiva (SOR)

Si se parte del método de Gauss-Seidel y sobrerrelajando este esquema, se obtiene

$$u_i^{m+1} = (1-\omega)u_i^m + \omega\left[\sum_{j=1}^{i-1} b_{ij}u_j^{m+1} + \sum_{j=i+1}^{N} b_{ij}u_j^m + k_i\right] \qquad \text{(IV.18)}$$

y cuando la matriz $\underline{\underline{\mathbf{A}}}$ es simétrica con entradas en la diagonal positivas, éste método converge si y solo si $\underline{\underline{\mathbf{A}}}$ es definida positiva y $\omega \in (0,2)$. En la práctica encontrar el valor de ω puede resultar muy costoso computacionalmente y las diversas estrategias para encontrar ω dependen de las características propias del problema (más información puede encontrarse en (Saad, 2000)). Sin embargo, comúnmente se usan valores menores que 1 cuando se quiere acelerar la convergencia de un problema que converge, pero que lo hace de manera muy lenta. Por el contrario, se usan valores mayores que 1 si se pretende asegurar la convergencia de un sistema que tiende a la divergencia. Por último cabe mencionar que si el valor del coeficiente de relajación es igual a 1, el método se vuelve igual al de Gauss-Seidel.

8 Apéndice V. Parámetros físicos del medio y del fluido

- **Porosidad**: se define como el porcentaje del volumen total de roca o de sedimento formado por poros o espacios vacíos. Los huecos son con frecuencia espacios que quedan entre las partículas sedimentarias, pero también son comunes las fallas, las cavidades formadas por disolución de la roca soluble, como la caliza, y las vesículas (vacíos dejados por los gases que escapan de la lava). La porosidad normalmente se da en (%).
- **Permeabilidad**: es la capacidad de un material para que un fluido lo atraviese sin alterar su estructura interna. Se afirma que un material es permeable si deja pasar a través de él una cantidad apreciable de fluido en un tiempo dado, e impermeable si la cantidad de fluido es despreciable. La velocidad con la que el fluido atraviesa el material depende de tres factores básicos: la porosidad del material; la densidad del fluido considerado, afectada por su temperatura; la presión a que está sometido el fluido. Para ser permeable, un material debe ser poroso, es decir, debe contener espacios vacíos o poros que le permitan absorber fluido. A su vez, tales espacios deben estar interconectados para que el fluido disponga de caminos para pasar a través del material. En el Sistema Internacional de Unidades se mide la permeabilidad en $\left[m^2\right]$, aunque también se usa mucho la unidad *Darcy* $\left(1\,Darcy = 9.86923^{-13}\ m^2\right)$.
- **Conductividad térmica**: es una propiedad física de los materiales que mide la capacidad de conducción de calor. En otras palabras la conductividad térmica es también la capacidad de una sustancia de transferir la energía cinética de sus moléculas a otras moléculas adyacentes o a substancias con las que está en contacto. En el Sistema Internacional de Unidades la conductividad térmica se mide en $\left[\frac{W}{K \cdot m}\right]$. La inversa de la conductividad térmica es la resistencia térmica, que es la capacidad de los materiales para oponerse al paso del calor.
- **Densidad**: se denomina en ocasiones masa específica y es una magnitud referida a la cantidad de masa contenida en un determinado volumen. Puede utilizarse en términos absolutos o relativos. La densidad absoluta es la magnitud que expresa la relación entre la masa y el volumen de un cuerpo. Su unidad en el Sistema Internacional es $\left[\frac{kg}{m^3}\right]$. La densidad es una magnitud intensiva. La densidad relativa de una sustancia es la relación existente entre su densidad y la de otra sustancia de referencia; en consecuencia, es una magnitud adimensional.
- **Calor específico**: también se le conoce como capacidad calorífica específica. Es una magnitud física que indica la capacidad de un material para almacenar energía

interna en forma de calor. De manera formal es la energía necesaria para incrementar en una unidad de temperatura una cantidad de sustancia. Usando el Sistema Internacional de Unidades, es la cantidad de *Joules* de energía necesaria para elevar en 1 *K* la temperatura de 1 *kg* de masa. Sus unidades en Sistema Internacional están dadas por $\left[\frac{J}{kg \cdot K}\right]$.

- **Capacidad calorífica**: es el cociente entre la cantidad de energía calorífica transferida a un cuerpo o sistema en un proceso cualquiera y el cambio de temperatura que experimenta. Es la energía necesaria para aumentar 1 *K* su temperatura. Indica la mayor o menor dificultad que presenta dicho cuerpo para experimentar cambios de temperatura bajo el suministro de calor. Puede interpretarse como una medida de inercia térmica. Es una propiedad extensiva, ya que su magnitud depende, no solo de la sustancia, sino también de la cantidad de materia del cuerpo o sistema; por ello, es característica de un cuerpo o sistema particular. En el Sistema Internacional de Unidades esta dado por $\left[\frac{J}{m^3 \cdot K}\right]$.
- **Coeficiente de dilatación volumétrica**: es el cociente que mide el cambio relativo de volumen que se produce cuando un cuerpo sólido o un fluido dentro de un recipiente experimenta un cambio de temperatura, presentándose así una dilatación térmica.
- **Viscosidad**: la viscosidad es la oposición de un fluido a las deformaciones tangenciales. En la práctica se utilizan dos tipos de viscosidad: viscosidad dinámica y viscosidad cinemática. La viscosidad dinámica, denominada también viscosidad absoluta o simplemente viscosidad, es una propiedad característica de cada fluido y es además dependiente de la temperatura y la presión. La unidad de viscosidad en el Sistema Internacional es el *Pascalsegundo* $\left[Pa \cdot s\right]$. La viscosidad cinemática se define como el cociente entre la viscosidad dinámica y la densidad, siendo sus unidades $\left[\frac{m^2}{s}\right]$. Si bien la viscosidad de los fluidos depende tanto de la presión como de la temperatura, la dependencia con la presión es, por lo general, despreciable. La viscosidad dinámica de los líquidos decrece con la temperatura y la de los gases crece. Esta diferencia puede ser explicada por la diferencia de la estructura molecular. La resistencia al corte o deformación depende de la cohesión molecular y de la rapidez de transferencia de cantidad de movimiento molecular. En los líquidos predominan las fuerzas cohesivas entre las moléculas y como éstas decrecen con la temperatura la viscosidad también decrece con la temperatura.

9 Apéndice VI. Código fuente del programa

Base.java

```
package FEM;

/**
 * Clase abstracta para definir una Base usada en el FEM.
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public abstract class Base {

    /** Número de funciones que componen la base*/
    protected final int numFunciones;

    public Base()
    {
        numFunciones = defineNumFunciones();
    }
    /**
     * Evalúa una función de la base.
     * @param funcionArg  Indica que función de la base será evaluada.
     * @param xArg        Punto en el que será evaluada la función base.
     * @return  El valor de la función de la base.
     */
    public abstract double evaluaP0 (int funcionArg, double[]xArg);

    /**
     * Evalúa el gradiente de una función de la base
     * @param elemArg       Elemento sobre el cual se trabaja.
     * @param funcionArg    Indica que función de la base será evaluada.
     * @param xArg          Punto en el que será evaluada la función base.
     * @return  Un arreglo que contiene el valor del gradiente de la función
     *          de la base.
     */
    public abstract double []evaluaP1 (Elemento elemArg, int funcionArg, double[]xArg);

    /**
     * Establece el número de funciones que componen la base.
     * @return  Un número entero correspondiente al número de funciones que
     *          componen la base.
     */
    protected abstract int defineNumFunciones();
}
```

BaseLinealTriangulo.java

```
package FEM;

/**
 * Base que usa funciones lineales (Chapeau), definida sobre
 * un elemento bidimensional triangular,formado por 3 nodos coincidentes
 * en los vértices del triángulo.
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public class BaseLinealTriangulo extends Base{

    /**
     * Evalúa una función de la base Chapeaux
     * definida sobre triángulos formados por 3 nodos coincidentes
     * en los vértices del triángulo.
     * @param funcionArg    Indica que función de la base será evaluada.
     * @param xArg          Punto dentro del elemento estándar
     *                      en el que será evaluada la función base.
     * @return  El valor de la función de la base.
     */
```

```
public double evaluaP0 (int funcionArg, double[]xArg)
{
    //Elige cual función base será valuada en el punto xArg. En
    //este caso particular la base se compone de tres funciones.
    switch (funcionArg)
    {
        case 0:
            return 1.0-xArg[0]-xArg[1];

        case 1:
            return xArg[0];

        case 2:
            return xArg[1];

        default:
            return -9999;
    }
}

/**
 * Evalúa el gradiente de una función de la base Chapeaux
 * definida sobre triángulos formados por 3 nodos coincidentes
 * en los vértices del triángulo.
 * @param elemArg       Elemento sobre el cual se trabaja.
 * @param funcionArg    Indica que función de la base será evaluada.
 * @param xArg          Punto dentro del elemento estándar
 *                      en el que será evaluada la función base.
 * @return  Un arreglo que contiene el valor del gradiente de la función
 *          de la base.
 */
public double []evaluaP1 (Elemento elemArg, int funcionArg, double[]xArg)
{
    double arreglo[]= new double [2] ;

    Triangulo3Nodos elem = (Triangulo3Nodos)elemArg;

    double [][] M = new double[2][2];

    M=elem.calculaM();

    switch (funcionArg)
    {
        case 0:
            arreglo[0]=(-1.0*M[0][0])+(-1.0*M[1][0]);
            arreglo[1]=(-1.0*M[0][1])+(-1.0*M[1][1]);
            return arreglo;

        case 1:
            arreglo[0]=(1.0*M[0][0])+(0.0*M[1][0]);
            arreglo[1]=(1.0*M[0][1])+(0.0*M[1][1]);
            return arreglo;

        case 2:
            arreglo[0]=(0.0*M[0][0])+(1.0*M[1][0]);
            arreglo[1]=(0.0*M[0][1])+(1.0*M[1][1]);
            return arreglo;

        default:
            arreglo[0]=-99999;
            arreglo[1]=-99999;
            return arreglo;
    }

}

/**
 * Establece el número de funciones que componen la base.
 * @return  Un número entero correspondiente al número de funciones que
 *          componen la base.
 */
protected int defineNumFunciones()
{
    return 3;
}
```

```
}
```

Discretizacion.java

```
package FEM;

import AlgebraLineal.MatrizDensa;
import AlgebraLineal.Vector;

/**
 * Clase abstracta de las discretizaciones espaciales.
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public abstract class Discretizacion {

    /** Número de nodos que componen el elemento.*/
    protected final int nodosPorElemento;

    /** Número máximo de nodos adyacentes que puede tener un nodo
     * en particular en la discretización.
     */
    protected final int maxNodosAdyacentes;

    /** Dominio que es discretizado espacialmente.*/
    public Dominio dominio;

    /**
     * Arreglo que contiene los elementos que componen la
     * discretización espacial de del dominio.
     * Cada renglón del arreglo corresponde a un elemento de la discretización
     * del dominio.
     */
    protected Elemento[] elemento;

    /** Arreglo que contiene los nodos que componen la
     * discretización espacial del dominio.
     * Cada renglón del arreglo corresponde a un nodo de la discretización del
     * dominio.
     */
    public Nodo[] nodo;

    /** Arreglo que contiene los nodos que son incógnita.*/
    protected Nodo[] nodoIncognita;

    /** Número de nodos que componen la discretización.*/
    public int totalNodos;

    /** Número de elementos que componen la discretización.*/
    protected int totalElementos;

    /** Base usada en la discretización espacial*/
    protected Base base;

    /** Constructor de la clase abstracta referente a una discretización
     * espacial.
     * @param dominioarg    Dominio que será discretizado.
     */
    public Discretizacion(Dominio dominioarg)
    {
        dominio = dominioarg;
        nodosPorElemento = defineNodosPorElemento();
        maxNodosAdyacentes = defineMaxNodosAdyacentes();
        base = defineBase();
    }

    /**
     * Establece el número de nodos que tendrá cada elemento dependiendo
     * de la discretización en particular que sea implementada.
     * @return  Un número entero correspondiente al número de nodos
     *          que componen cada elemento de la discretización.
     */
```

```
public abstract int defineNodosPorElemento();

/**
 * Establece el número máximo de nodos adyacentes que puede llegar a tener
 * cada nodo de la discretización en particular que sea implementada.
 * @return  Un número entero correspondiente al número máximo de nodos
 *          que puede llegar a tener cada nodo.
 */
public abstract int defineMaxNodosAdyacentes();

/**
 * Establece la base que será usada por la discretización.
 * @return  Base usada en la discretización.
 */
public abstract Base defineBase();

/**
 * Obtiene los puntos de integración dentro del elemento estándar.
 * @return  Un arreglo bidimensional que contiene las coordenadas estándar
 *          de los puntos usados para realizar la integración.
 *          Cada renglón representa un punto y las columnas el valor
 *          correspondiente al eje coordenado (el cual varía dependiendo
 *          de la dimensión en que esté definido el elemento).
 */
protected abstract double [][] getPtInt();

/**
 * Obtiene los puntos de integración sobre una frontera dentro del elemento
 * estándar.
 * @param eArg      Elemento del cual se obtendrán los puntos de integración
 * @param frontArg  Frontera de la cual se obtendrán los puntos de
 *                  integración.
 * @return  Un arreglo bidimensional que contiene las coordenadas
 *          estándar de los puntos de integración.
 *          Cada renglón representa un punto y las columnas el valor
 *          correspondiente al eje coordenado (el cual varía dependiendo
 *          de la dimensión en que esté definido el elemento).
 */
protected abstract double [][] getPtIntFront(Elemento eArg,
        Frontera frontArg);

/**
 * Integra sobre una frontera de un elemento de la discretización.
 * @param eArg          Elemento sobre del cual se integrará.
 * @param ptIntStdArg   Puntos de integración dentro del elemento estándar.
 * @param fnPtIntArg    Arreglo que contiene los valores de la función a
 *                      integrar en cada uno de los puntos de integración.
 * @return      El valor de la integral de la función dentro de una frontera
 *              del elemento.
 */
protected abstract double integraFront(Elemento eArg,
        double [][]ptIntStdArg, double [] fnPtIntArg);

/**
 * Obtiene el valor de una Función valuada en un punto dentro de un elemento
 * @param elemArg   Elemento en el cual se valuará la función.
 * @param ptStdArg  Punto dentro del elemento estándar donde se valuará
 *                  la función.
 * @param funArg    Función que será valuada.
 * @param tArg      Valor del tiempo.
 * @return          Regresa el valor de la función valuada en el punto dado.
 */
public double valuaFuncion(Elemento elemArg, double [] ptStdArg,
        Funcion funArg, double tArg)
{
    //Verifica el tipo de Funcion que será interpolada en el elemento

    //Si es una función del tipo FuncionExpresion
    if (funArg instanceof FuncionExpresion)
    {
        //Se realiza una conversión descendente de la Funcion
        //(de una Funcion a una FuncionExpresion)
        FuncionExpresion funArgExp = (FuncionExpresion)funArg;

        //Se valua la función convirtiendo a coordenadas regulares
```

```
            return funArgExp.evaluaFuncion(elemArg.stdToReg(ptStdArg), tArg);
        }

        //Si es una función del tipo FuncionValor, se interpola el valor
        //dentro del elemento
        else if (funArg instanceof FuncionValor)
        {
            //Se realiza una conversión descendente de la Funcion
            //(de una Funcion a una FuncionValor)
            FuncionValor funArgVal = (FuncionValor)funArg;

            return this.valuaFuncionValor(elemArg, ptStdArg, funArgVal,tArg);
        }

        else return 0.0;
    }

    /**
     * Obtiene el valor de una Función del tipo FuncionValor
     * valuada en un punto dentro del elemento.
     * @param elemArg   Elemento en el cual se valuará la función.
     * @param ptStdArg  Punto dentro del elemento estándar donde se valuará
     *                  la función.
     * @param funArg    Función que será valuada.
     * @param tArg      Valor del tiempo.
     * @return          Regresa el valor de la función valuada en el punto dado.
     */
    protected abstract double valuaFuncionValor(Elemento elemArg,
            double [] ptStdArg, FuncionValor funArg, double tArg);

    /**
     * Obtiene las coordenadas de los nodos de la discretización.
     * @return  Una MatrizDensa que contiene los valores de las coordenadas
     *          de los nodos de la discretización.
     */
    public abstract MatrizDensa getCoordenadas();

    /**
     * Evalúa el gradiente de una función escalar en un nodo perteneciente a una
     * discretización en particular.
     * @param nodoArg       Nodo en el cual se calculará el gradiente de la
     *                      función escalar.
     * @param propiedadArg  Función escalar o propiedad de la cual se obtendrá
     *                      el gradiente. Esta propiedad está dada por un vector
     *                      que tiene los valores de la función en cada uno
     *                      de los nodos de la discretización.
     * @return              Un vector con el gradiente de la función escalar
     *                      calculado en un nodo en particular.
     */
    protected abstract Vector evaluaGradFuncEsc(Nodo nodoArg,
            Vector propiedadArg);
}
```

Dominio.java

```
package FEM;

/**
 *  Clase abstracta para un Domino.
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public abstract class Dominio {

    /** Dimensión en la que esta definido el dominio.*/
    protected final int dim;

    /**
     * Constructor para crear un Dominio
```

```
     */
    public Dominio()
    {
        dim = DefineDim();
    }

    /**
     * Indica si un dominio es válido y esta listo
     * para ser usado.
     * @return  True:    si el dominio esta listo para ser usado.
     * False:   si el dominio no esta listo para ser usado.
     */
    public abstract boolean dominioValido();

    /**
     * Establece la dimensión en la que esta definido un dominio la cual
     * dependerá del tipo de dominio en particular que se cree.
     * @return  Un número entero correspondiente a
     *          la dimensión en que esta definido el dominio.
     */
    protected abstract int DefineDim();

}
```

DominioRectangular.java

```
package FEM;

/**
 * Dominio en 2 Dimensiones de forma rectangular.
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public class DominioRectangular extends Dominio {

    /** Mínimo valor en la dirección X del dominio rectangular.*/
    private double x1;

    /** Máximo valor en la dirección X del dominio rectangular.*/
    private double x2;

    /** Mínimo valor en la dirección Z del dominio rectangular.*/
    private double z1;

    /** Máximo valor en la dirección Z del dominio rectangular.*/
    private double z2;

    /** Altura del dominio rectangular.*/
    private double H;

    /** Ancho del dominio rectangular.*/
    private double L;

    /** Razón geométrica del dominio  rectangular (H/L).*/
    private double alphaSomb;

    /**
     * Constructor para crear un dominio rectangular definiendo el dominio.
     * @param x1arg  mínimo valor en la dirección X.
     * @param z1arg  mínimo valor en la dirección Z.
     * @param x2arg  máximo valor en la dirección X.
     * @param z2arg  máximo valor en la dirección Z.
     */
    public DominioRectangular(double x1arg, double z1arg, double x2arg,double z2arg)
    {
        //dim=2;
        this.defineDominioRectangular(x1arg, z1arg, x2arg, z2arg);

    }

    /** Constructor para crear un dominio rectangular sin definir el dominio.
```

```
     *
     */
    public DominioRectangular()
    {
        //dim=2;
    }

   /**
     * Define un dominio Rectángular.
     * @param x1arg  mínimo valor en la dirección X.
     * @param z1arg  mínimo valor en la dirección Z.
     * @param x2arg  máximo valor en la dirección X.
     * @param z2arg  máximo valor en la dirección Z.
     * @return   True:    si el dominio fué definido exitosamente.
     *           False:   si el dominio no fué definido exitosamente.
     */
    public boolean defineDominioRectangular(double x1arg, double z1arg, double x2arg,double
z2arg)
    {
        //Valida que los parámetros de definición del dominio seán válidos
        if(x1arg>=x2arg || z1arg>=z2arg) return false;
        else
        {
            x1=x1arg;
            x2=x2arg;
            z1=z1arg;
            z2=z2arg;
            H=z2-z1;
            L=x2-x1;
            alphaSomb=H/L;
            return true;
        }
    }

    /** Indica si un dominio es válido y esta listo
     * para ser usado.
     * @return  True:    si el dominio esta listo para ser usado.
     *          False:   si el dominio no esta listo para ser usado.
     */
    public boolean dominioValido()
    {
        if(x1<x2 || z1<z2) return true;
        else return false;
    }

    /**
     * Establece la dimensión en la que esta definido un dominio rectangular.
     * @return  El valor de la dimensión en la que esta definido un dominio
     *          rectangular, es decir: 2.
     */
    protected int DefineDim()
    {
        return 2;
    }

    /** Obtiene el mínimo valor en la dirección X del dominio rectangular*/
    public double getX1()
    {
        return x1;
    }

    /** Obtiene el máximo valor en la dirección X del dominio rectangular*/
    public double getX2()
    {
        return x2;
    }

    /** Obtiene el mínimo valor en la dirección Z del dominio rectangular*/
    public double getZ1()
    {
        return z1;
    }

    /** Obtiene el máximo valor en la dirección Z del dominio rectangular*/
    public double getZ2()
```

```
    {
        return z2;
    }

    /** Obtiene la altura del dominio rectangular*/
    public double getH()
    {
        return H;
    }

    /** Obtiene el ancho del dominio rectangular*/
    public double getL()
    {
        return L;
    }

    /** Obtiene la razón geométrica del dominio rectangular*/
    public double getAlphaSomb()
    {
        return alphaSomb;
    }
}
```

Elemento.java

```
package FEM;

/**
 * Clase abstracta de un Elemento.
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public abstract class Elemento {

    /** identificador del elemento*/
    protected final int idElemento;

    /** Arreglo que contiene los nodos que componen el elemento*/
    protected Nodo[] nodo;

    /**  Constructor para crear un elemento.
     *
     * @param idElementoArg Número entero que identifica al elemento.
     */
    public Elemento(int idElementoArg)
    {
        idElemento=idElementoArg;
    }

    /**
     * Va de coordenadas en el elemento estándar
     * a coordenadas en el elemento regular.
     * @param coordElemEst  Coordenadas dentro del elemento estandar.
     * @return  Coordenadas dentro del elemento regular.
     */
    public abstract double []stdToReg (double[]coordElemEst);

    /**
     * Va de coordenadas en el elemento regular
     * a coordenadas en el elemento estándar.
     * @param coordElemReg  Coordenadas dentro del elemento regular.
     * @return  Coordenadas dentro del elemento estandar.
     */
    public abstract double []regToStd (double[]coordElemReg);

    /**
     * Integra dentro del elemento.
     * @param fnPtInt   Arreglo que contiene los valores de la función a
     *                  integrar en cada uno de los puntos de integración.
     *                  Cada elemento del arreglo corresponde a un punto
     *                  de integración.
     * @return      El valor de la integral de la función dentro del
```

```
     *              elemento.
     */
    public abstract double integraElem(double [] fnPtInt);
}
```

FemEstandar.java

```
package FEM;

import AlgebraLineal.Vector;
import AlgebraLineal.MatrizBandada;

/**
 * Clase para el Método del Elemento Finito (FEM) estándar.
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public class FemEstandar {

    /**
     * Problema a resolverse mediante el FEM
     */
    protected final Problema prob;

    /**
     * Tolerancia permitida en el método SOR.
     * Valor por default 1 x 10-6
     */
    private double tolSOR=0.000001;

    /**
     * Número máximo de iteraciones permitidas en el método SOR.
     * Valor por default 1000 iteraciones.
     */
    private int maxIteraSOR=1000;

    /**
     * Coeficiente de relajación usado en el método SOR.
     * Valor por default 1.5.
     */
    private double coefRelSOR=1.5;

    /**
     * Constructor para crear un FEM.
     * @param probArg   Problema a ser resuelto mediante FEM.
     */
    public FemEstandar(Problema probArg)
    {
        prob = probArg;
    }

    /**
     * Resuelve el FEM en un "paso" en el tiempo.
     * @param UkArg     Solución del problema en el tiempo actual.
     * @param tArg      Tiempo actual dentro del FEM-CrankNicholson.
     * @param DtArg     Salto en el tiempo (Dt).
     * @param thetaArg  Coeficiente de relajación en Crank Nicholson.
     * @return  Uk en el Tiempo actual + Dt.
     */
    public Vector resuelve (Vector ukArg, double tArg, double DtArg,
            double thetaArg)
    {
        //Tiempo sobre el cual se realiza el cálculo de la matriz AR y el
        //vector F (tAct+Dt/2)
        double tAct=tArg+(DtArg/2.0);

        //Tiempo sobre el que se calcula el siguiente paso en Crank Nicholson
        double tNext=tArg+DtArg;

        //Número de nodos en la discretización del problema
        int numNodos = prob.discretizacion.totalNodos;
```

```
//Número de bandas del problema
int bandas = prob.discretizacion.maxNodosAdyacentes+1;

//Número de incógnitas del problema
int numIncog = prob.discretizacion.nodoIncognita.length;

// Vector de solución.
Vector X;
X = new Vector(numNodos);

// Vector del lado derecho de la ecuación.
Vector F;
F = new Vector(numNodos);

// Matriz bandada donde se guarda la "matriz de rigidez" global
MatrizBandada A;
A = new MatrizBandada(numNodos,numNodos,bandas);

// Vector de solución "reducido" (solo incógnitas).
Vector X2;

// Vector del lado derecho de la ecuación "reducido" (solo incógnitas).
Vector F2;

// Matriz bandada donde se guarda la "matriz de rigidez" global
// "reducida" (solo incógnitas)
MatrizBandada A2;

// Solución en el tiempo actual.
Vector uk=ukArg;

//Solución en el tiempo mas el salto en el tiempo.
Vector ukNext;

//Variable de apoyo para asignar valores de tipo double.
double val;

////////////////////////////////////////////////////////////////
// Procesamiento de cada uno de los elementos que componen la //
// discretización. Se calcula la matriz de rigidez global (AR)//
// y el lado derecho de la ecuación F.                        //
////////////////////////////////////////////////////////////////

//Recorre todos los elementos de la discretización.
for (Elemento elem : prob.discretizacion.elemento)
{
    //Variables usadas para indicar el número de función dentro de la
    //base
    int i=0,j=0;

    //Ciclos usados para recorrer los nodos del elemento
    for(Nodo nodoI:elem.nodo)
    {
        for(Nodo nodoJ:elem.nodo)
        {
            //Calcula el valor correspondiente a AR y lo asigna a la
            //matriz global
            val=calculaAR(elem,i,j,tAct,thetaArg,DtArg);
            A.AdicionaValor(nodoI.idNodo, nodoJ.idNodo, val);

            //Calcula el valor correspondiente a M
            //(Matriz de Masa, derivada de la aportación
            //del método de Crank Nicholson a la parte izquierda de
            //la EDP)
            val = calculaM(elem,i,j,tAct) * (1/(thetaArg*DtArg)) *
                    ukArg.RetornaValor(nodoJ.idNodo);
            F.AdicionaValor(nodoI.idNodo, val);

            //Verifica que los nodos usados sean parte de una frontera
            //y que no sea una frontera dirichlet. De esta manera
            //se calcula el aporte de las fronteras Neumann a la
            //matriz AR
            if ( (nodoI instanceof NodoBorde && nodoI.incognitaMap>=0)&&
                 (nodoJ instanceof NodoBorde && nodoJ.incognitaMap>=0) )
            {
```

```
                //Se limpia la variable de apoyo
                val=0.0;

                //Se realiza una conversión descendente de los
                //Nodos Borde
                NodoBorde nodBordI = (NodoBorde) nodoI;
                NodoBorde nodBordJ = (NodoBorde) nodoJ;

                //Ciclos usados para recorrer las fronteras a las
                //que pudiese pertenecer cada nodo borde
                for(Frontera frontI:nodBordI.frontera)
                {
                    for(Frontera frontJ:nodBordJ.frontera)
                    {
                        //Comprueba que se trate de la misma frontera
                        if (frontI==frontJ)
                        {
                            val+=calculaFrontNeu(elem, frontI, i, j,
                                    tAct);
                        }
                    }
                }//Terminan ciclos que recorren las fronteras Neumann

                //Adiciona el valor en la Fontera Neumann a la matriz AR
                A.AdicionaValor(nodoI.idNodo, nodoJ.idNodo, val);

            }//Termina condición que permite procesar fronteras Neumann
             //en el lado izquierdo de la EDP

            j++;
        }//Termina ciclo que controla la funcion j de la base

        //Calcula el valor correspondiente a F y lo asigna al vector F
        F.AdicionaValor(nodoI.idNodo, calculaF(elem,i,tAct));

        //Verifica que el nodoI sea parte de una frontera y que éste
        //no sea una frontera dirichlet. De esta manera
        //se calcula el aporte de las fronteras Neumann al vector F
        if ( nodoI instanceof NodoBorde && nodoI.incognitaMap>=0 )
        {
            //Se limpia la variable de apoyo
            val=0.0;

            //Se realiza una conversión descendente del Nodo Borde
            NodoBorde nodBordI = (NodoBorde) nodoI;

            //Ciclos usado para recorrer las fronteras a las que
            //pudiese pertenecer el nodo borde
            for(Frontera frontI:nodBordI.frontera)
            {
                val+=calculaFrontNeu(elem, frontI, i, tAct);
            }
            //Adiciona el aporte de las fronteras Neumann a F
            F.AdicionaValor(nodoI.idNodo, val);

        }//Termina condición que permite procesar fronteras Neumann
         //en el lado derecho de la EDP

        j=0;
        i++;
    }//Termina ciclo usado para recorrer los nodos del elemento
}//Termina ciclo usado para recorrer los elementos de la discretización

////////////////////////////////////////////////////////////////////
// Procesamiento de matriz de rigidez (AR) y el vector de         //
// resultados (F), y el vector de incógnitas (X), de manera       //
// que se introducen las fronteras Dirichlet (valores conocidos), //
// se calculan la matriz y vectores "reducidos" (donde se han     //
// eliminado los valores conocidos) y se dejan listas éstas para  //
// su solución numérica (solución de un sistema lineal)           //
////////////////////////////////////////////////////////////////////

//Ciclo usado para recorrer los nodos de la geometría en busca de los
//valores conocidos (Fronteras Dirichlet)
for (Nodo nodActual:prob.discretizacion.nodo)
```

```
{
    //Verifica que se trate de una frontera Dirichlet
    if (nodActual.incognitaMap<0)
    {
        //Se realiza una conversión descendente del Nodo Borde
        NodoBorde nodActualBord = (NodoBorde) nodActual;

        //Se asignan en el vector de resultados (X), los valores
        //conocidos

        //Ciclo usado para recorrer las fronteras a las que
        //pudiese pertenecer el nodo
        for(Frontera frontAct:nodActualBord.frontera)
        {
            //Verifica que se trate de una frontera Dirichlet
            if(frontAct.tipoFrontera == true)
            {
                //Asigna el valor de la frontera Dirichlet
                X.AsignaValor(nodActual.idNodo, frontAct.valorFrontera.
                        evaluaFuncion(nodActual, tAct));
                break;
            }
        }
    }
}

//Inicializa la matriz AR y los vectores F y X reducidos
A2 = new MatrizBandada(numIncog,numIncog, bandas);
F2 = new Vector(numIncog);
X2 = new Vector(numIncog);

//Rutina para eliminar los valores conocidos y dejar solo las incógnitas
//en F2

//Ciclo usado para recorrer los nodos de la geometría en busca de los
//valores conocidos (Fronteras Dirichlet)
for (Nodo nodActual:prob.discretizacion.nodo)
{
    //Si el nodo es un nodo incógnita
    if(nodActual.incognitaMap >=0)
    {
        F2.AsignaValor(nodActual.incognitaMap,
                F.RetornaValor(nodActual.idNodo));

        //Ciclo que recorre los nodos adyacentes al nodo actual,
        //en busca de nodos pertenecientes a fronteras Dirichlet
        //(valores conocidos), y así ponderar su valor en el vector
        //de resultados
        for(Nodo nodAdy:nodActual.nodoAdyacente)
        {
            //Verifica que el nodo adyacente exista
            if (nodAdy !=null)
            {
            //Verifica que el nodo adyacente sea parte de una frontera
            //Dirichlet
            if (nodAdy.incognitaMap<0)
            {
                //Se calcula el valor que se adicionará a F,
                //correspondiente al nodo adyacente que es conocido y
                //será "eliminado" del proceso de solución del
                //sistema lineal.
                val=A.retornaValor(nodActual.idNodo,
                        nodAdy.idNodo)*
                        X.RetornaValor(nodAdy.idNodo);

                F2.AdicionaValor(nodActual.incognitaMap, -val);
                F.AdicionaValor(nodActual.idNodo,-val);
            }
            }
        }
    }
}//Fin del ciclo usado para recorrer los nodos de la discretización
 //en busca de los nodos conocidos

//Rutina para construir la matriz A y X  "reducidos"
```

```
//Ciclo usado para recorrer los nodos de la geometría en busca de los
//valores incógnita
for (Nodo nodI:prob.discretizacion.nodoIncognita)
{
    for (Nodo nodJ:prob.discretizacion.nodoIncognita)
    {
        A2.asignaValor(nodI.incognitaMap, nodJ.incognitaMap,
                A.retornaValor(nodI.idNodo, nodJ.idNodo));
    }

    //Se prepara al vector de resultados correspondiente a las
    //incógnitas para la solución iterativa (X2). Se asigna al vector de
    //resultados los valores correspondientes a la solución en el tiempo
    //actual (Uk)
    X2.AdicionaValor(nodI.incognitaMap, uk.RetornaValor(nodI.idNodo));
}

////////////////////////////////////////////////////////////////
// Solución del sistema lineal y obtención del resultado en el  //
// siguiente paso del tiempo (mediante Crank Nicholson)         //
////////////////////////////////////////////////////////////////

//Vector donde se guarda la solución completa
ukNext = new Vector(numNodos);

//Resuelve el sistema lineal mediante el método iterativo SOR
MetodosNumericos.LinearSolve.BandSOR(A2, X2, F2, this.tolSOR,
        this.maxIteraSOR, this.coefRelSOR);

//Rutina que asigna los valores de la solución al vector de solución
//completo
for (Nodo nodIncog:prob.discretizacion.nodoIncognita)
{
    X.AsignaValor(nodIncog.idNodo,
            X2.RetornaValor(nodIncog.incognitaMap));
}

//Rutina para obtener la solución con Crank Nicholson
for(Nodo nodAct:prob.discretizacion.nodo)
{
    //Verifica que el nodo adyacente sea parte de una frontera Dirichlet
    if(nodAct.incognitaMap<0)
    {
        //Se realiza una conversión descendente del Nodo Borde
        NodoBorde nodActBord = (NodoBorde) nodAct;

        //Asigna el valor de la frontera Dirichlet en el siguiente
        //paso del tiempo, al nodo correspondiente

        //Ciclo usado para recorrer las fronteras a las que
        //pudiese pertenecer el nodo
        for(Frontera frontAct:nodActBord.frontera)
        {
            //Verifica que se trate de una frontera Dirichlet
            if(frontAct.tipoFrontera == true)
            {
                //Asigna el valor de la frontera Dirichlet
                ukNext.AsignaValor(nodAct.idNodo, frontAct.valorFrontera
                        .evaluaFuncion(nodAct, tNext));
                break;
            }
        }
    }
    //Si el nodo es una incógnita
    else
    {
        //Calcula el valor en el siguiente paso del tiempo usando
        //Crank Nicholson y lo asigna al nodo
        val=(1.0/thetaArg)*(X.RetornaValor(nodAct.idNodo) -
                ((1.0-thetaArg)*uk.RetornaValor(nodAct.idNodo)) );

        ukNext.AsignaValor(nodAct.idNodo, val);
    }
}
return ukNext;
```

```
}

/**
 * Calcula el valor de AR del FEM Estándar
 * @param elemArg   Elemento sobre el cual se hace el cálculo.
 * @param iArg      Índice i de la función de la base usada.
 * @param jArg      Índice j de la función de la base usada.
 * @param tArg      Tiempo usado dentro del FEM-CrankNicholson.
 * @param thetaArg  Coeficiente de relajación en Crank Nicholson.
 * @param DtArg     Salto en el tiempo.
 * @return      El valor de AR.
 */
private double calculaAR(Elemento elemArg, int iArg, int jArg, double tArg,
        double thetaArg, double DtArg)
{
    //Obtiene los puntos de integración usados en la discretización
    //del problema. Puntos dentro del elemento estándar.
    double [][]ptIntStd = prob.discretizacion.getPtInt();

    //Arreglo usado para guardar los valores a integrarse
    double []fnPtInt = new double [ptIntStd.length];

    //Arreglo usado para guardar los primeros 7 coeficientes de la EDP
    double[]EDP= new double[7];

    //Variables usadas para guardar los valores de las funciones de la base
    double p0i, p0j;

    //Variables usadas para guardar los valores del gradiente de las
    //funciones de la base
    double[] p1i, p1j;

    //Variables de apoyo usadas para facilitar la lectura de la obtención
    //de los resultados.
    double RA, RB, RC;

    //Recorre todos los puntos de integración y calcula los valores a
    //integrar
    for (int i=0; i<ptIntStd.length; i++)
    {
        //Obtiene los valores de los coeficientes de la EDP en el punto
        //de integración.
        for (int j=0; j<EDP.length; j++)
        {
            EDP[j]=prob.discretizacion.valuaFuncion(elemArg,ptIntStd[i],
                    prob.coefEDP[j], tArg);
        }

        //Define los valores de las funciones de la base
        p0i=prob.discretizacion.base.evaluaP0(iArg, ptIntStd[i]);
        p0j=prob.discretizacion.base.evaluaP0(jArg, ptIntStd[i]);

        //Define los valores de los gradientes de las funciones de la base
        p1i=prob.discretizacion.base.evaluaP1(elemArg, iArg, ptIntStd[i]);
        p1j=prob.discretizacion.base.evaluaP1(elemArg, jArg, ptIntStd[i]);

        //Calcula el valor de la EDP correspondiente al coeficiente A
        RA = (p1i[0] * (EDP[0] * p1j[0] + EDP[2] * p1j[1])) +
                (p1i[1] * (EDP[2] * p1j[0] + EDP[1] * p1j[1]));

        //Calcula el valor de la EDP correspondiente al coeficiente B
        RB = p0j * (p1i[0] * EDP[3] + p1i[1] * EDP[4]);

        //Calcula el valor de la EDP correspondiente al coeficiente C
        RC = (EDP[5] + (EDP[6] / (thetaArg * DtArg))) * p0i * p0j;

        //Define el valor de la función valuada en el punto de integración
        fnPtInt[i]= RA - RB + RC;
    }

    return elemArg.integraElem(fnPtInt);
}

/**
 * Calcula el valor de M (Matriz de Masa) utilizada en el FEM Estándar
```

```
 * @param elemArg   Elemento sobre el cual se hace el cálculo.
 * @param iArg      Índice i de la función de la base usada.
 * @param jArg      Índice j de la función de la base usada.
 * @param tArg      Tiempo usado dentro del FEM-CrankNicholson.
 * @return          El valor de M.
 */
private double calculaM(Elemento elemArg, int iArg, int jArg, double tArg)
{
    //Obtiene los puntos de integración usados en la discretización
    //del problema. Puntos dentro del elemento estándar.
    double [][]ptIntStd = prob.discretizacion.getPtInt();

    //Arreglo usado para guardar los valores a integrarse
    double []fnPtInt = new double [ptIntStd.length];

    //Arreglo usado para guardar el valor de alpha de la EDP
    double alphaEDP;

    //Variables usadas para guardar los valores de las funciones de la base
    double p0i, p0j;

    //Recorre todos los puntos de integración y calcula los valores a
    //integrar
    for (int i=0; i<ptIntStd.length; i++)
    {
        //Obtiene el valor del coeficiente alpha de la EDP en el punto
        //de integración.
        alphaEDP=prob.discretizacion.valuaFuncion(elemArg,ptIntStd[i],
                    prob.coefEDP[6], tArg);

        //Define los valores de las funciones de la base
        p0i=prob.discretizacion.base.evaluaP0(iArg, ptIntStd[i]);
        p0j=prob.discretizacion.base.evaluaP0(jArg, ptIntStd[i]);

        //Define el valor de la función valuada en el punto de integración
        fnPtInt[i]= p0i * p0j * alphaEDP;
    }

    return elemArg.integraElem(fnPtInt);
}

/**
 * Calcula el valor de F del FEM Estándar
 * @param elemArg   Elemento sobre el cual se hace el cálculo.
 * @param iArg      Índice i de la función de la base usada.
 * @param tArg      Tiempo usado dentro del FEM-CrankNicholson.
 * @return      El valor de F.
 */
private double calculaF(Elemento elemArg, int iArg, double tArg)
{
    //Obtiene los puntos de integración usados en la discretización
    //del problema. Puntos dentro del elemento estándar.
    double [][]ptIntStd = prob.discretizacion.getPtInt();

    //Arreglo usado para guardar los valores a integrarse
    double []fnPtInt = new double [ptIntStd.length];

    //Arreglo usado para guardar el valor de F de la EDP
    double fEDP;

    //Variable usada para guardar el valor de las función i de la base
    double p0i;

    //Recorre todos los puntos de integración y calcula los valores a
    //integrar
    for (int i=0; i<ptIntStd.length; i++)
    {
        //Obtiene el valor del coeficiente F de la EDP en el punto
        //de integración.
        fEDP=prob.discretizacion.valuaFuncion(elemArg,ptIntStd[i],
                    prob.coefEDP[7], tArg);

        //Define el valor de la función de la base
        p0i=prob.discretizacion.base.evaluaP0(iArg, ptIntStd[i]);
```

```
        //Define el valor de la función valuada en el punto de integración
        fnPtInt[i]= p0i*fEDP;
    }

    return elemArg.integraElem(fnPtInt);
}

/**
 * Calcula el valor de la frontera Neumann en su aporte a la matriz de
 * rigidez (AR), usando las funciones I y J de la base.
 * @param eArg      Elemento sobre el cual se hace el cálculo
 * @param frontArg  Frontera que se calculará
 * @param iArg      Índice i de la función usada.
 * @param jArg      Índice j de la función usada.
 * @param tArg      Tiempo usado dentro del FEM-CrankNicholson.
 * @return          El calculo de la frontera Neumann.
 */
private double calculaFrontNeu(Elemento eArg, Frontera frontArg, int iArg,
        int jArg, double tArg)
{
    //Obtiene los puntos de integración de una frontera
    //usados en la discretización del problema.
    //Puntos dentro del elemento estándar.
    double [][]ptIntStd = prob.discretizacion.getPtIntFront(eArg,frontArg);

    //Arreglo usado para guardar los valores a integrarse
    double []fnPtInt = new double [ptIntStd.length];

    //Vector usado para guardar los valores de B de la EDP
    Vector B = new Vector(2);

    //Variable para guardar los valores de Bn
    double bn;

    //Variables usadas para guardar los valores de las funciones de la base
    double p0i, p0j;

    //Recorre todos los puntos de integración y calcula los valores a
    //integrar
    for (int i=0; i<ptIntStd.length; i++)
    {
        //Obtiene los valores del coeficiente B de la EDP en el punto
        //de integración.
        B.AsignaValor(0, prob.discretizacion.valuaFuncion(eArg,
                ptIntStd[i], prob.coefEDP[3], tArg));
        B.AsignaValor(1, prob.discretizacion.valuaFuncion(eArg,
                ptIntStd[i], prob.coefEDP[4], tArg));

        //Calcula Bn (B.N)
        bn = B.ProdPunto(B, frontArg.normal);

        //Define los valores de las funciones de la base
        p0i=prob.discretizacion.base.evaluaP0(iArg, ptIntStd[i]);
        p0j=prob.discretizacion.base.evaluaP0(jArg, ptIntStd[i]);

        fnPtInt[i]= p0i*p0j*bn;
    }

    return prob.discretizacion.integraFront(eArg, ptIntStd, fnPtInt);
}

/**
 * Calcula el valor de la frontera Neumann en su aporte al lado derecho
 * de la EDP (F), usando la función I de la base.
 * @param eArg      Elemento sobre el cual se hace el cálculo
 * @param frontArg  Frontera que se calculará
 * @param iArg      Índice i de la función usada.
 * @param tArg      Tiempo usado dentro del FEM-CrankNicholson.
 * @return          El calculo de la frontera Neumann.
 */
private double calculaFrontNeu(Elemento eArg, Frontera frontArg, int iArg,
        double tArg)
{
    //Obtiene los puntos de integración de una frontera
    //usados en la discretización del problema.
```

```
    //Puntos dentro del elemento estándar.
    double [][]ptIntStd = prob.discretizacion.getPtIntFront(eArg,frontArg);

    //Arreglo usado para guardar los valores a integrarse
    double []fnPtInt = new double [ptIntStd.length];

    //Variable usada para guardar el valor de las función i de la base
    double p0i;

    //Recorre todos los puntos de integración y calcula los valores a
    //integrar
    for (int i=0; i<ptIntStd.length; i++)
    {
        //Define el valor de las función i de la base
        p0i=prob.discretizacion.base.evaluaP0(iArg, ptIntStd[i]);

        //Calcula el aporte de la frontera Neumann al lado derecho de
        //la ecuación (F), en un punto de integración
        fnPtInt[i]= p0i*frontArg.valorFrontera.evaluaFuncion(
                eArg.stdToReg(ptIntStd[i]), tArg );
    }
    return prob.discretizacion.integraFront(eArg, ptIntStd, fnPtInt);
}

/**
 * Calcula el error que existe entre un vector solución y la solución
 * exacta (cuando ésta se conoce).
 * @param uArg  Vector solución en un paso del tiempo particular
 * @param tArg  Tiempo en el cual se calcula el error
 * @return      El valor del máximo error que existe en un paso en el tiempo
 */
public double calculaError(Vector uArg, double tArg)
{
    //Variable donde se guarda el valor del error de un nodo
    double err=0.0;

    //Variable donde se guarda el valor del máximo error en todos los nodos
    double maxErr=0.0;

    //Ciclo que recorre todos los nodos de la discetización
    for(Nodo nodAct:prob.discretizacion.nodo)
    {
        //Se calcula el error en el nodo
        err = Math.abs( prob.solucionExacta.evaluaFuncion(nodAct, tArg)-
                uArg.RetornaValor(nodAct.idNodo) );

        //Comprueba si el error en el nodo es el máximo
        if( maxErr<err) maxErr = err;
    }
    return maxErr;
}

/**
 * Ejecuta una "corrida" de un problema transitorio.
 * @param tIniArg   Tiempo inicial a partir de la cual se resolverá
 *                  el problema mediante el FEM.
 * @param tFinArg   Tiempo final hasta la cual se resolverá el problema
 *                  mediante el FEM.
 * @param DtArg     Salto en el tiempo.
 * @param thetaArg  Coeficiente de relajación usado en Crank Nicholson.
 * @param tArr      Arreglo donde se guardarán los tiempos usados en
 *                  cada uno de los pasos en el tiempo.
 * @return          Un arreglo de Vector que contiene la solución en
 *                  cada paso en el tiempo.
 */
public Vector[] corridaTransitoria(double tIniArg, double tFinArg,
        double DtArg, double thetaArg, double[] tArr)
{
    //Número de soluciones en el tiempo que tendrá la corrida
    int nSol = (int)((tFinArg-tIniArg)/DtArg)+1;

    //Contador utilizado para indexar las soluciones en el tiempo
    int iSol=0;

    //Arreglo de Vector usado para guardar las soluciones en el tiempo
```

```
        Vector [] Uk = new Vector[nSol];

        //Calcula la solución inicial y la guarda en el arreglo de vectores
        //solución
        Uk[iSol]=this.prob.CalculaCondIni();

        //Guarda el tiempo inicial
        tArr[iSol]=tIniArg;

        //Incrementa el contador
        iSol++;

        //Ciclo que recorre en el tiempo la corrida
        for (double t=tIniArg; t<=tFinArg-DtArg; t = t+DtArg)
        {
            //Calcula la solución en el paso del tiempo siguiente y la guarda
            //en el arreglo de vectores solución
            Uk[iSol]=this.resuelve(Uk[iSol-1], t, DtArg, thetaArg);

            //Guarda el tiempo correspondiente a este paso
            tArr[iSol]=t+DtArg;

            //Incrementa el contador
            iSol++;
        }

        //Regresa el arreglo que contiene los vectores solución
        return Uk;
    }

    /**
     * Define o cambia el valor de la tolerancia permitida en el método SOR
     * que resuelve el sistema lineal.
     * @param tolSORArg Tolerancia permitida en SOR.
     */
    public void setTolSOR(double tolSORArg)
    {
        this.tolSOR = tolSORArg;
    }

    /**
     * Define o cambia el valor del número máximo de iteraciones permitidas
     * en el método SOR que resuelve el sistema lineal.
     * @param maxIteraSORArg    Número máximo de iteraciones en SOR
     */
    public void setMaxIteraSOR(int maxIteraSORArg)
    {
        this.maxIteraSOR = maxIteraSORArg;
    }

    /**
     * Define o cambia el valor del coeficiente de relajación usado en el método
     * SOR que resuelve el sistema lineal.
     * @param coefRelSORArg Coeficiente de relajación usado en SOR.
     */
    public void setCoefRelSOR(double coefRelSORArg)
    {
        this.coefRelSOR = coefRelSORArg;
    }
}
```

Frontera.java

```
package FEM;

import AlgebraLineal.Vector;

/**
 * Clase usada para definir las fronteras en un problema de una EDP.
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
```

```
public class Frontera {

    /**Tipo de frontera:
     * True: Dirichlet.
     * False: Neumann.
     */
    protected final boolean tipoFrontera;

    /**
     * Identificador de la frontera.
     */
    protected final int idFrontera;

    /**
     * Funcion usada para definir los valores en las fronteras.
     */
    protected final FuncionExpresion valorFrontera;

    /**
     * Vector normal a la frontera (apuntando hacia afuera del dominio).
     * Se considera un vector normal único a la frontera.
     */
    protected final Vector normal;

    /**
     * Constructor para crear una frontera
     * @param tipoFronteraArg   Tipo de frontera.
     *                          True: Dirichlet.
     *                          False: Neumann.
     * @param valorArg          Valor en la frontera.
     * @param normArg           Vector normal a la frontera (apuntando hacia
     *                          afuera del dominio.
     */
    public Frontera(boolean tipoFronteraArg, int idFronteraArg,
            String valorArg, Vector normArg)
    {
        //Define el tipo de frontera
        tipoFrontera = tipoFronteraArg;

        //Define el id de la frontera
        idFrontera = idFronteraArg;

        //Define la función que servirá para valuar la frontera
        valorFrontera = new FuncionExpresion (valorArg);

        //Define el vector normal (apuntando hacia afuera del dominio) a
        //la frontera
        normal = new Vector(normArg.RetornaTamanio());
        for(int i=0; i<normArg.RetornaTamanio();i++)
        {
            normal.AsignaValor(i, normArg.RetornaValor(i));
        }
    }
}
```

Funcion.java

```
package FEM;

/**
 * Interfaz para valuar una Funcion en un nodo.
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public interface Funcion {
    /**
     * Regresa el valor de la función correspondiente al nodo.
     * @param nodoArg   Nodo a partir del cual se regresará el valor de la
     * función.
     * @return  El valor de la función en el nodo.
     */
    public double evaluaFuncion(Nodo nodoArg, double tArg);
```

```
}
```

FuncionExpresion.java

```
package FEM;

import org.nfunk.jep.*;

/**
 * Función que queda definida en una expresión que se valúa en el nodo
 * correspondiente.
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public class FuncionExpresion implements Funcion{

    /**
     * Arreglo donde se guarda el valor correspondiente a la función en
     * cada nodo. Cada renglón representa un nodo.
     */
    private String valorExpresion;

    /**
     * Bandera que indica si el valor de la expresión es una constante.
     * True: constante.
     * False: expresión matemática.
     */
    private boolean constFlag;

    /**
     * Valor de la expresión cuando éste es un valor constante.
     */
    private double doubleConst;

    /**
     * Constructor para crear una función a la cual se le asigna
     * un valor correspondiente a cada nodo de una discretización.
     * @param valorEnNodoArg    Arreglo que contiene los valores
     *                          correspondientes a cada nodo en una
     *                          discretización. Cada renglón corresponde
     *                          a un nodo.
     */
    public FuncionExpresion (String valorExpresionArg)
    {
        valorExpresion = valorExpresionArg;

        //Intenta obtener un valor doble de la expresión. Esto solo se logra
        //si se trata de un valor constante.
        try {
                //Asigna el valor y marca la bandera para indicar que es un
                //valor constante.
                doubleConst = Double.parseDouble(valorExpresion);
                constFlag = true;
            } catch (Exception exception) {
                //Marca la bandera para indicar que no es un valor constante.
                constFlag = false;
            }
    }

    /**
     * Regresa el valor de la función correspondiente al nodo.
     * @param nodoArg   Nodo a partir del cual se regresará el valor de la
     *                  función.
     * @param tArg      Tiempo en el cual se evaluará la función.
     * @return  El valor de la función en el nodo.
     */
    public double evaluaFuncion(Nodo nodoArg, double tArg)
    {

        //Regresa el valor de la función correspondiente a las coordenadas
        //del nodo.
        return this.evaluaFuncion(nodoArg.coordenada, tArg);
```

```
    }

    /**
     * Regresa el valor de la función correspondiente al nodo.
     * @param coordArg  Arreglo que contiene las coordenadas donde se valuará
     *                  la función.
     * @param tArg      Tiempo en el cual se evaluará la función.
     * @return          El valor de la función en el nodo.
     */
    public double evaluaFuncion(double[] coordArg, double tArg)
    {
        //Si se trata de una expresión constante regresa el valor de la
        //constante.
        if (this.constFlag) return this.doubleConst;

        //Crea el objeto de la librería JEP
        JEP miEvaluador = new JEP();

        //Inicializa un arreglo donde se guardarán las constantes a usarse.
        //Pueden ser hasta 4 constantes, es decir, puede manejar hasta un
        //espacio tridimensional y el tiempo
        String[] constante = new String[4];
        constante[0]="x";
        constante[1]="z";
        constante[2]="y";

        //Inicializa el objeto de la librería JEP
        miEvaluador.addStandardFunctions();
        miEvaluador.addStandardConstants();

        //Indica cuales serán las variables
        int i=0;
        for (i=0; i<coordArg.length; i++)
        {
            miEvaluador.addVariable(constante[i], i);
        }
        miEvaluador.addVariable("t", i);

        //Convierte la expresión en una fórmula
        miEvaluador.parseExpression(valorExpresion);

        //Asigna los valores de las variables
        for (i=0; i<coordArg.length; i++)
        {
            miEvaluador.addVariable(constante[i], coordArg[i]);
        }
        miEvaluador.addVariable("t", tArg);
        boolean errorEnExpresion = miEvaluador.hasError(); //hay error?

        //Valúa la exresión
        return miEvaluador.getValue();
    }
}
```

FuncionValor.java

```
package FEM;

/**
 * Función a la cual se le ha definido un valor (double) correspondiente
 * a cada nodo.
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public class FuncionValor implements Funcion{

    /**
     * Arreglo donde se guarda el valor correspondiente a la función en
     * cada nodo. Cada renglón representa un nodo.
     */
    protected double []valorEnNodo;
```

```
    /**
     * Constructor para crear una función a la cual se le asigna
     * un valor correspondiente a cada nodo de una discretización.
     * @param valorEnNodoArg    Arreglo que contiene los valores
     *                          correspondientes a cada nodo en una
     *                          discretización. Cada renglón corresponde
     *                          a un nodo.
     */
    public FuncionValor (double [] valorEnNodoArg)
    {
        valorEnNodo= new double[valorEnNodoArg.length];
        for (int i=0; i<valorEnNodoArg.length; i++)
        {
            valorEnNodo[i]=valorEnNodoArg[i];
        }
    }

    /**
     * Regresa el valor de la función correspondiente al nodo.
     * @param nodoArg   Nodo a partir del cual se regresará el valor de la
     *                  función.
     * @param tArg      Tiempo en el cual se evaluará la función.
     * @return  El valor de la función en el nodo.
     */
    public double evaluaFuncion(Nodo nodoArg, double tArg)
    {
        return valorEnNodo[nodoArg.idNodo];
    }
}
```

ModeloFlujoConveccion.java

```
package ModeloConveccion;

import AlgebraLineal.Vector;
import AlgebraLineal.MatrizDensa;
import FEM.ProblemaStream;
import FEM.ProblemaTemperatura;
import FEM.FemEstandar;
import FEM.DominioRectangular;
import FEM.TriangulacionDominioRectangular;

/**
 * Clase que controla la definición y solución de un modelo de flujo
 * convectivo en medio poroso.
 * @author Conrado
 * @version 1.0
 * @since 1.0
 */
public class ModeloFlujoConvectivo {

    /** Objeto que implementa la interfaz E/S de la cual se obtiene la
     * información necesaria para definir un modelo.
     */
    protected DefinicionModeloConveccion2D mod;

    /**Problema que se usará para modelar la ecuación de flujo de calor.*/
    protected ProblemaTemperatura probTemp;

    /**Problema que se usará para modelar la ecuación de flujo de masa.*/
    protected ProblemaStream probStream;

    /** Objeto FEM usado para resolver la ecuación de flujo de calor. */
    protected FemEstandar femTemp;

    /** Objeto FEM usado para resolver la ecuación de flujo de masa. */
    protected FemEstandar femStream;

    /**
     * Constructor de la clase.
     * @param defModArg Objeto que contiene la información necesaria para
     *                  definir un modelo.
     */
```

```
public ModeloFlujoConvectivo(DefinicionModeloConveccion2D defModArg)
{
    mod = defModArg;
    this.defineModelo();
}

/**
 * Rutina usada para definir las ecuaciones de flujo de calor y masa
 */
private void defineModelo()
{
    //Define el dominio que se usará en el modelo. En este caso se trata
    //de un dominio rectangular.
    DominioRectangular dominio = new DominioRectangular();
    dominio.defineDominioRectangular(
            mod.getX1()/mod.getL(), mod.getZ1()/mod.getH(),
            mod.getX2()/mod.getL(), mod.getZ2()/mod.getH());

    //Define la la discretización usada para la ecuación de flujo de calor.
    TriangulacionDominioRectangular triangTemp=new
           TriangulacionDominioRectangular(dominio,mod.getNx(),mod.getNz());

    //Define el problema que se usará para modelar la ecuación de flujo
    //de calor
    probTemp= new ProblemaTemperatura(triangTemp,mod);

    //Define el objeto FEM que resolverá la ecuación de flujo de calor.
   femTemp = new FemEstandar(probTemp);

    //Define la la discretización usada para la ecuación de flujo de calor.
    TriangulacionDominioRectangular triangStream=new
           TriangulacionDominioRectangular(dominio,mod.getNx(),mod.getNz());

    //Define el problema que se usará para modelar la ecuación de flujo
    //de masa
    probStream= new ProblemaStream(triangStream,mod);

    //Define el objeto FEM que resolverá la ecuación de flujo de masa.
   femStream = new FemEstandar(probStream);
}

/**
 * Ejecuta una "corrida" del modelo de flujo convectivo.
 * @param tIniArg   Tiempo inicial a partir de la cual se resolverá
 *                  el problema mediante el FEM.
 * @param tFinArg   Tiempo final hasta la cual se resolverá el problema
 *                  mediante el FEM.
 * @param tArr      Arreglo donde se guardarán los tiempos usados en
 *                  cada uno de los pasos en el tiempo.
 * @return          Un arreglo que contiene 2 vectores que incluyen la
 *                  solución de Temperatura y Stream.
 */
public Vector[][] corridaTransitoria(double tIniArg, double tFinArg,
        double[] tArr)
{
    //Soluciones de temperatura y Stream
    Vector temp = new Vector(probTemp.discretizacion.totalNodos);
    Vector stream = new Vector(probTemp.discretizacion.totalNodos);

    //Variables Vector de apoyo
    Vector tempAnt = new Vector(probTemp.discretizacion.totalNodos);
    Vector streamAnt = new Vector(probTemp.discretizacion.totalNodos);
    Vector tempAntT = new Vector(probTemp.discretizacion.totalNodos);

    //Número de soluciones en el tiempo que tendrá la corrida
    int nSol = (int)Math.ceil((tFinArg-tIniArg)/mod.getDt())+1;

    //Contador utilizado para indexar las soluciones en el tiempo
    int iSol=0;

    //Arreglos Vector usados para guardar las soluciones en el tiempo
    Vector [] ukTemp = new Vector[nSol];
    Vector [] ukStream = new Vector[nSol];

    //Asigna el valor del vector solución del tiempo anterior, en este caso
```

```
//el tiempo inicial.
tempAntT = probTemp.CalculaCondIni();

//Calcula la solución inicial y la guarda en el arreglo de vectores
//solución. La solución inicial se asume está originalmente definida
//por valores de temperatura adimensionales.

ukTemp[iSol]= new Vector(probTemp.discretizacion.totalNodos);
ukTemp[iSol]=VectorTemAdimtoReal(tempAntT);

//Asigna el valor de Stream para el tiempo inicial, el cual se supone
//es igual a cero pues se parte de que el fluido está estático en el
//medio.
ukStream[iSol]= new Vector(probTemp.discretizacion.totalNodos);
ukStream[iSol].Inicializa(0.0);

//Inicializa los valores de Temperatura y Stream que se usarán en
//los ciclos
temp = ukTemp[iSol];
stream = ukStream[iSol];

//Guarda el tiempo inicial
tArr[iSol]=tIniArg;

//Incrementa el contador
iSol++;

//Variables usadas para ponderar la diferencia entre las soluciones
//en un mismo paso en el tiempo
double difTem=0.0;
double difStream=0.0;

//Variable usada para contar las iteraciones en el ciclo de acoplamiento
int cont=0;

//Ciclo que recorre en el tiempo la corrida
for (double t=tIniArg; t<tFinArg; t = t+mod.getDt())
{
    //Ciclo de acoplamiento entre las soluciones de Temperatura y
    //Stream
    do
    {
        //Actualiza el valor de Stream que usa la ecuación de calor
        probTemp.asignaStream(stream);

        //Resuelve la ecuación de calor.
        temp= femTemp.resuelve(tempAntT, t, mod.getDt(),
                mod.getTheta());

        //Actualiza el valor de Temperatura que usa la ecuación de masa
        probStream.asignaTemp(temp);

        //Resuelve la ecuación de masa. El hecho de que el parámetro
        //Theta sea igual a 1.0, hace que el problema sea análogo
        //a un problema estacionario.
        stream = femStream.resuelve(ukStream[iSol-1], t, mod.getDt(),
                1.0);

        //Calcula la diferencia entre dos soluciones consecutivas de
        //Temperatura y Stream

        try
        {
            difTem=temp.maximaDiferiencia(tempAnt);
            difStream=stream.maximaDiferiencia(streamAnt);
        }
        catch (Exception exception)
        {
            System.err.printf("%s\n\n", exception.getMessage());
        }

        //Incrementa el contador de iteraciones.
        cont++;

        //Actualiza los vectores de resultados Temperatura y Stream
```

```
            //anteriores, los cuales se usarán en la siguiente iteración.
            tempAnt = temp;
            streamAnt = stream;

        //El ciclo se realiza mientras no se cumplan los criterios de
        //convergencia o bien mientras no se sobrepase el número máximo de
        //iteraciones
        } while ( ( difTem>=mod.getDifTempMax() ||
                difStream>=mod.getDifStreamMax() )
                && cont<=mod.getMaxIt() );

        //Reinicia el contador de iteraciones.
        cont = 0;

        //Guarda la solución de Temperatura en el paso del tiempo actual.
        ukTemp[iSol]= new Vector(probTemp.discretizacion.totalNodos);
        ukTemp[iSol]=VectorTemAdimtoReal(temp);

        //Guarda la solución de Stream en el paso del tiempo actual.
        ukStream[iSol]= new Vector(probTemp.discretizacion.totalNodos);
        ukStream[iSol]=stream;

        //Asigna el valor del vector solución del tiempo anterior
        tempAntT= temp;

        //Guarda el tiempo correspondiente a este paso
        tArr[iSol]=t+mod.getDt();

        //Incrementa el contador que indexa el arreglo de vectores de
        //solución.
        iSol++;
    }

    //Preparación de los vectores de resultado para la salida.
    Vector [][] resultado = {ukTemp,ukStream};

    //Regresa el arreglo que contiene los vectores solución.
    return resultado;
}

/**
 * Transforma el valor de una temperatura "real" a una temperatura
 * adimensional.
 * @param TRealArg  Valor de la temperatura real.
 * @return  El valor de la temperatura adimensional.
 */
public double temRealtoAdim(double TRealArg)
{
    //return (TRealArg-TRefReal)/(T2Real-T1Real);
    return (TRealArg-mod.getTRef())/(mod.getT2()-mod.getT1());
}

/**
 * Transforma el valor de una temperatura adimensional a una temperatura
 * real.
 * @param TRealArg  Valor de la temperatura adimensional.
 * @return  El valor de la temperatura "real".
 */
public double temAdimtoReal(double TAdimArg)
{
    return ((mod.getT2()-mod.getT1())*TAdimArg)+mod.getTRef();
}

/**
 * Realiza la conversión de temperaturas adimensionales a temperaturas
 * "reales" de los valores de un Vector.
 * @param TempArg   Vector de temperaturas adimensionales.
 * @return          Vector de temperaturas reales.
 */
public Vector VectorTemAdimtoReal(Vector TempArg)
{
    //Variable donde se guarda el vector resultado.
    Vector TempRes = new Vector(TempArg.RetornaTamanio());

    //Ciclo que recorre el vector
```

```
        for(int i=0; i<TempArg.RetornaTamanio();i++)
        {
            TempRes.AsignaValor(i, this.temAdimtoReal(TempArg.RetornaValor(i)));
        }
        return TempRes;
    }

    /**
     * Obtiene las coordenadas de los nodos de la triangulación de un dominio
     * rectangular.
     * @return  Una MatrizDensa que contiene los valores de las coordenadas
     *          de los nodos de la triangulación del dominio rectangular.
     */
    public MatrizDensa getCoordenadasReales()
    {
        //Matriz donde se guardan las coordenadas de los nodos del dominio
        MatrizDensa coord = new MatrizDensa(
                this.probStream.discretizacion.totalNodos,2);

        //Variables de apoyo usadas para obtener las coordenadas de los nodos
        double x,z;

        //Obtiene los valores iniciales de x y x
        x=mod.getX1();
        z=mod.getZ1();

        //Variables usadas para definir la separación entre nodos en el eje
        //x y x
        double Hx,Hz;
        Hx=mod.getL()/(mod.getNx());
        Hz=mod.getH()/(mod.getNz());

        //Contador usado para indexar las coordenadas de los nodos
        int contNod=0;

        for (int i=0; i<=mod.getNz();i++)
        {
            for (int j=0; j<=mod.getNx();j++)
            {
                coord.asignaValor(contNod, 0, x);
                coord.asignaValor(contNod, 1, z);
                contNod++;
                x=x+Hx;
            }
            x=mod.getX1();
            z=z+Hz;
        }
        return coord;
    }
}
```

Nodo.java

```
package FEM;

/**
 /**
 * Clase de un Nodo.
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public class Nodo {

    /** Identificador del nodo*/
    public final int idNodo;

    /** Coordenada del nodo*/
    public double coordenada[];

    /**
     * Indica el número de "incógnita" (o número de ecuación usada en el FEM)
```

```
     * que se le asigna a cada nodo de la discretización.
     * Este valor será igual a -1 si se trata de un nodo correspondiente a
     * una frontera Dirichlet(debido a que es un valor conocido).
     */
    protected int incognitaMap;

    /** Arreglo que contiene los nodos adyacentes al nodo*/
    protected Nodo[] nodoAdyacente;

    /**  Constructor para crear un nodo.
     *
     * @param idNodoArg Número entero que identifica al nodo.
     * @param coordArg  Arreglo de números que contiene las coordenadas
     *                  del nodo.
     */
    public Nodo(int idNodoArg, double[] coordArg)
    {
        idNodo=idNodoArg;

        //Inicializa las coordenadas del nodo
        coordenada = new double [coordArg.length];
        for(int i=0; i< coordArg.length; i++)
        {
            coordenada[i] = coordArg[i];
        }
    }
}
```

NodoBorde.java

```
package FEM;

import java.util.List;
import java.util.ArrayList;

/**
 * Nodo borde en un espacio bidimensional
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public class NodoBorde extends Nodo{

    /**
     * Lista de Fronteras a las cuales pertenece el nodo.
     * Un nodo pude pertener a varias fronteras.
     */
    public List <Frontera> frontera = new ArrayList<Frontera>();

    /** Constructor para crear un nodo borde en un espacio bidimensional
     *
     * @param idNodoArg Número entero que identifica al nodo.
     * @param xarg  Valor en x del sistema coordenado bidimensional (x,z)
     * @param zarg  Valor en z del sistema coordenado bidimensional (x,z)
     */
    public NodoBorde(int idNodoArg, double[] coordArg)
    {
        super(idNodoArg, coordArg);
    }
}
```

Problema.java

```
package FEM;

import AlgebraLineal.Vector;
import ModeloConveccion.DefinicionModelo;

 /**
 * Clase abstracta para definir un problema completo definido modelado
 * por una EDP
 * @author Conrado Montealegre
```

```
 * @version 1.0
 * @since 1.0
 */
public abstract class Problema {

    /**
     * Arreglo de objetos Funcion, el cual servirá para definir los
     * "coeficientes" de la Ecuación Diferencial Parcial (EDP)
     * a resolver.
     * Los "coeficientes" de la EDP quedan definidos (por renglón del arreglo)
     * como:
     * 0: a11
     * 1: a22
     * 2: a12=a21
     * 3: b1
     * 4: b2
     * 5: c
     * 6: alpha
     * 7: f
     */
    protected final Funcion[] coefEDP;

    /**
     * Arreglo de objetos Frontera, que contiene la definición de
     * todas las fronteras del problema.
     */
    protected final Frontera[] frontera;

    /**
     * Funcion usada para definir el valor de las condiciones Iniciales
     * del problema.
     */
    protected final FuncionExpresion condIniciales;

    /**
     * Funcion usada para definir el valor de la solución exacta si es
     * que esta existe.
     */
    public final FuncionExpresion solucionExacta;

    /**
     * Discretización usada en el problema.
     */
    public Discretizacion discretizacion;

    /** Interfaz de E/S usada para leer los parámetros que definen el problema*/
    protected DefinicionModelo defMod;

    /**
     * Constructor de un Problema.
     * @param discretArg    Discretización usada en el problema.
     */
    public Problema(Discretizacion discretArg)
    {
        //Asigna la discretización que se usará en el problema
        this.discretizacion = discretArg;

        //Inicializa las fronteras, los "coeficientes" de la EDP,
        // las condiciones iniciales y la solución exacta del problema.
        this.frontera = defineFrontera();
        this.coefEDP = defineCoefEDP();
        this.condIniciales= defineCondIniciales();
        this.solucionExacta= defineSolucionExacta();
        this.incializaNodos();
    }

    /**
     * Constructor de un Problema, cuando el problema está definido a partir
     * de un medio externo.
     * @param discretArg    Discretización usada en el problema.
     * @param modArg        Interfaz de E/S usada para leer los parámetros
     *                      que definen el problema.
     */
    public Problema(Discretizacion discretArg,
            DefinicionModelo modArg)
```

```
{
    //Establece cual será la interfaz E/S usada para definir el modelo.
    defMod = modArg;

    //Asigna la discretización que se usará en el problema
    this.discretizacion = discretArg;

    //Inicializa las fronteras, los "coeficientes" de la EDP,
    // las condiciones iniciales y la solución exacta del problema.
    this.frontera = defineFrontera();
    this.coefEDP = defineCoefEDP();
    this.condIniciales= defineCondIniciales();
    this.solucionExacta= defineSolucionExacta();
    this.incializaNodos();
}

/**
 * Calcula la solución Uk en el tiempo 0 (Condiciones Iniciales)
 * @return  El vector de solución Uk valuado en t=0
 */
public Vector CalculaCondIni()
{
    //Vector donde se guarda el valor de Uk en el tiempo 0 (condiciones
    //iniciales)
    Vector ukIni = new Vector(this.discretizacion.totalNodos);

    //Ciclo que recorre todos los nodos de la discetización
    for(Nodo nodAct:this.discretizacion.nodo)
    {
        //Se calcula el valor inicial en cada nodo de la discretización
        ukIni.AsignaValor( nodAct.idNodo, this.condIniciales.
                evaluaFuncion(nodAct, 0.0) );
    }
    return ukIni;
}

/**
 * Rutina donde se inicializan los nodos de la discretización con los
 * valores que le corresponden según el problema en particular.
 * @return  El vector de solución Uk valuado en t=0
 */
private void incializaNodos()
{
    ////////////////////////////////////////////////
    //Inicializa la numeración de los nodos incógnita//
    ////////////////////////////////////////////////

    //Define la variable usada para numerar los nodos incógnita
    int contIncognita=0;

    //Recorre todos los nodos de la discretización
    for (Nodo nodoActual: this.discretizacion.nodo)
    {
        //Verifica si el nodo es un nodo borde del dominio
        if (nodoActual instanceof NodoBorde)
        {
            //Bandera que indica si en el nodo existió
            //alguna frontera dirichlet
            boolean frontDir=false;

            //Se realiza una conversión descendente del nodo
            //(de un Nodo a un NodoBorde)
            //para poder verificar el tipo de frontera
            NodoBorde nodoBordeActual = (NodoBorde)nodoActual;

            //Ciclo para recorrer todas las fronteras a las que
            //pueda pertenecer el nodo
            for (Frontera frontAct:nodoBordeActual.frontera)
            {
                //Si alguna de las fronteras es del tipo Dirichlet
                //se le asigna un valor de -1 pues se trata de
                //un valor conocido
                if(frontAct.tipoFrontera==true)
                {
                    nodoActual.incognitaMap=-1;
```

```
                        frontDir=true;
                        break;
                    }
                }
                if (frontDir==false)
                {
                    //Cuando ninguna frontera fué del tipo Dirichlet
                    //(es decir, todas fueron Neumann)
                    //se le asigna un número consecutivo de incógnita
                    nodoActual.incognitaMap=contIncognita;
                    contIncognita++;
                }
            }
            else
            {
                //Cuando el nodo no es un nodo borde se le asigna
                //un número consecutivo de incógnita
                nodoActual.incognitaMap=contIncognita;
                contIncognita++;
            }
        }

        ////////////////////////////////////////////////
        //Se crea el arreglo de nodos incógnita          //
        ////////////////////////////////////////////////

        discretizacion.nodoIncognita = new Nodo[contIncognita];

        //Recorre todos los nodos de la discretización
        for (Nodo nodoActual: this.discretizacion.nodo)
        {
            //Verifica que el nodo sea un nodo incógnita
            if(nodoActual.incognitaMap >=0)
            {
               discretizacion.nodoIncognita[nodoActual.incognitaMap]=nodoActual;
            }
        }
    }

    /**
     * Define las fronteras del problema y las asigna a los nodos borde.
     * @return Un arreglo con las fronteras del problema.
     */
    protected abstract Frontera[] defineFrontera();

    /**
     * Define los "coeficientes" de la EDP y los asigna (de ser el caso) a
     * los nodos de la discretización.
     * @return  Un arreglo con los "coeficientes" de la EDP.
     */
    protected abstract Funcion[] defineCoefEDP();

    /**
     * Defincón de las condiciones inciales del problema.
     * @return
     */
    protected abstract FuncionExpresion defineCondIniciales();

    /**
     * Definición de la solución exacta, si es que existe, del problema.
     * @return
     */
    protected abstract FuncionExpresion defineSolucionExacta();
}
```

ProblemaStream.java

```
package FEM;

import ModeloConveccion.DefinicionModelo;
import ModeloConveccion.DefinicionModeloConveccion2D;
import AlgebraLineal.Matriz;
import AlgebraLineal.Vector;
```

```
import AlgebraLineal.MatrizDensa;
import java.util.List;
import java.util.ArrayList;

/**
 * Clase usada para manejar el flujo de masa en un modelo de flujo convectivo
 * definido en un dominio rectangular (2D).
 * Las fronteras y los parámetros que definen el modelo se leen a través
 * de una interfaz externa.
 *
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public class ProblemaStream extends Problema{

    /** Interfaz de E/S usada para leer los parámetros que definen el modelo
     * de convección. */
    protected DefinicionModeloConveccion2D mod;

    /** Razón geométrica adimensional (H/L)*/
    protected double alphaSomb;

    /** Tensor métrico asociado al sistema coordenado adimensional (x,z)*/
    protected Matriz alpha;

    /** Capacidad calorífica del fluido*/
    protected double Rhoc;

    /** Densidad media del fluido*/
    protected double Rho;

    /** Viscosidad dinámica del fluido a la temperatura de referencia*/
    protected double Neta;

    /** Coeficiente de dilatación térmica del fluido*/
    protected double Beta;

    /** Aceleración de la gravedad*/
    protected double g;

    /** Coeficiente de dependencia de la viscosidad dinámica del fluido con la
     * temperatura.
     */
    protected double Gamma;

    /** Número de Rayleigh modificado para un medio heterogéneo*/
    protected double [] Ra;

    /** Tensor simétrico de permeabilidad anisotrópica*/
    protected Matriz []KAst;

    /** Tensor de permeabilidad adimensional anisotrópica en 2D */
    protected Matriz []K;

    /** Tensor de permeabilidad adimensional en el sistema coordenado
     * adimensional (x,z) */
    protected Matriz []KSomb;

    /** Tensor simétrico de conductividad térmica anisotrópica*/
    protected Matriz []LambAst;

    /** Tensor de conductividad térmica adimensional anisotrópica en 2D */
    protected Matriz []Lamb;

    /** Tensor de conductividad térmica adimensional en el sistema coordenado
     * adimensional (x,z) */
    protected Matriz []LambSomb;

    /**Arreglo de apoyo usado para calcular la traza de LambAst en cada nodo*/
    private double []trLambAst;

    /**Arreglo de apoyo usado para calcular el determinante de KSomb
     * en cada nodo*/
    private double []detKSomb;
```

```
/**Arreglo de apoyo usado para calcular la traza de KAst en cada nodo*/
private double [] trKAst;

/**
 * Costructor de un problema usado en el flujo de masa de un modelo
 * de convección.
 * @param discretArg    Discretización usada en el problema.
 * @param modArg        Interfaz de E/S usada para leer los parámetros
 *                      que definen el modelo de convección.
 */
public ProblemaStream(TriangulacionDominioRectangular discretArg,
        DefinicionModeloConveccion2D modArg)
{
    super(discretArg,(DefinicionModelo) modArg);
}

/**
 * Establece los "coeficientes" de la EDP a partir de los parámetros de
 * que definen el flujo de masa, los cuales se leen de un medio externo.
 * @return  Un arreglo con los "coeficientes" de la EDP.
 */
protected Funcion[] defineCoefEDP()
{
    ////////////////////////////////////////////////////////////
    //  Se calculan los valores de los parámetros usados en la  //
    //  ecuación de flujo de masa.                              //
    ////////////////////////////////////////////////////////////

    //Matriz donde se guardarán los valores de los coeficientes de la EDP
    double [][] coef = new double[8][discretizacion.totalNodos];

    //Realiza una conversión descendente de la triangulación
    DominioRectangular dominio =
            (DominioRectangular) this.discretizacion.dominio;

    //Obtiene y calcula los parámetros de la ecuación de calor
    alphaSomb=mod.getH()/mod.getL();
    alpha = new MatrizDensa(2,2);
    alpha.Inicializa(0.0);
    alpha.asignaValor(0, 0, Math.sqrt(alphaSomb));
    alpha.asignaValor(1, 1, 1.0/Math.sqrt(alphaSomb));
    Rhoc = mod.getRhoc();
    Rho = mod.getRho();
    Neta = mod.getNeta();
    Beta = mod.getBeta();
    g = mod.getG();
    Gamma = mod.getGamma();
    Ra = new double[discretizacion.totalNodos];
    KAst = new MatrizDensa[discretizacion.totalNodos];
    K = new MatrizDensa[discretizacion.totalNodos];
    KSomb = new MatrizDensa[discretizacion.totalNodos];
    trKAst = new double[discretizacion.totalNodos];
    detKSomb = new double[discretizacion.totalNodos];
    LambAst = new MatrizDensa[discretizacion.totalNodos];
    Lamb = new MatrizDensa[discretizacion.totalNodos];
    LambSomb = new MatrizDensa[discretizacion.totalNodos];
    trLambAst = new double[discretizacion.totalNodos];

    //Ciclo que recorre los nodos de la discretización
    for (Nodo nodAct:discretizacion.nodo)
    {
        int i=nodAct.idNodo;
        KAst[i] = new MatrizDensa(2,2);
        K[i] = new MatrizDensa(2,2);
        KSomb[i] = new MatrizDensa(2,2);
        LambAst[i] = new MatrizDensa(2,2);
        Lamb[i] = new MatrizDensa(2,2);
        LambSomb[i] = new MatrizDensa(2,2);

        //Leé el valor de LambAst y KAst correspondiente al nodo de la
        //discretización.
        LambAst[i]=mod.getLambAst(i);
        KAst[i] = mod.getKAst(i);
```

```
try
{
    trLambAst[i]=LambAst[i].obtenerTraza();
    trKAst[i]=KAst[i].obtenerTraza();
}
catch (Exception exception)
{
    System.err.printf("%s\n\n", exception.getMessage());
}

Lamb[i].asignaValor(0, 0, LambAst[i].retornaValor(0, 0));
Lamb[i].asignaValor(0, 1, LambAst[i].retornaValor(0, 1));
Lamb[i].asignaValor(1, 0, LambAst[i].retornaValor(1, 0));
Lamb[i].asignaValor(1, 1, LambAst[i].retornaValor(1, 1));
K[i].asignaValor(0, 0, KAst[i].retornaValor(0, 0));
K[i].asignaValor(0, 1, KAst[i].retornaValor(0, 1));
K[i].asignaValor(1, 0, KAst[i].retornaValor(1, 0));
K[i].asignaValor(1, 1, KAst[i].retornaValor(1, 1));
try
{
    Lamb[i].multiplicaEscalar((2/trLambAst[i]));
    K[i].multiplicaEscalar((2/trKAst[i]));
}
catch (Exception exception)
{
    System.err.printf("%s\n\n", exception.getMessage());
}
try
{
    LambSomb[i] = alpha.postMultiplicaMatriz(Lamb[i]);
    LambSomb[i] = LambSomb[i].postMultiplicaMatriz(alpha);
    KSomb[i] = alpha.postMultiplicaMatriz(K[i]);
    KSomb[i] = KSomb[i].postMultiplicaMatriz(alpha);
}
catch (Exception exception)
{
    System.err.printf("%s\n\n", exception.getMessage());
}
try
{
    detKSomb[i]= KSomb[i].obtenerDeterminante2x2();
}
catch (Exception exception)
{
    System.err.printf("%s\n\n", exception.getMessage());
}

Ra[i]=(Rho*g*Rhoc*Beta*(mod.getT2()-mod.getT1())*mod.getH()
        *trKAst[i])/(Neta*trLambAst[i]);

// Se asignan los valores de los coeficientes fijos de
// la EDP en cada nodo. Además asigna un valor "basura" a los
//coeficientes que dependen del valor de Temperatura.

//Asigna a11
coef[0][nodAct.idNodo]= KSomb[nodAct.idNodo].retornaValor(0, 0);

//Asigna a22
coef[1][nodAct.idNodo]= KSomb[nodAct.idNodo].retornaValor(1, 1);

//Asigna a12=a21
coef[2][nodAct.idNodo]= KSomb[nodAct.idNodo].retornaValor(0, 1);

//Asigna un valor "basura" a b1
coef[3][nodAct.idNodo]= 0.0;

//Asigna un valor "basura" a b2
coef[4][nodAct.idNodo]= 0.0;

//Asigna c
coef[5][nodAct.idNodo]= 0.0;

//Asigna alpha
coef[6][nodAct.idNodo]= 0.0;
```

```
            //Asigna un valor "basura" a f
            coef[7][nodAct.idNodo]= 0.0;
        }

        //Inicializa el arreglo de Funciones
        Funcion[] EDP = new Funcion[8];

        //Asigna a11
        EDP[0]= new FuncionValor(coef[0]);

        //Asigna a22
        EDP[1]= new FuncionValor(coef[1]);

        //Asigna a12=a21
        EDP[2]= new FuncionValor(coef[2]);

        //Asigna b1
        EDP[3]= new FuncionValor(coef[3]);

        //Asigna b2
        EDP[4]= new FuncionValor(coef[4]);

        //Asigna c
        EDP[5]= new FuncionValor(coef[5]);

        //Asigna alpha
        EDP[6]= new FuncionValor(coef[6]);

        //Asigna f
        EDP[7]= new FuncionValor(coef[7]);
        return EDP;
    }

    /**
     * Define las fronteras del problema y las asigna a los nodos borde, a
     * partir de la información obtenida de un medio externo.
     * @return Un arreglo con las fronteras del problema.
     */
    protected Frontera[] defineFrontera()
    {
        //Como esta es la primera rutina que se corre en el constructor
        //del problema, aquí es donde se especifica la definición del problema

        //Conversion descendente que permite usar el tipo particular
        //de definicion de modelo que usa este problema
        mod = (DefinicionModeloConveccion2D) defMod;

        ////////////////////////////////////////////
        //Definición de las fronteras del problema  //
        ////////////////////////////////////////////

        //Asigna el arreglo de Fronteras leyendolas de un medio externo.
        Frontera[] fronteraArg = mod.getStreamFront();

        ////////////////////////////////////////
        //Asignación de las fronteras a los nodos//
        ////////////////////////////////////////

        // Arreglo donde se guardan las listas que incluyen los índices de los
        // nodos borde que pertenecen a una frontera
        List <Integer> [] listNodFront = new ArrayList[fronteraArg.length];

        //Lee la "asignación" de los nodos a las fronteras a través de la
        //interfaz de lectura de un medio externo.
        listNodFront = mod.getListStreamNodFront();

        //Recorre los nodos de la discretización para asignar a cada nodo borde
        //su correspondiente frontera
        for (Nodo nodoActual: this.discretizacion.nodo)
        {
            //Ciclo usado para recorrer las listas que contienen los índices
            //de los nodos que pertenecen a cada frontera.
            for (int i=0; i<listNodFront.length; i++)
            {
                //Verifica que el índice del nodo actual esté contenido en una
```

```
            //de las listas de las fronteras.
            if (listNodFront[i].contains(nodoActual.idNodo))
            {
                //Verifica que se trate de un nodo borde
                if(nodoActual instanceof NodoBorde)
                {
                    //Se realiza una conversión descendente al nodo actual
                    //(De un Nodo a un NodoBorde
                    NodoBorde nodoBorde = (NodoBorde)nodoActual;

                    //Se asigna al nodo borde la frontera correspondiente.
                    nodoBorde.frontera.add(fronteraArg[i]);
                }

                //Si no es un nodo borde y esta contenido en una de las
                //listas, hay un error en la definición de las fronteras.
                else
                {
                    System.out.println("Hay un error en la asignación" +
                            "de un nodo a una frontera");
                }
            }
        }
    }
    return fronteraArg;
}

/**
 * Defincón de las condiciones inciales del problema.
 * @return
 */
protected FuncionExpresion defineCondIniciales()
{
    FuncionExpresion condIniArg = new FuncionExpresion("0");

    return condIniArg;
}

/**
 * Definición de la solución exacta, si es que existe, del problema.
 * @return
 */
protected FuncionExpresion defineSolucionExacta()
{
    //Inicializa y asigna la función correspondiente a la solucion exacta.
    FuncionExpresion solExactArg = new FuncionExpresion("0");
    return solExactArg;
}

/**
 * Asigna el valor de Temperatura que se usa para acoplar la ecuación de
 * flujo de masa con la ecuación de flujo de calor. Además actualiza
 * los valores de los parámetros dependientes al valor de Temperatura.
 * @param tempArg   Vector que contiene el valor de Temperatura en cada nodo
 *                  de la discretización.
 */
public void asignaTemp(Vector tempArg)
{
    //Calcula los parámetros dependientes del valor de Temperatura

    //Define variables de apoyo
    double gFun;
    Vector lnGFun = new Vector(discretizacion.totalNodos);
    Vector gradienteTemp = new Vector(2);
    Vector gradienteLnGFun = new Vector(2);
    Matriz A= new MatrizDensa(1,2);
    Matriz C= new MatrizDensa(1,2);
    double [] b1 = new double[discretizacion.totalNodos];
    double [] b2 = new double[discretizacion.totalNodos];
    double [] f = new double[discretizacion.totalNodos];

    //Realiza una conversión descendente de la triangulación
    DominioRectangular dominio =
            (DominioRectangular) this.discretizacion.dominio;
```

```
        //Ciclo que recorre los nodos de la discretización. Este ciclo se
        //realiza por separado puesto que los valores que se calculan
        //se necesitan en todos los nodos para el posterior ciclo de
        //cálculos.
        for (int i=0; i<discretizacion.nodo.length; i++)
        {
            gFun=(1/detKSomb[i])*(trLambAst[i]/trKAst[i])*(Neta/Rhoc)*
                    (1-(Gamma*tempArg.RetornaValor(i)));
            lnGFun.AsignaValor(i, Math.log(gFun));
        }

        //Ciclo que recorre los nodos de la discretización
        for (Nodo nodAct:discretizacion.nodo)
        {
            //Calcula los gradientes en el nodo
            gradienteTemp =
                    discretizacion.evaluaGradFuncEsc(nodAct, tempArg);
            gradienteLnGFun =
                    discretizacion.evaluaGradFuncEsc(nodAct, lnGFun);

            A.asignaValor(0, 0, gradienteLnGFun.RetornaValor(0)
                   /mod.getL());
            A.asignaValor(0, 1, gradienteLnGFun.RetornaValor(1)
                   /mod.getH());
            try {
                C = A.postMultiplicaMatriz(KSomb[nodAct.idNodo]);
            } catch (Exception exception) {
                System.err.printf("%s\n\n", exception.getMessage());
            }

            //Asigna b1
            b1[nodAct.idNodo] = -(C.retornaValor(0, 0));

            //Asigna b2
            b2[nodAct.idNodo] = -(C.retornaValor(0, 1));

            //Asigna f
            f[nodAct.idNodo] = ((Ra[nodAct.idNodo]*detKSomb[nodAct.idNodo])/
                    (1-(Gamma*tempArg.RetornaValor(nodAct.idNodo))))
                    *(gradienteTemp.RetornaValor(0));
        }

        //Actualiza el valor de los coeficientes de la EDP dependientes
        //de la Temperatura

        //Realiza una conversión descendente de la FuncionValor en los tres
        //coeficientes dependientes de Stream
        FuncionValor coefB1 = (FuncionValor) this.coefEDP[3];
        FuncionValor coefB2 = (FuncionValor) this.coefEDP[4];
        FuncionValor coefF = (FuncionValor) this.coefEDP[7];

        //Asigna b1
        coefB1.valorEnNodo = b1;

        //Asigna b2
        coefB2.valorEnNodo = b2;

        //Asigna f
        coefF.valorEnNodo = f;
    }
}
```

ProblemaTemperatura.java

```
package FEM;

import ModeloConveccion.DefinicionModelo;
import ModeloConveccion.DefinicionModeloConveccion2D;
import AlgebraLineal.Matriz;
import AlgebraLineal.Vector;
import AlgebraLineal.MatrizDensa;
import java.util.List;
import java.util.ArrayList;
```

```
/**
 * Clase usada para manejar el flujo de calor en un modelo de flujo convectivo
 * definido en un dominio rectangular (2D).
 * Las fronteras y los parámetros que definen el modelo se leen a través
 * de una interfaz externa.
 *
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public class ProblemaTemperatura extends Problema{

    /** Interfaz de E/S usada para leer los parámetros que definen el modelo
     * de convección. */
    protected DefinicionModeloConveccion2D mod;

    /** Razón geométrica adimensional (H/L)*/
    protected double alphaSomb;

    /** Tensor métrico asociado al sistema coordenado adimensional (x,z)*/
    protected Matriz alpha;

    /** Tensor simétrico de conductividad térmica anisotrópica*/
    protected Matriz []LambAst;

    /** Tensor de conductividad térmica adimensional anisotrópica en 2D */
    protected Matriz []Lamb;

    /** Tensor de conductividad térmica adimensional en el sistema coordenado
     * adimensional (x,z) */
    protected Matriz []LambSomb;

    /** Vector donde se guarda el logaritmo natural de la traza de la
     * conductividad térmica anisotrópica.
     */
    protected Vector lnTrLambAst;

    /**
     * Costructor de un problema usado en el flujo de calor de un modelo
     * de convección.
     * @param discretArg    Discretización usada en el problema.
     * @param modArg        Interfaz de E/S usada para leer los parámetros
     *                      que definen el modelo de convección.
     */
    public ProblemaTemperatura(TriangulacionDominioRectangular discretArg,
            DefinicionModeloConveccion2D modArg)
    {
        super(discretArg,(DefinicionModelo) modArg);
    }

    /**
     * Establece los "coeficientes" de la EDP a partir de los parámetros de
     * que definen el flujo de calor, los cuales se leen de un medio externo.
     * @return  Un arreglo con los "coeficientes" de la EDP.
     */
    protected Funcion[] defineCoefEDP()
    {
        ////////////////////////////////////////////////////////////
        //  Se calculan los valores de los parámetros usados en la  //
        //  ecuación de flujo de calor.                             //
        ////////////////////////////////////////////////////////////

        //Matriz donde se guardarán los valores de los coeficientes de la EDP
        double [][] coef = new double[8][discretizacion.totalNodos];

        //Variable de apoyo
        double []trLambAst;

        //Realiza una conversión descendente de la triangulación
        DominioRectangular dominio =
                (DominioRectangular) this.discretizacion.dominio;

        //Obtiene y calcula los parámetros de la ecuación de calor
        alphaSomb=mod.getH()/mod.getL();
```

```
alpha = new MatrizDensa(2,2);
alpha.Inicializa(0.0);
alpha.asignaValor(0, 0, Math.sqrt(alphaSomb));
alpha.asignaValor(1, 1, 1.0/Math.sqrt(alphaSomb));
LambAst = new MatrizDensa[discretizacion.totalNodos];
Lamb = new MatrizDensa[discretizacion.totalNodos];
LambSomb = new MatrizDensa[discretizacion.totalNodos];
trLambAst = new double[discretizacion.totalNodos];
lnTrLambAst = new Vector(discretizacion.totalNodos);

//Ciclo que recorre los nodos de la discretización
for (Nodo nodAct:discretizacion.nodo)
{
    int i=nodAct.idNodo;
    LambAst[i] = new MatrizDensa(2,2);
    Lamb[i] = new MatrizDensa(2,2);
    LambSomb[i] = new MatrizDensa(2,2);

    //Leé el valor de LambAst correspondiente al nodo de la
    //discretización.
    LambAst[i]=mod.getLambAst(i);

    try
    {
        trLambAst[i]=LambAst[i].obtenerTraza();
    }
    catch (Exception exception)
    {
        System.err.printf("%s\n\n", exception.getMessage());
    }

    Lamb[i].asignaValor(0, 0, LambAst[i].retornaValor(0, 0));
    Lamb[i].asignaValor(0, 1, LambAst[i].retornaValor(0, 1));
    Lamb[i].asignaValor(1, 0, LambAst[i].retornaValor(1, 0));
    Lamb[i].asignaValor(1, 1, LambAst[i].retornaValor(1, 1));
    try
    {
        Lamb[i].multiplicaEscalar((2/trLambAst[i]));
    }
    catch (Exception exception)
    {
        System.err.printf("%s\n\n", exception.getMessage());
    }

    try
    {
        LambSomb[i] = alpha.postMultiplicaMatriz(Lamb[i]);
        LambSomb[i] = LambSomb[i].postMultiplicaMatriz(alpha);
    }
    catch (Exception exception)
    {
        System.err.printf("%s\n\n", exception.getMessage());
    }

    lnTrLambAst.AsignaValor(i, Math.log(trLambAst[i]));

    // Se asignan los valores de los coeficientes fijos de
    // la EDP en cada nodo. Además asigna un valor "basura" a los
    //coeficientes que dependen del valor de Stream.

    //Asigna a11
    coef[0][nodAct.idNodo]= LambSomb[nodAct.idNodo].retornaValor(0, 0);

    //Asigna a22
    coef[1][nodAct.idNodo]= LambSomb[nodAct.idNodo].retornaValor(1, 1);

    //Asigna a12=a21
    coef[2][nodAct.idNodo]= LambSomb[nodAct.idNodo].retornaValor(0, 1);

    //Asigna un valor "basura" a b1
    coef[3][nodAct.idNodo]= 0.0;

    //Asigna un valor "basura" a b2
    coef[4][nodAct.idNodo]= 0.0;
```

```
        //Asigna c
        coef[5][nodAct.idNodo]= 0.0;

        //Asigna alpha
        coef[6][nodAct.idNodo]= 1.0;

        //Asigna f
        coef[7][nodAct.idNodo]= (2.0 * mod.getAAst(i) * mod.getH()
                * mod.getL())
                / (trLambAst[i]*(mod.getT2()-mod.getT1()));
    }

    //Inicializa el arreglo de Funciones
    Funcion[] EDP = new Funcion[8];

    //Asigna a11
    EDP[0]= new FuncionValor(coef[0]);

    //Asigna a22
    EDP[1]= new FuncionValor(coef[1]);

    //Asigna a12=a21
    EDP[2]= new FuncionValor(coef[2]);

    //Asigna b1
    EDP[3]= new FuncionValor(coef[3]);

    //Asigna b2
    EDP[4]= new FuncionValor(coef[4]);

    //Asigna c
    EDP[5]= new FuncionValor(coef[5]);

    //Asigna alpha
    EDP[6]= new FuncionValor(coef[6]);

    //Asigna f
    EDP[7]= new FuncionValor(coef[7]);

    return EDP;
}

/**
 * Define las fronteras del problema y las asigna a los nodos borde, a
 * partir de la información obtenida de un medio externo.
 * @return Un arreglo con las fronteras del problema.
 */
protected Frontera[] defineFrontera()
{
    //Como esta es la primera rutina que se corre en el constructor
    //del problema, aquí es donde se especifica la definición del problema

    //Conversion descendente que permite usar el tipo particular
    //de definicion de modelo que usa este problema
    mod = (DefinicionModeloConveccion2D) defMod;

    ////////////////////////////////////////////
    //Definición de las fronteras del problema  //
    ////////////////////////////////////////////

    //Asigna el arreglo de Fronteras leyendolas de un medio externo.
    Frontera[] fronteraArg = mod.getTempFront();

    ////////////////////////////////////////
    //Asignación de las fronteras a los nodos//
    ////////////////////////////////////////

    // Arreglo donde se guardan las listas que incluyen los índices de los
    // nodos borde que pertenecen a una frontera
    List <Integer> [] listNodFront = new ArrayList[fronteraArg.length];

    //Lee la "asignación" de los nodos a las fronteras a través de la
    //interfaz de lectura de un medio externo.
    listNodFront = mod.getListTempNodFront();
```

```
    //Recorre los nodos de la discretización para asignar a cada nodo borde
    //su correspondiente frontera
    for (Nodo nodoActual: this.discretizacion.nodo)
    {
        //Ciclo usado para recorrer las listas que contienen los índices
        //de los nodos que pertenecen a cada frontera.
        for (int i=0; i<listNodFront.length; i++)
        {
            //Verifica que el índice del nodo actual esté contenido en una
            //de las listas de las fronteras.
            if (listNodFront[i].contains(nodoActual.idNodo))
            {
                //Verifica que se trate de un nodo borde
                if(nodoActual instanceof NodoBorde)
                {
                    //Se realiza una conversión descendente al nodo actual
                    //(De un Nodo a un NodoBorde
                    NodoBorde nodoBorde = (NodoBorde)nodoActual;

                    //Se asigna al nodo borde la frontera correspondiente.
                    nodoBorde.frontera.add(fronteraArg[i]);
                }

                //Si no es un nodo borde y esta contenido en una de las
                //listas, hay un error en la definición de las fronteras.
                else
                {
                    System.out.println("Hay un error en la asignación" +
                            "de un nodo a una frontera");
                }
            }
        }
    }

    return fronteraArg;
}

/**
 * Defínción de las condiciones inciales del problema.
 * @return
 */
protected FuncionExpresion defineCondIniciales()
{
    FuncionExpresion condIniArg = mod.getTempCondIni();
    return condIniArg;
}

/**
 * Definición de la solución exacta, si es que existe, del problema.
 * @return
 */
protected FuncionExpresion defineSolucionExacta()
{
    //Inicializa y asigna la función correspondiente a la solucion exacta.
    FuncionExpresion solExactArg =
            new FuncionExpresion("0");
    return solExactArg;
}

/**
 * Asigna el valor de Stream que se usa para acoplar la ecuación de flujo
 * de calor con la ecuación de flujo de masa. Además actualiza los valores
 * de los parámetros dependientes al valor de Stream.
 * @param Streamarg Vector que contiene el valor de Stream en cada nodo
 *                  de la discretización.
 */
public void asignaStream(Vector StreamArg)
{
    //Calcula los parámetros dependientes del valor de Stream

    //Define variables de apoyo
    Vector gradienteStream = new Vector(2);
    Vector gradienteLnTrLambAst = new Vector(2);
    Matriz A= new MatrizDensa(1,2);
    Matriz C= new MatrizDensa(1,2);
```

```
        double [] b1 = new double[discretizacion.totalNodos];
        double [] b2 = new double[discretizacion.totalNodos];

        //Realiza una conversión descendente de la triangulación
        DominioRectangular dominio =
                (DominioRectangular) this.discretizacion.dominio;

        //Ciclo que recorre los nodos de la discretización
        for (Nodo nodAct:discretizacion.nodo)
        {
            //Calcula los gradientes en el nodo
            gradienteStream =
                    discretizacion.evaluaGradFuncEsc(nodAct, StreamArg);
            gradienteLnTrLambAst =
                    discretizacion.evaluaGradFuncEsc(nodAct, lnTrLambAst);
            A.asignaValor(0, 0, gradienteLnTrLambAst.RetornaValor(0)
                    /mod.getL());
            A.asignaValor(0, 1, gradienteLnTrLambAst.RetornaValor(1)
                    /mod.getH());
            try {
                C = A.postMultiplicaMatriz(LambSomb[nodAct.idNodo]);
            } catch (Exception exception) {
                System.err.printf("%s\n\n", exception.getMessage());
            }

            //Asigna b1
            b1[nodAct.idNodo]=
                    (gradienteStream.RetornaValor(1))-(C.retornaValor(0, 0));

            //Asigna b2
            b2[nodAct.idNodo]=
                    -(gradienteStream.RetornaValor(0))-(C.retornaValor(0, 1));
        }

        //Actualiza el valor de los coeficientes de la EDP dependientes
        //de Stream

        //Realiza una conversión descendente de la FuncionValor en los dos
        //coeficientes dependientes de Stream
        FuncionValor coefB1 = (FuncionValor) this.coefEDP[3];
        FuncionValor coefB2 = (FuncionValor) this.coefEDP[4];

        //Asigna b1
        coefB1.valorEnNodo = b1;

        //Asigna b2
        coefB2.valorEnNodo = b2;
    }
}
```

TriangulacionDominioRectangular.java

```
package FEM;

import AlgebraLineal.MatrizDensa;
import AlgebraLineal.Vector;
import MetodosNumericos.Diferenciador;

/**
 * Discretización espacial de un dominio rectangular consistente en dividir el
 * dominio en triángulos conformados por 3 nodos. Todos los triángulos
 * resultantes son del mismo tamaño.
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public class TriangulacionDominioRectangular extends Discretizacion {

    /** Número de particiones en el eje x.
     * Estas particiones sirven para definir los rectángulos en que se
     * divide el dominio, rectángulos que a su vez se dividen en dos
     * para formar finalmente los triángulos.
     */
```

```
private int Nx;

/** Número de particiones en el eje z.
 * Estas particiones sirven para definir los rectángulos en que se
 * divide el dominio, rectángulos que a su vez se dividen en dos
 * para formar finalmente los triángulos.
 */
private int Nz;

/** Área de los triángulos que resultan de esta discretización.*/
private final double areaTriangulos;

/** Constructor para crear una triangulación de un dominio bidimensional
 * definiendo un número de particiones equidistantes
 * en el eje x y en el eje x.
 * @param dominioarg    Dominio bidimensional rectángular que
 *                      será discretizado.
 * @param Nxarg         Número de particiones iguales en el eje x.
 * @param Nzarg         Número de particiones iguales en el eje x.
 */
public TriangulacionDominioRectangular(DominioRectangular dominioarg,
        int Nxarg, int Nzarg)
{
    super(dominioarg);

    // Define el número de particiones en el eje x
    Nx = Nxarg;

    //Define el número de particiones en el eje x
    Nz = Nzarg;

    //Define el número de nodos que componen la triangulación
   totalNodos=(Nx+1)*(Nz+1);

    //Define el número de elementos que componen la triangulación
    totalElementos=Nx*Nz*2;

    //Incializa el arreglo de nodos que componen la triangulación
    nodo = new Nodo[totalNodos];

    //Inicializa el arreglo de elementos que componen la triangulación
    elemento = new Triangulo3Nodos[totalElementos];

    ////////////////////////////////////////////////////////
    //Definición de los nodos que componen la triangulación//
    ////////////////////////////////////////////////////////

    //Contador para numerar e identificar los nodos
    int contNodo=0;

    //Tamaño de la partición en el eje x
    double valx;

    //Tamaño de la partición en el eje x
    double valz;

    //Coordenadas del nodo en el eje x y eje x
    double[] coord= new double[2];

    //Definición de los tamaños de particiones
    valx = (dominioarg.getX2()-dominioarg.getX1()) / Nx;
    valz = (dominioarg.getZ2()-dominioarg.getZ1()) / Nz;

    //Ciclo que recorre todo el dominio definiendo los nodos, numerándolos
    //a partir de cero, de izquierda a derecha y de abajo hacia arriba
    for (int i=0; i<=Nz; i++)
    {
        for (int j=0; j<=Nx; j++)
        {
            //Cálculo de las coordenadas del nodo
            coord[0]=dominioarg.getX1()+(valx*j);
            coord[1]=dominioarg.getZ1()+(valz*i);

            //Definición del nodo cuando éste forma parte del borde del
            // dominio
```

```
        if(coord[0] ==dominioarg.getX1() ||
                coord[0] ==dominioarg.getX2() ||
                coord[1] ==dominioarg.getZ1() ||
                coord[1] ==dominioarg.getZ2())
        {
            nodo[contNodo] = new NodoBorde(contNodo, coord);
        }

        //Definición del nodo cuando éste no forma parte del borde del
        // dominio
        else
        nodo[contNodo]= new Nodo(contNodo,coord);
        contNodo++;
    }
}

////////////////////////////////////////////////////////////
//Definición de los elementos que componen la triangulación//
////////////////////////////////////////////////////////////

//Contador para numerar e identificar los elementos
int contEl=0;

//Los elementos se definirán a partir de formar
//un rectángulo formado por 4 nodos.
//A partir de cada rectángulo se formarán 2 elementos triangulares:
//Uno inferior formado por los nodos (0,1,2) en ese orden.
//Uno superior formado por los nodos (1,3,2) en ese orden
//Siendo los nodos:
//Nodo0: nodo inferior izquierdo del rectángulo
//Nodo1: nodo inferior derecho del rectángulo
//Nodo2: nodo superior izquierdo del rectángulo
//Nodo3: nodo superior derecho del rectángulo

// Índices de los nodos que serán usados para definir los elementos.
//Índice del Nodo0
int idNodo0;
//Índice del Nodo1
int idNodo1;
//Índice del Nodo2
int idNodo2;
//Índice del Nodo3
int idNodo3;

//Ciclo que recorre los nodos del dominio de izquierda a derecha
//y de abajo hacia arriba hasta un nodo antes de los bordes
//superior e inferior
for (int i=0; i<Nz; i++)
{
    for (int j=0; j<Nx; j++)
    {
        // Determina los índices de los nodos
        idNodo0=(i*(Nx+1)) + j;
        idNodo1=(i*(Nx+1)) + j+1;
        idNodo2=((i+1)*(Nx+1)) + j;
        idNodo3=((i+1)*(Nx+1)) + j+1;

        //Definición de los nodos usados para formar los elementos
        Nodo nodo0 = nodo[idNodo0];
        Nodo nodo1 = nodo[idNodo1];
        Nodo nodo2 = nodo[idNodo2];
        Nodo nodo3 = nodo[idNodo3];

        //Definición del elemento triangular superior
        elemento[contEl]=
                new Triangulo3Nodos(contEl,nodo0,nodo1,nodo2);
        contEl++;

        //Definición del elemento triangular inferior
        elemento[contEl]=
                new Triangulo3Nodos(contEl, nodo1, nodo3, nodo2);
        contEl++;
    }
}
```

```
////////////////////////////////////////////////////////////////////
//Definición del área de los elementos que componen la triangulación//
////////////////////////////////////////////////////////////////////

//Se hace una conversión descente a cualquiera de los elementos
//para obtener su área, debido a que a priori se sabe que todos
//los triángulos tendrán una misma área.
Triangulo3Nodos trianguloArea=(Triangulo3Nodos)elemento[0];
areaTriangulos = trianguloArea.calculaArea();

////////////////////////////////////////////////
//Definición de los nodos adyacentes a cada nodo//
//dentro de la triangulación.                  //
////////////////////////////////////////////////

//La definición de los nodos adyacentes se da a partir
//de llenar el arreglo "Nodo.nodoAdyacente" de la siguiente manera:
//Cada columna representa un tipo de nodo adyacente
//Columna 0: Nodo adyacente izquierdo
//Columna 1: Nodo adyacente derecho
//Columna 2: Nodo adyacente inferior
//Columna 3: Nodo adyacente superior
//Columna 4: Nodo adyacente diagonal inferior/derecho
//Columna 5: Nodo adyacente diagonal superior/iaquierdo

//Recorre los nodos de la discretización
for (Nodo nodoActual :this.nodo)
{
    //Inicializa el arreglo Nodo.nodoAdyacente
    nodoActual.nodoAdyacente= new Nodo[defineMaxNodosAdyacentes()];

    //Asigna los nodos adyacentes izquierdos siempre y cuando
    //no sea un nodo de la frontera izquierda
    if (nodoActual.coordenada[0]!=dominioarg.getX1())
    {
        nodoActual.nodoAdyacente[0]=nodo[nodoActual.idNodo-1];
        //NAD.AsignaValor(i, 0, i-1);
    }

    //Asigna los nodos adyacentes derechos siempre y cuando
    //no sea un nodo de la frontera derecha
    //if (ND.retornaValor(i, 0)!=ptIntReg2)
    if (nodoActual.coordenada[0]!=dominioarg.getX2())
    {
        nodoActual.nodoAdyacente[1]=nodo[nodoActual.idNodo+1];
        //NAD.AsignaValor(i, 1, i+1);
    }

    //Asigna los nodos adyacentes inferiores siempre y cuando
    //no sea un nodo de la frontera inferior
    //if (ND.retornaValor(i, 1)!=y1)
    if (nodoActual.coordenada[1]!=dominioarg.getZ1())
    {
        nodoActual.nodoAdyacente[2]=nodo[nodoActual.idNodo-(this.Nx+1)];
        //NAD.AsignaValor(i, 2, i-Nx);
    }

    //Asigna los nodos adyacentes superiores siempre y cuando
    //no sea un nodo de la frontera superior
    //if (ND.retornaValor(i, 1)!=y2)
    if (nodoActual.coordenada[1]!=dominioarg.getZ2())
    {
        nodoActual.nodoAdyacente[3]=nodo[nodoActual.idNodo+(this.Nx+1)];
        //NAD.AsignaValor(i, 3, i+Nx);
    }

    //Asigna los nodos adyacentes diagonal inferior siempre y cuando
    //no sea un nodo de la frontera inferior
    //o de la frontera derecha
    //if (ND.retornaValor(i, 1)!=y1 && ND.retornaValor(i, 0)!=ptIntReg2)
    if (nodoActual.coordenada[1]!=dominioarg.getZ1()&&
            nodoActual.coordenada[0]!=dominioarg.getX2())
    {
        nodoActual.nodoAdyacente[4]=nodo[nodoActual.idNodo-this.Nx];
        //NAD.AsignaValor(i, 4, i-Nx+1);
```

```
        }

        //Asigna los nodos adyacentes diagonal superior siempre y cuando
        //no sea un nodo de la frontera superior
        //o de la frontera izquierda
        //if (ND.retornaValor(i, 1)!=y2 && ND.retornaValor(i, 0)!=ptIntReg1)
        if (nodoActual.coordenada[1]!=dominioarg.getZ2()&&
                nodoActual.coordenada[0]!=dominioarg.getX1())
        {
            nodoActual.nodoAdyacente[5]=nodo[nodoActual.idNodo+this.Nx];
            //NAD.AsignaValor(i, 5, i+Nx-1);
        }
    }
}

/**
 * Establece el número de nodos que tiene cada elemento de una
 * triangulación implementada de esta manera.
 * @return  El número de nodos que tiene cada elemento: 3.
 */
public int defineNodosPorElemento()
{
    return 3;
}

/**
 * Establece el número máximo de nodos adyacentes que tiene cada nodo
 * en una triangulación implementada de esta manera.
 * @return  El número máximo de nodos adyacentes que puede llegar
 * a tener cada nodo: 6.
 */
public int defineMaxNodosAdyacentes()
{
    return 6;
}

/**
 * Establece la base que será usada por la discretización.
 * @return  Base usada en la discretización.
 */
public Base defineBase()
{
    //Define la base que se usa en este tipo de triangulación.
    Base baseArg= new BaseLinealTriangulo();
    return baseArg;
}

/** Obtiene el val de N en la dirección x*/
public int getNx()
{
   return Nx;
}

/** Obtiene el val de N en la dirección y*/
public int getNz()
{
   return Nz;
}

/** Obtiene el val del área de los triángulos que componen la
 triangulación*/
public double getAreaTriangulos()
{
   return areaTriangulos;
}

    /**
 * Obtiene los puntos de integración dentro del elemento estándar
 * triangular. En este caso se ocuparán 3 puntos de integración
 * correspondientes a los puntos medios de cada uno de los lados
 * del triángulo.
 * @return  Un arreglo bidimensional que contiene las coordenadas estándar
 *          de los puntos usados para realizar la integración.
 *          Cada renglón representa un punto (en este caso se usan 3 puntos
 *          de integración) y las columnas el val
```

```
 *          correspondiente al eje coordenado (en este caso 2).
 */
protected double [][] getPtInt()
{
    double ptInt[][];
    //Se define un arreglo bidimensional de 3x2
    ptInt = new double [3][2];

    // Punto de integración 0
    ptInt[0][0]=0.5;
    ptInt[0][1]=0.0;

    // Punto de integración 1
    ptInt[1][0]=0.0;
    ptInt[1][1]=0.5;

    // Punto de integración 2
    ptInt[2][0]=0.5;
    ptInt[2][1]=0.5;
    return ptInt;
}

/**
 * Obtiene los puntos de integración lineal sobre una frontera dentro de un
 * elemento triangular estándar. El método de integración lineal que
 * se usará es el de dos puntos.
 * @param eArg      Elemento del cual se obtendrán los puntos de integración
 * @param frontArg  Frontera de la cual se obtendrán los puntos de
 *                  integración.
 * @return  Un arreglo bidimensional de 2x2 que contiene las coordenadas
 *          estándar de los puntos de integración.
 *          Cada renglón representa un punto de integración y las columnas
 *          el val correspondiente al eje coordenado.
 */
protected double [][] getPtIntFront(Elemento eArg,
        Frontera frontArg)
{
    double ptInt[][];
    //Se define un arreglo bidimensional de 2x2. Puesto que la integración
    //sobre una frontera, en este caso particular de discretización, se
    //realiza usando dos puntos de integración.
    ptInt = new double [2][2];

    //En este caso particular de discretización, para obtener los puntos
    //de integración en la frontera, basta con verificar que los nodos
    //del elemento pertenezcan a la frontera, siendo estos nodos,
    //precisamente, los puntos de integración.

    //Variable usada para indexar los puntos de integración.
    int iPtInt=0;

    //Ciclos usados para recorrer los nodos del elemento
    for(Nodo nod:eArg.nodo)
    {
        //Comprueba que el nodo sea un nodo borde
        if (nod instanceof NodoBorde)
        {
            //Se realiza una conversión descendente de los
            //Nodos Borde
            NodoBorde nodBord = (NodoBorde) nod;

            //Recorre las fronteras a las que puede pertenecer el nodo
            for(Frontera front:nodBord.frontera)
            {
                //Si la frontera del nodo es igual a la frontera de la
                //cual se pretenden obtener los puntos de integración
                if (front==frontArg)
                {
                    //Asigna los puntos de integración dentro del elemento
                    //estándar
                    ptInt[iPtInt]=eArg.regToStd(nodBord.coordenada);
                    iPtInt++;
                    break;
                }
            }
```

```
        }
    }
    //Comprueba que se hallan encontrado dos nodos que
    //pertenezcan a la frontera. Si no es así, se trata
    //de un caso cuando el nodo solo toca en una "esquina"
    //a la frontera, en este caso la integral de línea debe
    //de ser cero. Para lograr esto, se repite el nodo
    //que tocó la frontera, para lograr posteriormente una
    //integral igual a cero.
    if (iPtInt<2)
    {
        ptInt[1][0]=ptInt[0][0];
        ptInt[1][1]=ptInt[0][1];
    }
    return ptInt;
}

/**
 * Integra sobre una frontera de un elemento de una triangulación de
 * un dominio rectangular, en este caso es la integral de línea del
 * lado del triángulo que corresponde a la frontera, usando dos
 * puntos correspondientes a dos vértices del triángulo.
 * @param eArg          Elemento sobre del cual se integrará.
 * @param ptIntStdArg   Puntos de integración dentro del elemento estándar.
 * @param fnPtIntArg    Arreglo que contiene los valores de la función a
 *                      integrar en cada uno de los puntos de integración.
 * @return      El val de la integral de la función dentro de una frontera
 *              del elemento.
 */
protected double integraFront(Elemento eArg,
        double [][]ptIntStdArg, double [] fnPtIntArg)
{
    //Longitud o distancia entre los puntos de integración.
    double lon;

    //Puntos de integración en coordenadas del elemento regular
    double ptIntReg1[]=eArg.stdToReg(ptIntStdArg[0]);
    double ptIntReg2[]=eArg.stdToReg(ptIntStdArg[1]);

    //Calcula la longitud entre los dos puntos
    lon=Math.sqrt( Math.pow(ptIntReg2[0]-ptIntReg1[0], 2)+
            Math.pow(ptIntReg2[1]-ptIntReg1[1], 2) );
    return (lon/2)* (fnPtIntArg[0]+ fnPtIntArg[1]);
}

/**
 * Obtiene el val de una Función del tipo FuncionValor
 * valuada en un punto dentro del elemento triangular.
 * @param elemArg   Elemento triangular en el cual se valuará la función.
 * @param ptStdArg  Punto dentro del elemento estándar donde se valuará
 *                  la función.
 * @param funArg    Función que será valuada.
 * @param tArg      Valor del tiempo.
 * @return          Regresa el val de la función valuada en el punto dado.
 */
protected double valuaFuncionValor(Elemento elemArg, double [] ptStdArg,
        FuncionValor funArg, double tArg)
{
    //Resultado
    double val=0;

    //Se realiza una conversión descendente del Elemento
    //(de un Elemento a un Triangulo3Nodos, que es el elemento usado en
    //este tipo particular de discretización)
    Triangulo3Nodos e = (Triangulo3Nodos)elemArg;

    //Se identifica si se trata de un triángulo superior o inferior

    //Cuando se trata de un triángulo inferior
    if (e.idElemento%2==0)
    {
        try
        {
          val = MetodosNumericos.Interpolador.Inter2DCuadReg(
             e.stdToReg(ptStdArg),
```

```
                e.nodo[0].coordenada, funArg.evaluaFuncion(e.nodo[0], tArg),
                e.nodo[1].coordenada, funArg.evaluaFuncion(e.nodo[1], tArg),
                e.nodo[2].coordenada, funArg.evaluaFuncion(e.nodo[2], tArg),
                this.nodo[e.nodo[2].idNodo+1].coordenada,
                funArg.evaluaFuncion(this.nodo[e.nodo[2].idNodo+1], tArg));
            }
            catch (Exception exception)
            {
                System.err.printf("%s\n\n", exception.getMessage());
            }
        }

        //Cuando se trata de un triángulo superior
        else
        {
            try
            {
              val = MetodosNumericos.Interpolador.Inter2DCuadReg(
                e.stdToReg(ptStdArg),
                this.nodo[e.nodo[0].idNodo-1].coordenada,
                funArg.evaluaFuncion(this.nodo[e.nodo[0].idNodo-1], tArg),
                e.nodo[0].coordenada, funArg.evaluaFuncion(e.nodo[0], tArg),
                e.nodo[2].coordenada, funArg.evaluaFuncion(e.nodo[2], tArg),
                e.nodo[1].coordenada, funArg.evaluaFuncion(e.nodo[1], tArg)
                );
            }
            catch (Exception exception)
            {
                System.err.printf("%s\n\n", exception.getMessage());
            }
        }
        return val;
    }

    /**
     * Obtiene las coordenadas de los nodos de la triangulación de un dominio
     * rectangular.
     * @return  Una MatrizDensa que contiene los valores de las coordenadas
     *          de los nodos de la triangulación del dominio rectangular.
     */
    public MatrizDensa getCoordenadas()
    {
        //Matriz donde se guardan las coordenadas de los nodos del dominio
        MatrizDensa coord = new MatrizDensa(this.totalNodos,2);

        //Variables de apoyo usadas para obtener las coordenadas de los nodos
        double x,z;

        //Conversion descendente para trabajar sobre el dominio en particular
        //que usa esta discretización
        DominioRectangular dom = (DominioRectangular) this.dominio;

        //Obtiene los valores iniciales de x y x
        x=dom.getX1();
        z=dom.getZ1();

        //Variables usadas para definir la separación entre nodos en el eje
        //x y x
        double Hx,Hz;
        Hx=dom.getL()/(this.Nx);
        Hz=dom.getH()/(this.Nz);

        //Contador usado para indexar las coordenadas de los nodos
        int contNod=0;

        for (int i=0; i<=this.Nz;i++)
        {
            for (int j=0; j<=this.Nx;j++)
            {
                coord.asignaValor(contNod, 0, x);
                coord.asignaValor(contNod, 1, z);
                contNod++;
                x=x+Hx;
            }
            x=dom.getX1();
```

```
        z=z+Hz;
    }
    return coord;
}

/**
 * Evalúa el gradiente de una función escalar en un nodo perteneciente a una
 * triangulación de un dominio rectangular.
 * @param nodoArg       Nodo en el cual se calculará el gradiente de la
 *                      función escalar.
 * @param propiedadArg  Función escalar o propiedad de la cual se obtendrá
 *                      el gradiente. Esta propiedad está dada por un vector
 *                      que tiene los valores de la función en cada uno
 *                      de los nodos de la discretización.
 * @return              Un vector con el gradiente de la función escalar
 *                      calculado en un nodo en particular de la
 *                      triangulación.
 */
protected Vector evaluaGradFuncEsc(Nodo nodoArg,
        Vector propiedadArg)
{
    //Variable de apoyo para obtener resultados
    double val;

    //Vector donde se guardará el valor del gradiente.
    Vector gradiente= new Vector(2);

    //Se realiza una conversión descendente del Dominio Rectangular
    DominioRectangular domRect = (DominioRectangular) this.dominio;

    //Dependiendo de la ubicación del nodo en la triengulación, se realizará
    //el proceso de obtención del gradiente.

    //Se obtiene el valor del gradiente en la dirección del eje X
    //Si el nodo en cuestión pertenece al borde izquierdo del dominio.
    if (nodoArg.coordenada[0]==domRect.getX1())
    {
        //Variables usadas para definir los puntos usados para obtener
        //el gradiente
        double pki[], pkiMas1[], pkiMas2[];

        //Variables usadas para definir los valores de la función en los
        //puntos usados para obtener el gradiente
        double fpki, fpkiMas1, fpkiMas2;

        //Nodos que serán usados para calcular el gradiente
        Nodo nodoi, nodoiMas1, nodoiMas2;

        //Asigna los nodos
        nodoi= nodoArg;
        nodoiMas1= nodoi.nodoAdyacente[1];
        nodoiMas2= nodoiMas1.nodoAdyacente[1];

        //Asigna los puntos
        pki = nodoi.coordenada;
        pkiMas1=nodoiMas1.coordenada;
        pkiMas2=nodoiMas2.coordenada;

        //Asigna los valores de la función
        fpki=propiedadArg.RetornaValor(nodoi.idNodo);
        fpkiMas1=propiedadArg.RetornaValor(nodoiMas1.idNodo);
        fpkiMas2=propiedadArg.RetornaValor(nodoiMas2.idNodo);

        //Obtiene la diferenciación numérica hacia adelante usando 3 puntos
        try
        {
            val=Diferenciador.difAdelante(pki[0], fpki, pkiMas1[0],
                    fpkiMas1, pkiMas2[0], fpkiMas2);
            gradiente.AsignaValor(0, val);
        }
        catch (Exception exception)
        {
            System.err.printf("%s\n\n", exception.getMessage());
        }
    }
```

```
//Si el nodo en cuestión pertenece al borde derecho del dominio.
else if (nodoArg.coordenada[0]==domRect.getX2())
{
    //Variables usadas para definir los puntos usados para obtener
    //el gradiente
    double pki[], pkiMenos1[], pkiMenos2[];

    //Variables usadas para definir los valores de la función en los
    //puntos usados para obtener el gradiente
    double fpki, fpkiMenos1, fpkiMenos2;

    //Nodos que serán usados para calcular el gradiente
    Nodo nodoi, nodoiMenos1, nodoiMenos2;

    //Asigna los nodos
    nodoi= nodoArg;
    nodoiMenos1= nodoi.nodoAdyacente[0];
    nodoiMenos2= nodoiMenos1.nodoAdyacente[0];

    //Asigna los puntos
    pki = nodoi.coordenada;
    pkiMenos1=nodoiMenos1.coordenada;
    pkiMenos2=nodoiMenos2.coordenada;

    //Asigna los valores de la función
    fpki=propiedadArg.RetornaValor(nodoi.idNodo);
    fpkiMenos1=propiedadArg.RetornaValor(nodoiMenos1.idNodo);
    fpkiMenos2=propiedadArg.RetornaValor(nodoiMenos2.idNodo);

    //Obtiene la diferenciación numérica hacia atras usando 3 puntos
    try
    {
        val=Diferenciador.difAtras(pki[0], fpki, pkiMenos1[0],
                fpkiMenos1, pkiMenos2[0], fpkiMenos2);
        gradiente.AsignaValor(0, val);
    }
    catch (Exception exception)
    {
        System.err.printf("%s\n\n", exception.getMessage());
    }
}

//Si el nodo no pertenece ni al borde izquierdo ni derecho
else
{
    //Variables usadas para definir los puntos usados para obtener
    //el gradiente
    double x0[], x0MasH[], x0MenosH[];

    //Variables usadas para definir los valores de la función en los
    //puntos usados para obtener el gradiente
    double fx0MasH, fx0MenosH;

    //Nodos que serán usados para calcular el gradiente
    Nodo nodox0, nodox0MasH, nodox0MenosH;

    //Asigna los nodos
    nodox0= nodoArg;
    nodox0MasH= nodox0.nodoAdyacente[1];
    nodox0MenosH= nodox0.nodoAdyacente[0];

    //Asigna los puntos
    x0 = nodox0.coordenada;
    x0MasH=nodox0MasH.coordenada;
    x0MenosH=nodox0MenosH.coordenada;

    //Asigna los valores de la función
    fx0MasH=propiedadArg.RetornaValor(nodox0MasH.idNodo);
    fx0MenosH=propiedadArg.RetornaValor(nodox0MenosH.idNodo);

    //Obtiene la diferenciación numérica central usando 3 puntos
    try
    {
        val=Diferenciador.difCentral(x0[0], x0MenosH[0], fx0MenosH,
                x0MasH[0], fx0MasH);
```

```
            gradiente.AsignaValor(0, val);
        }
        catch (Exception exception)
        {
            System.err.printf("%s\n\n", exception.getMessage());
        }
    }
    //Se obtiene el valor del gradiente en la dirección del eje Z
    //Si el nodo en cuestión pertenece al borde inferior del dominio.
    if (nodoArg.coordenada[1]==domRect.getZ1())
    {
        //Variables usadas para definir los puntos usados para obtener
        //el gradiente
        double pki[], pkiMas1[], pkiMas2[];

        //Variables usadas para definir los valores de la función en los
        //puntos usados para obtener el gradiente
        double fpki, fpkiMas1, fpkiMas2;

        //Nodos que serán usados para calcular el gradiente
        Nodo nodoi, nodoiMas1, nodoiMas2;

        //Asigna los nodos
        nodoi= nodoArg;
        nodoiMas1= nodoi.nodoAdyacente[3];
        nodoiMas2= nodoiMas1.nodoAdyacente[3];

        //Asigna los puntos
        pki = nodoi.coordenada;
        pkiMas1=nodoiMas1.coordenada;
        pkiMas2=nodoiMas2.coordenada;

        //Asigna los valores de la función
        fpki=propiedadArg.RetornaValor(nodoi.idNodo);
        fpkiMas1=propiedadArg.RetornaValor(nodoiMas1.idNodo);
        fpkiMas2=propiedadArg.RetornaValor(nodoiMas2.idNodo);

        //Obtiene la diferenciación numérica hacia adelante usando 3 puntos
        try
        {
            val=Diferenciador.difAdelante(pki[1], fpki, pkiMas1[1],
                    fpkiMas1, pkiMas2[1], fpkiMas2);
            gradiente.AsignaValor(1, val);
        }
        catch (Exception exception)
        {
            System.err.printf("%s\n\n", exception.getMessage());
        }
    }

    //Si el nodo en cuestión pertenece al borde superior del dominio.
    else if (nodoArg.coordenada[1]==domRect.getZ2())
    {
        //Variables usadas para definir los puntos usados para obtener
        //el gradiente
        double pki[], pkiMenos1[], pkiMenos2[];

        //Variables usadas para definir los valores de la función en los
        //puntos usados para obtener el gradiente
        double fpki, fpkiMenos1, fpkiMenos2;

        //Nodos que serán usados para calcular el gradiente
        Nodo nodoi, nodoiMenos1, nodoiMenos2;

        //Asigna los nodos
        nodoi= nodoArg;
        nodoiMenos1= nodoi.nodoAdyacente[2];
        nodoiMenos2= nodoiMenos1.nodoAdyacente[2];

        //Asigna los puntos
        pki = nodoi.coordenada;
        pkiMenos1=nodoiMenos1.coordenada;
        pkiMenos2=nodoiMenos2.coordenada;

        //Asigna los valores de la función
```

```
            fpki=propiedadArg.RetornaValor(nodoi.idNodo);
            fpkiMenos1=propiedadArg.RetornaValor(nodoiMenos1.idNodo);
            fpkiMenos2=propiedadArg.RetornaValor(nodoiMenos2.idNodo);

            //Obtiene la diferenciación numérica hacia atras usando 3 puntos
            try
            {
                val=Diferenciador.difAtras(pki[1], fpki, pkiMenos1[1],
                        fpkiMenos1, pkiMenos2[1], fpkiMenos2);
                gradiente.AsignaValor(1, val);
            }
            catch (Exception exception)
            {
                System.err.printf("%s\n\n", exception.getMessage());
            }
        }

        //Si el nodo no pertenece ni al borde inferior ni superior
        else
        {
            //Variables usadas para definir los puntos usados para obtener
            //el gradiente
            double x0[], x0MasH[], x0MenosH[];

            //Variables usadas para definir los valores de la función en los
            //puntos usados para obtener el gradiente
            double fx0MasH, fx0MenosH;

            //Nodos que serán usados para calcular el gradiente
            Nodo nodox0, nodox0MasH, nodox0MenosH;

            //Asigna los nodos
            nodox0= nodoArg;
            nodox0MasH= nodox0.nodoAdyacente[3];
            nodox0MenosH= nodox0.nodoAdyacente[2];

            //Asigna los puntos
            x0 = nodox0.coordenada;
            x0MasH=nodox0MasH.coordenada;
            x0MenosH=nodox0MenosH.coordenada;

            //Asigna los valores de la función
             fx0MasH=propiedadArg.RetornaValor(nodox0MasH.idNodo);
            fx0MenosH=propiedadArg.RetornaValor(nodox0MenosH.idNodo);

            //Obtiene la diferenciación numérica central usando 3 puntos
            try
            {
                val=Diferenciador.difCentral(x0[1], x0MenosH[1], fx0MenosH,
                        x0MasH[1], fx0MasH);
                gradiente.AsignaValor(1, val);
            }
            catch (Exception exception)
            {
                System.err.printf("%s\n\n", exception.getMessage());
            }
        }
        return gradiente;
    }
}
```

Triangulo3Nodos.java

```
package FEM;

/**
 * Elemento bidimensional triangular, formado por 3 nodos coincidentes
 * en los vértices del triángulo.
 * @author Conrado Montealegre
 * @version 1.0
 * @since 1.0
 */
public class Triangulo3Nodos extends Elemento {
```

```
/**
 * Constructor para crear un elemento bidimensional
 * triangular formado por 3 nodos.
 *
 * @param idElementoArg Número entero que identifica al elemento.
 * @param nodo0Arg  Nodo 0 que conforma el elemento triangular
 * @param nodo1Arg  Nodo 1 que conforma el elemento triangular
 * @param nodo2Arg  Nodo 2 que conforma el elemento triangular
 */
public Triangulo3Nodos(int idElementoArg, Nodo nodo0Arg, Nodo nodo1Arg, Nodo nodo2Arg)
{
    super(idElementoArg);
    nodo= new Nodo[3];
    nodo[0]=nodo0Arg;
    nodo[1]=nodo1Arg;
    nodo[2]=nodo2Arg;
}

/**
 * Calcula el área del triángulo.
 * @return  el área del triángulo.
 */
public double calculaArea()
{
    return(Math.abs((nodo[2].coordenada[1]-nodo[0].coordenada[1]))*
                    Math.abs((nodo[2].coordenada[0]-nodo[1].coordenada[0])))
                    /2;
}

/**
 * Calcula el Jacobiano del elemento.
 * @return  el Jacobiano del elemento.
 */
public double calculaJacobiano()
{
    return(2.0*this.calculaArea());
}

/**
 * Calcula la matriz M (de transformación) del elemento actual.
 * @return  una arreglo bidimensional de 2x2 con los valores de M.
 */
protected double [][] calculaM()
{
    double [][] M = new double[2][2];
    double area=this.calculaArea();

    //Asigna las coordenadas de los vertices del elemento
    double v0[] = new double[2];
    double v1[] = new double[2];
    double v2[] = new double[2];
    v0=this.nodo[0].coordenada;
    v1=this.nodo[1].coordenada;
    v2=this.nodo[2].coordenada;

    //Calcula la matriz M (de transformación) del elemento actual
    M[0][0]=(1.0/(2.0*area))*(v2[1]-v0[1]);
    M[0][1]=-(1.0/(2.0*area))*(v2[0]-v0[0]);
    M[1][0]=-(1.0/(2.0*area))*(v1[1]-v0[1]);
    M[1][1]=(1.0/(2.0*area))*(v1[0]-v0[0]);
    return M;
}

/**
 * Va de coordenadas en el elemento estándar
 * a coordenadas en el elemento regular.
 * @param coordElemEst  Coordenadas dentro del elemento estandar.
 * @return  Coordenadas dentro del elemento regular.
 */
public double []stdToReg (double[]coordElemEst)
{
    double coordElemReg[]= new double [2];

    //Asigna las coordenadas de los vertices del elemento
```

```
    double v0[] = new double[2];
    double v1[] = new double[2];
    double v2[] = new double[2];
    v0=this.nodo[0].coordenada;
    v1=this.nodo[1].coordenada;
    v2=this.nodo[2].coordenada;
    coordElemReg[0]= (v0[0]*(1.0-coordElemEst[0]-coordElemEst[1]))+
            (v1[0]*coordElemEst[0])+(v2[0]*coordElemEst[1]);
    coordElemReg[1]= (v0[1]*(1.0-coordElemEst[0]-coordElemEst[1]))+
            (v1[1]*coordElemEst[0])+(v2[1]*coordElemEst[1]);
    return coordElemReg;
}

/**
 * Va de coordenadas en el elemento regular
 * a coordenadas en el elemento estándar.
 * @param coordElemReg  Coordenadas dentro del elemento regular.
 * @return  Coordenadas dentro del elemento estandar.
 */
public double []regToStd (double[]coordElemReg)
{
    double coordElemEst[]= new double [2] ;

    //Asigna las coordenadas de los vertices del elemento
    double v0[] = new double[2];
    v0=this.nodo[0].coordenada;

    //Asigna M (matriz de transformación)
    double [][]M = new double[2][2];
    M=this.calculaM();
    coordElemEst[0]= (M[0][0]*(coordElemReg[0]-v0[0]))+
            (M[0][1]*(coordElemReg[1]-v0[1]));
    coordElemEst[1]= (M[1][0]*(coordElemReg[0]-v0[0]))+
            (M[1][1]*(coordElemReg[1]-v0[1]));

    if (coordElemEst[0]<0)
    {
        coordElemEst[0]=0.0;
    }
    if(coordElemEst[1]<0)
    {
        coordElemEst[1]=0.0;
    }
    if (coordElemEst[0]>1)
    {
        coordElemEst[0]=1.0;
    }
    if(coordElemEst[1]>1)
    {
        coordElemEst[1]=1.0;
    }
    return coordElemEst;
}

/**
 * Integra dentro del elemento. En este caso realiza la integración de una
 * función dentro de un triángulo, usando 3 puntos de integración
 * correspondientes a los puntos medios de cada uno de los lados del
 * triángulo.
 * @param fnPtInt   Arreglo que contiene los valores de la función a
 *                  integrar en cada uno de los puntos de integración.
 * @return  El valor de la integración de la función dentro del triángulo.
 */
public double integraElem(double [] fnPtInt)
{
    //Variable usada para guardar el resultado de la integración.
    double resultado=0;

    //Realiza la integración dentro del triángulo estándar, usando los
    //puntos medios de cada uno de los lados del triángulo.
    try
    {
        resultado= MetodosNumericos.Integrador.integraTriangulo(0.5,
                fnPtInt[0],fnPtInt[1],fnPtInt[2]);
    }
```

```
        catch (Exception exception)
        {
            System.err.printf("%s\n\n", exception.getMessage());
        }
        return resultado*this.calculaJacobiano();
    }
}
```