UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

POSGRADO EN CIENCIAS DE LA TIERRA
INSTITUTO DE GEOFÍSICA
MODELACIÓN MATEMÁTICA

MODELACIÓN BIDIMENSIONAL DE FLUJOS CONVECTIVOS EN MEDIOS POROSOS

T E S I S

QUE PARA OPTAR POR EL GRADO DE:
MAESTRA EN CIENCIAS DE LA TIERRA

PRESENTA:
ALMA MARTÍNEZ NICOLÁS

Directora de Tesis:
Dra. Elsa Leticia Flores Márquez
Instituto de Geofísica

Ciudad Universitaria, CDMX
Octubre 2021

A mi familia

Agradecimientos

El presente trabajo de investigación fue posible gracias a un esfuerzo conjunto y a la generosa ayuda de muchas personas e instituciones:

Al Consejo Nacional de Ciencia y Tecnología (CONACyT), a través del Programa Nacional de Posgrados de Calidad (PNPC). Agradezco la beca otorgada que disfruté durante 2 años.

Al Programa de Posgrado en Ciencias de la Tierra - UNAM y a su personal e instituciones participantes, en especial al Instituto de Geofísica, que me abrió las puertas para desarrollar mi trabajo de investigación.

A mi directora de tesis, Dra. Elsa Leticia Flores Márquez, por su tiempo, paciencia, observaciones y recomendaciones durante la elaboración de este trabajo. Es una motivación para mí, su compromiso y dedicación para hacer investigación científica de calidad y formar profesionales. También agradezco la confianza brindada, su consejo y apoyo en una etapa muy difícil de mi vida.

Al Comité Tutor y Jurado de Examen, conformados por: Dr. David Parra Guevara, Dra. Claudia Arango Galván, Dra. Elsa Leticia Flores Márquez, Dr. Ernesto Rubio Acosta y Dr. Óscar Valdivieso Mijangos. Gracias por su tiempo, observaciones, recomendaciones e imprescindibles comentarios para mejorar mi trabajo.

A mis profesores del Posgrado en Ciencias de la Tierra que ayudaron en mi formación a lo largo de estos años, sus conocimientos y experiencia sirvieron también de inspiración para realizar mi trabajo de investigación.

A Conrado Montealegre, por su importante colaboración en el precedente del desarrollo computacional de mi tesis.

A Yitzhak Gómez, por su amable ayuda para resolver los problemas técnicos de mi computadora.

A mis compañeros de Posgrado, por la convivencia tanto en la Universidad como fuera de ella, por su amable ayuda, por su confianza y por hacer más agradable este tiempo en la maestría. Agradezco especialmente a Alicia y Martín la amistad brindada, me llevo por siempre gratos recuerdos de ellos.

A Araceli Chaman, Erika Ulloa, Gloria Alba, Lupita Ávila y Carlos Contreras, por su apoyo administrativo durante mis estudios de posgrado.

A mi papá, Agustín Martínez, quien es para mí un ejemplo de responsabilidad, de generosidad, de motivación por seguir aprendiendo y de que hay que seguir adelante a pesar de las adversidades. Gracias por todo, papá.

A mi mamá, Macrina Nicolás (†). Aunque te fuiste poco antes de que comenzara esta etapa, sabes que eres parte de ella, como siempre. Las últimas conversaciones que tuvimos me permitieron conocerte más y entender tu historia de vida un poco mejor. Gracias por todo, mamá.

A mis hermanas Adri, Gaby y Nancy, porque sé que siempre he contado con su apoyo.

A mis amigas Araceli y Maricela, gracias por estar conmigo, siempre.

A la Universidad Nacional Autónoma de México...

Muchas Gracias

Índice General

Índice de Figuras

Índice de Tablas

Resumen

La convección es un mecanismo de transferencia de calor que está presente tanto en la naturaleza como en algunos procesos de ingeniería. En ambos casos, para que este mecanismo ocurra se requiere de algún fluido que transporte la masa fluida y la energía calorífica; por esta razón el estudio del flujo convectivo es indispensable para comprender la evolución térmica de las formaciones geológicas que pueden contener fluidos, como: agua, vapor, gas e hidrocarburos. Con los procedimientos adecuados de la modelación de sistemas se puede predecir el comportamiento de un sistema en particular y tomar decisiones convenientes sobre su evolución o explotación.

Las ecuaciones matemáticas que representan al sistema físico a modelar pueden resolverse de manera analítica cuando el sistema es sencillo; sin embargo, cuando se quiere modelar un sistema que tenga consideraciones e hipótesis más cercanas a la realidad, las ecuaciones se vuelven complicadas y es aquí cuando los métodos numéricos se vuelven un recurso imprescindible para incorporarse al modelo como solucionador de las ecuaciones. Uno de estos métodos numéricos es el método de elemento finito. Entre más heterogeneidad y anisotropía haya en la naturaleza, el modelo requerirá más puntos que representan propiedades desconocidas así que los cálculos serán numerosos; por lo que será bastante útil crear algún código programable que resuelva el sistema de ecuaciones generado.

En particular, los modelos de flujo convectivo involucran el transporte de masa en conjunto con el transporte de calor. Este tipo de modelos abarcan normalmente el estudio de áreas extensas o de nivel regional, las cuales por naturaleza tienden a ser bastante heterogéneas y presentar además un comportamiento anisotrópico. Los modelos de convección requieren resolver numericamente las ecuaciones de balance de las propiedades de los fluidos (masa y energía), así como las ecuaciones que rigen el movimiento de los mismos en el medio, que puede ser poroso o fracturado.

El objetivo de la presente investigación es modelar numéricamente un flujo convectivo de fluido entre un medio poroso y una fractura, desarrollando un programa numérico computacional que resuelva las ecuaciones que rigen este fenómeno. Para ello se presentan los fundamentos teóricos de la convección natural. También se describen los fundamentos del método de elementos finitos, el cual se ocupa para resolver numéricamente las ecuaciones de flujo convectivo. Posteriormente, se consideró un sistema geotérmico subterráneo de forma rectangular, constituido por un medio poroso heterogéneo y anisotrópico, saturado con un fluido de una fase. Dicho sistema está atravesado por una fractura que es representada como una zona de muy alta permeabilidad. Se realizaron varios ensayos numéricos que esquematizan condiciones geológicas reales, en los ejemplos se varía la inclinación de la fractura, así como las propiedades físicas del medio poroso, como conductividad térmica y/o permeabilidad para las distintas capas horizontales. Los resultados obtenidos son congruentes con modelos previos y de laboratorio, así mismo los modelos esquemáticos muestran un comportamiento esperado para los campos de temperaturas y de corrientes.

Abstract

Convection is a heat transfer mechanism that is present both in nature and in some engineering processes. In both cases, for this mechanism to occur, some fluid is required to transfer fluid mass and heat. Because of this reason, the study of convective flow is essential to understand the thermal evolution of geological formations that can contain fluids, such as: water, steam, gas and hydrocarbons. With proper system modeling procedures, you can predict the behavior of a particular system and make appropriate decisions about its evolution or exploitation.

The mathematical equations that represent the physical system to be modeled can be solved analytically when the system is simple; however, when you want to model a system with specific considerations and hypotheses, the equations become complicated and then the numerical methods become an essential resource to be incorporated into the model as a solver of the equations. One of these numerical methods is the finite element method. While more heterogeneity and anisotropy are in nature, more points that represent unknown properties will be required by the model, so the calculations will be numerous; so it will be quite useful to create some programmable code that solves the generated system of equations.

In particular, convective flow models involve mass transport in conjunction with heat transport. This type of model normally encompasses the study of large or regional areas, which by nature tend to be quite heterogeneous and also present an anisotropic behavior. Convection models require numerically solving the balance equations of the properties of fluids (mass and energy), as well as the equations that govern their movement in the medium, which can be porous or fractured.

The objective of this research is to numerically model a convective fluid flow between a porous medium and a fracture, developing a computational numerical program that solves the equations

that govern this phenomenon. For this, the theoretical foundations of natural convection are presented. It also describes the fundamentals of the finite element method, which is used to solve the equations of convective flow numerically. Subsequently, an underground geothermal system of rectangular shape was considered, consisting of a heterogeneous and anisotropic porous medium, saturated with a one-phase fluid. Said system is traversed by a fracture that is represented as a zone of very high permeability. Several numerical tests were carried out that outline real geological conditions, in the examples the inclination of the fracture is varied, as well as the physical properties of the porous medium, such as thermal conductivity and / or permeability for the different horizontal layers. The results obtained are consistent with previous and laboratory models, likewise the schematic models show an expected behavior for the temperature and streamlines.

Capítulo 1

Introducción

1.1 Antecedentes y Motivación

Uno de los mecanismos de transferencia de calor es la convección, que está presente tanto en la naturaleza como en algunos procesos industriales o de ingeniería. En ambos casos, para que el fenómeno de convección ocurra se requiere de algún fluido para el transporte de la propia masa fluida y la energía calorífica; por esta razón el estudio del flujo convectivo es indispensable para comprender la evolución térmica de las formaciones geológicas, que pueden contener fluidos tales como agua, vapor, gas e hidrocarburos. El estudio del flujo de fluidos y transferencia de calor en medios porosos, se ha venido realizando con éxito gracias al uso de la modelación de sistemas terrestres. Algunos casos de estudio pueden hallarse en [Báez et al., 2004], [Báez y Nicolás, 2006], [Báez, 2008], [Basak et al., 2006], [Deng y Tang, 2002], [Evans y Raffensperger, 1992], [Flores-Márquez, 1992], [Kumar et al., 2015], [McKibbin y Tyvand, 1982], [Montealegre, 2010], [Richard y Gounot, 1981], [Royer y Flores-Márquez, 1994], [Zhang et al., 2016], [Zhao et al., 2019], entre otros.

La modelación de sistemas es un procedimiento en el cual se construyen modelos o esquemas de carácter físico, matemático, numérico y computacional, los cuales son una aproximación o sustituto, cuyo comportamiento imita al del sistema en estudio. 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 sistema, heterogeneidad y anisotropía de los sedimentos, mecanismos de transporte de partículas y reacciones químicas. Es debido a estas

hipótesis y a la incertidumbre en los valores de los datos requeridos por el modelo, que éste debe ser visto como una aproximación y no como un duplicado exacto de las condiciones de campo.

Las ecuaciones matemáticas que representan al sistema físico a modelar pueden resolverse de manera analítica cuando el sistema es sencillo; sin embargo, cuando se quiere modelar un sistema que tenga consideraciones e hipótesis muy específicas, las ecuaciones se vuelven complicadas y es aquí cuando los métodos numéricos se vuelven un recurso imprescindible para incorporarse al modelo como solucionador de las ecuaciones. Uno de estos métodos numéricos es el método del elemento finito. Entre más heterogeneidad y anisotropía haya en la naturaleza, el modelo requerirá más puntos que representan propiedades desconocidas, así que los cálculos serán numerosos; por lo que será bastante útil crear algún código programable que resuelva el sistema de ecuaciones generado.

El análisis, comprensión y aplicaciones derivadas del fenómeno de flujo convectivo en un medio poroso saturado con un fluido, son una importante motivación para desarrollar una investigación en esta área. Conocer el campo de temperaturas y de flujo de fluidos (stream) en un sistema, con condiciones específicas en espacio y tiempo, tiene aplicaciones de particular interés para las Ciencias de la Tierra.

En el aspecto matemático, la formulación del problema de convección y su forma de solución presentados en este trabajo constituyen una gran motivación de estudio, debido a que se usa el mismo algoritmo para hallar las funciones desconocidas de temperatura y stream. Esto representa un ahorro considerable de los recursos computacionales y, por lo tanto, una gran ventaja sobre otros trabajos hallados en la literatura, en los cuales se presenta un método de solución numérica diferente para cada función involucrada en el problema de convección natural en medios porosos.

1.2 Justificación

En particular, los modelos de flujo convectivo involucran el transporte de masa en conjunto con el transporte de calor. Este tipo de modelos abarcan normalmente el estudio de áreas extensas o de nivel regional, las cuales por naturaleza tienden a ser bastante heterogéneas y presentan, además,

un comportamiento anisotrópico. Para hacer modelos de convección, se requiere resolver de manera numérica las ecuaciones de balance de las propiedades de los fluidos (masa y energía), así como las ecuaciones que rigen el movimiento de los mismos en el medio, que puede ser poroso o fracturado. Los resultados obtenidos para el modelo de convección, en particular, tienen muchas aplicaciones en geofísica, geohidrología, geotermia, modelos de petróleo negro y algunos procesos de recuperación mejorada de petróleo, entre otros.

En el presente trabajo se considera como hipótesis un sistema geotérmico subterráneo, constituido por un medio poroso y una fractura, que también puede ser representada como un elemento de muy alta permeabilidad que atraviesa al medio poroso. Se consideran también las ecuaciones de balance para el estudio de las propiedades físicas, fluido-dinámicas y térmicas de estos medios y del fluido que en este caso es agua.

1.3 Objetivo

El objetivo de la presente investigación es modelar numéricamente un flujo convectivo de fluido entre un medio poroso y una fractura, desarrollando un programa numérico computacional que resuelva las ecuaciones que rigen este fenómeno. Para ello utilizaremos un algoritmo basado en el Método del Elemento Finito que proporcione la solución numérica a la ecuación de Darcy acoplada con la ecuación de calor. Se modelará el comportamiento del flujo convectivo para distintos casos que representen condiciones geológicas reales; con lo cual se obtendrá la distribución de temperaturas y el comportamiento de flujo para un sistema con características particulares de permeabilidad, conductividad térmica, razón geométrica, número de Rayleigh, inclinación de la falla, heterogeneidad y anisotropía.

Para cumplir este objetivo se realizó una implementación computacional del problema de convección modificando algunas clases (en Java - NetBeans) del programa presentado por [Montealegre, 2010] y se crearon otras clases más. Posteriormente, se realizaron ensayos numéricos para reproducir ejemplos sintéticos y de laboratorio que permitieran validar las salidas del programa. En una tercera etapa se realizaron ensayos numéricos que representaran condiciones geológicas reales; los primeros con geometrías cuadradas y posteriormente, además de cambiar las razones geométricas

alto/ancho, se introdujeron varios estratos y fracturas con diferentes inclinaciones y anchos.

Este trabajo está conformado por seis capítulos y tres apéndices. El Capítulo 1 corresponde a la presente introducción, el Capítulo 2 corresponde a los fundamentos teóricos de la convección y su modelación matemática. Los capítulos 3 y 4 corresponden, respectivamente, a la solución numérica y a la implementación computacional del problema de convección. En el Capítulo 5 se presentan ensayos numéricos que podrían representar situaciones geológicas reales de convección en medios porosos. Finalmente, en el Capítulo 6 se presentan las conclusiones de esta tesis.

Los apéndices A, B y C corresponden a las propiedades físicas del medio poroso, a la nomenclatura utilizada y a los códigos programables, respectivamente.

Capítulo 2

Modelación Matemática de la Convección

En esta capítulo se hace una breve introducción a la modelación matemática de los sistemas terrestres, se presentan los fundamentos teóricos de la convección natural en medios porosos, así como la formulación matemática adimensional de las ecuaciones que gobiernan este fenómeno.

2.1 Modelación de Sistemas Terrestres

Todas las propiedades físicas del planeta están sujetas a cambios tanto en el espacio como en el tiempo, en particular el calor y los fluidos en su interior, presentan una dinámica compleja de interés en investigación y en aplicaciones energéticas, como por ejemplo, la caracterización y explotación de un campo geotérmico o petrolero; algunas de ellas pueden describirse de manera aproximada mediante modelos que interactúan entre ellos siguiendo el método científico, con el objetivo de lograr una mejor comprensión de dichos fenómenos.

El **modelo de un sistema** es un sustituto que representa al fenómeno, con sus condiciones, del sistema original o real; por lo tanto del comportamiento del modelo del sistema es posible derivar el correspondiente al sistema original. Para construir el modelo integral de un sistema, se empieza por generar el *Modelo Conceptual*, que contiene a las leyes físicas, químicas, biológicas, etc. que rigen los fenómenos involucrados del problema científico de interés y un conjunto de hipótesis que delimitan ese problema. Uno de los puntos más importantes para la delimitación, es indicar como varía un fenómeno en el dominio del espacio y el tiempo.

Las variaciones del fenómeno de interés, que ya han sido conceptualizadas y delimitadas para su estudio, se representan mediante ecuaciones diferenciales en un *Modelo Matemático* con condiciones específicas para el fenómeno. Estas ecuaciones son por lo general de difícil solución analítica así que se deben aplicar técnicas que aproximen una solución a las mismas. Dichas técnicas son conocidas como métodos numéricos, están incluidas en el llamado *Modelo Numérico* y se pueden implementar en un código computacional cuya resolución es, relativamente, más sencilla de encontrar.

Por último en el *Modelo Computacional* se resuelven las ecuaciones previamente obtenidas en el modelo numérico con un código específico según las características del software utilizado. Una de las ventajas principales es el ahorro de recursos gracias a que estos programas solucionan y adaptan cambios de las propiedades en grandes sistemas de ecuaciones en un periodo corto en comparación con el tiempo que llevaría resolver las ecuaciones de manera analítica o algebraica.

La modelación matemática, llamando así a la integración de todos los modelos mencionados anteriormente, es un recurso muy utilizado en ciencias e ingeniería debido a que constituye el método más efectivo de predecir el comportamiento de diversos sistemas de interés. En nuestro país son usados ampliamente en la industria petrolera, exploración geotérmica, exploración geofísica y en muchas otras disciplinas. El presente trabajo está enfocado en la modelación de la convección en medios porosos, así que conviene empezar por definir (conceptualizar) a los sistemas geotérmicos que es donde se presentan este de tipo de fenómenos, aunque no son los únicos.

Los sistemas geotérmicos son una variedad de combinaciones de las características geológicas, físicas y químicas que dan origen a diferentes tipos de estos sistemas. Por otro lado, se dice que existe un yacimiento geotérmico, cuando en un área geográfica concreta se dan determinadas condiciones geológicas y geotérmicas favorables para que se puedan explotar de forma económica los recursos geotérmicos del subsuelo. Una descripción detallada de los tipos de sistemas geotérmicos y yacimientos se encuentra en [Calzada y Olivares, 2015].

El Instituto Nacional de Electricidad y Energías Limpias (antes Instituto de Investigaciones Eléctricas) presenta la siguiente clasificación de los sistemas geotérmicos en la naturaleza: hidrotermales, de roca seca caliente (sistemas geotérmicos mejorados, EGS), geopresurizados, marinos y magmáticos [Arellano-Gómez et al., 2008].

De particular interés son los sistemas hidrotermales, los cuales están constituidos por una fuente de calor, un fluido (agua en fase líquida o vapor) y la roca en donde se almacena el fluido. Estos sistemas, a su vez, pueden clasificarse en tres tipos principales: vapor dominante, líquido dominante de alta entalpía y líquido dominante de baja entalpía. Por otro lado, dentro de la clasificación de yacimientos, los sistemas hidrotermales incluyen los siguientes tipos de yacimientos: dominados por líquido, dominados por líquido con capa de vapor, dominados por vapor y yacimientos de baja permeabilidad (dominados por roca).

La modelación de sistemas geotérmicos, como los descritos anteriormente, requiere el estudio teórico de los procesos de transferencia de calor en tales sistemas, lo cual se describe en la siguiente sección.

2.2 Fundamentos Teóricos de la Convección en Medios Porosos

Una sustancia en fase líquida o gaseosa se conoce como *fluido*. La mecánica es la ciencia física más antigua que trata de los cuerpos en reposo y en movimiento bajo la influencia de fuerzas. En particular, la *mecánica de fluidos* es la ciencia que estudia el comportamiento de los fluidos en reposo (*estática de fluidos*) o en movimiento (*dinámica de fluidos*). Debido a que los fenómenos considerados en esta disciplina son macroscópicos, un fluido es considerado como un medio continuo cuyo movimiento se describe con pocas características básicas como la velocidad, la densidad, la compresibilidad, la viscosidad, etc. [Skiba, 2008].

Lo anterior significa que cualquier volumen pequeño o elemental en el fluido es bastante grande y contiene muchas moléculas. Por lo tanto, cuando se habla de un volumen de fluido infinitesimal se sobreentiende que dicho volumen es muy pequeño en comparación con el volumen del dominio total, pero es bastante grande en comparación con la distancia entre moléculas, es decir, contiene muchas moléculas [Landau y Lifshitz, 1987] [Skiba, 2008].

Las expresiones *partícula de fluido* y *punto en un fluido* tienen el mismo significado. Por ejemplo, cuando se habla del desplazamiento de alguna partícula de fluido no significa el desplazamiento

de una molécula individual, sino el del volumen elemental que contiene muchas moléculas y que aún se considera un punto [Landau y Lifshitz, 1987].

La descripción matemática del estado de un fluido en movimiento se efectúa por medio de las funciones que dan la distribución de la velocidad del flujo $\underline{u} = (x, y, z, t)$ y de cualesquiera dos cantidades termodinámicas pertenecientes al fluido, por ejemplo presión $p = (x, y, z, t)$ y densidad $\rho = (x, y, z, t)$. Todas las variables termodinámicas están determinadas por los valores de dos de ellas, junto con la ecuación de estado. Por lo tanto, si se tienen estas cinco cantidades: las tres componentes de la velocidad, la presión y la densidad, el estado del movimiento del fluido está completamente determinado [Landau y Lifshitz, 1987] [Skiba, 2008].

La *energía térmica* representa la energía interna total de un objeto: la suma de las energías potencial y cinética de todas sus moléculas. La mecánica clásica no puede explicar cambios en los estados de la energía térmica; por esta razón se define a la *temperatura* como la propiedad fundamental que tienen todos los objetos y que determina si estarán en equlibrio térmico con otros objetos.

La temperatura $T = (x, y, z, t)$ puede ser determinada por la ecuación de estado $f(\rho, p, T) = 0$

Cuando dos objetos con diferentes temperaturas se ponen en contacto, la energía del objeto con mayor temperatura se transfiere al objeto de menor temperatura hasta que el sistema alcanza una condición estable llamada equilibrio térmico, que se da cuando los dos objetos llegan a la misma temperatura.

En termodinámica se introduce una forma especial del movimiento de materia: el *calor*, que representa la energía térmica generada por el movimiento de partículas (moléculas, átomos, etc.) debido a una diferencia (gradiente) de temperaturas entre varias partes de una sustancia. El calor se transfiere entre dos sistemas en contacto (no necesariamente físico) y puede presentarse entre medios distintos o dentro de un mismo medio. Dicha transferencia se realiza por medio de 3 mecanismos:

- **Conducción**
- **Radiación**
- **Convección**

La *conducción* es el proceso por el cual la energía térmica se transfiere mediante colisiones de las moléculas de una sustancia, de las más energéticas a las menos energéticas. Por lo general se presenta en sólidos y fluidos estáticos, se basa en el contacto directo entre cuerpos y está regida por la *Ley de Fourier* de conducción del calor, la cual afirma que la velocidad de conducción del calor a través de un cuerpo por unidad de sección transversal (flujo de calor), es proporcional al gradiente de temperatura que existe en el cuerpo [Bear, 1972].

La *radiación* se atribuye a la energía calorífica transmitida por medio de ondas electromagnéticas o fotones, debido a cambios en las configuraciones electrónicas de los átomos o moléculas. El calor emitido por este mecanismo viene dado por la *Ley de Stefan-Boltzmann*.

La *convección* ocurre por la transferencia de energía calorífica entre el fluido en movimiento y un sólido adyacente a él debido a dos submecanismos: difusión y advección. La *difusión* es el movimiento aleatorio de las moléculas (nivel microscópico), mientras que la *advección* es el movimiento de grandes cantidades de fluido (nivel macroscópico), que en presencia de un gradiente de temperatura contribuye a la transferencia de calor [Montealegre, 2010]. 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 (movimiento de difusión más el de advección) 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. La cantidad de calor transferido por convección, se rige por la *Ley de enfriamiento de Newton*.

El proceso de transferencia de calor en fluidos es más complejo que en sólidos porque el fluido puede estar en movimiento; es decir, no hay equilibrio mecánico. Un fluido puede estar en equilibrio mecánico (no hay advección o movimiento macroscópico del mismo) sin estar en equilibrio térmico. Si hay equilibrio mecánico en un fluido en un campo gravitacional, la distribución de temperatura puede depender solamente de la profundidad z: $T = T(z)$. Si esta condición no se satisface, sino que la temperatura también es función de otras coordenadas, el equilibrio mecánico no es posible [Landau y Lifshitz, 1987].

La ausencia de equilibrio mecánico induce la formación de corrientes internas en el fluido, las cuales intentan mezclar el fluido con el fin de igualar la temperatura; es decir el fluido en movimiento se calienta de manera no uniforme. Esta diferencia de temperaturas en el sistema genera diferencias de densidad, lo cual hace que el fluido menos caliente y más pesado tienda a hundirse (debido a la fuerza de gravedad), y éste empuje, a su vez, al fluido más caliente y menos pesado hacia arriba. Tal movimiento en un campo gravitacional es llamado convección libre o natural [Bear, 1972] [Landau y Lifshitz, 1987].

Otro tipo de convección es la convección forzada, en la cual el flujo del fluido se debe a causas externas, por ejemplo un ventilador, bombeo o el viento en la atmósfera. Si se presentan ambos tipos de convección: forzada y natural, y ambas son de magnitudes similares, se dice que se tiene una convección mixta [Bear, 1972] [Montealegre, 2010].

La convección puede presentarse en un medio libre o en un medio poroso. En un medio libre el fluido no tiene restricciones en su movimiento; por ejemplo, el agua en las corrientes marinas o en ríos y canales, así como el aire que circula en la atmósfera [Báez, 2008].

Por otro lado, un medio poroso se define como un cuerpo heterogéneo conformado por un material sólido (matriz) que contiene orificios llamados poros, los cuales están llenos de fluidos (ver **Figura 1**). Algunos de estos poros pueden estar interconectados entre sí permitiendo el flujo de fluido a través de ellos . A partir de esta definición, la casi totalidad de los elementos que constituyen la corteza terrestre puede considerarse como un medio poroso [Flores-Márquez, 1992]. En el **Apéndice A** se describen las propiedades físicas del medio poroso y del fluido que lo satura.

En un medio poroso, el movimiento del fluido se verá restringido al espacio disponible dentro de los poros. Ejemplos de fluidos en medios porosos son los hidrocarburos en sistemas petroleros, el agua en sistemas geotérmicos o acuíferos moviéndose a través del subsuelo, y el aire que circula entre los granos almacenados en silos. Además, un medio poroso provee una gran dispersión térmica y un area de contacto fluido-sólido mucho más grande que el area de transferencia de calor de un fluido en un medio libre, lo que trae como consecuencia que la transferencia de calor sea también más elevada en medios porosos.

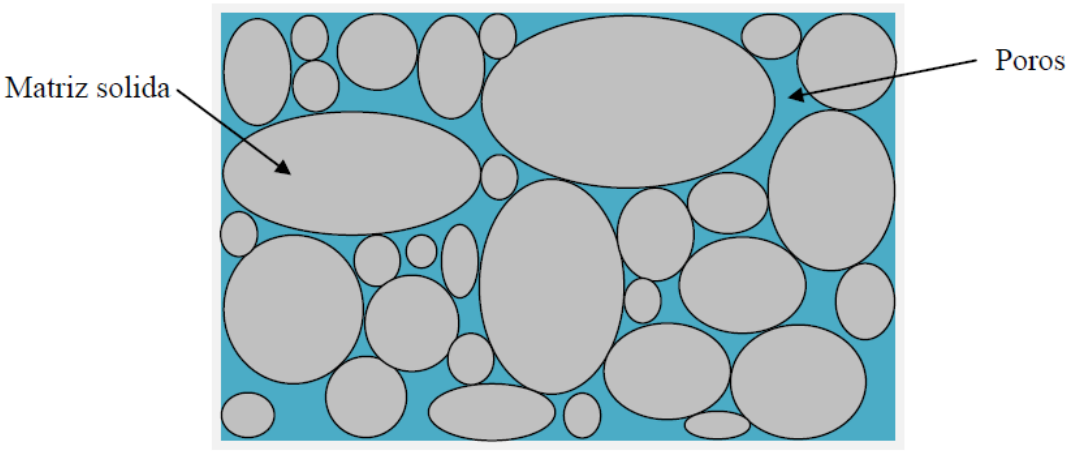


Figura 1: *Medio poroso saturado con un fluido.*

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 por movimiento del fluido [Combarnous y Bories, 1975].

Los movimientos convectivos en una capa porosa tienen dos efectos principales: homogeneizar el volumen del fluido donde la convección toma lugar, y distribuir la temperatura in situ la cual se caracteriza por zonas calientes y frías.

En resumen, en un medio poroso saturado con un fluido en movimiento, los mecanismos de transferencia de calor son principalmente dos: conducción en la fase sólida; mientras que en la fase fluida el calor se transfiere por conducción y convección (más manifestaciones de los mecanismos de transferencia de calor pueden verse en [Bear, 1972]). A continuación se describirá la formulación matemática para la transferencia de calor por convección en medios porosos.

2.3 Formulación Matemática de la Convección Natural en Medios Porosos

En esta sección se presenta el estudio de la convección natural en un medio poroso heterogéneo y anisotrópico saturado con un fluido en una sola fase. La fase líquida se comporta de acuerdo con la Ley de Darcy. La fase sólida es un medio poroso saturado, heterogéneo y anisotrópico, 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 de Darcy. Para problemas bidimensionales las ecuaciones resultantes son similares y pueden resolverse numéricamente usando el mismo procedimiento [Royer y Flores-Márquez, 1994].

2.3.1 Análisis

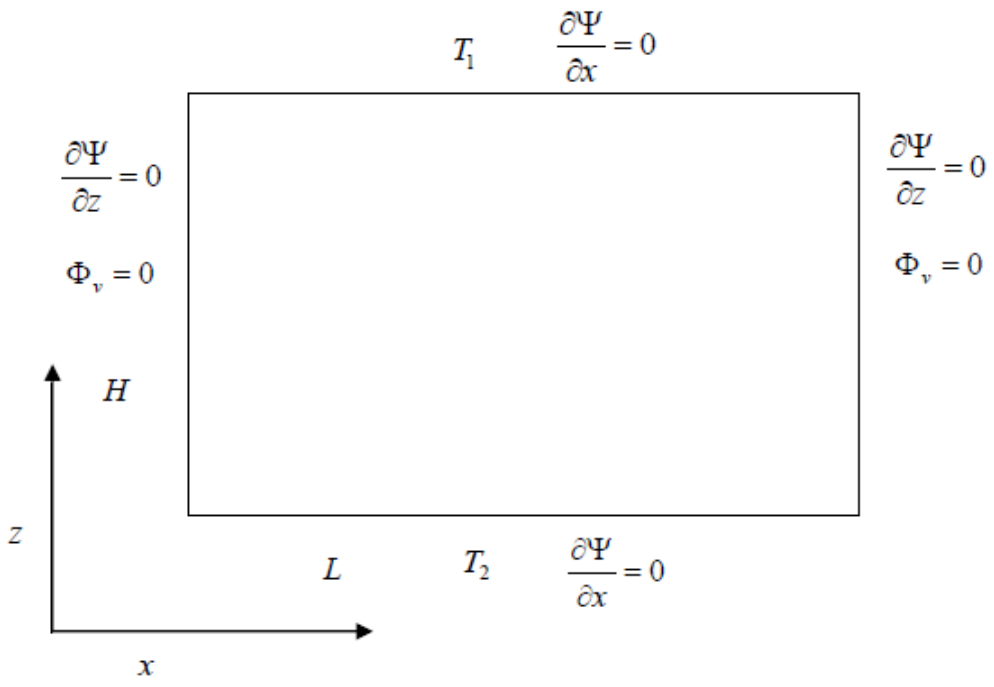


Figura 2: *Medio poroso saturado confinado, con bordes horizontales superior e inferior isotérmicamente definidos, bordes verticales adiabáticos, sin intercambio de fluidos entre las fronteras.*

Un estudio clásico de la convección natural de fluidos incompresibles en un medio poroso representado por el dominio rectangular de la **Figura 2**, incluye hacer un estudio de la distribución de la

temperatura y velocidad de filtración. Para ello se requieren ecuaciones de energía, momento lineal y conservación de la masa, representativas del medio poroso saturado con un fluido en movimiento; así como las ecuaciones de estado del fluido, en función de la temperatura. Todo esto con el fin de constituir el modelo del sistema continuo.

A continuación se presenta el conjunto de ecuaciones para modelar la transferencia térmica en estado transitorio de un fluido en movimiento en un medio poroso, siguiendo la nomenclatura propuesta en el **Apéndice B**. También se describe, para cada una de las ecuaciones, la ley física que les da origen.

- **Ecuación de transferencia de calor o ecuación de Fourier**. Se basa en la ley de conservación de la energía o Primera Ley de la Termodinámica, la cual establece que la energía total de un sistema aislado permanece constante. Si un sistema no está aislado, su energía se incrementa solamente si la energía de sus alrededores experimenta una disminución correspondiente.

$$\nabla \cdot (\Lambda^* \nabla T^*) = (\rho c_p)_f \, \mathbf{v} \cdot \nabla T^* + (\rho c_p)^* \frac{\partial T^*}{\partial t^*} - A^* \tag{2.1}$$

- **Ecuación generalizada de Darcy o de momento lineal**. Se basa en la ley generalizada de Darcy, la cual establece que la cantidad de movimiento de un sistema permanece constante si no hay fuerzas externas que actúen en él.

$$\nabla p - \rho \underline{\hat{g}} - \frac{\mu}{\mathbf{K}} \cdot \mathbf{v} = 0 \tag{2.2}$$

- **Ecuación de continuidad o de conservación de la masa**. Se basa en la ley de la conservación de la masa, la cual establece que la cantidad de masa que entra en un sistema menos la cantidad de masa que sale es igual a la cantidad de masa que se acumula en el mismo; es decir, la cantidad de masa total en el sistema permanece constante, o bien, los cuerpos conservan su masa en su movimiento. En este caso, se considera un fluido incompresible ($\partial\rho/\partial t = 0$)

$$\nabla \cdot (\rho \mathbf{v}) = 0 \tag{2.3}$$

- **Ecuaciones de estado del fluido**. Se basan en leyes constitutivas que describen la variación de las propiedades del fluido con respecto a la temperatura. La ecuación (2.4) muestra la dependencia de la densidad con la temperatura, mientras que la ecuación (2.5) muestra la dependencia de la viscosidad con la temperatura, ambas ecuaciones son para agua pura.

$$\rho_f = \rho_r \left(1 - \beta_{th} \left(T^* - T_r^*\right)\right) \tag{2.4}$$

$$\eta = \eta_r \left(1 - \gamma^* \left(T^* - T_r^*\right)\right) \tag{2.5}$$

En investigaciones prácticas comúnmente se asume que la velocidad de filtración y su gradiente son muy pequeños por lo cual se pueden despreciar las fuerzas inerciales [Royer y Flores-Márquez, 1994]; esto se debe a la existencia de la matriz sólida en el medio poroso con un tamaño pequeño del diámetro de poro promedio [Combarnous y Bories, 1975]. Como puede observarse, las ecuaciones de calor, momento lineal y continuidad están formuladas en términos de las variables primitivas velocidad y presión; mientras que las expresiones para la densidad y la viscosidad son aproximaciones lineales, a presión constante, de la ley de estado dada por $\rho = \rho(p, T)$ y $\eta = \eta(p, T)$ [Báez et al., 2004].

Si consideramos la aproximación de Boussinesq en la termodinámica y efectos térmicos de flujo, las ecuaciones de balance de masa y de momento lineal se simplifican debido a que se asume un medio homogéneo donde las variaciones de la densidad del fluido se consideran despreciables, excepto en el término de fuerza externa gravitacional (flotabilidad) dado por ρg, donde ρ es la densidad y g es el vector de aceleración gravitacional [Báez et al., 2004] [Royer y Flores-Márquez, 1994].

Sin embargo, para grandes sistemas hidrotermales algunos parámetros característicos del medio como la viscosidad dinámica y la conductividad térmica (η y Λ^*, respectivamente) pueden depender de la litología y del campo de temperaturas; mientras que otros parámetros como la permeabilidad K, el coeficiente de expansión volumétrico β_{th} y la densidad de masa del fluido ρ_r son menos sensibles. Debido a la heterogeneidad y a la dependencia térmica del medio poroso, el conjunto de

ecuaciones debe establecerse bajo las simplificaciones mencionadas [Royer y Flores-Márquez, 1994].

A continuación se presenta la formulación matemática que da lugar al conjunto de ecuaciones adimensionales acopladas (2.11) y (2.22), que gobiernan el estudio de la convección libre en medios porosos.

2.3.2 Ecuación de Transferencia de Calor

En la ecuación (2.1) de balance de energía o transferencia de calor, las características térmicas del medio poroso son tomadas en cuenta por los parámetros de capacidad calorífica $(\rho c)^*$ y el coeficiente de capacidad térmica equivalente Λ. El término del lado izquierdo de esta ecuación representa la contribución de la conducción al balance de energía; mientras que el aporte debido a la convección está dado por el primer término del lado derecho de la ecuación de calor [Combarnous y Bories, 1975].

La ecuación de transferencia de calor es válida suponiendo que la diferencia entre las temperaturas de la fase sólida (T_s^*) y de la fase líquida (T_f^*) es despreciable y que la velocidad de filtración no es muy grande. De esta manera el medio se comporta como un único continuo donde la temperatura promedio es $T^* = T_s^* = T_f^*$; es decir, hay equilibrio térmico local. Esta aproximación es válida para la mayoría de los medios porosos saturados que se presentan en el ámbito geológico tales como formaciones sedimentarias e incluso, con limitaciones, para modelar la transferencia de calor a través de rocas fracturadas.

La capacidad calorífica del medio poroso saturado $(\rho c)^*$ se considera que depende de la porosidad, de acuerdo con el siguiente modelo simple:

$$(\rho c)^* = (1-\varepsilon)\,(\rho c)_s + \varepsilon\,(\rho c)_f \tag{2.6}$$

El tensor de conductividad térmica Λ^* disminuye generalmente con el incremento en la temperatura, de acuerdo con la siguiente relación [Royer y Flores-Márquez, 1994]:

$$\Lambda^* = \Lambda_r^* / \left(1 + a_r \left(T^* - T_r^*\right)\right) \tag{2.7}$$

donde Λ_r^* es la conductividad térmica a $25°C$; a es un parámetro que depende del medio poroso y toma valores en un rango de $5 \times 10^{-4}\ °C^{-1}$ a $10^{-3}\ °C^{-1}$. La conductividad térmica de la roca depende de varios parámetros tales como composición de la roca, textura, tamaño de grano y composición mineral.

Esta termodependencia de la conductividad térmica hace que la ecuación de transferencia de calor sea no lineal, lo cual es una seria complicación en comparación con la aproximación tradicional que supone una conductividad térmica constante para el medio poroso. Sin embargo para problemas que involucran un medio poroso de grandes dimensiones como lo es un sistema geotérmico, el efecto de la temperatura en las propiedades térmicas no debe ser descartado.

Para medios isotrópicos, la conductividad térmica equivalente puede relacionarse experimentalmente con la porosidad de acuerdo con la siguiente relación:

$$\lambda_0 = \lambda_s^{(1-\varepsilon)} * \lambda_f^{\varepsilon} \tag{2.8}$$

donde λ_s y λ_f son las conductividades térmicas del sólido y del fluido, respectivamente. Esta relación es válida solamente si las fases sólida y fluida están bien dispersas y si el contraste entre sus propiedades térmicas no es muy grande.

Para medios anisotrópicos, el tensor de conductividad térmica equivalente Λ^* es una función más complicada que depende de las propiedades térmicas de cada fase que constituye al medio, las cuales pueden ser determinadas experimentalmente.

2.3.3 Ecuación de Transferencia de Calor Adimensional

Al introducir la conductividad adimensional Λ (**Apéndice B**) y considerando la equivalencia siguiente:

$$\frac{2}{tr\Lambda^*}\nabla\cdot(\Lambda\nabla T^*)=\nabla\cdot(\Lambda\nabla T^*)+\nabla\cdot(\ln tr\Lambda^*)\cdot\Lambda\nabla T^* \tag{2.9}$$

la ecuación 2.1 puede reescribirse como:

$$\nabla\cdot(\Lambda\nabla T^*)=\frac{2\,(\rho c)_f}{tr\Lambda^*}\mathbf{v}\cdot\nabla T^*+\frac{2\,(\rho c)^*}{tr\Lambda}\frac{\partial T^*}{\partial t^*}-\frac{2A^*}{tr\Lambda^*}-\nabla\cdot(\ln tr\Lambda^*)\cdot\Lambda\nabla T^* \tag{2.10}$$

La ecuación (2.10) muestra que las heterogeneidades de la conductividad térmica dentro del medio poroso se toman en cuenta como un término fuente (similar a una generación interna de calor) en el segundo miembro de la ecuación de calor. Usando las variables adimensionales definidas en la nomenclatura del **Apéndice B**, la ecuación (2.10) puede simplificarse de la siguiente manera:

$$\nabla\cdot\left(\underline{\underline{\Lambda}}\nabla T\right)=\mathbf{v}\cdot\nabla T+\frac{\partial T}{\partial t}-A \tag{2.11}$$

$$v=v_{Darcy}-\nabla\left(\ln tr\Lambda^*\right)\cdot\underline{\underline{\Lambda}} \tag{2.12}$$

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

2.3.4 Ecuación de Darcy

La ecuación (2.2) en la cual se han omitido algunos términos inerciales, es una forma generalizada de la ecuación de Darcy en estado estable (la ecuación completa puede verse en Combarnous y Bories, 1975). Esta ecuación es válida para describir el movimiento de la fase fluida en sistemas convectivos cuya velocidad de filtración no es demasiado importante y para sistemas en los cuales la razón entre la permeabilidad media isotrópica y la altura cuadrada del dominio (κ/H^2) es menor que 10^{-3}. Suponiendo que se mantiene constante la capacidad calorífica para el sólido y el fluido, y una termo-dependencia de las variaciones de las características físicas del fluido, ρ y η, la ecuación (2.2) puede reescribirse como:

$$-\left(1-\gamma T\right)v_{Darcy}=\frac{tr\mathbf{K}}{tr\Lambda^{*}}\frac{\left(\rho c\right)_{f}}{\eta_{r}}\underline{\underline{K}}\left(\nabla p\right)^{t}\ -Ra^{*}\left[T-\frac{1}{\beta}\right]\underline{\underline{K}}\cdot e_{3} \tag{2.13}$$

donde v_{Darcy}, $\underline{\underline{K}}$, T y β son las variables adimensionales definidas en la nomenclatura del Apéndice B. Ra^{*} es el número de Rayleigh de filtración local para un medio anisotrópico y heterogéneo, definido como:

$$Ra^{*}=\frac{\rho_{r}g\left(\rho c\right)_{f}\beta_{th}\Delta T^{*}Htr\mathbf{K}}{\eta_{r}tr\Lambda^{*}} \tag{2.14}$$

Esta ecuación describe explicítamente el criterio adimensional para el comienzo de la convección hidrotermal libre, midiendo la influencia relativa de la fuerza "impulsora" de la convección sobre los efectos "estabilizadores" debidos a la viscosidad del fluido η_r y a su difusividad térmica.

Los medios homogéneos han sido ampliamente estudiados [Combarnous y Bories, 1975], en éstos la condición para la ocurrencia de las celdas termoconvectivas es que el número Rayleigh de filtración sea mayor que un valor crítico $Ra_{cr}=4\pi^{2}$. Para valores mayores del número de Rayleigh las celdas se vuelven inestables y por lo tanto la convección es turbulenta.

La fórmula para el valor crítico del número de Rayleigh [Royer y Flores-Márquez, 1994] es:

$$Ra_{cr}^{*}=\frac{tr\mathbf{K}\left[tr\left(\Lambda^{*}\mathbf{K}^{-1}\right)^{1/2}\right]^{2}}{tr\Lambda^{*}}\pi^{2} \tag{2.15}$$

si se usan tensores adimensionales de permeabilidad o de conductividad, el valor crítico de Rayleigh se obtiene de la siguiente manera:

$$Ra_{cr}^{*}=\left[tr\left(\Lambda K^{-1}\right)^{1/2}\right]^{2}\pi^{2} \tag{2.16}$$

2.3.5 Ecuación de Darcy Adimensional

Al introducir las cantidades adimensionales v_{Darcy}, f y g', la ecuación de Darcy puede ser reescrita como una función de stream (función de corriente) de la siguiente manera:

$$v_{Darcy} = (\mathbf{D}\psi)^t \tag{2.17}$$

$$f = Ra^* \frac{tr\Lambda^*}{trK} \frac{\eta_r}{(\rho c)_f} \left[T - \frac{1}{\beta}\right] \tag{2.18}$$

$$g' = \frac{tr\Lambda^*}{tr\mathbf{K}} \frac{\eta_r}{(\rho c)_f} [1 - \gamma T] \tag{2.19}$$

$$(\nabla p)^t = f e_3 - g' \underline{\underline{K}}^{-1} (\mathbf{D}\psi)^t \tag{2.20}$$

Por medio de la diferenciación cruzada y aplicando la relación tensorial 2D $tr\left(\mathbf{D}^t \nabla p\right) = 0$ la ecuación (2.20) puede ser simplificada en la siguiente ecuación diferencial no lineal que involucra la función de stream ψ:

$$tr\left(\mathbf{D}^t \left(f e_3^t\right)\right) = tr\left(\mathbf{D}^t \left(g' \left(\mathbf{D}\psi\right) \underline{\underline{K}}^{-1}\right)\right) \tag{2.21}$$

Como se observa en [Royer y Flores-Márquez, 1994], podrían usarse más simplificaciones con respecto a las propiedades tensoriales en 2D para los operadores $\mathbf{D}$ y ∇, con lo que se obtiene una forma más simple y práctica para la ecuación de Darcy:

$$\nabla \cdot \left(\underline{\underline{K}} \nabla \psi\right) = u \cdot \nabla \psi - S \tag{2.22}$$

con las siguientes cantidades adimensionales:

$$\mathbf{u} = -\nabla \ln g \underline{\underline{K}} \tag{2.23}$$

$$S = (h \nabla \ln g + \nabla h) \cdot e_1 \tag{2.24}$$

$$h = Ra^* \det \underline{\underline{K}} \frac{\left[T - \frac{1}{\beta}\right]}{[1 - \gamma T]} \tag{2.25}$$

$$bg = \frac{1}{det \underline{\underline{K}}} \frac{tr\Lambda^*}{trK} \frac{\eta_r}{(\rho c)_f} [1 - \gamma T] \tag{2.26}$$

Se observa claramente la similitud entre las ecuaciones adimensionales (2.11) y (2.22) las cuales son, respectivamente, la ecuación de transferencia de calor y la ecuación de Darcy. La cantidad vectorial u sería equivalente a la velocidad de filtración $\mathbf{v}$ en la ecuación de calor, mientras que S sería similar a un término fuente, equivalente a la producción de calor A en la ecuación de Fourier de transferencia de calor. Esta similitud es de gran interés al resolver numéricamente las ecuaciones acopladas de transferencia de calor y masa en estado estable debido a que puede usarse el mismo procedimiento para resolver ambas ecuaciones.

2.3.5.1 Evaluación del término fuente S

El término fuente S se evalúa usando técnicas de derivación logarítmica, de la siguiente manera:

$$\nabla \ln g = \nabla \ln (tr\Lambda^*) - \nabla \ln (tr\mathbf{K}) - \nabla \ln \left(det \underline{\underline{K}}\right) - \nabla \ln \left((\rho c)_f\right) - \frac{\gamma \nabla T}{(1 - \gamma T)} \tag{2.27}$$

$$\nabla \ln h = \nabla \ln \left(det \underline{\underline{K}}\right) + \nabla \ln (Ra^*) + \frac{\nabla T}{\left(T - \frac{1}{\beta}\right)} + \frac{\gamma \nabla T}{(1 - \gamma T)} \tag{2.28}$$

$$\nabla \ln (Ra^*) = \nabla \ln \left((\rho c)_f \right) + \nabla \ln (tr\mathbf{K}) - \nabla \ln (tr\Lambda^*) \tag{2.29}$$

$$S = \frac{\nabla T}{\left(T - \frac{1}{\beta}\right)} h \cdot e_1 = \frac{Ra^* det\underline{\underline{K}}}{(1 - \gamma T)} \nabla T \cdot e_1 \tag{2.30}$$

La ecuación (2.30) resalta el significado físico del término fuente S, el cual depende de las propiedades anisotrópicas del medio poroso a través de $det\underline{\underline{K}}$ y de las variaciones de la viscosidad del fluido con la temperatura; mientras que el término **u** depende de la heterogeneidad del medio a través de los términos $\underline{\underline{K}}$, $\nabla \ln (trK)$, $\nabla \ln (tr\Lambda^*)$, $\nabla \ln \left(det\underline{\underline{K}}\right)$.

Es de notar que para un medio homogéneo e isotrópico saturado con un fluido newtoniano de viscosidad η constante e independiente de la temperatura se tiene $det\underline{\underline{K}} = 1$. Entonces el término fuente en la ecuación de Darcy se reduce a la forma clásica $S = Ra^* \nabla T \cdot e_1$.

2.3.6 Ecuación de Continuidad

La ecuación de Darcy ha sido obtenida considerando que las variaciones de la densidad del fluido son despreciables, excepto en el término de flotabilidad $\rho \underline{g}$. Esta suposición también implica una simplificación para la ecuación de balance de masa (2.3), la cual puede ser reescrita usando la velocidad de filtración adimensional v_{Darcy}:

$$\nabla \cdot (tr\Lambda^* v_{Darcy}) = 0 \tag{2.31}$$

En las aplicaciones más comunes, las variaciones de la conductividad térmica con respecto al espacio y que están representadas por el término $tr\Lambda^*$ pueden descartarse en la ecuación (2.31) en comparación con la variación de la velocidad de filtración debida a la permeabilidad. Por ejemplo, la conductividad térmica de las rocas comunes varía en un rango entre 1 y 3 $\left[Wm^{-1}K^{-1}\right]$ para un intervalo de temperatura de 25 a 300 $[^\circ C]$, mientras que la permeabilidad varía de 10^{-20} a 10^{-12} $\left[m^2\right]$. Esta aproximación permite simplificar la ecuación (2.31) de la siguiente manera:

$$\nabla \cdot v_{Darcy} = 0 \tag{2.32}$$

Es necesario recalcar que la velocidad de Darcy en la ecuación (2.17) se obtiene a partir de la derivación cruzada de la función de corriente $(\mathbf{D}\psi)^t$. Esta propiedad se asume automaticamente en la ecuación (2.32) debido a que $\nabla \cdot (\mathbf{D}\psi)^t$ es siempre igual a cero [Royer y Flores-Márquez, 1994].

Cuando la variación de la conductividad térmica con respecto al espacio no puede ser despreciada, desarrollando la ecuación (2.31) e incorporando la ecuación (2.32), la ecuación de continuidad queda de la siguiente manera [Royer y Flores-Márquez, 1994]:

$$\nabla tr\Lambda^* \cdot v_{Darcy} = 0 \tag{2.33}$$

La ecuación (2.33) 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 la interfaz entre dos capas homogéneas con diferente conductividad térmica, esto implica que la velocidad de Darcy sea paralela a la interfaz, lo cual ocurre comúnmente cuando las dos capas tienen diferentes permeabilidades. En otros casos la ecuación (2.33) debe incluirse en las ecuaciones fundamentales, sin embargo no es caso del presente trabajo.

2.3.7 El Conjunto Completo de Ecuaciones

Finalmente, del desarrollo previo, se tiene que las ecuaciones que gobiernan el estudio de la convección libre se reducen a las ecuaciones adimensionales acopladas (2.11) y (2.22), las cuales se resumen a continuación:

$$\nabla \cdot \left(\underline{\underline{\Lambda}} \nabla T\right) = v \cdot \nabla T + \frac{\partial T}{\partial t} - A \tag{2.34}$$

$$\nabla \cdot \left(\underline{\underline{K}} \nabla \psi\right) = u \cdot \nabla \psi - S \tag{2.35}$$

$$v = v_{Darcy} - \nabla \left(\ln tr\Lambda^* \right) \cdot \underline{\underline{\Lambda}}$$

$$v_{Darcy} = (\mathbf{D}\psi)^t = \left(\frac{\partial \psi}{\partial z}, -\frac{\partial \psi}{\partial x} \right)^t$$

$$u = -\nabla \ln g \underline{\underline{K}}$$

$$S = \frac{Ra^* det\underline{\underline{K}}}{(1 - \gamma T)} \nabla T \cdot e_1$$

$$g = \frac{1}{det\underline{\underline{K}}} \frac{tr\Lambda^*}{trK} \frac{\eta_r}{(\rho c)_f} \left[1 - \gamma T \right]$$

$$Ra^* = \frac{\rho_r g \left(\rho c \right)_f \beta_{th} \Delta T^* H tr\mathbf{K}}{\eta_r tr\Lambda^*} \tag{2.36}$$

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

$$(\nabla p)^t = f e_3 - g' \underline{\underline{K}}^{-1} (\mathbf{D}\psi)^t \tag{2.37}$$

El conjunto de ecuaciones (2.36) relaciona dos funciones escalares desconocidas: la temperatura y la función de corriente (T, ψ), las cuales se resolverán simultáneamente de forma numérica con las condiciones de frontera adecuadas. Los principales parámetros del medio poroso que representan estas ecuaciones son la permeabilidad y conductividad térmica.

Como se describe en [Evans y Raffensperger, 1992], las ventajas de usar la función de stream ψ son las siguientes: la función de stream provee una descripción escalar del campo de velocidades de tal manera que sus líneas de contorno representan las trayectorias que el fluido sigue durante el

flujo. Además las velocidades del flujo son proporcionales al espacio entre las líneas de contorno, así que la magnitud relativa de las velocidades de fluido puede inferirse a partir de una gráfica de ψ (el vector velocidad en un punto es tangente a la línea de contorno de la función stream).

En particular, cualquier solución del campo de flujo en términos de la función de stream garantiza la conservación de la masa del fluido; esto último hace que las soluciones intermedias sean físicamente razonables y tiendan a la convergencia. Numéricamente, esto es quizás la mejor razón para usar la formulación de la función de stream. Las soluciones del campo de flujo en términos de ψ tienden a converger más rápidamente y ser más estables numéricamente que los cálculos similares para presión o carga hidráulica (en el caso de flujo de aguas subterráneas) [Evans y Raffensperger, 1992].

El uso de la función de stream implícitamente supone flujo en estado estable o cambios despreciables en el almacenamiento, lo cual no es una suposición severa en problemas de convección libre; pero sí aplicable en el cálculo de flujo transitorio cercano a pozos.

2.3.8 Número de Rayleigh

El criterio de aparición de la convección en una capa horizontal depende fuertemente del medio. De acuerdo con los modelos propuestos por [Combarnous y Bories, 1975] sobre la convección térmica en medios porosos, se estableció que la transición entre el flujo estable y los regímenes convectivos para un medio horizontal infinito tiene lugar cuando el número de Rayleigh de filtración sobrepasa el valor de $4\pi^2$ y el número de Nusselt es superior a 1. No existe una correlación única entre los números de Nusselt y de Rayleigh, pero sus valores son indicadores de la aparición de la convección en todos los casos [Flores-Márquez, 1992].

2.3.9 Condiciones de Frontera

El contexto de un estudio de convección térmica natural está definido por las condiciones de frontera hidrodinámicas, las condiciones de frontera térmicas y la geometría del medio poroso, que en este caso es un dominio rectangular cuya razón geométrica H/L puede variar. Para las condiciones de temperatura en las fronteras, se imponen:

1. Frontera superior. *Condiciones de frontera Neumann*, las cuales imponen un flujo de calor constante conocido a lo largo de la frontera. O bien, *condiciones de frontera Dirichlet*, las cuales imponen un potencial a lo largo de la frontera, es decir que el valor de la temperatura sea constante (frontera isotérmica), la cual es la condición que se utiliza en la mayoría de los casos.

2. Frontera inferior. Una temperatura constante (Dirichlet) o bien un flujo de calor constante (Neumann), esta última condición es la más comúnmente utilizada.

3. Fronteras laterales. Se impone una condición de flujo horizontal nulo (fronteras adiabáticas), lo cual corresponde a una transferencia conductiva puramente vertical.

Para las condiciones de frontera hidrodinámicas (ecuación de Darcy) se imponen:

1. Un valor constante de la función stream ψ: cero en los cuatro límites del dominio, en la mayoría de los casos, y, eventualmente un flujo impuesto en la superficie.

En un problema clásico de convección se pueden definir los dos tipos de condiciones de frontera (Dirichlet y Neumann) para la resolución de cada una de las funciones desconocidas temperatura y stream (T, ψ) del conjunto de ecuaciones (2.34) y (2.35). El método de solución numérica se describe en el siguiente capítulo.

Capítulo 3

Solución Numérica

En este capítulo se presenta el método del elemento finito que permite obtener las funciones desconocidas de temperatura y flujo (stream) (T, ψ) del conjunto de ecuaciones (2.34) y (2.35).

3.1 Introducción

Un problema de ecuaciones diferenciales con valores en la frontera se puede resolver de manera analítica para obtener la solución exacta; sin embargo, algunas veces es muy difícil llegar a esta solución sobre todo cuando se trata de ecuaciones con condiciones muy específicas o grandes sistemas de ecuaciones. Debido a eso se crearon diversos métodos numéricos de aproximación a la solución de las ecuaciones diferenciales, algunos de estos métodos son: diferencias finitas, elementos finitos, elementos de frontera, método espectral y funciones de base radial, entre otros.

El fundamento de estos métodos numéricos de aproximación es discretizar el conjunto de ecuaciones diferenciales del sistema continuo, el cual tiene un número infinito de grados de libertad, para así transformarlo en un sistema discreto de ecuaciones algebraicas, con un número finito de grados de libertad, con el objetivo de que pueda ser resuelto usando una computadora.

Uno de los métodos que más popularidad ha ganado es el método de los elementos finitos, el cual fue desarrollado por primera vez por Richard Courant en 1943. Este matemático utilizó la formulación variacional de Ritz (un método de análisis numérico) para obtener soluciones aproximadas a un problema de condición de frontera en un sistema de vibración. Desde 1950 hasta 1970

éstos métodos fueron desarrollados por ingenieros y matemáticos hasta llegar a una metodología general para la solución de ecuaciones diferenciales parciales [Montealegre, 2010].

En el método de los elementos finitos las condiciones de frontera generales, las geometrías complejas, y propiedades materiales variables pueden ser manejadas de una manera relativamente más sencilla, en ventaja de otros métodos como el de diferencias finitas. No obstante, en la práctica, las derivadas temporales son discretizadas casi exclusivamente usando el método de diferencias finitas; mientras que las derivadas espaciales son discretizadas usando diferencias finitas, elementos finitos o usando el método espectral; así que un problema general puede ocupar más de un método numérico [Barreiro, 2017].

La base para la formulación del método del elemento finito es la formulación variacional de Galerkin. En las siguientes secciones se presentarán estos métodos, con los cuales se hará la discretización espacial del conjunto de ecuaciones (2.34) y (2.35).

3.2 Formulación Variacional

En la sección 2.1 del capítulo anterior se hizo un esbozo de la metodología de modelación matemática de sistemas con los elementos que la integran (modelos conceptual, matemático, numérico y computacional). Desde el punto de vista matemático, un modelo de un sistema está completo si define un problema de ecuaciones diferenciales bien planteado. Un problema de ecuaciones diferenciales con valores iniciales y condiciones de frontera es bien planteado si cumple que:

1. existe una y solo una solución,

2. la solución depende de manera continua de las condiciones iniciales y de frontera del problema.

Es decir, un modelo completo es aquél en el cual se incorporan condiciones iniciales y de frontera que definen conjuntamente con las ecuaciones diferenciales un problema bien planteado [Carrillo et al., 2008]. Los modelos de los sistemas continuos están constituidos por sistemas de ecuaciones diferenciales cuyo número es igual al número de propiedades intensivas (o extensivas) que intervienen en la formulación del modelo básico; más las relaciones que ligan a las propiedades intensivas entre sí, las cuales son las leyes constitutivas. Para mayores detalles consulte [Herrera y Pinder, 2012].

Como ya se vio en el capítulo anterior el sistema de ecuaciones diferenciales del modelo son las ecuaciones de balance de energía (2.1), momento lineal (2.2) y masa (2.3), en el modelo básico, y las ecuaciones acopladas de temperatura (2.34) y stream (2.35) que se obtienen después de hacer el proceso de adimensionalización. En cuanto a las leyes constitutivas, éstas son las ecuaciones de estado de densidad y viscosidad (2.4) y (2.5), respectivamente.

Un problema de ecuaciones diferenciales con valores en la frontera puede reescribirse en su forma variacional. La formulación variacional es más débil que la formulación convencional ya que ésta demanda menor suavidad de la solución $\mathbf{u}$, sin embargo cualquier problema variacional con valores en la frontera corresponde a un problema con valor en la frontera y viceversa. Además, la formulación variacional facilita el tratamiento de los problemas al usar métodos numéricos de ecuaciones diferenciales parciales.

Los métodos empleados en la formulación variacional se dividen en dos tipos: directos e indirectos. Los métodos directos están asociados a la minimización de algún funcional, mientras que los métodos indirectos o de residuos pesados (ponderados) se aplican directamente sobre la ecuación diferencial y sus condiciones de frontera sin que precise de algún funcional asociado.

- Métodos Variacionales
  - Directos
    - Rayleigh-Ritz
  - Indirectos o de Residuos Pesados
    - Colocación
    - Emparejamiento de puntos
    - Subdominio
    - Galerkin
    - Mínimos Cuadrados

Los métodos de residuos pesados suponen que la solución puede ser representada por un conjunto de funciones base o funciones de prueba. Con estos métodos es posible obtener soluciones aproximadas de ecuaciones diferenciales que no tienen un funcional asociado.

3.2.1 Método de Galerkin

Para la formulación variacional, se considera el siguiente problema variacional con valores en la frontera (VBVP, por sus siglas en inglés: *Variational Boundary Value Problem*) de la forma:

$$\begin{aligned} \mathcal{L}u &= f_\Omega && en \ \ \Omega \\ u &= g && en \ \ \partial\Omega \end{aligned} \tag{3.1}$$

donde:

$$\mathcal{L}u = -\nabla \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u + cu \tag{3.2}$$

con $\underline{\underline{\mathbf{a}}}$ una matriz simétrica, definida positiva y $c \geq 0$. Aquí $\Omega \subset R^2$ es un dominio poligonal, es decir, Ω es un conjunto abierto acotado y conexo tal que su frontera $\partial\Omega$ es la unión de un número finito de polígonos.

También se define V como un subespacio de un espacio de Hilbert H. Un espacio de Hilbert es un espacio vectorial lineal con producto interior $(H, \langle\cdot,\cdot\rangle)$, que es completo bajo la norma inducida por el mismo producto interior.

Al multiplicar la ecuación $-\nabla \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u + cu = f_\Omega$ por una función $v \in V = H_0^1(\Omega)$, se obtiene:

$$-v\left(\nabla \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u + cu\right) = v f_\Omega \tag{3.3}$$

aplicando el teorema de Green, se obtiene:

$$\int_\Omega \left(\nabla v \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u + cuv\right) \, d\underline{x} = \int_\Omega v f_\Omega \, d\underline{x} \tag{3.4}$$

Definiendo el operador bilineal como:

$$a\left(u, v\right) = \int_\Omega \left(\nabla v \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u + cuv\right) \, d\underline{x} \tag{3.5}$$

y la funcional lineal como:

$$l(v) = \langle f, v \rangle = \int_{\Omega} v f_{\Omega} \, d\underline{x} \tag{3.6}$$

se puede reescribir el problema dado por la ecuación (3.1) de orden 2 en forma variacional, haciendo uso de la forma bilineal $a(\cdot, \cdot)$ y de la funcional lineal $l(\cdot)$.

La idea básica detrás del método de Galerkin es, considerando el problema variacional con valor en la frontera (3.1), encontrar $u \in V = H_0^1(\Omega)$ que satisfaga:

$$a(u, v) = \langle f, v \rangle \qquad \forall v \in V \tag{3.7}$$

donde V es un subespacio de un espacio de Hilbert H (por conveniencia se restringirá a los números reales).

El problema al tratar de resolver la ecuación (3.7) está en el hecho de que el espacio V es de dimensión infinita, por lo que en general no es posible encontrar el conjunto solución. Así que en lugar de tener el problema en el espacio V, se supone que se tienen funciones linealmente independientes $\phi_1, \phi_2, \ldots, \phi_N$ en V y se define el espacio V^h a partir del subespacio dimensionalmente finito de V generado por las funciones ϕ_i, es decir:

$$V^h = Generado\{\phi_i\}_{i=1}^{N}, \qquad V^h \subset V \tag{3.8}$$

El índice $h = 1/N$ es un parámetro que estará entre 0 y 1, cuya magnitud da alguna indicación de cuan cerca V^h está de V, h se relaciona con la dimensión de V^h. Como el número N de las funciones base se escoge de manera que sea grande y haga que h sea pequeño, en el límite, cuando $N \to \infty$, entonces $h \to 0$.

Después de definir el espacio V^h, es posible trabajar con V^h en lugar de V y encontrar una función u_h que satisfaga:

$$a\left(u_h, v_h\right) = \langle f, v_h \rangle \qquad \forall v_h \in V^h \tag{3.9}$$

Esta es la esencia del método de Galerkin, cabe notar que u_h y v_h solamente son combinaciones lineales de las funciones base de V^h, tales que:

$$u_h = \sum_{i=1}^{N} c_i \phi_i \qquad v_h = \sum_{j=1}^{N} d_j \phi_j \tag{3.10}$$

donde v_h es arbitraria, como los coeficientes de d_j y sin pérdida de generalidad podemos hacer $v_h = \phi_j$. Así, para encontrar la solución u_h se sustituyen las ecuaciones (3.10) en la ecuación (3.9) y usando el hecho de que $a\left(\cdot,\cdot\right)$ es una forma bilineal y $l\left(\cdot\right)$ es una funcional lineal, se obtiene la ecuación:

$$\sum_{i=1}^{N} a\left(\phi_i, \phi_j\right) c_i = \langle f, \phi_j \rangle \tag{3.11}$$

o más concisamente como:

$$\sum_{i=1}^{N} K_{ij} c_i - F_j = 0 \qquad j = 1, 2, \ldots, N \tag{3.12}$$

en la cual:

$$K_{ij} = a\left(\phi_i, \phi_j\right) \qquad F_j = \langle f, \phi_j \rangle \tag{3.13}$$

se nota que tanto K_{ij} como F_j pueden ser evaluados, ya que ϕ_i, $a\left(\cdot,\cdot\right)$ y $l\left(\cdot\right)$ son conocidas.

Entonces el problema se reduce a resolver el sistema de ecuaciones lineales:

$$\sum_{i=1}^{N} K_{ij} c_i - F_j \qquad\qquad j = 1, 2, \ldots, N \tag{3.14}$$

o de manera matricial:

$$\underline{\underline{K}}\,\underline{u} = \underline{F} \tag{3.15}$$

en la cual $\underline{\underline{K}}$ y $\underline{F}$ son, respectivamente, la matriz y el vector cuyas entradas son K_{ij} y F_j. Una vez que el sistema es resuelto, la solución aproximada u_h es encontrada.

Cabe destacar que la forma bilineal $a(\cdot,\cdot)$ define un producto interior sobre V, si $a(\cdot,\cdot)$ es simétrica y V-elíptica, entonces las propiedades de linealidad y simetría son obvias, mientras que la propiedad de V-elipticidad de $a(\cdot,\cdot)$ es por:

$$a(v,v) \geq \alpha \|v\|^2 > 0 \qquad\qquad \forall V \neq 0 \tag{3.16}$$

además, si $a(\cdot,\cdot)$ es continua, entonces la norma $\|v\|_a = a(v,v)$ generada por este producto interior es equivalente a la norma estándar sobre V, tal que si V es completa con respecto a la norma estándar, esta también es completa con respecto a la norma $\|v\|_a$.

Por otro lado, si el conjunto de funciones base $\{\phi_i\}_{i=1}^{N}$ se eligen de tal forma que sean ortogonales entre sí, entonces el sistema (3.12) se simplifica considerablemente ya que:

$$K_{ij} = a(\phi_i, \phi_j) = 0 \qquad\qquad si \;\; i \neq j \tag{3.17}$$

$$K_{ii} c_i = F_i \qquad\qquad o \qquad\qquad c_i = \frac{F_i}{K_{ii}} \tag{3.18}$$

Así el problema (3.1) definido en $V^h = H_0^1(\Omega)$ reescrito como el problema (3.7) genera una forma bilineal V^h-elíptica cuyo producto interior sobre V^h es simétrico y positivo definido ya que:

$$a(v_h, v_h) \geq \alpha \|v_h\|_{V^h}^2 > 0 \qquad \forall v_h \in V^h, v_h \neq 0 \tag{3.19}$$

reescribiéndose el problema (3.9) como el problema aproximado en el cual debemos encontrar $u_h \in V^h \subset V$ tal que:

$$a(u_h, v_h) = \langle f, v_h \rangle - a(u_0, v_h) \tag{3.20}$$

donde $u_0 = g = 0$ en $\partial\Omega$, para toda $v_h \in V^h$, es decir:

$$\int_\Omega \left(\nabla v_h \cdot \underline{\underline{a}} \cdot \nabla u_h + c u_h v_h \right) dxdy = \int_\Omega f_\Omega v_h \, dxdy \tag{3.21}$$

para todo $v_h \in V^h$.

Entonces el problema planteado en (3.1) al aplicarle el método Galerkin se obtiene la ecuación (3.4), el cual puede reescribirse como (3.21).

Lo anterior es la idea básica del método de elementos finitos, el cual puede interpretarse como un método de aproximación donde las funciones base se definen en forma local en cada elemento y son llamadas *funciones de forma*, estas funciones se combinan para dar lugar a una aproximación por tramos.

3.3 Método del Elemento Finito Estándar

El método del elemento finito (FEM: Finite Element Method) es un método numérico para la resolución de sistemas de ecuaciones en derivadas parciales, en un espacio de dimensión finita.

En su forma más simple, el método FEM consiste en hacer elecciones especiales para el subespacio V_h, de tal manera que la solución u_h del problema (3.9) converge a la solución u del VBVP (3.1).

La aproximación clásica para el desarrollo de este método utiliza la formulación de Galerkin en donde la función de peso más comúnmente usada es la misma función base que se usa para aproximar la función conocida. Se debe verificar que el producto escalar del operador diferencial con todas las funciones de la base sea nulo; esto se logra aplicando el teorema de Green e integrando por partes.

La formulación variacional del método de elementos finitos es útil para la formulación de problemas no estructurales. En un principio el método de elementos finitos nació para el análisis de estructuras como vigas; así que tal deducción encontró serias limitaciones cuando se quiso extender a problemas no estructurales [Jouglard, 2002]. En la teoría, los métodos de elementos finitos son más abstractos; sin embargo, tienen ventajas significativas sobre otros métodos en aplicaciones prácticas. Por lo tanto, son una estrategia numérica muy eficiente para resolver problemas en la modelación de sistemas.

FEM provee una manera sistemática y simple de generar las funciones base en un dominio con geometría Ω poligonal. Las funciones base son polinomios definidos por pedazos (elementos Ω_i) que son diferentes de cero sólo en una pequeña parte de Ω, proporcionando a la vez una gran ventaja computacional al método ya que las matrices generadas resultan bandadas ahorrando memoria al implantarlas en una computadora.

Así, partiendo del problema aproximado (3.21), se elegirá una familia de espacios $V_h (h \in (0,1))$ definido por el procedimiento de elementos finitos (que se verá en las siguientes secciones), en este caso para interpoladores lineales, teniendo la propiedad de que V_h se aproxima a V cuando h se aproxima a cero en un sentido apropiado, esto es, por supuesto una propiedad indispensable para la convergencia del método Galerkin [Carrillo et al., 2008].

3.3.1 Principio del Método

Sea Ω un dominio de $\mathbb{R}^n$ ($n = 1, 2\ o\ 3$, en la práctica), de frontera $\Gamma = \partial\Omega$ y sobre la cual se busca resolver una ecuación en derivadas parciales con condiciones de frontera definidas. Se asume

que el problema de ecuaciones diferenciales es bien planteado, por lo que se garantiza la existencia y unicidad de su solución. Este problema puede escribirse como un problema variacional lineal abstracto:

$$\text{Encontrar} \quad u \in V \quad \text{tal que} \quad a(u,v) = f(v) = \langle f, v \rangle, \qquad \forall v \in V \tag{3.22}$$

donde el espacio V (es un espacio de Hilbert), la forma bilineal $a(\cdot,\cdot)$ y la forma lineal f sastisfacen las hipótesis del Teorema de Lax-Milgram [Ciarlet, 2002].

En el método del elemento finito se propone discretizar el problema considerado (3.22), utilizando el *método de Galerkin* para tal fin y de esta manera aproximar la solución del problema. Como ya se vio en la sección **3.2.1**, el método de Galerkin consiste en definir problemas similares en subespacios finito-dimensionales del espacio V; más específicamente, con cualquier subespacio de dimensión finita V_h de V, se asocia el siguiente problema discreto:

$$\text{Encontrar} \quad u_h \in V_h \quad \text{tal que} \quad a(u_h, v_h) = f(v_h) = \langle f, v_h \rangle, \qquad \forall v_h \in V_h \tag{3.23}$$

Aplicando el teorema de Lax-Milgram, se infiere que el problema tiene solución única u_h, la cual se llama *solución discreta*. En el caso de que la forma bilineal es simétrica, la solución discreta también se caracteriza por la siguiente propiedad:

$$\mathcal{J}(u_h) = \mathbf{inf}_{v_h \in V_h} \mathcal{J}(v_h) \tag{3.24}$$

donde el funcional $\mathcal{J}$ está dado por:

$$\mathcal{J}(v) = \frac{1}{2} a(v,v) - f(v) \tag{3.25}$$

Esta definición alternativa de la solución discreta se conoce como el *método de Ritz*.

El método del elemento finito, en su forma más simple, es una especificación del proceso de construcción del subespacio V_h, el cual se llama espacio del elemento finito, $V_h \subset V$. Esta construcción está caracterizada por tres aspectos [Ciarlet, 2002]:

- *Discretización.* Es necesario disponer de una descripción del dominio sobre el cual se desea trabajar. Para ello se establece un mallado o triangulación K_h sobre el conjunto $\bar{\Omega}$; es decir, el conjunto $\bar{\Omega}$ se escribe como la unión finita de elementos finitos $K \in K_h$.

- *Interpolación.* A continuación, es necesario tener alguna forma de representar el campo o los campos desconocidos. Para ello, estos campos se aproximan por funciones más simples (por ejemplo, polinomios de primer o segundo grado) definidas sobre cada uno de los elementos de la malla. La función $v_h \in V_h$ representa polinomios a trozos o por partes (*piecewise*), en el sentido de que, para cada elemento $K \in K_h$, los espacios $P_k = \{v_{h|k}\,;\quad v_h \in V_h\}$ consisten en polinomios.

- *Aproximación.* Debe existir una base en el espacio V_h cuyas funciones tengan pequeños soportes. Según el tipo de aproximación, se reemplaza no solamente el espacio V, de dimensión infinita, por el espacio V_h de dimensión finita; sino que igualmente la forma bilineal y el funcional lineal que definen el problema (ver [Manet, 2015]).

En la siguiente sección se muestra el método del elemento finito, con un ejemplo ilustrativo, para resolver un problema particular en dos dimensiones.

3.3.2 Discretización Espacial en Dos Dimensiones

3.3.2.1 Planteamiento del Problema

A manera de ilustración del método FEM, se considera el siguiente problema estacionario en dos dimensiones:

$$\begin{aligned} -\Delta p &= f && en\ \ \Omega \\ p &= 0 && en\ \ \Gamma \end{aligned} \tag{3.26}$$

donde Ω es un dominio acotado en un plano con frontera Γ, f es una función real en Ω continua por partes y acotada. En este contexto, Δ es el operador de Laplace definido por:

$$\Delta p = \frac{\partial^2 p}{\partial x_1^2} + \frac{\partial^2 p}{\partial x_2^2} \tag{3.27}$$

Se define el espacio lineal:

$$\begin{aligned} V = \{\, v : v \quad & \text{es una función continua en } \Omega, \\ & \frac{\partial v}{\partial x_1} \text{ y } \frac{\partial v}{\partial x_2} \text{ son continuas por tramos y acotadas en } \Omega, \\ & v = 0 \text{ en } \Gamma \,\} \end{aligned} \tag{3.28}$$

Por otro lado, para una función vectorial $\underline{\mathbf{b}} = (b_1, b_2)$, el *teorema de la divergencia* o *teorema de Gauss* indica que:

$$\int_\Omega \nabla \cdot \underline{\mathbf{b}} \; d\underline{x} = \int_\Gamma \underline{\mathbf{b}} \cdot \underline{\mathbf{n}} \; ds \tag{3.29}$$

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

$$\nabla \cdot \underline{\mathbf{b}} = \frac{\partial b_1}{\partial x_1} + \frac{\partial b_2}{\partial x_2} \tag{3.30}$$

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

$$\underline{\mathbf{b}} \cdot \underline{\mathbf{n}} = b_1 n_1 + b_2 n_2 \tag{3.31}$$

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 (3.29), respectivamente, entonces se observa que:

$$\int_\Omega \frac{\partial^2 v}{\partial x_i^2} w \; d\underline{x} + \int_\Omega \frac{\partial v}{\partial x_i} \frac{\partial w}{\partial x_i} \; d\underline{x} = \int_\Gamma \frac{\partial v}{\partial x_i} w v_i \; ds \qquad\qquad i = 1, 2 \tag{3.32}$$

Usando la definición del operador gradiente:

$$\nabla v = \left(\frac{\partial v}{\partial x_1}, \frac{\partial v}{\partial x_2} \right) \tag{3.33}$$

y sumando sobre $i = 1, 2$ en (3.32) se obtiene:

$$\int_\Omega \Delta v \, w \; d\underline{x} = \int_\Gamma \frac{\partial v}{\partial \underline{\mathbf{n}}} w \; ds \; - \int_\Omega \nabla v \, \nabla w \; d\underline{x} \tag{3.34}$$

donde la derivada normal es la *derivada direccional*:

$$\frac{\partial v}{\partial \underline{\mathbf{n}}} = \nabla v \cdot \underline{\mathbf{n}} = \frac{\partial v}{\partial x_1} n_1 + \frac{\partial v}{\partial x_2} n_2 \tag{3.35}$$

La relación (3.34) es la *fórmula de Green*, y también puede aplicarse en tres dimensiones.

3.3.2.2 Formulación Variacional o Forma Débil del Problema

Ahora, se introduce la notación:

$$\begin{aligned} a(p, v) &= \int_\Omega \nabla p \cdot \nabla v \; d\underline{x} \\ (f, v) &= \int_\Omega f v \; \underline{x} \end{aligned} \tag{3.36}$$

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

$$\begin{aligned} a\left(u, \alpha v + \beta w\right) &= \alpha a\left(u, v\right) + \beta a\left(u, w\right) \\ a\left(\alpha u + \beta v, w\right) &= \alpha a\left(u, w\right) + \beta a\left(v, w\right) \end{aligned} \tag{3.37}$$

para $\alpha, \beta \in \mathbb{R}$ y $u, v, w \in V$.

También se define el funcional $F : V \to \mathbb{R}$ como:

$$F(v) = \frac{1}{2}a(v,v) - (f,v) \qquad v \in V \tag{3.38}$$

Entonces, el problema (3.26) puede formularse como un problema de minimización:

$$\text{Encontrar} \quad p \in V \quad \text{tal que} \quad F(p) \leq F(v) \qquad \forall v \in V \tag{3.39}$$

El problema (3.39) se conoce como la *forma variacional de Ritz* de (3.26), y es equivalente al *problema variacional de Galerkin* (3.42) que se presenta más adelante [Chen et al., 2006].

Multiplicando la primera ecuación de (3.26) por $v \in V$ e integrando sobre Ω, se observa que:

$$-\int_\Omega \nabla p\, v \; d\underline{x} = \int_\Omega f v \; dx \tag{3.40}$$

Aplicando (3.34) a la ecuación anterior y usando las condiciones de frontera homogéneas, se tiene que:

$$\int_\Omega \nabla p \nabla v \; d\underline{x} = \int_\Omega f v \; dx \qquad \forall v \in V \tag{3.41}$$

Entonces se puede plantear en la forma variacional de Galerkin:

$$\text{Encontrar} \quad p \in V \quad \text{tal que} \quad a(p,v) = (f,v) \qquad \forall v \in V \tag{3.42}$$

3.3.2.3 Construcción del Mallado

Una vez planteado el problema en su forma variacional, se construyen los elementos finitos para resolver (3.26). Por simplicidad se asume que Ω es un dominio poligonal. El tratamiento para un dominio curvo puede observarse en [Chen, 2005].

El método de elementos finitos considera a la región entera (el dominio) como una suma de subregiones o elementos discretos, los cuales poseen características y condiciones determinadas de vinculo entre sí. Por lo tanto, se debe obtener una discretización del dominio; es decir, se debe generar una malla de elementos finitos que cubra todo el dominio, como se observa en la **Figura 3**.

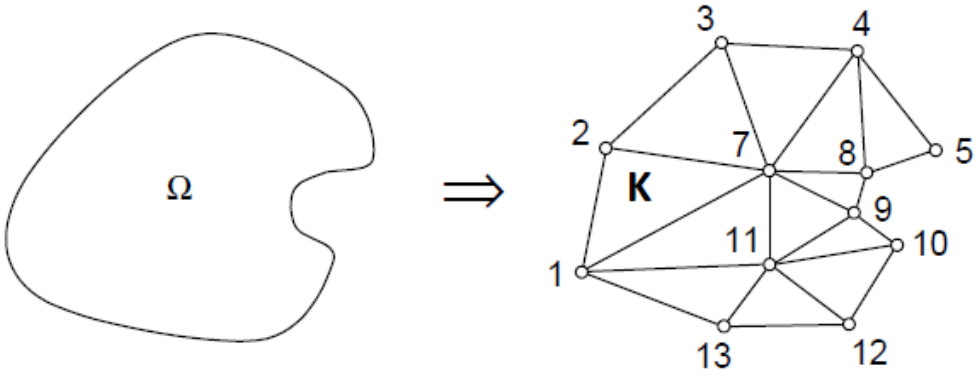


Figura 3: *Discretización del dominio Ω, en elementos finitos.*

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

$$\bar{\Omega} = \bar{K}_1 \cup \bar{K}_2 \cup \ldots \cup \bar{K}_M \tag{3.43}$$

tal que ningún vértice de un triángulo cae dentro del interior del borde de otro triángulo, donde $\bar{\Omega}$ representa la cerradura de Ω (es decir $\bar{\Omega} = \Omega \cup \Gamma$) y se tiene un significado similar para cada K_i (**Figura 3**).

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

$$\begin{aligned} diam(K) &= el\ borde\ más\ largo\ de\ \ \bar{K} \\ h &= máx\ diam(K) \qquad\qquad K \in K_h \end{aligned} \tag{3.44}$$

Ahora, se introduce el espacio del elemento finito:

$$V_h = \{\, v : v \quad \text{es una función continua en } \Omega,$$
$$v \text{ es lineal en cada triángulo } K \in K_h,$$
$$v = 0 \text{ en } \Gamma \,\} \tag{3.45}$$

Se nota que $V_h \subset V$. Por lo tanto, el método del elemento finito para (3.26) se formula como:

$$\text{Encontrar} \quad p_h \in V_h \quad \text{tal que} \quad a(p_h, v) = (f, v) \qquad \forall v \in V_h \tag{3.46}$$

También, se puede observar que (3.46) es equivalente a un problema de minimización discreto:

$$\text{Encontrar} \quad p_h \in V_h \quad \text{tal que} \quad F(p_h) \leq F(v) \qquad \forall v \in V_h \tag{3.47}$$

3.3.2.4 Interpolación y Aproximación

Se nombran los *nodos* (vértices) de los triángulos en K_h como $\underline{x}_1, \underline{x}_2, \ldots, \underline{x}_M$. Las funciones base ϕ_i en $V_h, i = 1, 2, \ldots, M$, están definidas por la función sombrero o *función Chapeau*, que se muestra a continuación:

$$\phi_i\left(x_j\right) = \begin{cases} 1 & , si \;\; i = j \\ 0 & , si \;\; i \neq j \end{cases} \tag{3.48}$$

El soporte de ϕ_i, por el ejemplo el conjunto de $\underline{x}$ donde $\phi_i(\underline{x}) \neq 0$, está formado por los triángulos con el nodo común $\underline{x}_i$, como se observa en la **Figura 4**.

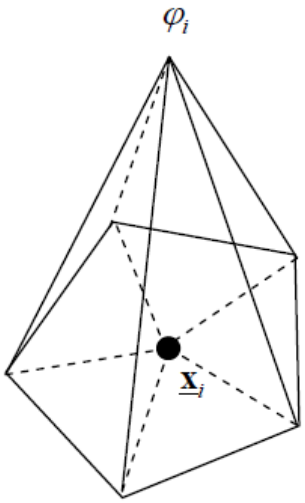


Figura 4: *Función base en dos dimensiones.*

Haciendo que M sea el número de vértices en K_h, y que por conveniencia los primeros M vértices sean los interiores. Entonces, se tiene que cualquier función $v \in V_h$ tiene una única representación:

$$v(\underline{x}) = \sum_{i=1}^{M} v_i \phi_i(\underline{x}) \qquad \underline{x} \in \Omega \tag{3.49}$$

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

El problema (3.46) puede escribirse en forma matricial:

$$\underline{\underline{\mathbf{A}}}\,\underline{\mathbf{p}} = \underline{\mathbf{f}} \tag{3.50}$$

donde la matriz $\underline{\underline{\mathbf{A}}}$ y los vectores $\underline{\mathbf{p}}$ y $\underline{\mathbf{f}}$ son:

$$\begin{aligned} \underline{\underline{\mathbf{A}}} &= (a_{ij}) & a_{ij} &= a(\phi_i, \phi_j) \\ \underline{\mathbf{p}} &= (p_j) & & \\ \underline{\mathbf{f}} &= (f_j) & f_j &= (f, \phi_j) \\ & & i, j &= 1, 2, \ldots, M \end{aligned} \tag{3.51}$$

A la matriz $\underline{\underline{\mathbf{A}}}$ se le llama *matriz de rigidez*, y a $\underline{\mathbf{f}}$ vector fuente.

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

$$a_{ij} = a(\phi_i, \phi_j) = \sum_{K \in K_h} a^K(\phi_i, \phi_j)$$

donde :

$$a_{ij}^K \equiv a^K(\phi_i, \phi_j) = \int_K \nabla\phi_i \cdot \nabla\phi_j \, d\underline{x} \tag{3.52}$$

La matriz de rigidez $\underline{\underline{\mathbf{A}}}$ es simétrica y positiva definida. De forma particular, es no singular. Consecuentemente, el sistema matricial (3.50) y por consiguiente el problema de elementos finitos (3.46) tienen una solución única. El sistema matricial se resuelve con alguno de los métodos que se describirán en la sección **3.6**.

Cabe recordar que la condición de frontera tipo Dirichlet se define como una condición de carga constante en la frontera. El resultado de sustituir valores constantes de la carga conocida en la frontera es una reducción en el número de filas de la matriz de rígidez. La matriz final contiene el número de ecuaciones no conocidas que son el número de nodos menos el número de nodos con condiciones de fronteras Dirichlet.

Para los nodos localizados en la frontera de la región de flujo dentro del elemento puede especificarse una condición de tipo Neumann, asumiendo que una condición Dirichlet no está definida a lo largo del segmento de frontera. Para ello se reemplaza el término en la frontera por un valor de flujo conocido. Así se crea el término que representa el flujo a través de la frontera del dominio.

El tratamiento de un problema para condiciones de frontera generales se verá en la siguiente sección.

3.4 Método del Elemento Finito para Ecuaciones Advectivo - Difusivo - Reactivas con Condiciones de Frontera Generales

3.4.1 Forma de la Divergencia del Operador Diferencial

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

$$\mathcal{L}u = Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu \tag{3.53}$$

donde: $u_x = \frac{\partial u}{\partial x}$, $u_y = \frac{\partial u}{\partial y}$, $u_{xx} = \frac{\partial u}{\partial x \partial x}$, $u_{yy} = \frac{\partial u}{\partial y \partial y}$, $u_{xy} = \frac{\partial u}{\partial x \partial y}$.

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

$$\mathcal{L}u = -\nabla \cdot (\underline{\underline{\mathbf{a}}} \cdot \nabla u) + \nabla \cdot (\underline{\mathbf{b}} u) + \mathbf{c}u \tag{3.54}$$

donde: $\underline{\underline{\mathbf{a}}} = \left(\begin{smallmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{smallmatrix}\right)$ es una matriz simétrica, $\underline{\mathbf{b}} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$ es un vector y $\mathbf{c}$ es un escalar. Se observa que fácilmente es identificable con la forma diferencial de las ecuaciones (2.34) y (2.35).

Entonces se tiene que:

$$\mathcal{L}u = -\nabla \cdot \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \cdot \begin{pmatrix} u_x \\ u_y \end{pmatrix} + \nabla \cdot \begin{pmatrix} b_1 u \\ b_2 u \end{pmatrix} + \mathbf{c}u \tag{3.55}$$

Desarrollando (3.55) e igualando a la ecuación (3.53) se tiene que:

$$\begin{aligned} \mathcal{L}u = & -a_{11}u_{xx} - a_{11x}u_x - a_{12x}u_y - a_{12}u_{xy} - a_{12y}u_x - a_{12}u_{xy} - a_{22}u_{yy} - a_{22y}u_y \\ & + b_{1x}u + b_1 u_x + b_{2y}u + b_2 u_y + \mathbf{c}u \\ & = Au_{xx} + Bu_{xy} + Cu_{yy} + Du_x + Eu_y + Fu \end{aligned} \tag{3.56}$$

Por lo tanto:

$$\underline{\underline{\mathbf{a}}} = -\begin{pmatrix} A & \frac{B}{2} \\ \frac{B}{2} & C \end{pmatrix}$$

$$\underline{\mathbf{b}} = \begin{pmatrix} D - A_x - \frac{1}{2}B_y \\ E - C_y - \frac{1}{2}B_x \end{pmatrix}$$

$$\mathbf{c} = F - \left(D_x - A_{xx} - \frac{1}{2}B_{xy}\right) - \left(E_y - C_{yy} - \frac{1}{2}B_{xy}\right) = F + A_{xx} + B_{xy} + C_{yy} - D_x - E_y \quad (3.57)$$

La ecuación será elíptica si la matriz $\underline{\underline{\mathbf{a}}}$ es positiva definida, en cuyo caso:

$$det\ \underline{\underline{\mathbf{a}}} = a_{11}a_{22} - a_{12}^2 > 0 \quad (3.58)$$

La ecuación es parabólica si:

$$det\ \underline{\underline{\mathbf{a}}} = a_{11}a_{22} - a_{12}^2 = 0 \quad (3.59)$$

Y será hiperbólica si:

$$det\ \underline{\underline{\mathbf{a}}} = a_{11}a_{22} - a_{12}^2 < 0 \quad (3.60)$$

3.4.2 Formulación FEM para Condiciones de Frontera Generales

Se tiene el problema:

$$\begin{aligned} \mathcal{L}u &= f && en\ \ \Omega \\ u(\underline{\mathbf{x}}) &= g(\underline{\mathbf{x}}) && en\ \ \Gamma_D \\ \mathbf{a}_n \frac{\partial u}{\partial \underline{n}}(\underline{\mathbf{x}}) &= h\left(\underline{\mathbf{x}}\right) - e\left(\underline{\mathbf{x}}\right) u(\underline{\mathbf{x}}) && en\ \ \Gamma_N \end{aligned} \quad (3.61)$$

Donde Ω representa el dominio espacial, Γ_D y Γ_N las fronteras del dominio, $\mathcal{L}u$ la forma de la divergencia (3.54) y $\underline{\underline{\mathbf{a}}}(\underline{\mathbf{x}}, t)$ es simétrica y positiva definida. Esto se muestra en la **Figura 5**.

Además, las fronteras del dominio presentan las siguientes características:

$$\begin{aligned} \Gamma_N \cap \Gamma_D &= \varnothing \\ \Gamma_N \cup \Gamma_D &= \Gamma \end{aligned} \tag{3.62}$$

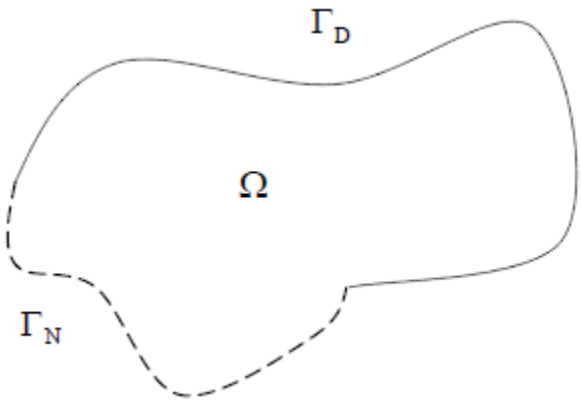


Figura 5: *Representación de un dominio bidimensional donde* $\Omega \in \mathbb{R}^2$, Γ_D *representa las fronteras Dirichlet y* Γ_N *las fronteras Neumann.*

Tomando en cuenta la última ecuación de (3.61), se tiene que:

$$\begin{aligned} &si \;\; e(x) = 0 \;\; \textit{entonces la frontera es Neumann} \\ &si \;\; e(x) \neq 0 \;\; \textit{entonces la frontera es Robin} \end{aligned} \tag{3.63}$$

Se tiene que:

$$\begin{aligned} u &= u_0(\underline{\mathbf{x}}) + v(\underline{\mathbf{x}}) && en \;\; \Gamma_D \\ u &= v(\underline{\mathbf{x}}) && en \;\; \Gamma_N \end{aligned} \tag{3.64}$$

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 la primera ecuación de (3.61) por una función $w(\underline{x})$, resulta:

$$
\begin{aligned}
w\mathcal{L}u &= -w\nabla\cdot(\underline{\underline{\mathbf{a}}}\cdot\nabla u) + w\nabla\cdot(\underline{\mathbf{b}}u) + \mathbf{c}wu \\
&= -\nabla\cdot\left(w\underline{\underline{\mathbf{a}}}\nabla u\right) + \nabla w\cdot\underline{\underline{\mathbf{a}}}\nabla u + \nabla\cdot(w\underline{\mathbf{b}}u) - u\underline{\mathbf{b}}\cdot\nabla w + cwu \\
&= fw
\end{aligned} \tag{3.65}
$$

Integrando sobre Ω y aplicando el Teorema de Gauss o de la divergencia (3.29):

$$
\begin{aligned}
\int_\Omega w\mathcal{L}u\,d\underline{\mathbf{x}} &= \int_\Omega \left[-w\nabla\cdot(\underline{\underline{\mathbf{a}}}\cdot\nabla u) + w\nabla\cdot(\underline{\mathbf{b}}u) + \mathbf{c}wu\right]\,d\underline{\mathbf{x}} \\
&= \int_\Omega \left[-\nabla\cdot\left(w\underline{\underline{\mathbf{a}}}\nabla u\right) + \nabla w\cdot\underline{\underline{\mathbf{a}}}\nabla u + \nabla\cdot(w\underline{\mathbf{b}}u) - u\underline{\mathbf{b}}\cdot\nabla w + cwu\right]\,d\underline{\mathbf{x}} \\
&= \int_\Gamma(-w\underline{\underline{\mathbf{a}}}\nabla u)\cdot\underline{\mathbf{n}}ds + \int_\Omega \nabla w\cdot(\underline{\underline{\mathbf{a}}}\nabla u)d\underline{\mathbf{x}} + \int_\Gamma w\underline{\mathbf{b}}u\cdot\underline{\mathbf{n}}ds - \int_\Omega \nabla w\cdot(\underline{\mathbf{b}}u)d\underline{\mathbf{x}} + \int_\Omega wcu\,d\underline{\mathbf{x}} \\
&= \int_\Omega \left[\nabla w\cdot(\underline{\underline{\mathbf{a}}}\nabla u) - \nabla w\cdot(\underline{\mathbf{b}}u) + cwu\right]\,d\underline{\mathbf{x}} - \int_\Gamma w(\underline{\underline{\mathbf{a}}}\nabla u - \underline{\mathbf{b}}u)\cdot\underline{\mathbf{n}}\,ds \\
&= \int_\Omega fwd\underline{\mathbf{x}}
\end{aligned} \tag{3.66}
$$

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

y tomando en cuenta la derivada direccional (3.35), se tiene que:

$$
\begin{aligned}
\int_\Omega w\mathcal{L}u\,d\underline{\mathbf{x}} &= \int_\Omega \left[\nabla w\cdot(\underline{\underline{\mathbf{a}}}\nabla u) - \nabla w\cdot(\underline{\mathbf{b}}u) + cwu\right]\,d\underline{\mathbf{x}} - \int_\Gamma w(\mathbf{a}_n\frac{\partial u}{\partial\underline{n}} - \mathbf{b}_n u)\,ds \\
&= \int_\Omega fwd\underline{\mathbf{x}}
\end{aligned} \tag{3.67}
$$

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

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

$$\int_{\Omega} w\mathcal{L}u\, d\underline{\mathbf{x}} = \int_{\Omega} \left[\nabla w \cdot (\underline{\underline{\mathbf{a}}}\nabla u) - \nabla w \cdot (\underline{\mathbf{b}} u) + cwu\right]\, d\underline{\mathbf{x}} - \int_{\Gamma_N} w(\mathbf{a}_n \frac{\partial u}{\partial \underline{n}} - \mathbf{b}_n u)\, ds$$
$$= \int_{\Omega} f w d\underline{\mathbf{x}} \tag{3.68}$$

Sustituyendo en la ecuación anterior las condiciones de frontera Neumann tomadas de (3.61) se tiene:

$$\int_{\Omega} \left[\nabla w \cdot (\underline{\underline{\mathbf{a}}}\nabla u) - \nabla w \cdot (\underline{\mathbf{b}} u) + cwu\right]\, d\underline{\mathbf{x}} + \int_{\Gamma_N} wu(e - \underline{\mathbf{b}}_n)\, ds = \int_{\Omega} f w d\underline{\mathbf{x}} + \int_{\Gamma_N} hw\, ds \tag{3.69}$$

Ahora, se elige un subespacio de dimensión finita V_N conformado por una base $\{w_1, w_2, \ldots, w_N\}$, tal que se cumpla lo siguiente:

$$\hat{v}(x) = \sum_{j=1}^{N} v_j w_j(x) \approx v(x) \tag{3.70}$$

y tomando en cuenta (3.64), se tiene:

$$\sum_{j=1}^{N} v_j \int_{\Omega} \left[\nabla w_i \cdot (\underline{\underline{\mathbf{a}}}\nabla w_j) - \nabla w_i \cdot (\underline{\mathbf{b}} w_j) + c w_i w_j\right]\, d\underline{\mathbf{x}} + \int_{\Gamma_N} w_i w_j (e - \underline{\mathbf{b}}_n)\, ds$$
$$= \int_{\Omega} f w_i d\underline{\mathbf{x}} + \int_{\Gamma_N} h w_i\, ds - \int_{\Omega} \left[\nabla w_i \cdot (\underline{\underline{\mathbf{a}}}\nabla u_0) - \nabla w_i \cdot (\underline{\mathbf{b}} u_0) + c w_i u_0\right]\, d\underline{\mathbf{x}} \tag{3.71}$$

Haciendo:

$$A_{ij} = \int_{\Omega} \left[\nabla w_i \cdot (\underline{\underline{\mathbf{a}}}\nabla w_j) - \nabla w_i \cdot (\underline{\mathbf{b}} w_j) + c w_i w_j\right]\, d\underline{\mathbf{x}} + \int_{\Gamma_N} w_i w_j (e - \underline{\mathbf{b}}_n)\, ds$$
$$r_{ij} = \int_{\Omega} f w_i d\underline{\mathbf{x}} + \int_{\Gamma_N} h w_i\, ds - \int_{\Omega} \left[\nabla w_i \cdot (\underline{\underline{\mathbf{a}}}\nabla u_0) - \nabla w_i \cdot (\underline{\mathbf{b}} u_0) + c w_i u_0\right]\, d\underline{\mathbf{x}} \tag{3.72}$$

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

$$\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{v}} = \underline{\mathbf{r}} \tag{3.73}$$

De forma que la solución del problema está dada por:

$$\underline{\mathbf{v}} = \underline{\underline{\mathbf{A}}}^{-1} \cdot \underline{\mathbf{r}} \tag{3.74}$$

3.5 Método de Crank-Nicholson: Discretización Temporal

En las sección anterior se presentó el método del elemento finito para el problema en estado estable. En esta sección se presenta la forma de resolver un problema transitorio.

Se considera el problema:

$$\begin{aligned} \alpha \frac{\partial u}{\partial t} + \mathcal{L}u &= f\left(\underline{\mathbf{x}}, t\right) && en \;\; \Omega \times [t_0, t_f] \\ u\left(x, t\right) & && en \;\; \Gamma_D \times [t_0, t_f] \\ a_n \frac{\partial u}{\partial n} &= h\left(\underline{\mathbf{x}}, t\right) - e\left(\underline{\mathbf{x}}, t\right) u && en \;\; \Gamma_N \times [t_0, t_f] \end{aligned} \tag{3.75}$$

Donde $\mathcal{L}u$ es el mismo que se definió en (3.54), Γ_D representa la frontera Dirichlet y Γ_N, la frontera Neumann. Además, se toma en cuenta la condición inicial:

$$u\left(\underline{\mathbf{x}}, t_0\right) = u_0(\underline{\mathbf{x}}) \tag{3.76}$$

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

$$u\left(t + \frac{1}{2}\Delta t\right) = \left(1 - \theta\right) u(t) + \theta u\left(t + \Delta t\right) + \mathcal{O}\left(\Delta t^2\right) \tag{3.77}$$

$$u_t\left(t+\frac{1}{2}\Delta t\right)=\frac{u\left(t+\Delta t\right)-u(t)}{\Delta t}+\mathcal{O}\left(\Delta t^2\right) \tag{3.78}$$

Donde $0<\theta\leq 1$, $\mathcal{O}$ es el orden del error y $u_t=\frac{\partial u}{\partial t}$. Ahora, usando la notación $u^k=u\left(x,t_k\right)=u\left(x,t_0+k\Delta t\right)$, y sustituyendo (3.77) en (3.78) se tiene:

$$u_t^{k+\frac{1}{2}}=\frac{1}{\theta}\left(u^{k+\frac{1}{2}}-\left(1-\theta\right)u^k-\theta u^k\right)+\mathcal{O}\left(\Delta t^2\right) \tag{3.79}$$

Si se supone que se avanza hasta un tiempo $k+\frac{1}{2}$, entonces la primera ecuación del problema (3.75), puede reescribirse como:

$$\alpha u_t^{k+\frac{1}{2}}+\mathcal{L}u^{k+\frac{1}{2}}=f^{k+\frac{1}{2}} \tag{3.80}$$

donde, incorporando (3.79) a la anterior ecuación se tiene:

$$\mathcal{L}u^{k+\frac{1}{2}}+\frac{\alpha u^{k+\frac{1}{2}}}{\theta\Delta t}=f^{k+\frac{1}{2}}+\frac{\alpha u^k}{\theta\Delta t} \tag{3.81}$$

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 (3.81), se puede obtener de forma iterativa el valor para el siguiente paso en el tiempo, usando:

$$u^{k+1}(x)=\frac{u^{k+\frac{1}{2}}-\left(1-\theta\right)u^k}{\theta} \tag{3.82}$$

Es de destacarse que el problema (3.80), puede reescribirse de la siguiente manera:

$$\mathcal{L}'v=f'(\underline{\mathbf{x}}) \tag{3.83}$$

donde:

$$\begin{aligned} \mathcal{L}'v &= -\nabla \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla v + \nabla(\underline{\mathbf{b}} v) + \left(\mathbf{c} + \frac{\alpha}{\theta \Delta t}\right) v \\ f'(\underline{\mathbf{x}}) &= f^{k+\frac{1}{2}} + \frac{\alpha u^k}{\theta \Delta t} \\ v(\underline{\mathbf{x}}) &= u\left(\underline{\mathbf{x}}, t^{k+\frac{1}{2}}\right) \end{aligned} \tag{3.84}$$

Las condiciones de frontera quedan definidas como:

$$\begin{aligned} v(\underline{\mathbf{x}}) &= g\left(\underline{\mathbf{x}}, t^{k+\frac{1}{2}}\right) && \forall \underline{\mathbf{x}} \in \Gamma_D \\ \mathbf{a}_n \frac{\partial v(\underline{\mathbf{x}})}{\partial n} &= h\left(x, t^{k+\frac{1}{2}}\right) - e\left(x, t^{k+\frac{1}{2}}\right) v && \forall \underline{\mathbf{x}} \in \Gamma_N \end{aligned} \tag{3.85}$$

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

$$u^{k+1}(x) = \frac{v - (1-\theta)u^k}{\theta} \tag{3.86}$$

Se observa que, para la resolución de (3.84) por el método del elemento finito, se requiere resolver la integral:

$$\int_\Omega u^k w_i \, d\underline{\mathbf{x}} \tag{3.87}$$

la cual tiene la siguiente solución:

$$\int_\Omega u^k w_i \, d\underline{\mathbf{x}} = \sum_j M_{ij} u_j^k \tag{3.88}$$

donde:

$$M_{ij} = \int_\Omega w_i w_j \, d\underline{\mathbf{x}} \tag{3.89}$$

3.6 Solución de Sistemas de Ecuaciones Lineales

El sistema de ecuaciones algebraicas lineales $\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}}$, en donde la matriz $\underline{\underline{\mathbf{A}}}$ es bandada (tiene muchos elementos nulos), no singular y de tamaño $N \times N$, y $b \in R^N$, tiene como solución $\underline{\mathbf{u}}^* = \underline{\underline{\mathbf{A}}}^{-1} \cdot \underline{\mathbf{b}} \in R^N$, mientras que $\underline{\mathbf{u}}$ representa una solución.

Todos los métodos numéricos desarrollados para resolver un sistema de ecuaciones algebraicas lineales $\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}}$ pueden clasificarse en dos grandes grupos:

- métodos exactos o directos
- métodos aproximados o iterativos

En los métodos directos los algoritmos permiten obtener la solución de un sistema mediante un número finito de operaciones aritméticas. Los metodos directos son convenientes en el caso de que $\underline{\underline{\mathbf{A}}}$ es una matriz rala o con una topología específica como: tridiagonal, triangular, simétrica, etc., o de dimensión pequeña (cuando el número N no es muy grande).

Sin embargo, si la matriz $\underline{\underline{\mathbf{A}}}$ es densa, o de dimensión grande (por ejemplo, que N sea mayor que 10^6), a menudo un método iterativo es más efectivo y económico. En este tipo de 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.

Los métodos iterativos rara vez se usan para resolver sistemas de ecuaciones 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 [Carrillo et al., 2008].

Cabe 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 el método numérico que minimice el tiempo invertido en este proceso.

3.6.1 Métodos Directos

Todos los métodos exactos de solución de los sistemas de ecuaciones algebraicas lineales están basados en una forma de factorización de la matriz $\underline{\underline{\mathbf{A}}}$ en el producto de dos matrices que tienen una estructura simple [Skiba, 2005]. En estos métodos, la solución $\underline{\mathbf{u}}$ se obtiene en un número fijo de pasos y sólo está sujeta 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 et al., 2008]. A continuación se describe el método de la factorización LU, debido a que se ocupará en la implementación computacional del problema de convección.

3.6.1.1 Eliminación Gaussiana

Este método quizá sea el más utilizado para encontrar la solución de un sistema de ecuaciones lineales, usando métodos directos. La eliminación Gaussiana se basa en la aplicación de operaciones elementales a renglones o columnas de tal forma que es posible obtener matrices equivalentes. Sea dado el sistema de un número arbitrario N de ecuaciones algebraicas lineales, con N incógnitas:

$$\sum_{j=1}^{N} a_{ij}^{(0)} x_j = a_{i,n+1}^{(0)} \qquad i = 1, 2, \ldots, N \tag{3.90}$$

Si $a_{11}^{(0)} \neq 0$ y los pivotes $a_{ii}^{(i-1)}, \quad i = 2, 3, \ldots, N$ de las demás filas, que se obtienen en el curso de los cálculos, son distintos de cero, entonces, el sistema (3.90) se reduce a la siguiente forma triangular (la carrera directa o eliminación hacia adelante):

$$x_i + \sum_{j=i+1}^{N} a_{ij}^{(i)} x_j = a_{i,n+1}^{(i)} \qquad i = 1, 2, \ldots, N \tag{3.91}$$

donde:

$$\begin{aligned} k &= 1, 2, \ldots, N\{ j = k+1, \ldots, N+1\{ \\ a_{kj}^{(k)} &= \frac{a_{kj}^{(k-1)}}{a_{kk}^{(k-1)}}; \\ i &= k+1, \ldots, N\{ \\ a_{ij}^{(k)} &= a_{ij}^{(k-1)} - a_{kj}^{(k)} \cdot a_{ik}^{(k-1)} \}\}\} \end{aligned} \tag{3.92}$$

La carrera inversa (sustitución hacia atrás), donde las incógnitas se calculan por sustitución regresiva, se realiza por medio de las siguientes fórmulas:

$$\begin{aligned} x_n &= a_{n,n+1}^{(n)}; \\ i &= N-1, N-2, \ldots, 1 \\ x_i &= a_{i,n+1}^{(i)} - \sum_{j=i+1}^{N} a_{ij}^{(i)} x_j \end{aligned} \tag{3.93}$$

El algoritmo de la eliminación gaussiana no es muy eficiente puesto que, en general, para el sistema de N ecuaciones, con una N grande, se requiere un almacenamiento computacional de N^2 para todas las entradas de la matriz $\underline{\underline{\mathbf{A}}}$; además $N^3/3 + \mathcal{O}(N^2)$ para multiplicaciones y $N^3/3 + \mathcal{O}(N^2)$ para adiciones, lo cual es muy costoso computacionalmente. No obstante, la eliminación Gaussiana puede observarse como un punto de partida en la teoría de los otros métodos directos [Allen et al., 1988].

3.6.1.2 Descomposición LU

Sea $\underline{\underline{\mathbf{U}}}$ la matriz triangular superior, obtenida por eliminación bandada de la matriz $\underline{\underline{\mathbf{A}}}$; 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 principal. Las entradas de $\underline{\underline{\mathbf{L}}}^{-1}$ pueden obtenerse a partir de los coeficientes m_{ij} definidas en el método de la eliminación bandada (ver siguiente sección), y pueden ser almacenadas estrictamente en lugar de las entradas de la diagonal inferior de $\underline{\underline{\mathbf{A}}}$ que ya fueron eliminadas. Esto proporciona una factorización $\underline{\underline{\mathbf{A}}} = \underline{\underline{\mathbf{L}}} \cdot \underline{\underline{\mathbf{U}}}$ en la misma matriz $\underline{\underline{\mathbf{A}}}$ ahorrando espacio de memoria.

De este modo el problema original $\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}}$ se escribe como $\underline{\underline{\mathbf{L}}} \cdot \underline{\underline{\mathbf{U}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}}$, y se reduce a la solución sucesiva de los dos sistemas lineales triangulares:

$$\underline{\underline{\mathbf{L}}} \cdot \underline{\mathbf{y}} = \underline{\mathbf{b}} \qquad\qquad \underline{\underline{\mathbf{U}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{y}} \tag{3.94}$$

La descomposición $\underline{\underline{\mathbf{L}}} \cdot \underline{\underline{\mathbf{U}}}$ requiere también $N^3/3$ operaciones aritméticas para la matriz llena, pero sólo Nb^2 operaciones aritméticas para la matriz con un ancho de banda b siendo esto más económico computacionalmente.

3.6.1.3 Eliminación Bandada

Cuando se usa el orden lexicográfico natural de los nodos se puede lograr un considerable ahorro de almacenamiento y recursos computacionales usando el algoritmo de la eliminación bandada. Este algoritmo consiste en triangularizar la matriz $\underline{\underline{\mathbf{A}}}$ por medio de una eliminación hacia adelante, operando solamente en las entradas de la matriz dentro de una banda central que contiene todos los elementos no-cero.

De este modo el renglón j (que inicia con $j = 1$) se multiplica por $m_{ij} = a_{ij}/a_{jj}$ y el resultado se resta del 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 principal. Así, es posible resolver el sistema resultante al sustituir en forma inversa las incógnitas. Si no se requieren intercambios de renglones (por ejemplo, para evitar la división entre cero al calcular m_{ij}) entonces todas las entradas de la matriz $\underline{\underline{\mathbf{U}}}$ serán cero, excepto en la diagonal principal.

3.6.2 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 sobre estos métodos, se recomienda ver [Allen et al., 1988] y [Saad, 2000].

En contraste con los métodos directos, los métodos iterativos comienzan con una aproximación inicial $\underline{\mathbf{u}}^0$ para el vector solución $\underline{\mathbf{u}}$ y producen mejores aproximaciones mediante una secuencia de iteraciones. Mientras que estos métodos no producen formalmente $\underline{\mathbf{u}}$ en un número finito de pasos, se puede terminar el proceso después de un número finito de iteraciones, cuando se haya producido una respuesta de aproximación lo suficientemente buena. La mayoría de los esquemas iterativos poseen el rasgo atractivo de requerir operaciones aritméticas solamente en las entradas no-cero de la matriz $\underline{\underline{\mathbf{A}}}$ [Allen et al., 1988].

3.6.2.1 Convergencia de las iteraciones

Cada método iterativo genera una sucesión de soluciones aproximadas $\{\underline{\mathbf{u}}^{(k)}\}_{k=1}^{\infty}$ a partir de un vector inicial $\underline{\mathbf{u}}^0$. Para hacer iteraciones es conveniente reescribir el problema $\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}}$ en la forma equivalente:

$$\underline{\mathbf{u}} = \underline{\underline{\mathbf{B}}} \cdot \underline{\mathbf{u}} + \underline{\mathbf{d}} \tag{3.95}$$

para alguna matriz fija $\underline{\underline{\mathbf{B}}}$ y un vector $\underline{\mathbf{d}}$, los cuales se pueden definir de varias maneras. Y se considera el método de las iteraciones sucesivas:

$$\underline{\mathbf{u}}^{(k)} = \underline{\underline{\mathbf{B}}} \cdot \underline{\mathbf{u}}^{(k-1)} + \underline{\mathbf{d}} \qquad \forall\, k = 1, 2, 3, \ldots \tag{3.96}$$

donde para iniciar los cálculos se elige un vector $\underline{\mathbf{u}}^0$ inicial. Este vector se considera como la aproximación inicial de la solución exacta $\underline{\mathbf{u}}^* = \underline{\underline{\mathbf{B}}} \cdot \underline{\mathbf{u}}^* + \underline{\mathbf{d}}$ del problema (3.95), y las iteraciones $\{\underline{\mathbf{u}}^{(k)}\}$ se llaman aproximaciones sucesivas de la solución exacta.

Como el problema es lineal, existen sólo dos opciones: las iteraciones $\{\underline{\mathbf{u}}^{(k)}\}$ convergen hacia la solución exacta $\underline{\mathbf{u}}^*$ o divergen. La convergencia depende sólo de las propiedades de la matriz $\underline{\underline{\mathbf{B}}}$ y no depende de la selección del vector inicial $\underline{\mathbf{u}}^0$; esta convergencia del método iterativo se garantiza con el siguiente teorema [Skiba, 2005]:

Teorema 1. *Si* $\|\underline{\underline{\mathbf{B}}}\| < 1$ *por lo menos en una norma matricial, entonces el sistema (3.95) tiene una solución única* $\underline{\mathbf{u}}^*$*, y las iteraciones* $\{\underline{\mathbf{u}}^{(k)}\}$ *definidas por la fórmula (3.96) convergen hacia*

la solución exacta $\underline{\mathbf{u}}^*$ *para cualquier vector inicial* $\underline{\mathbf{u}}^0$ *con la velocidad equivalente a la de una progresión geométrica con la razón* $\|\underline{\underline{\mathbf{B}}}\|$.

Se nota que mientras menor sea la norma de la matriz $\underline{\underline{\mathbf{B}}}$, más rápida es la convergencia; en el caso cuando $\|\underline{\underline{\mathbf{B}}}\|$ 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 $\|\underline{\underline{\mathbf{B}}}\|$ es pequeña ya que la convergencia es rápida.

La velocidad de convergencia de los métodos iterativos dependen de las propiedades espectrales de la matriz de coeficientes del sistema de ecuaciones, cuando el operador diferencial $\mathcal{L}$ de la ecuación del problema a resolver es auto-adjunto se obtiene una matriz simétrica y positivo definida y el número de condicionamiento de la matriz $\underline{\underline{\mathbf{A}}}$, es por definición [Carrillo et al., 2008]:

$$cond\left(\underline{\underline{\mathbf{A}}}\right) = \frac{\lambda_{mx}}{\lambda_{mn}} \geq 1 \tag{3.97}$$

donde λ_{mx} y λ_{mn} es el máximo y mínimo de los eigenvalores de la matriz $\underline{\underline{\mathbf{A}}}$. Si el número de condicionamiento es cercano a 1, los métodos numéricos al solucionar el problema convergerá en pocas iteraciones, en caso contrario se requerirán muchas iteraciones. Frecuentemente al usar el método de elemento finito se tiene una velocidad de convergencia de $\mathcal{O}\left(\frac{1}{h^2}\right)$ en el mejor de los casos, donde h es la máxima distancia de separación entre nodos continuos de la partición, es decir, que poseen una pobre velocidad de convergencia cuando $h \to 0$ [Carrillo et al., 2008].

Entre los métodos más usados para el tipo de problemas tratados en el presente trabajo podemos encontrar: Jacobi, Gauss-Seidel, Richardson, relajación sucesiva (SOR), gradiente conjugado (CG) y gradiente conjugado precondicionado. Los métodos antes mencionados se colocaron en orden descendente en cuanto al consumo de recursos computacionales y ascendente en cuanto al aumento en la eficiencia en su desempeño. A continuación se describen el método de Gauss-Seidel y el método de relajación sucesiva, los cuales se utilizaron para resolver el sistema de ecuaciones que se obtuvo después de aplicar el método de elemento finito al problema de convección.

3.6.2.2 Gauss-Seidel

Este método converge más rápido que el método de Jacobi, en algunos casos [Skiba, 2005]. Se asume que los elementos diagonales de la matriz $\underline{\underline{\mathbf{A}}}$ difieren de cero ($a_{ii} \neq 0\,,\, i = 1, \ldots, N$), y se escribe el sistema de ecuaciones lineales $\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}} = \underline{\mathbf{b}}$ en la forma (3.95):

$$\underline{\mathbf{u}} = \underline{\underline{\mathbf{B}}} \cdot \underline{\mathbf{u}} + \underline{\mathbf{d}} \tag{3.98}$$

donde $\underline{\underline{\mathbf{B}}}$ y $\underline{\mathbf{d}}$ se definen por:

$$d_i = \frac{b_i}{a_{ii}}\,, \qquad \underline{\underline{\mathbf{B}}} = \{b_{ij}\}\,, \qquad b_{ij} = \begin{cases} -a_{ij}/a_{ii}\,, & j \neq i \\ 0 & , j = i \end{cases} \tag{3.99}$$

Así las iteraciones se realizan por medio de la fórmula:

$$u_i^{(k)} = \sum_{j=1}^{i-1} b_{ij} u_j^{(k)} + \sum_{j=i+1}^{N} b_{ij} u_j^{(k-1)} + d_i \tag{3.100}$$

donde $u_i^{(0)}$ son arbitrarias ($i = 1, \ldots, N$; $k = 1, 2, \ldots$). La ecuación (3.100) es el método de Gauss-Seidel, en donde, para obtener el *i-ésimo* componente de la *k-ésima* aproximación se utilizan inmediatamente todos los componentes $x_j^{(k)}$ ya obtenidos (con $j < i$). Esto es muy conveniente para cálculos computacionales, ya que los valores nuevos pueden ser almacenados en los lugares ocupados por los valores viejos, lo que reduce los requerimientos de almacenaje. En su forma vectorial, el método de Gauss-Seidel se escribe de la siguiente manera:

$$\underline{\mathbf{u}}^{(k+1)} = \underline{\underline{\mathbf{E}}} \cdot \underline{\mathbf{u}}^{(k+1)} + \underline{\underline{\mathbf{F}}} \cdot \underline{\mathbf{u}}^{(k)} + \underline{\mathbf{d}} \tag{3.101}$$

donde $\underline{\underline{\mathbf{E}}}$ y $\underline{\underline{\mathbf{F}}}$ son las matrices triangular superior y triangular inferior, respectivamente.

3.6.2.3 Relajación sucesiva (SOR)

La idea básica de los métodos de relajación es multiplicar el término residual $\underline{\mathbf{b}} - \underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{u}}^{k-1}$ por un factor de peso ω llamado *coeficiente de relajación*. En ciertos casos, esta simple modificación puede

acelerar considerablemente la velocidad de convergencia del método [Skiba, 2005].

El método de Gauss-Seidel con relajación, también conocido como el *método de sobrerrelajaciones sucesivas*, o *método SOR (successive overrelaxations)*, se obtiene sobrerrelajando el método de Gauss-Seidel, de la siguiente manera:

$$u_i^{(k+1)} = (1-\omega)u_i^{(k)} + \omega \left[\sum_{j=1}^{i-1} b_{ij} u_j^{(k+1)} + \sum_{j=1+1}^{N} b_{ij} u_j^{(k)} + d_i \right] \tag{3.102}$$

Cuando la matriz $\underline{\underline{\mathbf{A}}}$ es simétrica con entradas positivas en la diagonal, este método converge sí y sólo sí $\underline{\underline{\mathbf{A}}}$ 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 encontrarlo dependen de las características propias del problema [Saad, 2000]. Sin embargo, comúnmente se usan valores menores que 1 cuando se quiere acelerar la convergencia de un problema que converge muy lentamente. 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.

Capítulo 4

Implementación Computacional

En este capítulo se describen las consideraciones generales que permiten la implementación del Método del Elemento Finito en dos dimensiones. Se presenta una implementación computacional del problema de convección, para lo cual se modificaron algunas clases del programa presentado por [Montealegre, 2010] y se crearon otras clases más. Como recursos computacionales se ocuparon el lenguaje de programación [Java-JDK15, 2020] en un entorno de desarrollo [NetBeans-12.0, 2020]; mientras que para las salidas gráficas se ocupó el entorno de desarrollo [Matlab, 2016].

4.1 Problema de Convección

Como ya se vio, el problema de convección está representado por dos ecuaciones diferenciales adimensionales, temperatura y stream, las cuales están acopladas por medio de la velocidad de filtración. Después del proceso de adimensionalización (ver sección 2.3), las ecuaciones adimensionales de calor y flujo (stream) son las ecuaciones (2.34) y (2.35), las cuales se reproducen a continuación:

$$\nabla \cdot \left(\underline{\underline{\Lambda}} \nabla T\right) = v \cdot \nabla T + \frac{\partial T}{\partial t} - A \tag{4.1}$$

$$\nabla \cdot \left(\underline{\underline{K}} \nabla \psi\right) = u \cdot \nabla \psi - S \tag{4.2}$$

Las velocidades adimensionales de filtración (o su equivalente, ver sección 2.3.5) del conjunto de ecuaciones (2.36), se sustituyen en las ecuaciones de calor y flujo adimensional, lo que da como resultado lo siguiente (ver nomenclatura en **Apéndice B**):

$$\nabla \cdot \left(\underline{\underline{\Lambda}} \nabla T\right) = \left[(\mathbf{D}\psi)^t - \nabla \left(\ln tr \Lambda^*\right) \cdot \underline{\underline{\Lambda}}\right] \cdot \nabla T + \frac{\partial T}{\partial t} - A \tag{4.3}$$

$$\nabla \cdot \left(\underline{\underline{K}} \nabla \psi\right) = \left[-\nabla \ln g \underline{\underline{K}}\right] \cdot \nabla \psi - S \tag{4.4}$$

Se observa, con mayor claridad, el acoplamiento entre las ecuaciones de calor (función T) y stream (función ψ), en el término que representa la velocidad de Darcy $(\mathbf{D}\psi)^t$.

Las condiciones de frontera generales para una función u, que puede representar la función de temperatura o la función de stream, son:

$$\begin{aligned} u\left(x,t\right) & \qquad en \quad \Gamma_D \times [t_0, t_f] \\ a_n \frac{\partial u}{\partial n} = h\left(\underline{\mathbf{x}},t\right) - e\left(\underline{\mathbf{x}},t\right) u & \qquad en \quad \Gamma_N \times [t_0, t_f] \end{aligned} \tag{4.5}$$

De acuerdo con lo presentado en la sección 2.3.9, estas condiciones de frontera se especifican con los valores apropiados de temperatura constante T (condición tipo Dirichlet), flujo de calor Φ (condición tipo Neumann) y velocidad de filtración v (condición tipo Dirichlet). Se recuerda que la velocidad de filtración es la derivada de la función de stream; es decir: $v_x = \frac{\partial \psi}{\partial z}$ y $v_z = \frac{\partial \psi}{\partial x}$ (condiciones de tipo Neumann). Un ejemplo particular de las condiciones de frontera, se observa en la **Figura 2**.

La condición inicial para una función u, que representa a la temperatura o a la función de corriente (stream), es:

$$u\left(\underline{\mathbf{x}}, t_0\right) = u_0(\underline{\mathbf{x}}) \tag{4.6}$$

Como puede notarse la forma de las ecuaciones adimensionales de calor y stream es fácilmente identificable con la del operador diferencial (3.54). Entonces, esta forma del operador diferencial puede expresarse de tal manera que se pueda integrar en un dominio Ω, según se presentó en el Capítulo 3. Por lo tanto, el problema de convección se puede resolver con el método del elemento finito.

4.2 Consideraciones de Programación

La solución de elementos finitos de un problema sigue una metodología estándar. Los aspectos esenciales que debe cubrir un programa computacional que implemente el método del elemento finito, independientemente de la plataforma o lenguaje de programación elegido, son los siguientes [Chen, 2005] [Montealegre, 2010] [Zienkiewics et al., 2005]:

1. Definir el problema a resolver en términos de ecuaciones diferenciales. Es decir, entrada de los datos referentes al dominio Ω, el lado derecho de la función f, las condiciones de frontera y los coeficientes que definen al operador diferencial lineal $\mathcal{L}$ (ecuación (3.54)), los cuales dependen del problema en cuestión.

2. Construir la forma integral para el problema como formulación débil o formulación variacional.

3. Seleccionar el tipo y orden de elementos finitos que serán usados en el análisis. Construcción de la triangulación K_h (mallado del problema).

4. Computo y ensamble de la matriz de rigidez $\underline{\underline{\mathbf{A}}}$, así como del vector $\underline{\mathbf{f}}$ correspondiente al lado derecho de la ecuación. La forma débil o variacional que se haya utilizado, proporciona las bases para calcular las relaciones específicas de cada elemento (ecuación (3.72)).

5. Solución del sistema lineal de la ecuación $\underline{\underline{\mathbf{A}}} \cdot \underline{\mathbf{v}} = \underline{\mathbf{r}}$ (ecuación (3.73)).

6. 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. Para los cinco primeros puntos se mo-

dificaron algunas clases del programa en (Java-NetBeans) presentado por [Montealegre, 2010] y se crearon algunas clases más, específicas del problema de convección para simular casos geológicos reales. La matriz de valores obtenidos en las corridas para cada clase que define a un problema específico de convección, se exportó a Matlab para graficarse y así se obtuvieron los campos de temperatura y de flujo para distintos casos con condiciones específicas. Esto se desarrollada con mayor detalle en el Capítulo 5 de Ensayos Numéricos; por ahora, se describirán los puntos restantes.

4.2.1 Triangulación del Dominio

En esta sección se tratará únicamente la creación de un mallado conformado por triángulos, lo cual se conoce como *triangulación* del dominio (en [Chen, 2005] y [Chen et al., 2006] se pueden revisar otras técnicas de mallado), puesto que se trata de un mallado fácil de implementar y lo suficientemente flexible para adaptarse a distintas formas de dominios.

La triangulación K_h puede construirse al 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, como se muestra en la **Figura 6**. 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 et al., 2006] para más referencia.

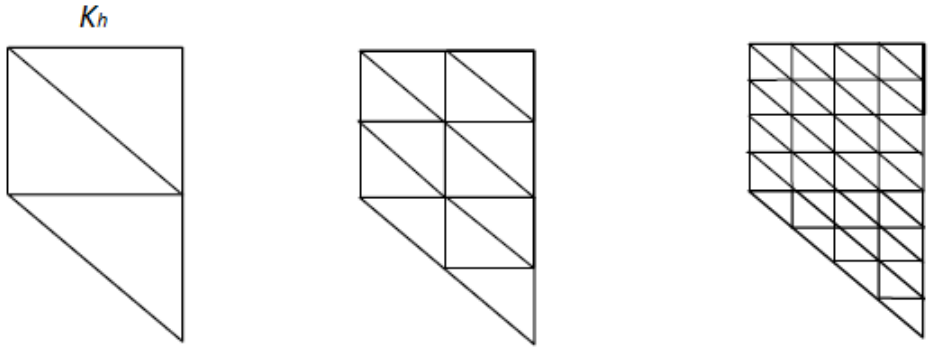


Figura 6: *Proceso de refinamiento uniforme de una malla*

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* y se muestra en la **Figura 7**. 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 7** la línea punteada en el tercer dibujo, representa un detalle que debe 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 et al., 2006].

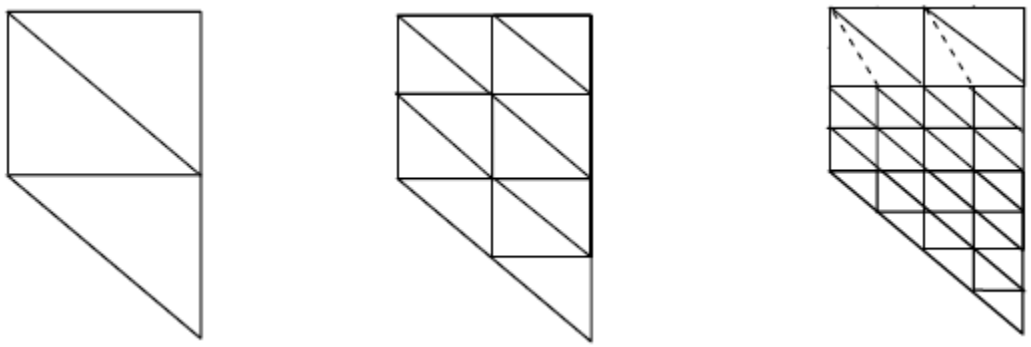


Figura 7: *Proceso de refinamiento no-uniforme de una malla, donde una estrategia de refinamiento local se lleva a cabo en el tercer ejemplo*

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; es decir, al unir con una línea recta dos de los vértices opuestos del mismo, como se observa en la **Figura 8**.

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

En la **Figura 9** se muestra un ejemplo de numeración, donde la numeración de los triángulos es

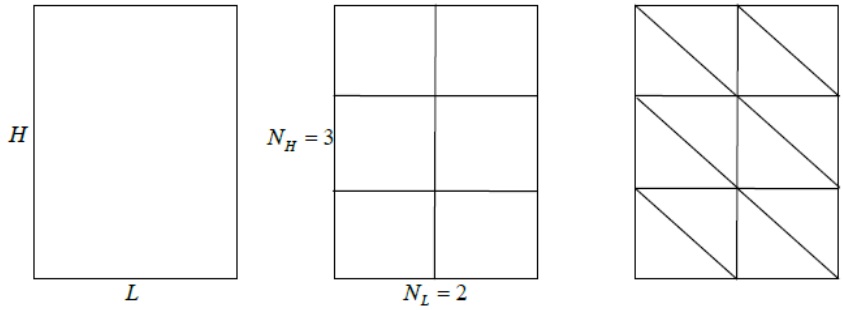


Figura 8: *Proceso de creación de un mallado triangular sencillo a partir de un dominio rectangular*

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 $\mathcal{Z}(3, \mathcal{M})$, tiene la siguiente forma (donde $M = \mathcal{M} = 12$):

$$\mathcal{Z} = \begin{pmatrix} 1 & 2 & 2 & 3 & 4 & 5 & 5 & 6 & 7 & 8 & 8 & 9 \\ 2 & 5 & 3 & 6 & 5 & 8 & 6 & 9 & 8 & 1 & 19 & 1 \\ 4 & 4 & 5 & 5 & 7 & 7 & 8 & 8 & 10 & 10 & 11 & 11 \end{pmatrix} \tag{4.7}$$

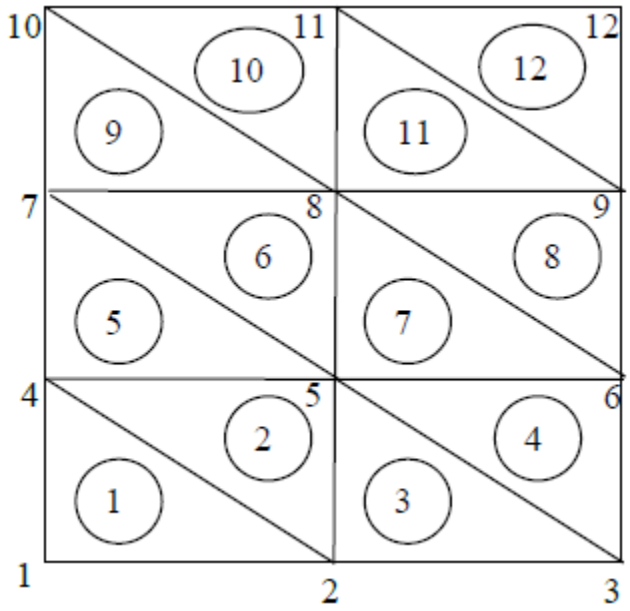


Figura 9: *Numeración de nodos y triángulos en una triangulación*

4.2.2 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 $\underline{\underline{A}}^{(k)} = \left(a_{ij}^k\right)$ dadas por la ecuación (3.72) (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, $\mathcal{Z}(m,k)$ $(m = 1,2,3)$ son los números de vértices de K_k y el elemento de la matriz de rigídez $\underline{\underline{A}}^{(k)} = \left(a_{mn}^k\right)_{m,n=1}^3$ y se calcula (basándose en el caso más general de la ecuación (3.72)):

$$a_{mn}^k = \int_{K_k} \left(\nabla w_m \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla w_n - \nabla w_m \cdot (\underline{\mathbf{b}} w_n) + c w_m w_n\right) d\underline{\mathbf{x}} + \int_{\Gamma_N} w_m w_n \left(e - \mathbf{b}_{normal}\right) ds$$
$$m,n = 1,2,3 \tag{4.8}$$

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

$$w_m\left(x_{\mathcal{Z}(n,k)}\right) = \begin{cases} 1 & , si \;\; m = n \\ 0 & , si \;\; m \neq n \end{cases} \tag{4.9}$$

El lado derecho de la ecuación (3.72), $\underline{\mathbf{r}}^{(k)} = \left(r_m^k\right)_{m=1}^3$ sobre K_k se calcula:

$$r_{mn}^k = \int_{K_k} f w_m \, d\underline{\mathbf{x}} + \int_{\Gamma_N} h w_m \, ds - \int_{K_k} \left(\nabla w_m \cdot \underline{\underline{\mathbf{a}}} \cdot \nabla u_0 - \nabla w_m \cdot (\underline{\mathbf{b}} u_0) + c w_m u_0\right) d\underline{\mathbf{x}}$$
$$m = 1,2,3 \tag{4.10}$$

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 (3.72) son los números globales de los vértices en K_h.

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

$$
\begin{aligned}
\text{For} \quad & k = 1, 2, \ldots, \mathcal{M}, \quad \text{calcula} \\
a_{\mathcal{Z}(m,k),\mathcal{Z}(n,k)} &= a_{\mathcal{Z}(m,k),\mathcal{Z}(n,k)} + a^k_{mn} \\
r_{\mathcal{Z}(m,k)} &= r_{\mathcal{Z}(m,k)} + r^k_m \\
m, n &= 1, 2, 3
\end{aligned} \tag{4.11}
$$

En este caso se usa la aproximación orientada a elementos, es decir, el bucle se realiza sobre los elementos (en este caso, los 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 $\underline{\mathbf{r}}$ [Chen, 2005].

4.3 Programa en Java-NetBeans

4.3.1 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. Ésta es una forma de programar que trata de encontrar una solución a los problemas que se presentan con otros paradigmas, como la programación estructurada tradicional (para mayor detalle véase [Montealegre, 2010]). La programación orientada a objetos introduce conceptos, tales como:

- *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.

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 [Montealegre, 2010].

4.3.2 Estructura y Descripción del Programa

Para realizar la implementación computacional se modificaron algunas de las clases del programa desarrollado por [Montealegre, 2010] y se crearon otras clases más. El lenguaje de programación utilizado fue Java SE Development Kit 15.0.2 [Java-JDK15, 2020], el cual es un lenguaje multiplataforma, es decir, el programa puede ejecutarse, sin necesidad de compilarse nuevamente, en diversos sistemas operativos, como podrían ser Windows, UNIX, LINUX, etc. La programación se realizó usando el entorno de desarrollo integrado (IDE, Integrated Development Environment) Apache NetBeans IDE 12.0 [NetBeans-12.0, 2020], 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.

La 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; por otro lado, NetBeans tiene la ventaja de ser un producto libre, gratuito y sin restricciones de uso. Después de ejecutar el programa en Java-NetBeans, la matriz de resultados se graficó en [Matlab, 2016] para una mejor visualización de los resultados: campo de temperaturas y de flujo para coordenadas específicas del dominio rectangular donde se resolvió el problema de convección.

El programa en Java-NetBeans está conformado por cuatro paquetes, como se muestra en la **Figura 10**:

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 método del Elemento Finito 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.

Figura 10: *Paquetes que conforman el programa en Java*

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.
- 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.

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 flujo (stream) de fluido y temperatura, respectivamente, a los datos que usan para calcular sus soluciones.

En la **Figura 11** se observa 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.

En el **Apéndice C** se muestra el código programable de las clases más importantes usadas en Java-NetBeans; así también se presenta el código del programa en MatLab para la salida gráfica de los resultados computacionales.

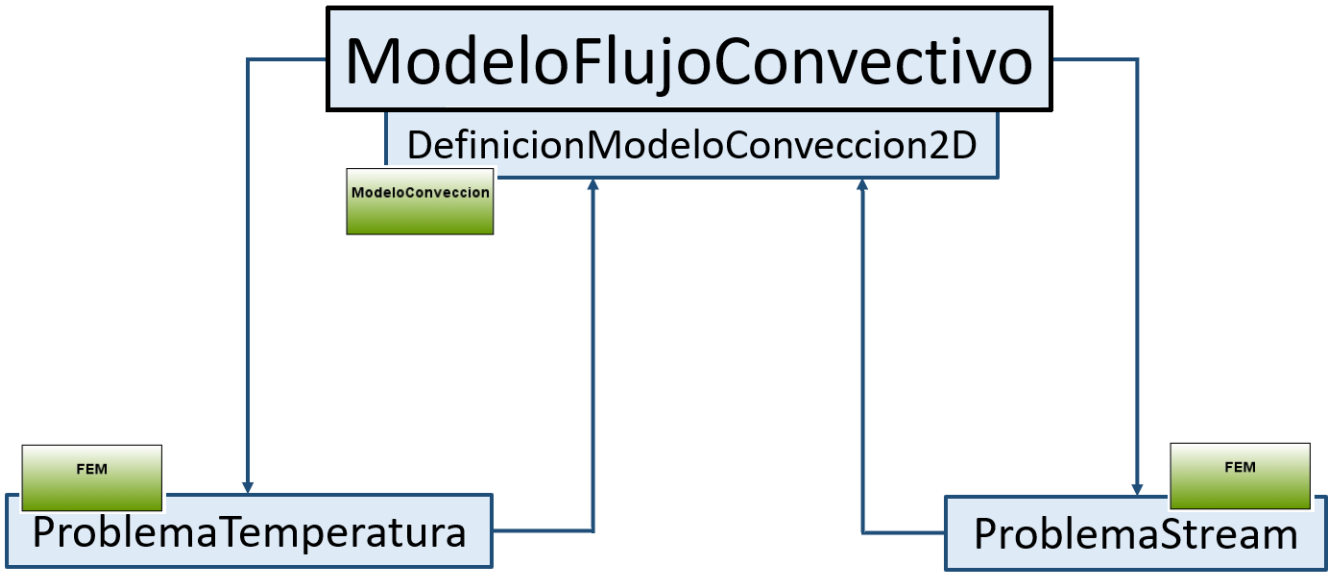


Figura 11: *Clases que conforman el acoplamiento del modelo completo con los problemas de temperatura y stream.*

Capítulo 5

Ensayos Numéricos

En este capítulo se presentan algunos de los ensayos numéricos que se realizaron para verificar el comportamiento del programa computacional, en geometrías rectangulares que contienen varias capas horizontales y una capa inclinada que atraviesa al medio poroso.

En esencia, el proceso de transferencia de calor por convección es una combinación del transporte de masa y calor, la visualización numérica de estos dos procesos de transporte, es la aproximación más directa para investigar la estructura o características del calor y flujo de fluido con conocimiento de la teoría de la transferencia de calor por convección [Deng y Tang, 2002].

Para que el modelo computacional represente condiciones reales de un medio geológico, debemos considerar valores reales de conductividades térmicas y permeabilidad, entre otros parámetros físicos de importancia como las propiedades del agua.

La configuración más estudiada en convección natural, como ya se ha visto previamente, es aquella en la que el medio poroso está saturado por un fluido y además está delimitado por dos fronteras horizontales impermeables, con la frontera inferior más caliente que la frontera superior. Al aplicar las condiciones de frontera mixtas al dominio rectangular se tiene temperatura fija en los bordes superior e inferior (condiciones Dirichlet) y flujo fijo con valor cero en las paredes verticales (condiciones Neumann), lo cual equivale a que las paredes verticales son adiabáticas.

Para las prueba que se analizaron, además de las propiedades más importantes que son la con-

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

Tabla 5.1: Propiedades del agua usada en los ensayos numéricos

ductividad térmica y la permeabilidad, se consideró la variación de los siguientes parámetros: razón geométrica, inclinación de la capa que atraviesa al medio poroso, número de capas horizontales. Esto dio lugar a las configuraciones que a continuación se presentan.

Además se considera que el fluido es agua, con las propiedades mostradas en la Tabla 5.1.

5.1 Medio Poroso Homogéneo Isotrópico

Para el primer ensayo numérico se consideró un dominio de 100×100, que representa un medio poroso compuesto por una sola capa con permeabilidad y conductividad térmica homogénea e isotrópica. Las fronteras superior e inferior están isotérmicamente fijadas, las fronteras verticales son adiabáticas y no se presenta intercambio de fluidos en las fronteras.

Los resultados del primer ensayo se presentan en la **Figura 12** que representa un medio homogéneo e isotrópico. Como se observa, en las paredes adiabáticas las isotermas llegan perpendiculares, lo cual es consistente con la propiedad geométrica del gradiente de temperatura, de ser perpendicular a la normal de la pared adiabática [Báez et al., 2004]. También se observa que las celdas convectivas son simétricas.

El flujo de calor dominado por conducción puede observarse por la apariencia de isotermas suaves y monótonas. En cambio, cuando el calor transferido es dominado por convección, las lineas de calor se distorsionan. Debido a la intensa convección, las isotermas están acopladas con el flujo de fluido [Kumar et al., 2015]. Por otro lado, cuando mayor sea el flujo de calor más nítido será el cambio en las isotermas [Deng y Tang, 2002].

La función de stream y las líneas de stream son muy eficientes y han sido ampliamente usadas como herramientas para visualizar el transporte de momento del flujo de fluido. Puesto que no hay intercambio de fluido o masa entre el sistema y el exterior, las lineas de stream circulan como vórtices.

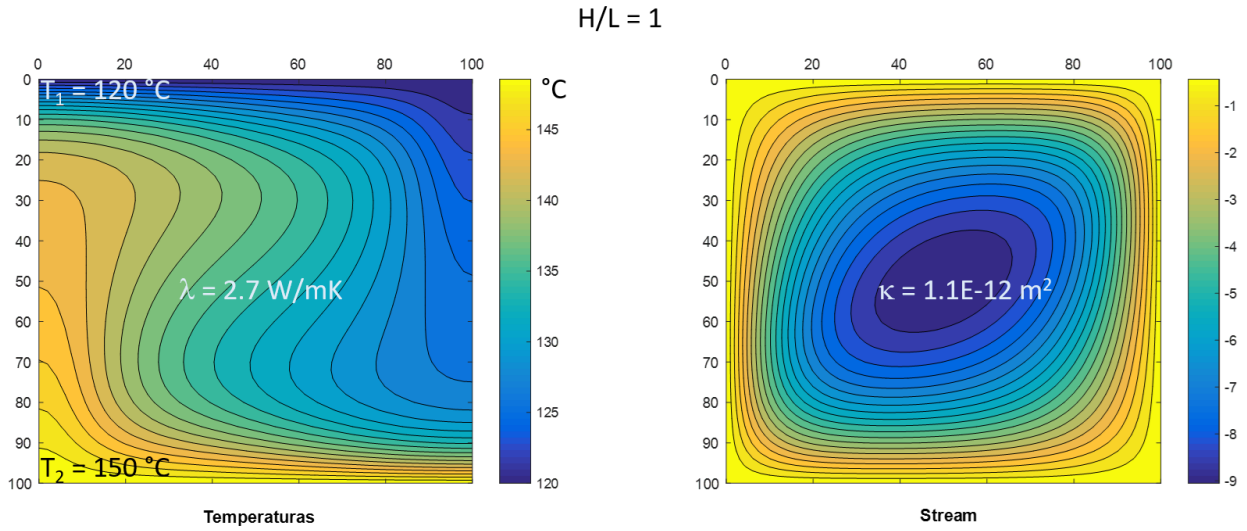


Figura 12: *Campo de temperaturas y de flujo (stream) de fluido en un estado convectivo estable. El ensayo numérico se realizó en un dominio cuadrado (H/L=1) que representa a un medio poroso homogéneo e isotrópico. Los valores de conductividad térmica λ y permeabilidad κ se observan en la imagen.*

Los resultados obtenidos de campo de temperaturas y función de corriente (**Figura 12**) son consistentes con los que ya se han presentado en trabajos anteriores de convección en medios porosos, en donde se muestran los resultados del campo de temperaturas y lineas de stream para un medio poroso homogéneo e isotrópico: [Al-Amiri, 2002], [Báez et al., 2004], [Báez y Nicolás, 2006], [Báez, 2008], [Basak y Roy, 2008], [Deng y Tang, 2002], [Flores-Márquez, 1992], [Ibrahim e Hirpho, 2021], [Jiménez-Islas, 1999], [Jiménez-Islas et al., 2009], [Kumar et al., 2015], [Monteale-gre, 2010], [Pandit y Chattopadhyay, 2014], [Riley y Winters, 1990], [Royer y Flores-Márquez, 1994], [Saeid y Pop, 2004] [Zhang et al., 2016] y [Zhao et al., 2019], entre otros.

Las **Figuras 13**, **14** y **15** muestran algunos resultados del campo de temperaturas y función de stream, en trabajos publicados en la literatura. Cabe señalar que los métodos numéricos de solución empleados en cada caso, abordan de diferente manera el problema de convección.

En [Flores-Márquez, 1992], el problema se resuelve por diferencias finitas, con sobrerrelajación para el sistema matricial de ecuaciones. En [Deng y Tang, 2002] se utiliza el algoritmo $SIMPLE$ junto con el esquema de corrección $QUICK$, para mejorar la precisión numérica del término de convección. En [Zhang et al., 2016] se muestran 3 diferentes formulaciones de elementos finitos, para obtener el campo de temperaturas, presiones y velocidades, respectivamente. Para más información, revisar estos artículos.

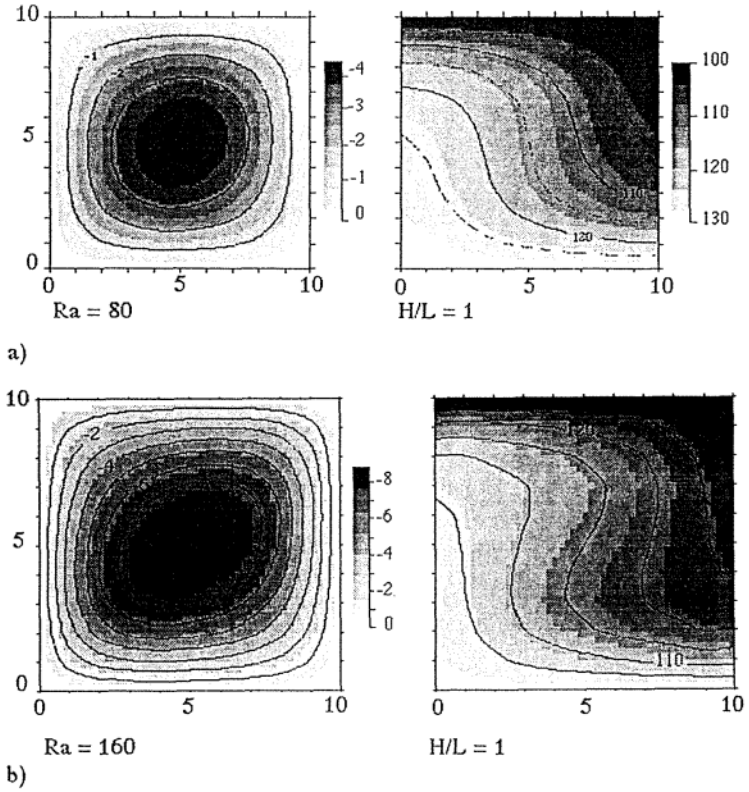


Figure 6.5 Essais numériques réalisés sur des couches homogènes et isotropes. a) Domaine carré (H/L =1) et Ra = 80, b) même domaine et Ra = 160, c) Domaine rectangulaire (H/L = 1.8) et Ra = 200 et d) domaine rectangulaire (H/L = 0.33) et Ra = 80.

(Pour chaque incise la première figure représente la fonction de courant et la deuxième le champ de températures).

Figura 13: *Campos de solución del problema de convección presentado por [Flores-Márquez, 1992].*

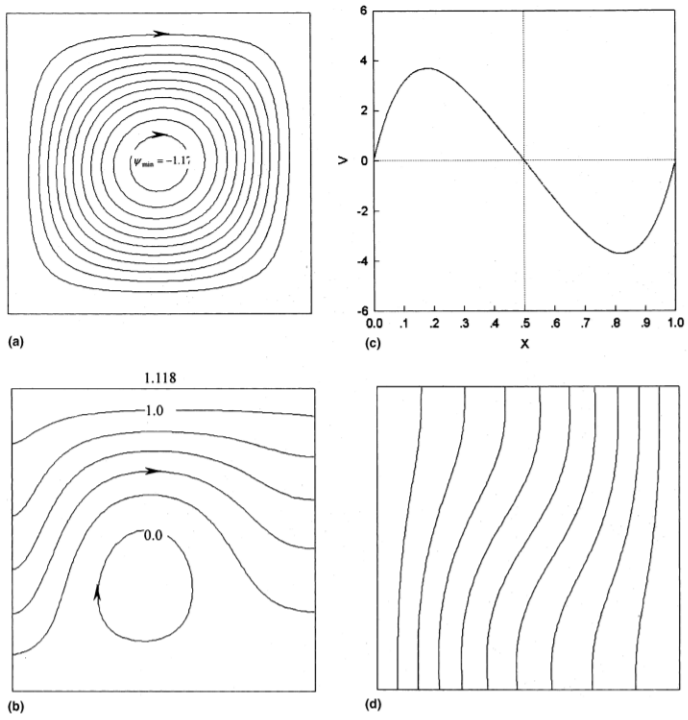


Fig. 3. Results of natural convection at $Ra = 10^3$ in an air-filled square cavity: (a) Streamlines whose incremental step is 0.1; (b) heatlines whose incremental step is 0.2; (c) velocity – V profiles along the horizontal centerline X; (d) isothermal whose incremental step is 0.1.

Figura 14: *Campo de solución del problema de convección presentado por [Deng y Tang, 2002].*

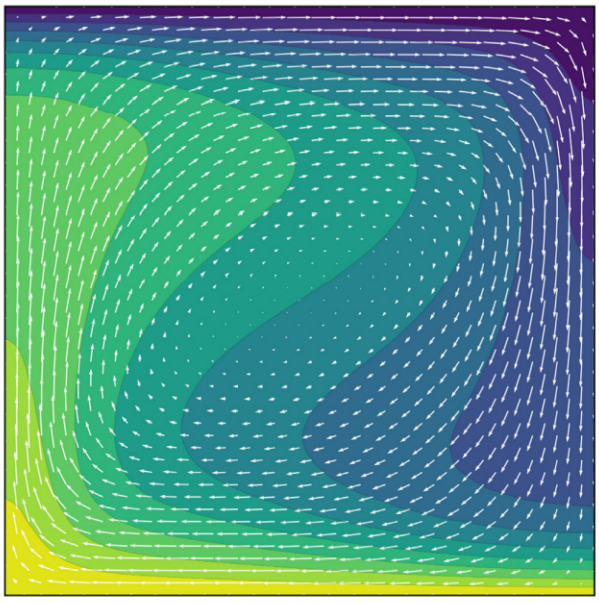


Fig. 6. Solution fields in the HRL problem with Rayleigh number Ra = 300. Colors indicate the temperature T from 0 (blue) to 1 (yellow). The same colormap is used in all contour plots in this paper. Arrows represent the velocity field. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

Figura 15: *Campo de solución del problema de convección presentado por [Zhang et al., 2016].*

5.2 Presencia de una Falla en el Medio Poroso

5.2.1 Fallas y Fracturas

Las fracturas y fallas son producto de la deformación frágil en cualquier tipo de roca, se forman por esfuerzos cortantes y en zonas de compresión o de tensión. Cada una de las zonas o ámbitos que resultan de una superficie de ruptura se denominan bloque; si la superficie de ruptura es horizontal o inclinada, al volumen que queda arriba de la superficie se denomina bloque de techo y al volumen inferior bloque de piso. El vector de desplazamiento que conecta a puntos originalmente contiguos entre el bloque del techo y el bloque de piso se conoce como desplazamiento neto o salto [Arellano-Gil et al., 2002].

Bajo el campo de la deformación frágil las rocas se rompen conforme a superficies más o menos planas. Las superficies de ruptura se denominan fracturas cuando no se aprecia desplazamiento entre los dos ámbitos definidos por la superficie de discontinuidad, en sentido paralelo a la propia superficie. Las fracturas son discontinuidades aproximadamente planas que separan bloques de roca con desplazamiento perpendicular al plano de ruptura.

Cuando la roca ha tenido un movimiento relativo a lo largo del plano de fractura, es llamada falla. Es decir, las fallas son superficies de discontinuidad que separan bloques de roca donde ha ocurrido desplazamiento de bloques con movimiento paralelo al plano de discontinuidad. Las fallas pueden ser normales, inversas, transcurrentes, rotacionales y de crecimiento. Generalmente se describen y clasifican por el echado de la falla, la dirección y el sentido del movimiento.

La inclinación del plano de falla es llamado echado. El desplazamiento a lo largo del plano de falla entre dos bloques puede tomar cualquier dirección en el plano; de manera ideal, puede ser en la dirección del rumbo o en la dirección del echado.

En la **Figura 16**, se muestra un esquema conceptual de una falla y sus componentes. La **Figura 17** muestra un ejemplo de una falla observada en un ambiente geológico.

$\overline{AC} =$	Desplazamiento neto	$\delta =$	Echado de falla
$AB =$	Dirección del desplazamiento neto	$\phi =$	Cabeceo (pitch)
$\overline{AD} =$	Desplazamiento a rumbo	$\beta =$	Buzamiento del desplazamiento neto
$\overline{DC} =$	Desplazamiento a echado		
$AF =$	Salto vertical		
$EF =$	Salto horizontal	$\alpha =$	Hade

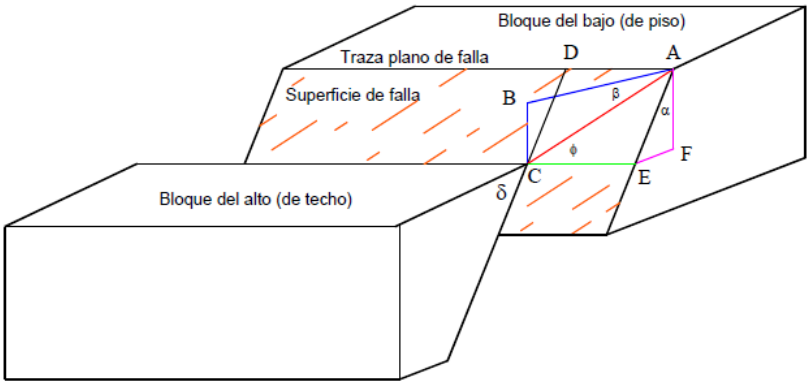


Figura 16: *Componentes principales de una falla. (Modificada de [Arellano-Gil et al., 2002]).*

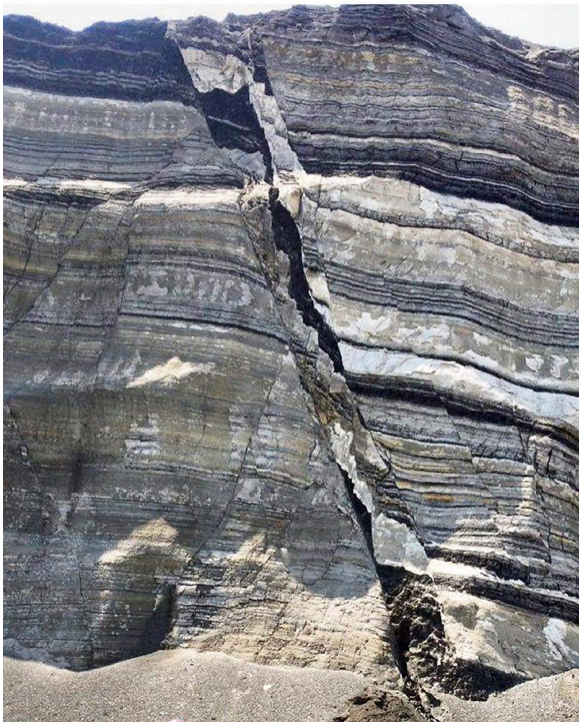


Figura 17: *Ejemplo de falla casi vertical. Imagen obtenida de* `https://geotectica.com/wp-content/uploads/2019/07/12-Falla-Geol%C3%B3gica.jpg`

5.2.2 Medio Poroso Homogéneo e Isotrópico con Falla Homogénea Anisotrópica

En el ensayo numérico de la **Figura** 18 se considera un dominio de 100 × 100, que representa un medio poroso compuesto por un estrato con permeabilidad y conductividad térmica homogénea e isotrópica, que es atravesado por una falla inclinada homogénea y anisotrópica. Las fronteras superior e inferior están isotérmicamente fijadas, las fronteras verticales son adiabáticas y no presentan intercambio de fluidos.

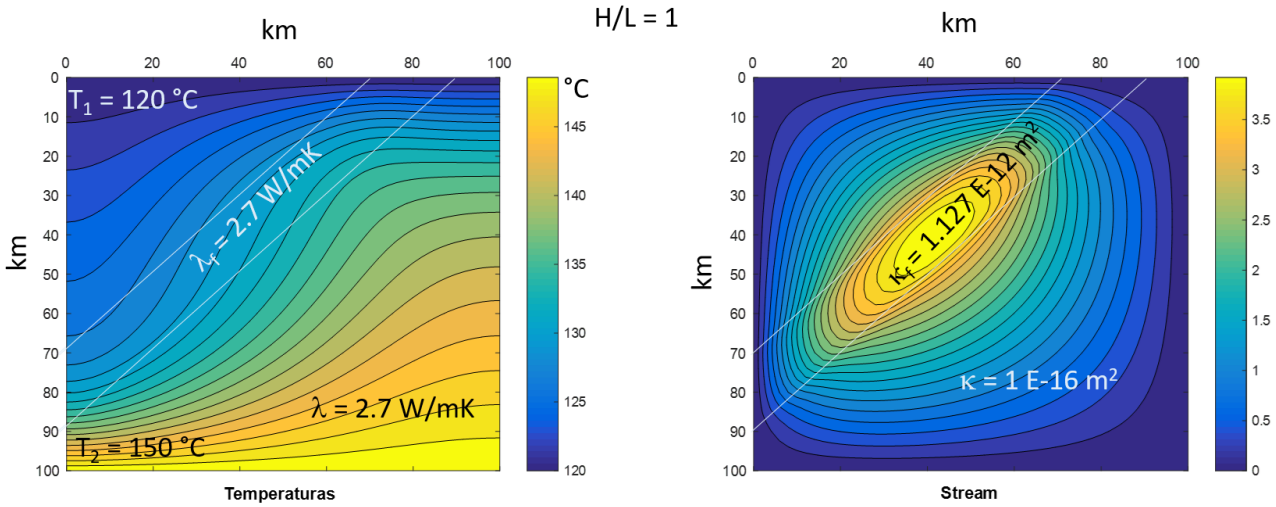


Figura 18: *Campo de temperaturas y de flujo (stream) de fluido en un estado convectivo estable. El ensayo numérico se realizó en un dominio cuadrado (H/L=1) que representa a un medio poroso homogéneo e isotrópico, con presencia de una falla inclinada. La conductividad térmica λ es la misma en todo el medio; mientras que la permeabilidad κ es mayor en la capa inclinada, que en el resto del medio poroso.*

En la **Figura** 18 se observa una gran celda convectiva en la zona de la falla inclinada, debido a que ésta presenta una mayor permeabilidad en comparación con el resto del medio poroso. Así mismo, se nota la elongación de la celda, en comparación con el caso homogéneo e isotrópico de la **Figura** 12, el cual no tiene una falla.

Por otro lado, el ligero estrechamiento de las líneas de corriente en la zona superior de la falla, 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). El caso contrario se nota en la zona inferior de la falla, en donde las líneas de corriente están más separadas, lo que corresponde a una disminución en la diferencia de temperaturas.

5.2.3 Medio Poroso Homogéneo e Isotrópico con Falla Homogénea Anisotrópica. Existencia de Flujo de Calor Desde la Base

En este ensayo numérico (**Figura 19**) se consideró un dominio de 100×100, que representa un medio poroso compuesto por un estrato con permeabilidad y conductividad térmica homogénea e isotrópica, que es atravesado por una falla inclinada homogénea y anisotrópica. La frontera superior está isotérmicamente fijada, la frontera inferior presenta entrada de calor, las fronteras verticales son adiabáticas y no presentan intercambio de fluidos.

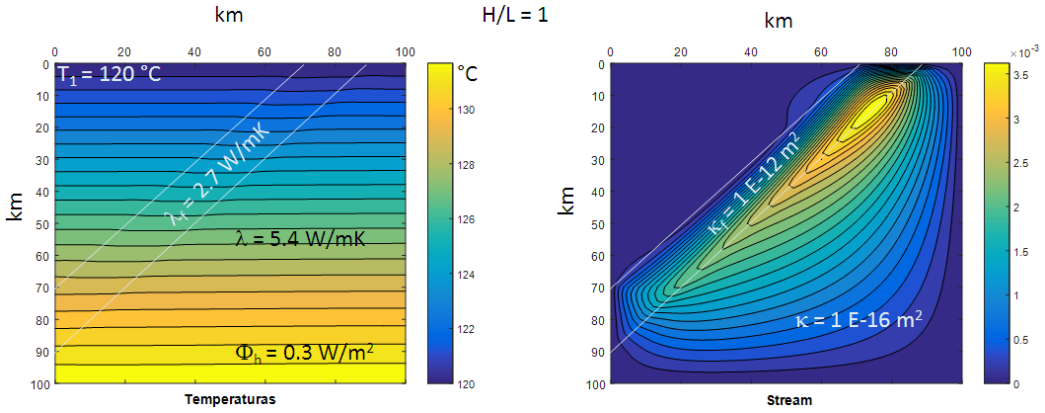


Figura 19: *Campo de temperaturas y de flujo (stream) de fluido en un estado convectivo estable. El ensayo numérico se realizó en un dominio cuadrado (H/L=1) que representa a un medio poroso homogéneo e isotrópico, con presencia de una falla inclinada y con flujo de calor Φ_h desde la base. La conductividad térmica λ es menor en la falla que en el resto del medio poroso; mientras que la permeabilidad κ es mayor en la capa inclinada, que en el resto del medio poroso.*

Igualmente que el caso anterior, la **Figura 19** muestra una gran celda convectiva que se elonga asimétricamente a lo largo de la falla. Esta asimetría de la celda es causada por los efectos gravitatorios presentes en el medio poroso. El flujo de calor en la base provoca que el campo de temperaturas tenga una distribución más homogénea, con lo cual la zona inferior se mantiene caliente a lo largo de toda su extensión, y existiendo un ligero aumento en la región de la falla.

En un modelo homogéneo anisotrópico la convección ocurre en todo el dominio; es decir la convección tiene lugar a "gran escala". Sin embargo, para grandes sistemas con múltiples estratos heterogéneos en permeabilidad o conductividad, es posible que la convección comience solamente en algunas de las capas, aquellas con mayor permeabilidad o menor conductividad. Tal movimiento se

denomina “convección local” y está prácticamente confinado a estas capas activas, debido a que allí se presenta el mayor flujo del fluido. Sin embargo, debido a la continuidad, también existe cierto movimiento en las capas menos activas; es decir, las que tienen permeabilidad menor o conductividad mayor [McKibbin y Tyvand, 1982]. Esto se ejemplifica en los siguientes ensayos numéricos.

5.2.4 Medio Poroso Heterogéneo Isotrópico con una Falla Homogénea Anisotrópica. Dominio Horizontal

El ensayo numérico de la **Figura 20** considera un dominio de 100 × 300, que representa un medio poroso compuesto por dos estratos con permeabilidad y conductividad térmica heterogénea e isotrópica, que es atravesado por una falla inclinada homogénea y anisotrópica. Las fronteras superior e inferior están isotérmicamente fijadas, las fronteras verticales son adiabáticas y no se presenta intercambio de fluidos en las fronteras.

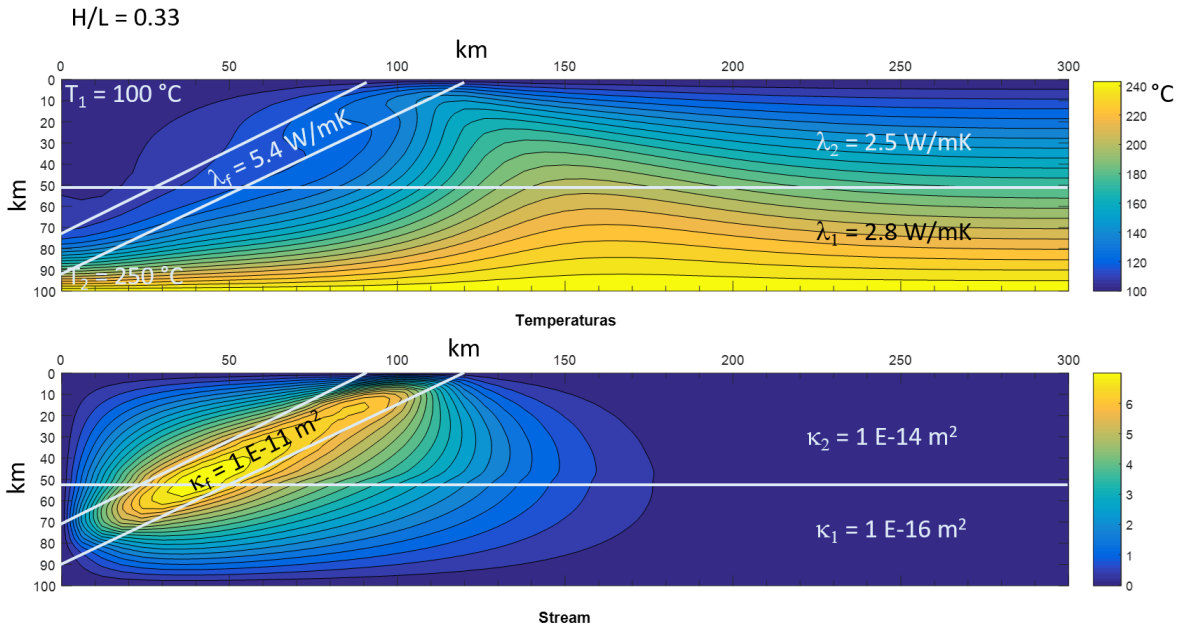


Figura 20: *Campo de temperaturas y de flujo (stream) de fluido en un estado convectivo estable. El ensayo numérico se realizó en un dominio rectangular horizontal (H/L=0.33) que representa a un medio poroso heterogéneo (2 estratos), con presencia de una falla inclinada que atraviesa a todo el medio. La conductividad térmica λ es mayor en el estrato inferior, y ambas son menores que la conductividad de la falla. La permeabilidad κ es mayor en la capa inclinada, que en el resto del medio poroso.*

En los estudios que se han hecho sobre el inicio de la convección en un medio poroso compuesto por dos capas, se encontró que la convección tiende a localizarse en las capas de mayor permeabilidad. Esta "convección local" también está asociada con el movimiento de un fluido cuya viscosidad disminuye con el aumento en la temperatura; tasas de flujo relativamente grandes se presentan cerca de la frontera inferior caliente en el sistema. Sin embargo, en una capa homogénea saturada con un fluido de velocidad constante la convección local no ocurre, así que el flujo se considera de "gran escala" [McKibbin y Tyvand, 1982].

Lo anterior también se muestra en [Richard y Gounot, 1981], en donde se presenta el caso de un sistema con dos estratos horizontales del mismo espesor; pero características físicas distintas (permeabilidad y conductividad térmica).

En la **Figura 20** puede observarse esta "convección local" cercana a la zona de la capa inclinada, la cual tiene mayor permeabilidad que el resto del medio poroso heterogéneo; mientras que la zona más alejada de la capa inclinada, permanece sin el "efecto" de la celda convectiva. Esto último también se debe a la geometría alargada horizontalmente del dominio.

La zona a la izquierda de la falla, en el estrato superior presenta una mayor diferencia de temperaturas, en comparación con la misma zona del estrato inferior, en donde hay una menor variación de la temperatura; esto también se nota en que las líneas de corriente están más estrechas en la capa inferior, justo en la zona de la falla.

A la derecha de la falla se nota que las líneas de corriente empiezan muy estrechas y se separan cada vez más hasta que se pierde el efecto de la convección. Esto corresponde a un aumento de la temperatura en la zona final de la convección y, posteriormente, una distribución homogénea de temperatura que se observa en el extremo derecho del dominio poroso, donde ya no hay efecto de convección.

5.2.5 Medio Poroso Heterogéneo Isotrópico con una Falla Homogénea Anisotrópica. Dominio Vertical

En este ensayo numérico se consideró un dominio de 2000×500, que representa un medio poroso compuesto por dos estratos con permeabilidad y conductividad térmica heterogénea e isotrópica, que es atravesado por una falla inclinada homogénea y anisotrópica. Las fronteras superior e inferior están isotérmicamente fijadas, las fronteras verticales son adiabáticas y no se presenta intercambio de fluidos en las fronteras.

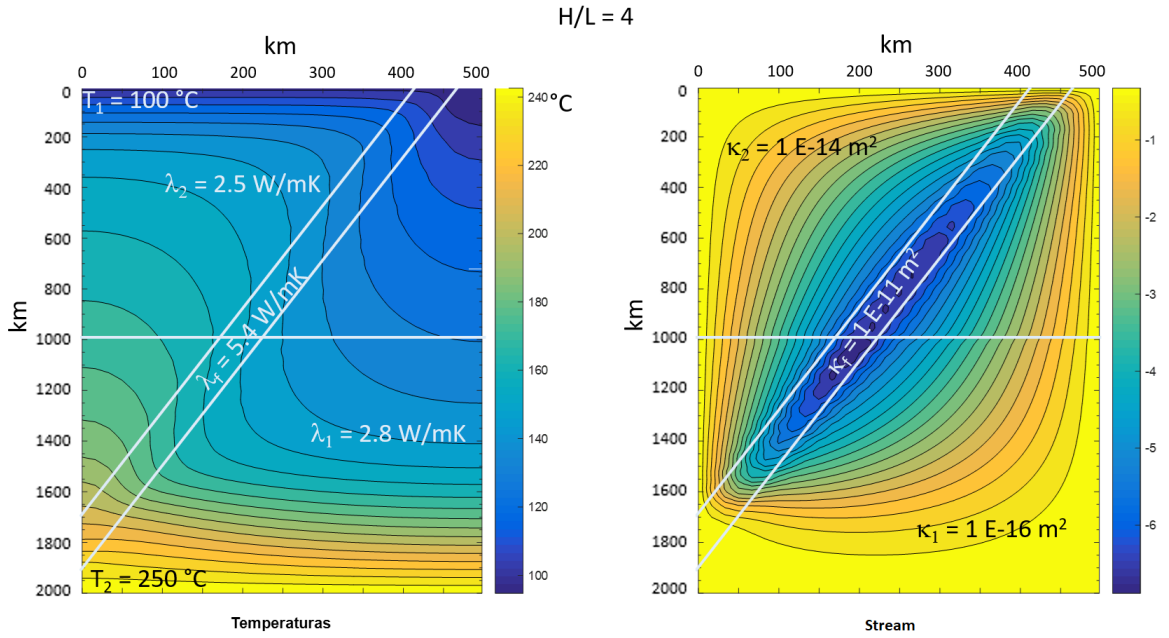


Figura 21: *Campo de temperaturas y de flujo (stream) de fluido en un estado convectivo estable. El ensayo numérico se realizó en un dominio rectangular vertical (H/L=4) que representa a un medio poroso heterogéneo (2 estratos), con presencia de una falla inclinada que atraviesa a todo el medio. La conductividad térmica λ es mayor en el estrato inferior, y ambas son menores que la conductividad de la falla. La permeabilidad κ es mayor en la capa inclinada, que en el resto del medio poroso.*

Como se observa en las **Figuras** 20 y 21 que tienen capa inclinada, la presencia de las misma resulta en celdas convectivas asimétricas, en la zona cercana a la falla. A mayores angulos aparece una única celda de convección y en este caso hay un mayor flujo de calor [Kumar et al., 2015]. Además, como el dominio rectangular es vertical, se observa una gran celda con intensa convección debida a las fuerzas de flotación.

5.2.6 Efecto Debido a la Inclinación de la Falla

Ahora se verán algunos ensayos en donde se varió la inclinación de la falla en un intervalo de 0 a 90°. Los dos primeros (**Figura 22**, incisos a y b) se hicieron para para una falla con inclinación de 1.7° y 11.3° en un dominio horizontal de $300\,km \times 100\,km$, compuesto por un estrato con permeabilidad y conductividad térmica homogénea e isotrópica, que es atravesado por una falla inclinada homogénea y anisotrópica. La frontera superior está isotérmicamente fijada, la frontera inferior presenta flujo de calor que entra desde la base, y las fronteras verticales son adiabáticas. No se presenta intercambio de fluidos en las fronteras.

Aquí puede observarse el efecto de las fallas casi horizontales, que ocasionan sólo perturbaciones locales en la temperatura, alrededor de la falla. En la base se imponen condiciones de flujo térmico constante y a pesar de que existe una leve convección de fluidos al interior de la falla, la distribución de temperaturas es casi la misma en los dos casos.

Los ensayos c y d de la **Figura 22** se realizaron para 81° y 87° de inclinación de la falla en un dominio vertical de $500\,km \times 2000\,km$, compuesto por un estrato con permeabilidad y conductividad térmica homogénea e isotrópica, que es atravesado por una falla inclinada homogénea y anisotrópica. La frontera superior está isotérmicamente fijada, la frontera inferior presenta flujo de calor que entra desde la base, y las fronteras verticales son adiabáticas. No se presenta intercambio de fluidos en las fronteras.

En estos ensayos se muestran el caso de la falla inclinada prácticamente vertical. Como se observa, el espesor de la falla es delgado, lo que provoca que en el inciso c la circulación del fluido tenga preferencia en la parte inferior del dominio poroso, y no se logre una convección estable en la zona superior. Por otro lado, en el inciso d, la falla está prácticamente vertical, por lo que la celda convectiva se forma solamente en la parte inferior del dominio poroso.

Debido a que el efecto de la convección no alcanza la zona superior, en los incisos c y d, las isotermas no presentan variación significativa en esa zona. En la zona inferior tampoco se presenta mucha variación debido a la presencia constante del flujo de calor. De hecho, la variación más notable en la diferencia de temperaturas es en la entrada inferior de la falla, sobre todo cuando

ésta es muy vertical. Así que, en conclusión los incisos c y d, prácticamente presentan la misma distribución de temperaturas.

En conclusión, el efecto de la inclinación de una falla intercalada en el medio poroso se observa notablemente en las líneas de corriente y muy poco en las isotermas. Cuando la falla es más bien horizontal y de espesor considerable, la convección toma lugar formando una gran celda dentro de la falla. En cambio, cuando la falla es delgada y está orientada verticalmente, la convección tiende a formarse en la zona inferior del dominio poroso, con una celda deformada que prácticamente no tiene influencia de la falla; pero sí de la gravedad.

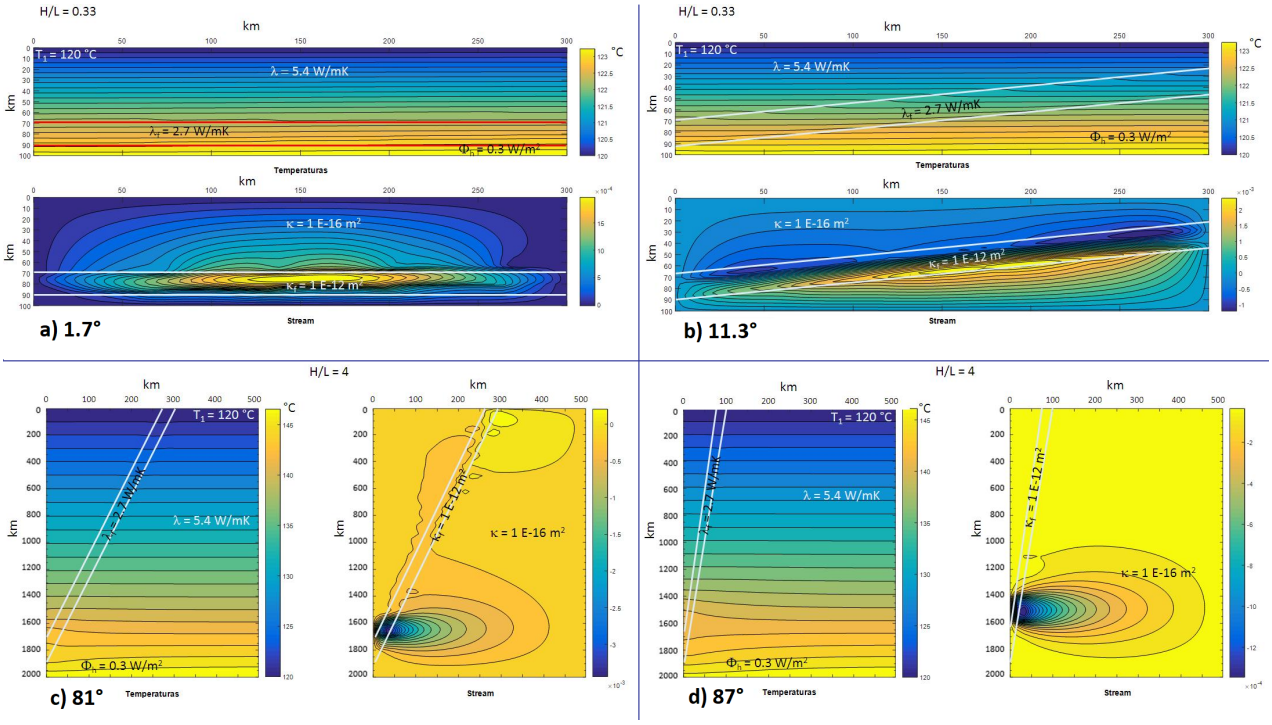


Figura 22: *Variación del campo de temperatura y de flujo debido a la inclinación de la falla.*

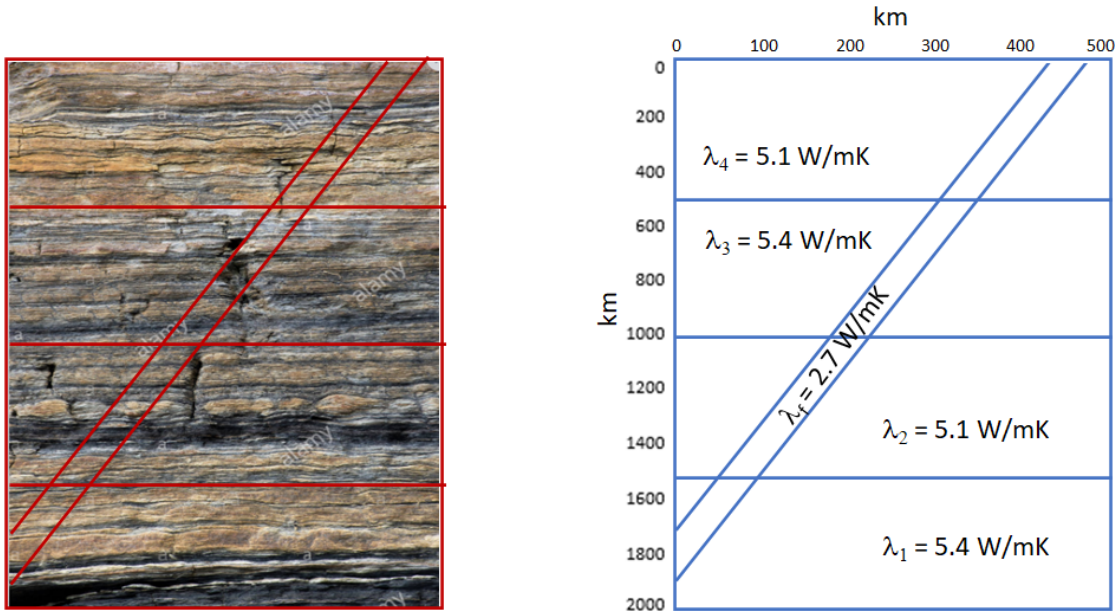
A continuación se modelan algunos ejemplos de fallas reales, en un dominio rectangular vertical compuesto por varias estratos; en un caso con espesores iguales, y en el segundo caso con espesores distintos de los estratos.

5.2.7 Medio Poroso Heterogéneo e Isotrópico con Falla Homogénea Anisotrópica. Estratos con el Mismo Espesor.

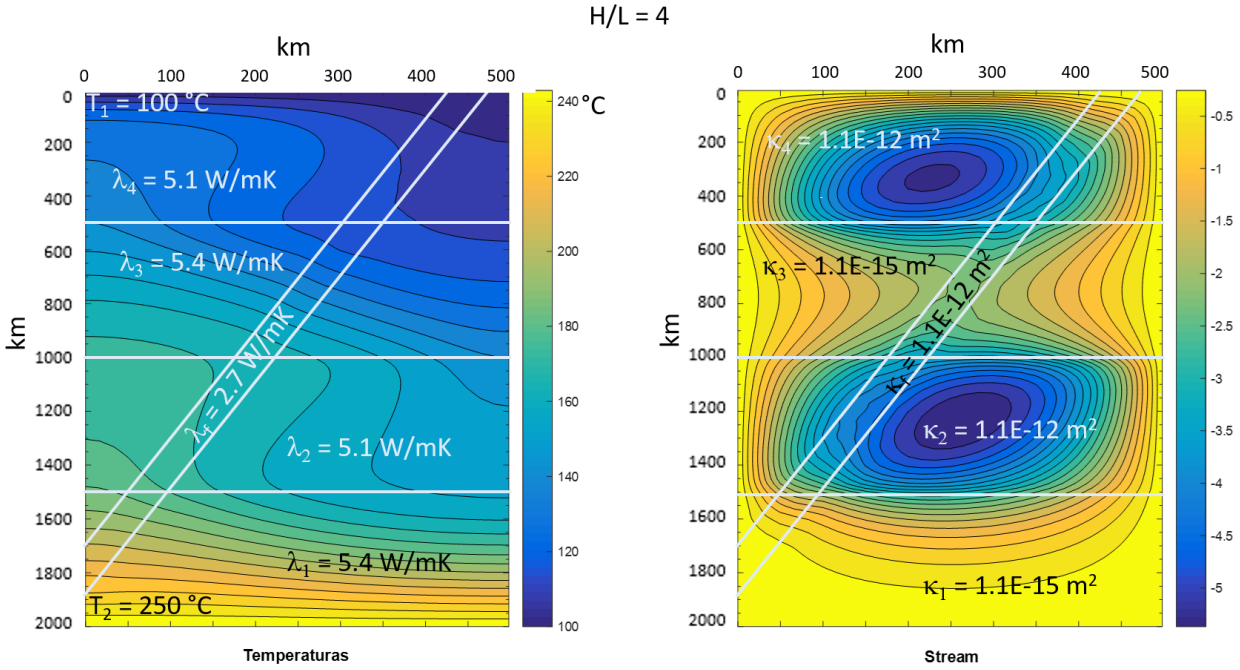
En la **Figura 23** se muestra el ensayo numérico que considera un dominio de $500\,km \times 2000\,km$, que representa un medio poroso compuesto por cuatro estratos de igual espesor con permeabilidad y conductividad térmica heterogénea e isotrópica, que es atravesado por una falla inclinada homogénea y anisotrópica. Las fronteras superior e inferior están isotérmicamente fijadas, las fronteras verticales son adiabáticas y no se presenta intercambio de fluidos en las fronteras.

Como puede observarse los estratos 2 y 4 tienen una permeabilidad superior en tres órdenes de magnitud que la de los estratos 1 y 3, por lo que la circulación de fluidos sólo se da en estos estratos, lo que ocasionan alteraciones en los campos de temperaturas observados.

Sin embargo, debido a la continuidad del sistema poroso, las líneas de corriente se suavizan y se separan más en los estratos 1 y 3, lo que provoca una variación más suave de la temperatura en dichos estratos.



(a) *Conceptualización geométrica de una falla real.*



(b) *Distribución de temperaturas y stream en un estado convectivo estable.*

Figura 23: *Modelación de un caso real de una falla que atraviesa a cuatro estratos de un mismo espesor. El ensayo numérico se realizó en un dominio rectangular vertical (H/L=4) que representa a un medio poroso heterogéneo (4 estratos, mismo espesor), con presencia de una falla inclinada que atraviesa a todo el medio. La conductividad térmica λ es menor en la falla que en el resto del medio poroso. La permeabilidad κ en la falla tiene el mismo valor que en 2 estratos alternos, y a su vez estas permeabilidades son mayores que en el resto de los estratos.*

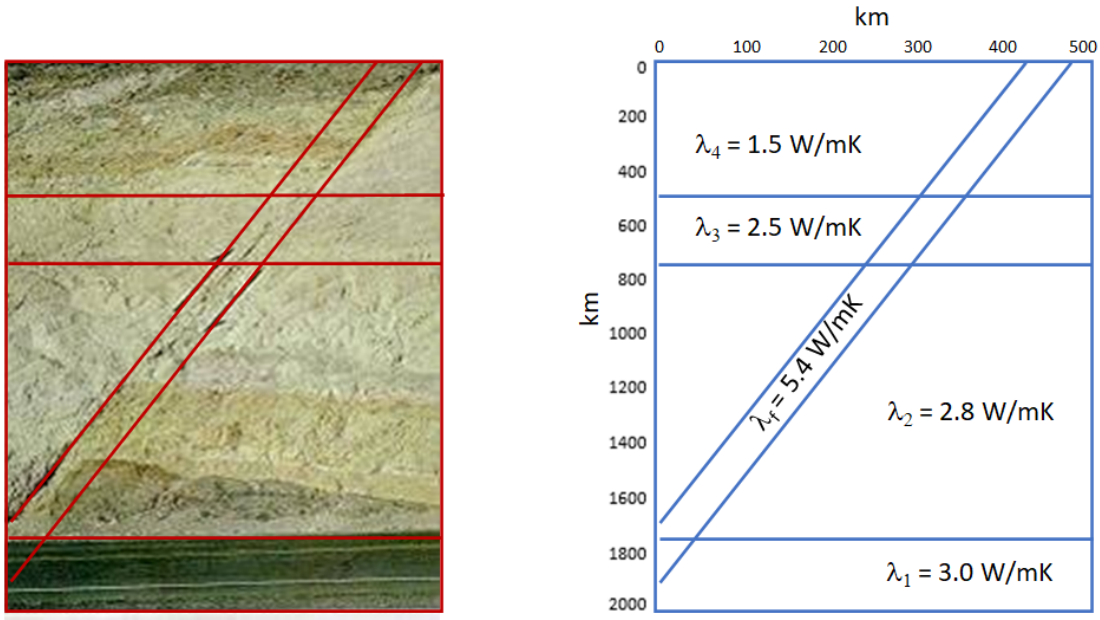
5.2.8 Medio Poroso Heterogéneo e Isotrópico con Falla Homogénea Anisotrópica. Estratos con Diferentes Espesores.

En este ensayo numérico se consideró un dominio de $500\,km \times 2000\,km$, que se muestra en la **Figura 24**, este muestra un medio poroso compuesto por cuatro estratos de diferentes espesores con permeabilidad y conductividad térmica heterogénea e isotrópica, que es atravesado por una falla inclinada homogénea y anisotrópica. Las fronteras superior e inferior están isotérmicamente fijadas, las fronteras verticales son adiabáticas y no se presenta intercambio de fluidos en las fronteras.

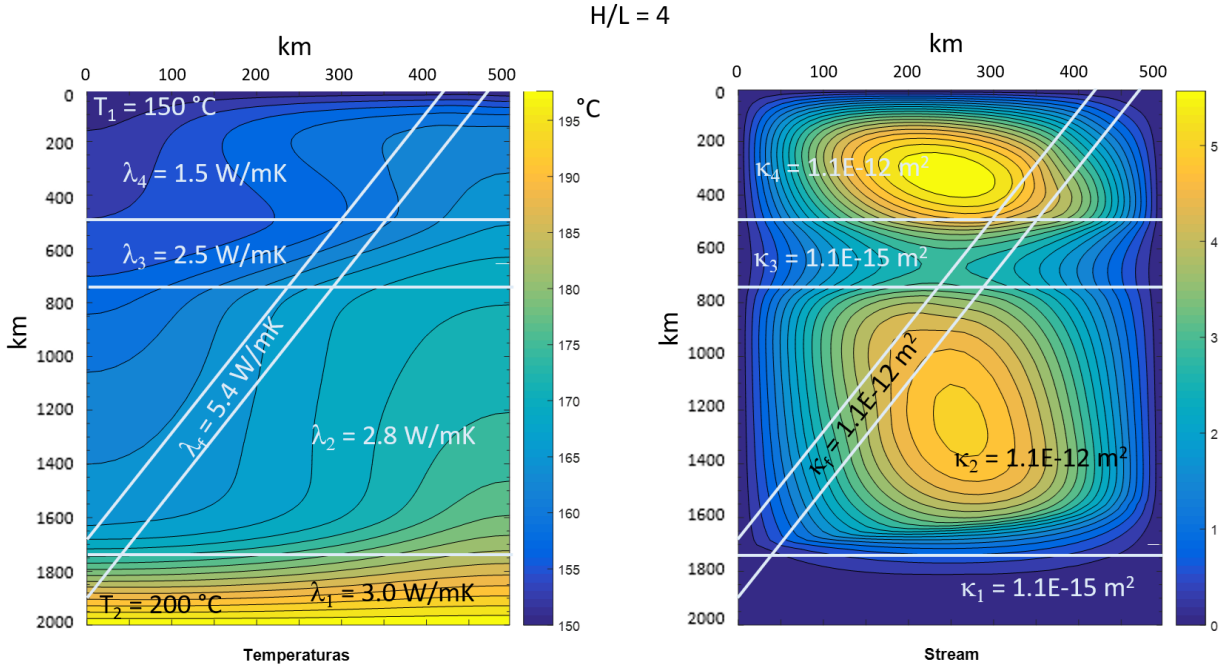
En este ensayo, los estratos 2 y 4 también tienen una permeabilidad superior en tres órdenes de magnitud que la de los estratos 1 y 3, por lo que la convección se presenta sólo se da en estos estratos, lo que ocasiona una fuerte alteración del campo de temperaturas en esos estratos.

Sin embargo, debido a la continuidad del sistema poroso, las líneas de corriente se suavizan y se separan más en los estratos 1 y 3, lo que provoca una variación más suave de la temperatura en dichos estratos.

También se observa en las **Figuras 23** y **24** que la celda convectiva abarca toda la extensión de los estratos activos, que son los de mayor permeabilidad y menor conductividad.



(a) *Conceptualización geométrica de una falla real*



(b) *Distribución de temperaturas y stream en un estado convectivo estable.*

Figura 24: *Modelación de un caso real de una falla que atraviesa a cuatro estratos de diferente espesor. El ensayo numérico se realizó en un dominio rectangular vertical (H/L=4) que representa a un medio poroso heterogéneo (4 estratos, diferente espesor cada uno), con presencia de una falla inclinada que atraviesa a todo el medio. La conductividad térmica λ es menor en la falla que en el resto del medio poroso. La permeabilidad κ en la falla tiene el mismo valor que en 2 estratos alternos, y a su vez estas permeabilidades son mayores que en el resto de los estratos.*

Capítulo 6

Conclusiones

Se utilizó la formulación matemática para el modelo de convección propuesta por [Flores-Márquez, 1992] y [Royer y Flores-Márquez, 1994] que acopla las ecuaciones de calor, de flujo, de conservación de la masa y de estado del agua, para resolver el problema de la convección natural en medios porosos. Esto fue una clara ventaja, ya que pudo usarse el mismo algoritmo para resolver las ecuaciones y hallar el campo de temperaturas y el de flujo (stream) de fluido.

La modificación de clases (en Java - NetBeans) para la implementación computacional del problema de convección del programa presentado por [Montealegre, 2010] dio resultados que fueron consistentes con los ya presentados en la literatura, lo cual indicó un buen funcionamiento del programa computacional modificado. Las clases desarrolladas en el programa para realizar ensayos numéricos que representaran condiciones geológicas reales también fueron corroboradas con la bibliografía existente.

Estos ensayos numéricos primero representaron geometrías sencillas, como cuadrados, con uno o dos estratos geológicos de propiedades isotrópicas y anisotrópicas, los resultados de estos ensayos reprodujeron observaciones previamente reportadas en la literatura. La complejidad geológica fue representada en los modelos sintéticos introduciendo varios estratos y fracturas con diferentes inclinaciones y anchos, además, de cambiar las razones geométricas alto/ancho en el dominio de los modelos y los resultados obtenidos muestran los comportamientos esperados tanto para el campo de temperaturas como para la función de corriente.

Las principales dificultades que se presentaron en abordar numéricamente los problemas de convección simulando casos geológicos reales, son: los concernientes a los valores de las dimensiones horizontal y vertical del dominio geométrico rectangular, a la inclinación de la falla y, no menos importante, a la sensibilidad del programa con los valores de permeabilidad. Sin embargo, debido a que el conjunto de ecuaciones de convección utiliza la razón geométrica como parámetro (y no las propias dimensiones del rectángulo) hizo que los ensayos numéricos se pudieran simplificar, manteniendo dicha razón. Por lo tanto, los numerosos ensayos que se realizaron para simular casos geológicos reales también obtuvieron resultados exitosos y consistentes con trabajos previos.

En cuanto a la interpretación física de los ensayos numéricos, se observó que la presencia de las celdas convectivas se debe principalmente a la permeabilidad, más que a la conductividad térmica y/o a las temperaturas o flujos de calor impuestos en las fronteras.

Los requerimientos computacionales para correr el programa son los siguientes: ambiente de programación Java SE Development Kit 15.0.2 [Java-JDK15, 2020], entorno Apache NetBeans IDE 12.0 [NetBeans-12.0, 2020], ejecutado en una computadora personal con sistema operativo Windows 10 de 64 bits, procesador Intel(R) Core(TM) i5-8250U, CPU de 1.6-1.8 Gz y RAM de 8 GB. Los tiempos de corrida (run time o total time) variaron desde $12s$ hasta $2m21s$. Los tiempos de convergencia variaron entre $0.53s$ y $0.6s$

6.1 Aplicaciones y Trabajos Futuros

El conocimiento del campo de temperaturas y de flujo (stream) de fluido cada una de las variables involucradas en el problema de convección, permite representar estas propiedades físicas en un yacimiento geotérmico o petrolero. En conjunto con otras técnicas y campos de conocimiento, se tendrá una mejor caracterización estática o dinámica del sistema de interés.

Otra aplicación importante es el análisis de pruebas de trazadores en conjunto con la simulación de streamlines, lo cual es necesario para los procesos de recuperación mejorada de yacimientos petroleros, en donde se hace inyección de agua o gas para extraer el petróleo atrapado en los yacimientos.

Algunos trabajos futuros que se podrían desarrollar a partir de la modelación bidimensional de flujos convectivos en medios porosos son:

- Modelación del problema de convección en tres dimensiones.
- Caracterización integral de un yacimiento geotérmico o de petróleo.
- Estudio y análisis con pruebas de trazadores y streamlines para determinar las trayectorias de flujo, en procesos de recuperación de hidrocarburos por inyección de fluidos.

Estos trabajos, podrían compararse con los ya existentes en la literatura. De esta manera se identificarían aspectos a mejorar en la modelación matemática de sistemas de flujos convectivos en medios porosos.

Cabe mencionar que las aplicaciones no se limitan únicamente a la modelación de sistemas terrestres, como fue el caso de este trabajo. Existen otras aplicaciones interesantes, tales como la caracterización del clima por medio de la dinámica de fluidos de la atmósfera, en donde el fluido de estudio es el aire. Otra aplicación es el tratamiento térmico de los alimentos envasados, en los cuales predomina la convección como mecanismo de calentamiento y enfriamiento.

Todas las aplicaciones mencionadas anteriormente comparten los mismos principios físicos. Por lo tanto, para finalizar este trabajo, se enfatiza más la motivación por comprender el fenómeno físico del flujo convectivo y su representación con la modelacion de sistemas, en particular los de interés para Ciencias de la Tierra.

Referencias

[Al-Amiri, 2002] Al-Amiri, A. M. (2002). *Natural convection in porous enclosures: The application of the two-energy equation model*. Numerical Heat Transfer, A(41), 817-834.

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

[Arellano-Gil et al., 2002] Arellano-Gil, J.; De Llata, R.; Carreón, M.A; Villareal, J.C. & Morales, W.V. (2002). *Ejercicios de Geología Estructural*. Ciudad de México: UNAM, Facultad de Ingeniería.

[Arellano-Gómez et al., 2008] Arellano-Gómez, V. M., Iglesias-Rodríguez, E. & García-Gutiérrez, A. (2008). *La energía geotérmica: una opción tecnológica y económicamente madura*. Boletín IIE, Tendencias tecnológicas, 102-114.

[Báez et al., 2004] Báez, E., Bermúdez, B. & Nicolás, A. (2004). *Convección natural en medios porosos y libres: simulación numérica*. Revista Mexicana de Física, 50(1), 36-48.

[Báez y Nicolás, 2006] Báez, E. & Nicolás, A. (2006). *2D natural convection flows in tilted cavities: Porous media and homogeneous fluids*. International Journal of Heat and Mass Transfer, 49, 4773–4785.

[Báez, 2008] Báez, E. (2008). *Convección natural y transferencia de calor en medios libres y porosos*. Tesis para obtener el grado de Doctora en Ciencias (Matemáticas). UAM, Unidad Iztapalapa.

[Barreiro, 2017] Barreiro, M. (2017). *Modelación numérica de la atmósfera: 2. Discretización de ecuaciones*. Montevideo, Uruguay. Apuntes de la asignatura. Universidad de la República, Facultad de Ciencias. Obtenido de $http://meteo.fisica.edu.uy/Materias/MNdelaA/Teorico\_MNdelaA/Tema2\_17.pdf$

[Basak et al., 2006] Basack, T., Roy, S., Paul, T. & Pop, I. (2006). *Natural convection in a square cavity filled with a porous medium: Effects of various thermal boundary conditions.* International Journal of Heat and Mass Transfer, 49, 1430–1441.

[Basak y Roy, 2008] Basack, T. & Roy, S. (2008). *Role of 'Bejan's heatlines' in heat flow visualization and optimal thermal mixing for differentially heated square enclosures.* International Journal of Heat and Mass Transfer, 51, 3486-3503.

[Bear, 1972] Bear, J. (1972). *Dynamic of fluids in porous media.* New York: Dover Publications, Inc.

[Bories y Combarnous, 1973] Bories, S. A. & Combarnous, M. A. (1973). *Natural convection in a sloping porous layer.* J. Fluid Mech., 57(1), 63-79.

[Calzada y Olivares, 2015] Calzada, R. y Olivares, J.C. (2010). *Conceptos de geotermía para ingenieros petroleros.* Tesis para obtener el grado de Licenciatura en Ingeniería Petrolera. UNAM, Facultad de Ingeniería.

[Carrillo et al., 2008] Carrillo, A., Herrera, I. & Yates, R. (2008). *Método de Elementos Finitos.* Ciudad de México: UNAM, Instituto de Geofísica-Grupo de Modelación Matemática y Computacional.

[Chen, 2005] Chen, Z. (2005). *Finite Element Methods and Their Applications.* New York: Springer-Verlag Berlin Heidelberg.

[Chen et al., 2006] Chen, Z., Huan, G. & Ma, Y. (2006). *Computational Methods for Multiphase Flows in Porous Media.* Philadelphia: SIAM.

[Ciarlet, 2002] Ciarlet, P. G. (2002). *The finite element method for elliptic problems.* Philadelphia: SIAM.

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

[De la Cruz - Salas, 2019] De la Cruz - Salas, L. M. (2019). *Modelación Matemática de Sistemas Terrestres II.* CdMx, México. Apuntes de la asignatura, Posgrado en Ciencias de la Tierra. UNAM, Instituto de Geofísica.

[Deng y Tang, 2002] Deng, Q. H. & Tang, G. F. (2002). *Numerical visualization of mass and heat transport for conjugate natural convection/heat conduction by streamline and heatline.* International Journal of Heat and Mass Transfer, 45, 2373–2385.

[Evans y Raffensperger, 1992] Evans, D. G. & Raffensperger, J. P. (1992). *On the Stream Function for Variable-Density Groundwater Flow.* Water Resources Research, 28(8), 2141-2145.

[Flores-Márquez, 1992] Flores-Márquez, E. 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.

[Herrera y Pinder, 2012] Herrera, I. & Pinder, G. P. (2012). *Mathematical modeling in science and engineering, An axiomatic approach.* New York: John Wiley & Sons.

[Horne, 1979] Horne, R. N. (1979). *Three-dimensional natural convection in a confined porous medium heated from below.* J. Fluid Mech., 92(4), 751-766.

[Ibrahim e Hirpho, 2021] Ibrahim, W. & Hirpho, M. (2021). *Finite element analysis of mixed convection flow in a trapezoidal cavity with non-uniform temperature.* Heliyon, 6(e05933), 1-12.

[Java-JDK15, 2020] Java SE Development Kit 15.0.2 (2020). Obtenido de $https://www.oracle.com/java/technologies/javase-jdk15-downloads.html$

[Jiménez-Islas, 1999] Jiménez-Islas, H. (1999). *Modelamiento matemático de los procesos de transferencia de momentum, calor y masa en medios porosos.* Tesis para obtener el grado de Doctor en Ciencias. UAM-I, División de Ciencias Básicas e Ingeniería.

[Jiménez-Islas et al., 2009] Jiménez-Islas, H.; Calderón-Ramírez, M.; Navarrete-Bolaños, J.L.; Botello-Álvarez, J.E.; Martínez-González, G.M. & López-Isunza, F. (2009). *Estudio numérico de la convección natural en una cavidad cuadrada en 2-D con interfase fluido-medio poroso y generación de calor.* Revista Mexicana de Ingeniería Química, 8(2), 169-185.

[Jouglard, 2002] Jouglard, C. E. (2002). *Introducción al método de los elementos finitos: Formulación variacional de elementos finitos*. Buenos Aires, Argentina. Curso de especialización. Universidad Tecnológica Nacional.

[Kimura y Bejan, 1983] Kimura, S. & Bejan, A. (1983). *The "Heatline" Visualization of Convective Heat Transfer*. Transactions of the ASME, 105, 916-919.

[Kumar et al., 2015] Kumar-Singh, A., Basak, T., Nag, A. & Roy, S. (2015). *Heatlines and thermal management analysis for natural convection within inclined porous square cavities*. International Journal of Heat and Mass Transfer, 87, 583–597.

[Landau y Lifshitz, 1987] Landau, L. D. & Lifshitz, E. M. (1987). *Fluid mechanics*. (2nd ed.) Great Britain: Pergamon Press, INC.

[Manet, 2015] Manet, V. (2015). *Méthode des éléments finis: Vulgarisation des aspects mathématiques et illustration de la méthode*. France. Versión libre obtenida de *https* : *//www.researchgate.net/publication/*278301498*_La_Methode_des_Elements_Finis_vulgarisation_des_aspects_mathematiques_illustration_des_capacites_de_la_methode/link/*5*c*8*b*865*d*299*bf*14*e*7*e*7*d*213*b/download*

[Matlab, 2016] Matlab versión 9.1.0 (R2016b) (2016).

[McKibbin y Tyvand, 1982] McKibbin, R. & Tyvand, P. A. (1982). *Anisotropic modelling of thermal convection in multilayered porous media*. Journal of Fluid Mechanics, 118, 315-339.

[Montealegre, 2010] Montealegre, C. (2010). *Modelación de flujo convectivo en un medio poroso anisotrópico y heterogéneo*. Tesis para obtener el grado de Maestría en Ciencias de la Tierra. UNAM, Instituto de Geofísica.

[Moya y Ramos, 1987] Moya, S. L. & Ramos, E. (1987). *Numerical study of natural convection in a tilted rectangular porous material*. Int. J. Heat Mass Transfer, 30(4), 741-756.

[NetBeans-12.0, 2020] Apache NetBeans IDE 12.0 (2020). Obtenido de *https* : *//netbeans.apache.org/download/nb*120*/nb*120*.html*

[Pandit y Chattopadhyay, 2014] Pandit, S. K. & Chattopadhyay, A. (2014). *Higher order compact computations of transient natural convection in a deep cavity with porous medium*. International Journal of Heat and Mass Transfer, 75, 624-636.

[Ribando y Torrance, 1976] Ribando, R. J. & Torrance, K. E. (1976). *Natural convection in a porous medium: Effects of confinement, variable permeability, and thermal boundary conditions*. Journal of Heat Transfer, 76, 42-48.

[Richard y Gounot, 1981] Richard, J. P. & Gounot, J. (1981). *Critere d'apparition de la convection naturelle dans des couches poreuses stratifiees*. Int. J. Heat Mass Transfer, 24(8), 1325-1334.

[Riley y Winters, 1990] Riley, D. S. & Winters, K. H. (1990). *A numerical bifurcation study of natural convection in a tilted two-dimensional porous cavity*. J. Fluid Mech., 215, 309-329.

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

[Saad, 2000] Saad, Y. (2000) (2a ed.). *Iterative Methods for Sparse Lineal Systems*. SIAM.

[Saeid y Pop, 2004] Saeid, N. H. & Pop, I. (2004). *Transient free convection in a square cavity filled with a porous medium*. International Journal of Heat and Mass Transfer, 47, 1917-1924.

[Skiba, 2005] Skiba, Y. N. (2005). *Métodos y esquemas numéricos: Un análisis computacional*. Ciudad de México: UNAM, CCA.

[Skiba, 2008] Skiba, Y. N. (2008). *Introducción a la dinámica de fluidos*. Ciudad de México: UNAM, CCA.

[Zhang et al., 2016] Zhang, C.; Zarrouk, S. J. & Archer, R. (2016). *A mixed finite element solver for natural convection in porous media using automated solution techniques*. Computers and Geosciences, 96, 181–192.

[Zhao et al., 2019] Zhao, C.; Hobbs, B. E. & Ord, A. (2019). *Computational modeling of convective seepage flow in fluid-saturated heterogeneous rocks: Steady-state approach.* Computers and Geosciences, 123, 103–110.

[Zienkiewics et al., 2005] Zienkiewics, O. C.; Taylor, R. L. & Zhu, J. Z. (2005). *The finite element method: Its basis and fundamentals.* (6th ed.) London: Elsevier.

Apéndice A

Propiedades Físicas del Medio Poroso y del Fluido

- **Porosidad**

  Desde el punto de vista cualitativo la porosidad es la capacidad de una roca de tener poros, entendiendo por poro cualquier espacio de una masa rocosa que no esté ocupado por un material sólido, sino por un fluido (agua, aire, petróleo, etc.). Cuantitativamente, la porosidad se define como el espacio total ocupado por poros en un volumen determinado de roca. La porosidad normalmente se da en [%].

  Los poros o 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 promedio de algunos tipos de rocas o sedimentos es de: gravas, 25-40%; arenas y gravas, 25-30%; arenas, 25 a 47%; arcillas, 44-50%; limos, 34 a 50%; caliza, 1-17%; arenisca, 4-26% y yeso, 4%.

  La porosidad eficaz es la fracción o porcentaje del volumen total de la roca formado por los poros interconectados. Este tipo de porosidad depende de la forma, tamaño y disposición de los granos; disminuye con el diámetro y aumenta con la uniformidad. Evidentemente para un sistema geotérmico, es necesario que las rocas tengan una buena porosidad eficaz, a fin de que los fluidos circulen lo más libremente posible. La porosidad eficaz da lugar a la permeabilidad.

- **Permeabilidad**

  La permeabilidad es una medida de 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, y la presión a que está sometido el fluido.

  Para ser permeable, un material debe ser poroso; es decir, debe contener espacios vacíos o poros con el fin de que el fluido tome lugar en esos espacios vacios, entre los granos de los minerales y en los espacios creados por fracturas y fallas (en el ámbito geológico). A su vez, tales espacios deben estar interconectados para que el fluido disponga de caminos para pasar a través del material; es decir, se requiere que la roca tenga porosidad efectiva. Una roca sin porosidad efectiva será una roca impermeable. En el Sistema Internacional de Unidades se mide la permeabilidad en m^2, aunque también se usa mucho la unidad Darcy; $1\,Darcy = 9.86923^{-13} m^2$.

- **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 $[kg/m^3]$. 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. La densidad es una magnitud intensiva.

  La densidad de un fluido se define como la masa del fluido por unidad de volumen. En general varia con la presión y temperatura de acuerdo con las ecuaciones de estado: $\rho = \rho(p, T)$ o $f(\rho, p, T) = 0$

- **Viscosidad**

  La viscosidad es la propiedad que tiene un fluido para resistir cualquier deformación. Es decir, es una medida de la oposición del fluido a ceder a las deformaciones tangenciales, cuando el fluido está en movimiento. 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 Pascal-segundo $[Pa \cdot s]$. La viscosidad cinemática se define como el cociente entre la viscosidad dinámica y la densidad, siendo sus unidades $[m^2/s]$. 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.

- **Conductividad Térmica**

  La 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 $[W/K \cdot m]$. En general, la conductividad térmica global de un medio poroso depende de forma compleja de la geometría del medio. La inversa de la conductividad térmica es la resistencia térmica, que es la capacidad de los materiales para oponerse al paso del calor.

- **Capacidad Calorífica**

  La 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 canti-

dad 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 $[J/m^3 \cdot K]$

El agua tiene como característica que presenta alta capacidad calorífica y un alto calor latente de vaporización, por lo tanto, es un fluido ideal para transferir calor (Torres, et al., 1993).

- **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 cierta 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 $[J/kg \cdot K]$.

  El valor del calor específico depende, por lo general, de la temperatura inicial del sistema, pero su influencia es leve y se asume constante. Esta magnitud es intensiva; y por lo tanto, una muestra será suficiente para evaluar el calor específico del afloramiento y de su hipotético volumen en profundidad. Esta magnitud también es isótropa y se requiere el conocimiento acotado de su valor para la estimación y modelamiento temporal del flujo termal. Al calentarse tanto rocas como sedimentos, el calor involucrado en el aumento termal de los respectivos cuerpos puede corresponder a gran parte del flujo calórico resultante. Si los procesos de erosión y sedimentación ocurren a una tasa baja, el calor específico no ejerce mayor efecto, y el gradiente geotérmico permanece medianamente constante.

- **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.

Apéndice B

Nomenclatura Utilizada en la Formulación Matemática

B.1 Parámetros Dimensionales

a_t	Coeficiente de variación de la conductividad térmica con respecto a la temperatura $\left[\frac{1}{K}\right]$
A^*	Producción de calor $\left[\frac{J}{m^3 \cdot s}\right]$
e_1	Vector unitario para el sistema coordenado horizontal $[m]$
e_3	Vector unitario para el sistema coordenado vertical $[m]$
$\underline{\hat{g}}$	Aceleración de la gravedad $\left[\frac{m}{s^2}\right]$
H	Altura del rectángulo $[m]$
κ	Permeabilidad media isotrópica $\left[m^2\right]$
$\mathbf{K}$	Tensor simétrico de permeabilidad anisotrópica $\left[m^2\right]$
L	Ancho del rectángulo $[m]$
p	Presión $[Pa]$
S	Matriz diagonal asociada al sistema coordenado (x^*, y^*). $\begin{pmatrix} 1/H & 0 \\ 0 & 1/L \end{pmatrix}$
t^*	Tiempo $[s]$
T^*	Temperatura $[K]$
T_r^*	Temperatura al nivel de referencia $[K]$. Por ejemplo: $T_r^* = \frac{T_1^* + T_2^*}{2}$
$\mathbf{v}$	Velocidad de filtración $\left[\frac{m}{s}\right]$
x^*	Coordenadas horizontales paralelas a L $[m]$
z^*	Coordenadas verticales paralelas a H $[m]$
β_{th}	Coeficiente de expansión volumétrica del fluido saturante $\left[\frac{1}{K}\right]$
γ^*	Coeficiente de variación de la viscosidad dinámica con relación a la temperatura del fluido saturante $\left[\frac{1}{K}\right]$

ΔT^* Diferencia de temperaturas $[K]$. $T_2^* - T_1^*$

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

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

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

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

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

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

Φ_h^* Densidad de flujo de calor vertical a través de los límites horizontales o paralelos a las isotermas $\left[\frac{W}{m^2}\right]$

Φ_v^* Densidad de flujo de calor horizontal a través de las superficiesnverticales o perpendiculares a las isotermas $\left[\frac{W}{m^2}\right]$

B.2 Parámetros Adimensionales

A Producción de calor adimensional. $\frac{2A^* LH}{tr\Lambda^* \Delta T^*}$

e_1 Vector unitario adimensional para el sistema coordenado horizontal.

e_3 Vector unitario adimensional para el sistema coordenado vertical.

K Tensor de permeabilidad adimensional en 2D. $\left(\frac{2}{tr\mathbf{K}}\right)\mathbf{K}$

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

Ra^* Número de Rayleigh modificado para filtración de Darcy en un medio heterogéneo (ec. 2.14)

t Tiempo adimensional. $\frac{t^* \cdot tr\Lambda}{2LH(\rho c)^*}$

T Temperatura adimensional. $\frac{T^* - T_r^*}{\Delta T^*}$

v Velocidad de flujo adimensional. $v_Darcy - \nabla(\ln tr\Lambda^*) \cdot \Lambda$

v_{Darcy} Velocidad de filtración adimensional, componentes $[v_x, v_z]$

v_x Velocidad de filtración adimensional horizontal. $\frac{2H(\rho c)_f v_x}{tr\Lambda^*}$

v_z Velocidad de filtración adimensional vertical. $\frac{2L(\rho c)_f v_z}{tr\Lambda^*}$

x Coordenada horizontal x adimensional. $\frac{x^*}{L}$

z Coordenada vertical z adimensional. $\frac{z^*}{H}$

α Razón geométrica adimensional, 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

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

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

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

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

ψ Función de corriente en condiciones estacionarias

B.3 Superídices

t	Operador matricial transpuesta
-1	Operador matricial inversa

B.4 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

B.5 Operadores

$\mathbf{j}$	Matriz de rotación $= \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$ se nota que $\mathbf{j} = -\mathbf{I}$, donde $\mathbf{I}$ es el operador de identidad.
α	Tensor métrico asociado al sistema coordenada adimensional (x, z). $\begin{bmatrix} \sqrt{\alpha} & 0 \\ 0 & \frac{1}{\sqrt{\alpha}} \end{bmatrix}$
tr	Operador traza
det	Determinante
$\bullet$	Producto escalar
$\otimes$	Producto tensorial
$\{a_i\}$	Sumatoria de términos a_i con respecto a i, $\sum_i a_i$
∇	Operador nabla 2D. $\left(\frac{\partial}{\partial x^*}, \frac{\partial}{\partial z^*}\right) = grad$
$\mathbf{J}$(f)	Jacobiano de la función vectorial f
$\mathbf{H}(f)$	Hessiano de la función escalar f
$\mathbf{D}$	Operador rotacional 2D. $\left(\frac{\partial}{\partial z^*}, -\frac{\partial}{\partial x^*}\right) = \nabla\mathbf{j} = curl$
Δf	Laplaciano de una función escalar f

Apéndice C

Códigos de Programación

C.1 Main.java

```
package ModeloConveccion;
import AlgebraLineal.Vector;

/**
* Clase usada para probar la solución de un
    modelo de flujo de calor
* convectivo.
* @author Conrado Montealegre
* @version 1.0
* @since 1.0
* Modificado por Alma MN
*/
public class Main {
    public static void main(String[] args)
        throws ClassNotFoundException
    {
        ModeloFlujoConvectivo modConv;

        //Prueba 1 Capa, Homogeneo, Ra=80, HL
            =1
        //DefinicionModeloConveccion2D
            defModConv = new Capa1HL1K1();

        //Prueba 1 Capa, Homogeneo, Ra=160,
            HL=1
        //DefinicionModeloConveccion2D
            defModConv = new Capa1HL1K2();

        //Prueba 1 Capa, Homogeneo, Ra=200, H
            /L=1.8
        //DefinicionModeloConveccion2D
            defModConv = new Capa1HL1_8K();

        //Prueba 1 Capa, Homogeneo, Ra=80, H/
            L=0.33
        //DefinicionModeloConveccion2D
            defModConv = new Capa1HL0_33_K1()
            ;

        //Prueba 2 Capas, Isotropico, Ra= , H
            /L=1.0
        //DefinicionModeloConveccion2D
            defModConv = new Capa2HL1();

        //Prueba 3 Capas, Isotropico, Ra= , H
            /L=0.33
        //DefinicionModeloConveccion2D
            defModConv = new Capa3HL0_33();

        //Prueba 5 Capas, Isotropico, Ra= , H
            /L=1.0
        //DefinicionModeloConveccion2D
            defModConv = new Capa5HL11();

        //Prueba Capa inclinada isotropica
        //DefinicionModeloConveccion2D
            defModConv = new CapaInclinada1()
            ;

        //Prueba Capas inclinada anisotropica
        //DefinicionModeloConveccion2D
            defModConv = new CapaInclinada2()
            ;

        // ** NUEVAS PRUEBAS AMN ** \\
        //Prueba AMN Medio Homogéneo e Isotró
            pico, sin Capa Inclinada
        //DefinicionModeloConveccion2D
            defModConv = new Capa1HL1_AMN();
```

```
//Prueba AMN Capa inclinada
    anisotropica, sin flujo de calor
//DefinicionModeloConveccion2D
    defModConv = new
    CapaInclinada1sinflujocalor_AMN()
    ;

//Prueba AMN Capa inclinada
    anisotropica con flujo de calor
//DefinicionModeloConveccion2D
    defModConv = new
    CapaInclinada2FlujoCalor_AMN();

//Prueba AMN Capa inclinada
    anisotropica con 2 capas
    horizontales
//DefinicionModeloConveccion2D
    defModConv = new Capa2CapInc_AMN
    ();

//Prueba AMN Capa inclinada
    anisotropica con 3 capas
    horizontales
//DefinicionModeloConveccion2D
    defModConv = new Capa3CapaInc_AMN
    ();

//Prueba AMN Capa inclinada
    anisotropica con 4 capas
    horizontales
DefinicionModeloConveccion2D
    defModConv = new Capa4CapaInc_AMN
    ();

//Prueba AMN Capa inclinada
    anisotropica con 5 capas
    horizontales
//DefinicionModeloConveccion2D
    defModConv = new Capa5CapaInc_AMN
    ();

//Prueba AMN Capa inclinada
    anisotropica con 6 capas
    horizontales
//DefinicionModeloConveccion2D
    defModConv = new Capa6CapaInc_AMN
    ();

//Prueba AMN Capa inclinada
    anisotropica con 8 capas
    horizontales
//DefinicionModeloConveccion2D
    defModConv = new Capa8CapaInc_AMN
    ();

modConv = new ModeloFlujoConvectivo(
    defModConv);

double tIni = 0.0;
double tFin = 0.523;
//double tFin = 1.113;
int nSol = (int)Math.ceil((tFin-tIni)
    /defModConv.getDt())+1;
double t[] = new double[nSol];

Vector [][] uk =
        new Vector[2][nSol];
uk = modConv.corridaTransitoria(tIni,
     tFin, t);

System.out.println("
    ***************************");
System.out.println("Tiempo "+ t[nSol
    -1]);
System.out.println("X\tZ\tTemp\
    tStream");
for (int i=0; i<modConv.probStream.
    discretizacion.totalNodos; i++)
{
    System.out.println(
            modConv.
                getCoordenadasReales().
                retornaValor(i, 0)+"\t"
                +
            modConv.
                getCoordenadasReales().
                retornaValor(i, 1)+"\t"
                +
```

```
                    uk[0][nSol-1].
                        RetornaValor(i)+"\t"+
                    uk[1][nSol-1].
                        RetornaValor(i)+"\t"
                    //modConv.mod.getKAst(i).
                        retornaValor(0, 0)+"\
                        t"+
                    //modConv.mod.getKAst(i).
                        retornaValor(1, 1)
                    ); } } }
```

C.2 Capa1HL1_AMN.java

```
package ModeloConveccion;
import AlgebraLineal.Matriz;
import AlgebraLineal.MatrizDensa;
import FEM.FuncionExpresion;
import FEM.Frontera;
import FEM.TriangulacionDominioRectangular;
import FEM.Nodo;
import FEM.NodoBorde;
import FEM.DominioRectangular;
import java.util.List;
import java.util.ArrayList;

/**
 * Clase usada para probar la interfaz
     DefinicionModeloConveccion2D. En esta
 * clase se simulará solamente la obtención
     de los parámetros del modelo de
 * convección.
 * @author Conrado
 * @version 1.0
 * @since 1.0
 * Modificado por Alma MN
 */
public class Capa1HL1_AMN extends
    DefinicionModelo
        implements
            DefinicionModeloConveccion2D {
    //Objetos usados para obtener la
        información que permite simular la
    //"lectura" de información de un modo
        externo
    private DominioRectangular dominio;
    private TriangulacionDominioRectangular
        discretizacion;

    public Capa1HL1_AMN()
    {   //En este caso, como se está
        simulando la lectura de la informació
        n
        //de un medio externo, en esta rutina
             se crea una discretización igual
        //a la que se creo con la información
             de esta "interfaz" para poder
        //simular la lectura de la informació
            n en los nodos.

        //Define el dominio que se usará en
            el modelo. En este caso se trata
        //de un dominio rectangular.
        this.dominio = new DominioRectangular
            ();
        this.dominio.defineDominioRectangular
            (this.getX1(), this.getZ1(),
                this.getX2(), this.getZ2());

        //Define la discretización usada para
             la ecuación de flujo de calor.
        this.discretizacion=new
         TriangulacionDominioRectangular(
            dominio,this.getNx(),this.getNz
            ());
    }

    /**
     * Obtiene el mínimo valor en la direcció
         n X del dominio rectangular.
     * @return
     */
    public double getX1()
    { return 0.0; }

    /**
     * Obtiene el máximo valor en la direcció
         n X del dominio rectangular.
     * @return
     */
```

```
public double getX2()
{
    return 100.0;
}

/**
 * Obtiene el mínimo valor en la direcció
     n Z del dominio rectangular.
 * @return
 */
public double getZ1()
{
    return 0.0;
}

/**
 * Obtiene el máximo valor en la direcció
     n Z del dominio rectangular.
 * @return
 */
public double getZ2()
{
    return 100.0;
}

/**
 * Obtiene la altura del dominio
     rectangular.
 * @return
 */
public double getH()
{
    return this.getZ2()-this.getZ1();
}

/**
 * Obtiene el ancho del dominio
     rectangular.
 * @return
 */
public double getL()
{
    return this.getX2()-this.getX1();
}
```

```
/**
 * Obtiene el 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.
 * @return
 */
public int getNx()
{
    return 50;
}

/**
 * Obtiene el 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.
 * @return
 */
public int getNz()
{
    return 50;
}

/**
 * Obtiene el coeficiente de relajación
     usado en Crank Nicholson.
 * @return
 */
public double getTheta()
{
    return 0.8;
}

/**
 * Obtiene el salto en el tiempo (Dt)
     usado en Crank Nicholson.
 * @return
 */
```

```
public double getDt()
{
    return 0.1;
}

/**
 * Obtiene el valor de la temperatura
     frontera superior.
 * @return
 */
public double getT1()
{
    return 120;
}

/**
 * Obtiene el valor de la temperatura
     frontera inferior.
 * @return
 */
public double getT2()
{
    return 150;
}

/**
 * Obtiene el valor de la temperatura de
     referencia.
 * @return
 */
public double getTRef()
{
    return (getT1()+getT2())/2.0;
}

/**
 * Obtiene el valor del tensor simétrico
     de conductividad térmica
 * anisotrópica en un  nodo de la
     triangulación.
 * @param iNodo Índice del nodo de la
     triangulación.
 * @return      Una matriz con el valor
     del tensor simétrico de
 *              conductividad térmica
     anisotrópica.
 */
public Matriz getLambAst(int iNodo)
{
    //Se simula la obtención del valor
        del tensor.
    MatrizDensa mat = new MatrizDensa
        (2,2);

    //En este caso particular se trata de
         un medio homogéneo isotrópico.
    mat.asignaValor(0, 0, 2.7);
    mat.asignaValor(1, 1, 2.7);
    mat.asignaValor(0, 1, 0.0);
    mat.asignaValor(1, 0, 0.0);

    return mat;
}

/**
 * Obtiene el valor del tensor simétrico
     de permeabilidad
 * anisotrópica en un  nodo de la
     discretización.
 * @param iNodo Índice del nodo de la
     discretización
 * @return      Una matriz con el valor
     del tensor simétrico de
 *              permeabilidad anisotró
     pica.
 */
public Matriz getKAst(int iNodo)
{
    //Se simula la obtención del valor
        del tensor.
    MatrizDensa mat = new MatrizDensa
        (2,2);

    //En este caso particular se trata de
         un medio homogéneo isotrópico.
    mat.asignaValor(0, 0, 0.000000000001)
        ;
    mat.asignaValor(1, 1, 0.000000000001)
        ;
```

```
        mat.asignaValor(0, 1, 0.0);
        mat.asignaValor(1, 0, 0.0);

        return mat;
    }

    /**
     * Obtiene el valor de la producción de
         calor en un  nodo
     * de la discretización.
     * @param iNodo Índice del nodo de la
         discretización
     * @return      El valor de la producción
          de calor en el nodo.
     */
    public double getAAst(int iNodo)
    {
        //En este caso particular no hay
            producción de calor.
        return 0.0;
    }

    /**
     * Obtiene el valor de la capacidad calor
         ífica del fluido.
     * @return
     */
    public double getRhoc()
    {
        return 4180000;
    }

    /**
     * Obtiene el valor de la densidad media
         del fluido.
     * @return
     */
    public double getRho()
    {
        return 968;
    }

    /**
     * Obtiene el valor de la viscosidad diná
         mica del fluido a la temperatura
     * de referencia.
     * @return
     */
    public double getNeta()
    {
        return 0.00018;
    }

    /**
     * Obtiene el valor del coeficiente de
         dilatación térmica del fluido.
     * @return
     */
    public double getBeta()
    {
        return 0.0008275;
    }

    /**
     * Obtiene el valor de la aceleración de
         la gravedad.
     * @return
     */
    public double getG()
    {
        return 9.81;
    }

    /**
     * Obtiene el valor del coeficiente de
         dependencia de la viscosidad
     * dinámica del fluido con la temperatura
         .
     * @return
     */
    public double getGamma()
    {
        return 0;
    }

    /**
     * Obtiene el valor de la diferencia má
         xima que debe de existir entre dos
     * valores consecutivos de la solución de
          Temperatura dentro del ciclo
```

```
     * iterativo de acoplamiento con Stream.
     * @return
     */
    public double getDifTempMax()
    {
        return 0.001;
    }

    /**
     * Obtiene el valor de la diferencia má
         xima que debe de existir entre dos
     * valores consecutivos de la solución de
          Stream dentro del ciclo
     * iterativo de acoplamiento con
         Temperatura.
     * @return
     */
    public double getDifStreamMax()
    {
        return 0.1;
    }

    /**
     * Obtiene el número máximo de
         iteraciones permitido dentro del
         ciclo
     * usado para acoplarlas soluciones de
         Stream y Temperatura.
     * @return
     */
    public int getMaxIt()
    {
        return 50;
    }

    /**
     * Obtiene la FuncionExpresión que define
          las condiciones iniciales de
     * temperatura del problema de conveccion
         .
     * @return  Una FuncionExpresion que
         define las condiciones iniciales de
     *          temperatura del problema de
         conveccion.
     */
    public FuncionExpresion getTempCondIni()
    {
        //Inicializa y asigna la función
            correspondiente a las C.I.
        FuncionExpresion condIniArg =
                //new FuncionExpresion("exp(x
                    )*cos(z)");
                //new FuncionExpresion("0");
                //new FuncionExpresion("cos(x
                    )");
                //new FuncionExpresion("sin
                    ((1-z)*1.5)*cos(x)");

                new FuncionExpresion("-sin(z*
                    pi-pi/2)/2");

        return condIniArg;
    }

    /**
     * Obtiene las fronteras que definen el
         modelo de flujo de calor en un
     * problema de convección.
     * @return  Un arreglo de Frontera que
         contiene las fronteras que definen
     *          el modelo de flujo de calor.
     */
    public Frontera[] getTempFront()
    {
        //Variable usada para definir el id
            de la frontera
        int idFrontCont = 0;

        //Inicializa el arreglo de Fronteras,
             las cuales en este caso serán
        //cuatro.
        Frontera[] fronteraArg = new Frontera
            [4];

        //Incializa el Vector que servirá
            para definir la normal a cada
        //frontera
        AlgebraLineal.Vector normFront =  new
             AlgebraLineal.Vector(2);
```

```
        //Frontera 0
        //Frontera Dirichlet correspondiente
            al borde inferior del dominio
        normFront.AsignaValor(0, 0);
        normFront.AsignaValor(1, -1);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                String.valueOf(this.
                    temRealtoAdim(this.getT2
                    ())), normFront);

        //Frontera 1
        //Frontera Neumann correspondiente al
             borde derecho del dominio
        normFront.AsignaValor(0, 1);
        normFront.AsignaValor(1, 0);
        fronteraArg[idFrontCont] = new
            Frontera(false, idFrontCont++,
                "0", normFront);

        //Frontera 2
        //Frontera Dirichlet correspondiente
            al borde superior del dominio
        normFront.AsignaValor(0, 0);
        normFront.AsignaValor(1, 1);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                String.valueOf(this.
                    temRealtoAdim(this.getT1
                    ())), normFront);

        //Frontera 3
        //Frontera Neumann correspondiente al
             borde izquierdo del dominio
        normFront.AsignaValor(0, -1);
        normFront.AsignaValor(1, 0);
        fronteraArg[idFrontCont] = new
            Frontera(false, idFrontCont++,
                "0", normFront);

        return fronteraArg;
    }


    /**
     * Obtiene la asignación de los nodos
         borde que pertenecen a cada una
     * de las fronteras en particular del
         modelo de flujo de calor en un
     * problema de convección.
     * @return  Un arreglo de objetos List
         que contienen las listas de los
     *          índices de los nodos que
         pertenecen a cada una de las
     *          fronteras del problema.
     */
    public List<Integer> []
        getListTempNodFront()
    {
        ////////////////////////////
        //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> [] nodFront = new
            ArrayList[getTempFront().length];

        //Inicializa las listas que contendrá
            n los índices de los nodos que
        //pertenecen a éstas
        for (int i= 0; i<getTempFront().
            length; i++)
        {
            nodFront[i]= new ArrayList<
                Integer>();
        }

        //En este ejemplo en particular se
            simula la lectura de los nodos
        //que pertenecen a cada frontera por
            medio de una rutina que define
        //y asigna las fronteras según los
            criterios particulares del
            ejemplo.
```

```
//Recorre los nodos de la
    discretización para asignar a
    cada nodo borde
//su correspondiente frontera
for (Nodo nodoActual: this.
    discretizacion.nodo)
{
    //Verifica que se trate de un
        nodo borde
    if(nodoActual instanceof
        NodoBorde)
    {


        //Se realiza una conversión
            descendente al dominio
        //(De un Domino a un
            DominioRectangular
        DominioRectangular dominio2d
            =
           (DominioRectangular) this.
               discretizacion.dominio
               ;

        //Se realiza una conversión
            descendente al nodo
            actual
        //(De un Nodo a un NodoBorde
        NodoBorde nodoBorde = (
            NodoBorde)nodoActual;

        //Frontera 0
        //Asigna la frontera inferior
        if(nodoBorde.coordenada[1]==
            dominio2d.getZ1())
        {
            nodFront[0].add(
                nodoActual.idNodo);
        }

        //Frontera 1
        //Asigna la frontera derecha
        if(nodoBorde.coordenada[0]==
            dominio2d.getX2())
        {
            nodFront[1].add(
                nodoActual.idNodo);
        }

        //Frontera 2
        //Asigna la frontera superior
        if(nodoBorde.coordenada[1]==
            dominio2d.getZ2())
        {
            nodFront[2].add(
                nodoActual.idNodo);
        }

        //Frontera 3
        //Asigna la frontera
            izquierda
        if(nodoBorde.coordenada[0]==
            dominio2d.getX1())
        {
            nodFront[3].add(
                nodoActual.idNodo);
        }
      }
    }
    return nodFront;
}

/**
 * Obtiene las fronteras que definen el
     modelo de flujo de masa en un
 * problema de convección.
 * @return  Un arreglo de Frontera que
     contiene las fronteras que definen
 *          el modelo de flujo de masa.
 */
public Frontera[] getStreamFront()
{
    //Variable usada para definir el id
        de la frontera
    int idFrontCont = 0;

    //Inicializa el arreglo de Fronteras,
         las cuales en este caso serán
    //cuatro.
```

```
        Frontera[] fronteraArg = new Frontera
            [4];

        //Incializa el Vector que servirá
            para definir la normal a cada
        //frontera
        AlgebraLineal.Vector normFront =  new
             AlgebraLineal.Vector(2);

        //Frontera 0
        //Frontera Dirichlet correspondiente
            al borde inferior del dominio
        normFront.AsignaValor(0, 0);
        normFront.AsignaValor(1, -1);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        //Frontera 1
        //Frontera Dirichlet correspondiente
            al borde derecho del dominio
        normFront.AsignaValor(0, 1);
        normFront.AsignaValor(1, 0);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        //Frontera 2
        //Frontera Dirichlet correspondiente
            al borde superior del dominio
        normFront.AsignaValor(0, 0);
        normFront.AsignaValor(1, 1);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        //Frontera 3
        //Frontera Dirichlet correspondiente
            al borde izquierdo del dominio
        normFront.AsignaValor(0, -1);
        normFront.AsignaValor(1, 0);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        return fronteraArg;
    }


    /**
     * Obtiene la asignación de los nodos
         borde que pertenecen a cada una
     * de las fronteras en particular del
         modelo de flujo de masa en un
     * problema de convección.
     * @return  Un arreglo de objetos List
         que contienen las listas de los
     *          índices de los nodos que
         pertenecen a cada una de las
     *          fronteras del problema.
     */
    public List <Integer> []
        getListStreamNodFront()
    {
        ////////////////////////////////
        //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> [] nodFront = new
            ArrayList[getTempFront().length];

        //Inicializa las listas que contendrá
            n los índices de los nodos que
        //pertenecen a éstas
        for (int i= 0; i<getTempFront().
            length; i++)
        {
            nodFront[i]= new ArrayList <
                Integer >();
        }

        //En este ejemplo en particular se
            simula la lectura de los nodos
```

```
//que pertenecen a cada frontera por
    medio de una rutina que define
//y asigna las fronteras según los
    criterios particulares del
    ejemplo.

//Recorre los nodos de la
    discretización para asignar a
    cada nodo borde
//su correspondiente frontera
for (Nodo nodoActual: this.
    discretizacion.nodo)
{
    //Verifica que se trate de un
        nodo borde
    if(nodoActual instanceof
        NodoBorde)
    {


        //Se realiza una conversión
            descendente al dominio
        //(De un Domino a un
            DominioRectangular
        DominioRectangular dominio2d
            =
            (DominioRectangular) this
                .discretizacion.
                dominio;

        //Se realiza una conversión
            descendente al nodo
            actual
        //(De un Nodo a un NodoBorde
        NodoBorde nodoBorde = (
            NodoBorde)nodoActual;

        //Frontera 0
        //Asigna la frontera inferior
        if(nodoBorde.coordenada[1]==
            dominio2d.getZ1())
        {
            nodFront[0].add(
                nodoActual.idNodo);
        }

        //Frontera 1
        //Asigna la frontera derecha
        if(nodoBorde.coordenada[0]==
            dominio2d.getX2())
        {
            nodFront[1].add(
                nodoActual.idNodo);
        }

        //Frontera 2
        //Asigna la frontera superior
        if(nodoBorde.coordenada[1]==
            dominio2d.getZ2())
        {
            nodFront[2].add(
                nodoActual.idNodo);
        }

        //Frontera 3
        //Asigna la frontera
            izquierda
        if(nodoBorde.coordenada[0]==
            dominio2d.getX1())
        {
            nodFront[3].add(
                nodoActual.idNodo);
        }
    }
  }
  return nodFront;
}

/**
 * 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.
 */
private double temRealtoAdim(double
    TRealArg)
{
```

```
        //return (TRealArg-TRefReal)/(T2Real-
            T1Real);
        return (TRealArg-this.getTRef())/(
            this.getT2()-this.getT1());
    }

}
```

C.3 CapInc_CSFlujoCal_AMN.java

```
package ModeloConveccion;

import AlgebraLineal.Matriz;
import AlgebraLineal.MatrizDensa;
import FEM.FuncionExpresion;
import FEM.Frontera;
import FEM.TriangulacionDominioRectangular;
import FEM.Nodo;
import FEM.NodoBorde;
import FEM.DominioRectangular;
import java.util.List;
import java.util.ArrayList;

/**
 * Clase usada para probar la interfaz
     DefinicionModeloConveccion2D. En esta
 * clase se simulará solamente la obtención
     de los parámetros del modelo de
 * convección.
 * @author Conrado
 * @version 1.0
 * @since 1.0
 * Modificado por Alma MN
 */
public class CapaInclinada1sinflujocalor_AMN
    extends DefinicionModelo
        implements
            DefinicionModeloConveccion2D {

    //Objeros usados para obtener la
        información que permite simular la
    //"lectura" de información de un modo
        externo
    private DominioRectangular dominio;
    private TriangulacionDominioRectangular
        discretizacion;

    public CapaInclinada1sinflujocalor_AMN()
    {
        //En este caso, como se está
            simulando la lectura de la
            información
        //de un medio externo, en esta rutina
             se crea una discretización igual
        //a la que se creo con la información
             de esta "interfaz" para poder
        //simular la lectura de la informació
            n en los nodos.

        //Define el dominio que se usará en
            el modelo. En este caso se trata
        //de un dominio rectangular.
        this.dominio = new DominioRectangular
            ();
        this.dominio.defineDominioRectangular
            (this.getX1(), this.getZ1(),
                this.getX2(), this.getZ2());

        //Define la la discretización usada
            para la ecuación de flujo de
            calor.
        this.discretizacion=new
          TriangulacionDominioRectangular(
              dominio,this.getNx(),this.getNz
              ());

    }

    /**
     * Obtiene el mínimo valor en la direcció
         n X del dominio rectangular.
     * @return
     */
    public double getX1()
    {
        return 0.0;
    }
```

```
    /**
     * Obtiene el máximo valor en la direcció
         n X del dominio rectangular.
     * @return
     */
    public double getX2()
    {
        //return 450.0;
        return 300.0;
    }

    /**
     * Obtiene el mínimo valor en la direcció
         n Z del dominio rectangular.
     * @return
     */
    public double getZ1()
    {
        return 0.0;
    }

    /**
     * Obtiene el máximo valor en la direcció
         n Z del dominio rectangular.
     * @return
     */
    public double getZ2()
    {
        //return 150.0;
        return 100.0;
    }

    /**
     * Obtiene la altura del dominio
         rectangular.
     * @return
     */
    public double getH()
    {
        return this.getZ2()-this.getZ1();
    }

    /**
     * Obtiene el ancho del dominio
         rectangular.
     * @return
     */
    public double getL()
    {
        return this.getX2()-this.getX1();
    }

    /**
     * Obtiene el 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.
     * @return
     */
    public int getNx()
    {
        return 50;
    }

    /**
     * Obtiene el 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.
     * @return
     */
    public int getNz()
    {
        return 50;
    }

    /**
     * Obtiene el coeficiente de relajación
         usado en Crank Nicholson.
     * @return
     */
    public double getTheta()
    {
        return 0.8;
```

```
    }

    /**
     * Obtiene el salto en el tiempo (Dt)
         usado en Crank Nicholson.
     * @return
     */
    public double getDt()
    {
        return 0.01;
    }

    /**
     * Obtiene el valor de la temperatura
         superficie.
     * @return
     */
    public double getT1()
    {
        //return 10;
        return 120;
    }

    /**
     * Obtiene el valor de la temperatura en
         la base.
     * @return
     */
    public double getT2()
    {
        //return 40;
        return 150;
    }

    /**
     * Obtiene el valor de la temperatura de
         referencia.
     * @return
     */
    public double getTRef()
    {
        return (getT1()+getT2())/2.0;
    }

    /**
     * Obtiene el valor del tensor simétrico
         de conductividad térmica
     * anisotrópica en un  nodo de la
         triangulación.
     * @param iNodo Índice del nodo de la
         triangulación.
     * @return      Una matriz con el valor
         del tensor simétrico de
     *              conductividad térmica
         anisotrópica.
     */
    public Matriz getLambAst(int iNodo)
    {
        //Se simula la obtención del valor
            del tensor.
        MatrizDensa mat = new MatrizDensa
            (2,2);

        //Medio con una conductividad térmica
             homogénea

        //En este caso particular se trata de
             un medio homogéneo isotrópico.
        mat.asignaValor(0, 0, 2.7);
        mat.asignaValor(1, 1, 2.7);
        mat.asignaValor(0, 1, 0.0);
        mat.asignaValor(1, 0, 0.0);

        return mat;
    }

    /**
     * Obtiene el valor del tensor simétrico
         de permeabilidad
     * anisotrópica en un  nodo de la
         discretización.
     * @param iNodo Índice del nodo de la
         discretización
     * @return      Una matriz con el valor
         del tensor simétrico de
     *              permeabilidad anisotró
         pica.
     */
    public Matriz getKAst(int iNodo)
    {
```

```
        //Se simula la obtención del valor
            del tensor.
        MatrizDensa mat = new MatrizDensa
            (2,2);

        //Capa inclinada
        if(this.discretizacion.nodo[iNodo].
            coordenada[1]>=
                ((0.15*this.discretizacion.
                    nodo[iNodo].coordenada
                    [0]+10))
          && this.discretizacion.nodo[iNodo].
              coordenada[1]<(
                (0.15*this.discretizacion.
                    nodo[iNodo].coordenada
                    [0])+30))
        {
            //En este caso particular se
                trata de un medio homogéneo
                isotrópico.
            //mat.asignaValor(0, 0,
                0.00000000001);
            //mat.asignaValor(1, 1,
                0.00000000001);
            mat.asignaValor(0, 0,
                0.00000000000127);
            mat.asignaValor(1, 1,
                0.00000000000127);
            mat.asignaValor(0, 1, 0.0);
            mat.asignaValor(1, 0, 0.0);
        }

        //Medio impermeable
        else
        {
            //mat.asignaValor(0, 0,
                0.0000000000000000001);
            //mat.asignaValor(1, 1,
                0.0000000000000000001);
            mat.asignaValor(0, 0,
                0.0000000000000001);
            mat.asignaValor(1, 1,
                0.0000000000000001);
            mat.asignaValor(0, 1, 0.0);
            mat.asignaValor(1, 0, 0.0);
        }

        return mat;
    }

    /**
     * Obtiene el valor de la producción de
         calor en un  nodo
     * de la discretización.
     * @param iNodo Índice del nodo de la
         discretización
     * @return      El valor de la producción
          de calor en el nodo.
     */
    public double getAAst(int iNodo)
    {
        //En este caso particular no hay
            producción de calor.
        return 0.0;
    }

    /**
     * Obtiene el valor de la capacidad calor
         ífica del fluido.
     * @return
     */
    public double getRhoc()
    {
        return 4180000;
    }

    /**
     * Obtiene el valor de la densidad media
         del fluido.
     * @return
     */
    public double getRho()
    {
        return 968;
    }

    /**
     * Obtiene el valor de la viscosidad diná
         mica del fluido a la temperatura
     * de referencia.
```

```
     * @return
     */
    public double getNeta()
    {
        return 0.00018;
    }

    /**
     * Obtiene el valor del coeficiente de
         dilatación térmica del fluido.
     * @return
     */
    public double getBeta()
    {
        return 0.0008275;
    }

    /**
     * Obtiene el valor de la aceleración de
         la gravedad.
     * @return
     */
    public double getG()
    {
        return 9.81;
    }

    /**
     * Obtiene el valor del coeficiente de
         dependencia de la viscosidad
     * dinámica del fluido con la temperatura
         .
     * @return
     */
    public double getGamma()
    {
        return -0.4;
    }

    /**
     * Obtiene el valor de la diferencia má
         xima que debe de existir entre dos
     * valores consecutivos de la solución de
          Temperatura dentro del ciclo
     * iterativo de acoplamiento con Stream.
     * @return
     */
    public double getDifTempMax()
    {
        return 0.001;
    }

    /**
     * Obtiene el valor de la diferencia má
         xima que debe de existir entre dos
     * valores consecutivos de la solución de
          Stream dentro del ciclo
     * iterativo de acoplamiento con
         Temperatura.
     * @return
     */
    public double getDifStreamMax()
    {
        return 0.1;
    }

    /**
     * Obtiene el número máximo de
         iteraciones permitido dentro del
         ciclo
     * usado para acoplarlas soluciones de
         Stream y Temperatura.
     * @return
     */
    public int getMaxIt()
    {
        return 100;
    }

    /**
     * Obtiene la FuncionExpresión que define
          las condiciones iniciales de
     * temperatura del problema de conveccion
         .
     * @return  Una FuncionExpresion que
         define las condiciones iniciales de
     *          temperatura del problema de
         conveccion.
     */
    public FuncionExpresion getTempCondIni()
```

```
{
    //Inicializa y asigna la función
        correspondiente a las C.I.
    FuncionExpresion condIniArg =

            //new FuncionExpresion("-cos(
                pi*z)/2*cos(x*pi)");
            //new FuncionExpresion("-cos(
                pi*z)/2*cos(x*pi/15.0)");

            //new FuncionExpresion("sin(z
                *pi-pi/2)/2");

            //new FuncionExpresion("sin(z
                *pi-pi/24)/2");

            //new FuncionExpresion("0");

            //new FuncionExpresion("-cos(
                pi*z)*cos(x*pi*4)/2");

            //-pi/2

            new FuncionExpresion("cos(z*
                pi*4+pi/2)*cos(x*pi*5+pi
                /2)/4");

            //new FuncionExpresion("0.5+
                cos(0.349*(100-1))*sin
                (0.349*(100-1))+((100-1)
                *1/(100-1))");

    return condIniArg;
}

/**
 * Obtiene las fronteras que definen el
     modelo de flujo de calor en un
 * problema de convección.
 * @return  Un arreglo de Frontera que
     contiene las fronteras que definen
 *          el modelo de flujo de calor.
 */
public Frontera[] getTempFront()
{
    //Variable usada para definir el id
        de la frontera
    int idFrontCont = 0;

    //Inicializa el arreglo de Fronteras,
         las cuales en este caso serán
    //cuatro.
    Frontera[] fronteraArg = new Frontera
        [4];

    //Incializa el Vector que servirá
        para definir la normal a cada
    //frontera
    AlgebraLineal.Vector normFront =  new
         AlgebraLineal.Vector(2);

    //Frontera 0
    //Frontera Dirichlet correspondiente
        al borde inferior del dominio
    normFront.AsignaValor(0, 0);
    normFront.AsignaValor(1, -1);
    fronteraArg[idFrontCont] = new
        Frontera(true, idFrontCont++,
            String.valueOf(this.
                temRealtoAdim(this.getT2
                ())), normFront);

    //Frontera 1
    //Frontera Neumann correspondiente al
         borde derecho del dominio
    normFront.AsignaValor(0, 1);
    normFront.AsignaValor(1, 0);
    fronteraArg[idFrontCont] = new
        Frontera(false, idFrontCont++,
            "0", normFront);

    //Frontera 2
    //Frontera Dirichlet correspondiente
        al borde superior del dominio
    normFront.AsignaValor(0, 0);
    normFront.AsignaValor(1, 1);
    fronteraArg[idFrontCont] = new
        Frontera(true, idFrontCont++,
            String.valueOf(this.
                temRealtoAdim(this.getT1
```

```
                ())), normFront);

    //Frontera 3
    //Frontera Neumann correspondiente al
        borde izquierdo del dominio
    normFront.AsignaValor(0, -1);
    normFront.AsignaValor(1, 0);
    fronteraArg[idFrontCont] = new
        Frontera(false, idFrontCont++,
            "0", normFront);

    return fronteraArg;
}


/**
 * Obtiene la asignación de los nodos
     borde que pertenecen a cada una
 * de las fronteras en particular del
     modelo de flujo de calor en un
 * problema de convección.
 * @return  Un arreglo de objetos List
     que contienen las listas de los
 *          índices de los nodos que
     pertenecen a cada una de las
 *          fronteras del problema.
 */
public List<Integer> []
    getListTempNodFront()
{
    ////////////////////////////////
    //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> [] nodFront = new
        ArrayList[getTempFront().length];

    //Inicializa las listas que contendrá
        n los índices de los nodos que
    //pertenecen a éstas
    for (int i= 0; i<getTempFront().
        length; i++)
    {
        nodFront[i]= new ArrayList<
            Integer>();
    }

    //En este ejemplo en particular se
        simula la lectura de los nodos
    //que pertenecen a cada frontera por
        medio de una rutina que define
    //y asigna las fronteras según los
        criterios particulares del
        ejemplo.

    //Recorre los nodos de la
        discretización para asignar a
        cada nodo borde
    //su correspondiente frontera
    for (Nodo nodoActual: this.
        discretizacion.nodo)
    {
        //Verifica que se trate de un
            nodo borde
        if(nodoActual instanceof
            NodoBorde)
        {


            //Se realiza una conversión
                descendente al dominio
            //(De un Domino a un
                DominioRectangular
            DominioRectangular dominio2d
                =
                (DominioRectangular) this.
                    discretizacion.dominio
                    ;

            //Se realiza una conversión
                descendente al nodo
                actual
            //(De un Nodo a un NodoBorde
```

```
                NodoBorde nodoBorde = (
                    NodoBorde)nodoActual;

                //Frontera 0
                //Asigna la frontera inferior
                if(nodoBorde.coordenada[1]==
                    dominio2d.getZ1())
                {
                    nodFront[0].add(
                        nodoActual.idNodo);
                }

                //Frontera 1
                //Asigna la frontera derecha
                if(nodoBorde.coordenada[0]==
                    dominio2d.getX2())
                {
                    nodFront[1].add(
                        nodoActual.idNodo);
                }

                //Frontera 2
                //Asigna la frontera superior
                if(nodoBorde.coordenada[1]==
                    dominio2d.getZ2())
                {
                    nodFront[2].add(
                        nodoActual.idNodo);
                }

                //Frontera 3
                //Asigna la frontera
                    izquierda
                if(nodoBorde.coordenada[0]==
                    dominio2d.getX1())
                {
                    nodFront[3].add(
                        nodoActual.idNodo);
                }
            }
        }
        return nodFront;
    }

    /**
     * Obtiene las fronteras que definen el
         modelo de flujo de masa en un
     * problema de convección.
     * @return  Un arreglo de Frontera que
         contiene las fronteras que definen
     *          el modelo de flujo de masa.
     */
    public Frontera[] getStreamFront()
    {
        //Variable usada para definir el id
            de la frontera
        int idFrontCont = 0;

        //Inicializa el arreglo de Fronteras,
             las cuales en este caso serán
        //cuatro.
        Frontera[] fronteraArg = new Frontera
            [4];

        //Incializa el Vector que servirá
            para definir la normal a cada
        //frontera
        AlgebraLineal.Vector normFront =  new
             AlgebraLineal.Vector(2);

        //Frontera 0
        //Frontera Dirichlet correspondiente
            al borde inferior del dominio
        normFront.AsignaValor(0, 0);
        normFront.AsignaValor(1, -1);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        //Frontera 1
        //Frontera Neumann correspondiente al
             borde derecho del dominio
        normFront.AsignaValor(0, 1);
        normFront.AsignaValor(1, 0);
        //fronteraArg[idFrontCont] = new
            Frontera(false, idFrontCont++,
          //        "0.0000000001", normFront);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);
```

```
        //Frontera 2
        //Frontera Dirichlet correspondiente
            al borde superior del dominio
        normFront.AsignaValor(0, 0);
        normFront.AsignaValor(1, 1);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        //Frontera 3
        //Frontera Neumann correspondiente al
             borde izquierdo del dominio
        normFront.AsignaValor(0, -1);
        normFront.AsignaValor(1, 0);
        //fronteraArg[idFrontCont] = new
            Frontera(false, idFrontCont++,
          //        "-0.0000000001", normFront)
              ;
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        return fronteraArg;
    }


    /**
     * Obtiene la asignación de los nodos
         borde que pertenecen a cada una
     * de las fronteras en particular del
         modelo de flujo de masa en un
     * problema de convección.
     * @return  Un arreglo de objetos List
         que contienen las listas de los
     *          índices de los nodos que
         pertenecen a cada una de las
     *          fronteras del problema.
     */
    public List<Integer> []
        getListStreamNodFront()
    {
        ////////////////////////////////
        //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> [] nodFront = new
            ArrayList[getTempFront().length];

        //Inicializa las listas que contendrá
            n los índices de los nodos que
        //pertenecen a éstas
        for (int i= 0; i<getTempFront().
            length; i++)
        {
            nodFront[i]= new ArrayList<
                Integer>();
        }

        //En este ejemplo en particular se
            simula la lectura de los nodos
        //que pertenecen a cada frontera por
            medio de una rutina que define
        //y asigna las fronteras según los
            criterios particulares del
            ejemplo.

        //Recorre los nodos de la
            discretización para asignar a
            cada nodo borde
        //su correspondiente frontera
        for (Nodo nodoActual: this.
            discretizacion.nodo)
        {
            //Verifica que se trate de un
                nodo borde
            if(nodoActual instanceof
                NodoBorde)
            {


                //Se realiza una conversión
                    descendente al dominio
```

```
//(De un Domino a un
    DominioRectangular
DominioRectangular dominio2d
    =
   (DominioRectangular) this.
       discretizacion.dominio
       ;

//Se realiza una conversión
    descendente al nodo
    actual
//(De un Nodo a un NodoBorde
NodoBorde nodoBorde = (
    NodoBorde)nodoActual;

//Frontera 0
//Asigna la frontera inferior
if(nodoBorde.coordenada[1]==
    dominio2d.getZ1())
{
    nodFront[0].add(
        nodoActual.idNodo);
}

//Frontera 1
//Asigna la frontera derecha
if(nodoBorde.coordenada[0]==
    dominio2d.getX2())
{
    nodFront[1].add(
        nodoActual.idNodo);
}

//Frontera 2
//Asigna la frontera superior
if(nodoBorde.coordenada[1]==
    dominio2d.getZ2())
{
    nodFront[2].add(
        nodoActual.idNodo);
}

//Frontera 3
//Asigna la frontera
    izquierda
                if(nodoBorde.coordenada[0]==
                    dominio2d.getX1())
                {
                    nodFront[3].add(
                        nodoActual.idNodo);
                }
            }
        }
        return nodFront;
    }

    /**
     * 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.
     */
    private double temRealtoAdim(double
        TRealArg)
    {
        //return (TRealArg-TRefReal)/(T2Real-
            T1Real);
        return (TRealArg-this.getTRef())/(
            this.getT2()-this.getT1());
    }

}
```

C.4 Capa2CapInc_AMN.java

```
package ModeloConveccion;

import AlgebraLineal.Matriz;
import AlgebraLineal.MatrizDensa;
import FEM.FuncionExpresion;
import FEM.Frontera;
import FEM.TriangulacionDominioRectangular;
import FEM.Nodo;
import FEM.NodoBorde;
import FEM.DominioRectangular;
```

```
import java.util.List;
import java.util.ArrayList;

/**
 * Clase usada para los ensayos con dos capas
      horizontales y una capa inclinada
 * @author Alma MN
 */
public class Capa2CapInc_AMN extends
    DefinicionModelo
        implements
            DefinicionModeloConveccion2D {

    //Objetos usados para obtener la
        información que permite simular la
    //"lectura" de información de un modo
        externo
    private DominioRectangular dominio;
    private TriangulacionDominioRectangular
        discretizacion;

    public Capa2CapInc_AMN()
    {
        //En este caso, como se está
            simulando la lectura de la
            información
        //de un medio externo, en esta rutina
             se crea una discretización igual
        //a la que se creo con la información
             de esta "interfaz" para poder
        //simular la lectura de la informació
            n en los nodos.

        //Define el dominio que se usará en
            el modelo. En este caso se trata
        //de un dominio rectangular,
            horizontal o vertical
        this.dominio = new DominioRectangular
            ();
        this.dominio.defineDominioRectangular
            (this.getX1(), this.getZ1(),
                this.getX2(), this.getZ2());

        //Define la discretización usada para
             la ecuación de flujo de calor.
        this.discretizacion=new
         TriangulacionDominioRectangular(
             dominio, this.getNx(), this.getNz
             ());

    }

    /**
     * Obtiene el mínimo valor en la direcció
         n X del dominio rectangular.
     * @return
     */
    public double getX1()
    {
        return 0.0;
    }

    /**
     * Obtiene el máximo valor en la direcció
         n X del dominio rectangular.
     * @return
     */
    public double getX2()
    {
        return 300.0;
    }

    /**
     * Obtiene el mínimo valor en la direcció
         n Z del dominio rectangular.
     * @return
     */
    public double getZ1()
    {
        return 0.0;
    }

    /**
     * Obtiene el máximo valor en la direcció
         n Z del dominio rectangular.
     * @return
     */
    public double getZ2()
    {
        return 100.0;
```

```
}

/**
 * Obtiene la altura del dominio
     rectangular.
 * @return
 */
public double getH()
{
    return this.getZ2()-this.getZ1();
}

/**
 * Obtiene el ancho del dominio
     rectangular.
 * @return
 */
public double getL()
{
    return this.getX2()-this.getX1();
}

/**
 * Obtiene el 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.
 * @return
 */
public int getNx()
{
    return 50;//30
}

/**
 * Obtiene el 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.
 * @return
 */
public int getNz()
{
    return 50;//120
}

/**
 * Obtiene el coeficiente de relajación
     usado en Crank Nicholson.
 * @return
 */
public double getTheta()
{
    return 0.8;
}

/**
 * Obtiene el salto en el tiempo (Dt)
     usado en Crank Nicholson.
 * @return
 */
public double getDt()
{
    return 0.1;
}

/**
 * Obtiene el valor de la temperatura
     superior.
 * @return
 */
public double getT1()
{
    return 100;
}

/**
 * Obtiene el valor de la temperatura
     superior.
 * @return
 */
public double getT2()
{
    return 250;
```

```
}

/**
 * Obtiene el valor de la temperatura de
     referencia.
 * @return
 */
public double getTRef()
{
    return (getT1()+getT2())/2.0;
}

/** CONDUCTIVIDAD TÉRMICA
 * 3 zonas con diferente conductividad:
     inferior, superior, cap.inclinada
 *
 * Obtiene el valor del tensor simétrico
     de conductividad térmica
 * anisotrópica en un  nodo de la
     triangulación.
 * @param iNodo Índice del nodo de la
     triangulación.
 * @return      Una matriz con el valor
     del tensor simétrico de
 *              conductividad térmica
     anisotrópica.
 */
 public Matriz getLambAst(int iNodo)
 {
    //Se simula la obtención del valor
        del tensor.
    MatrizDensa mat = new MatrizDensa
        (2,2);

    //Capa inferior
    if(this.discretizacion.nodo[iNodo].
        coordenada[1]<=0.50*this.getH())
    {
        //Capa Inclinada
        if(this.discretizacion.nodo[iNodo
            ].coordenada[1]>=
            ((0.75*this.discretizacion.
                nodo[iNodo].coordenada
                [0])+10)
        && this.discretizacion.nodo[iNodo
            ].coordenada[1]<(
             (0.75*this.discretizacion.
                 nodo[iNodo].coordenada
                 [0])+30))

            {
             mat.asignaValor(0, 0, 5.4);
             mat.asignaValor(1, 1, 5.4);
             mat.asignaValor(0, 1, 0.0);
             mat.asignaValor(1, 0, 0.0);
            }

        // fuera de la capa inclinada;
            pero aún en el estrato
            inferior
        else
            {
             mat.asignaValor(0, 0, 2.8);
             mat.asignaValor(1, 1, 2.8);
             mat.asignaValor(0, 1, 0.0);
             mat.asignaValor(1, 0, 0.0);
            }

}

//Capa superior
else

    {
     //Capa Inclinada
     if(this.discretizacion.nodo[
         iNodo].coordenada[1]>=
         ((0.75*this.discretizacion.
             nodo[iNodo].coordenada
             [0])+10)
     && this.discretizacion.nodo[
         iNodo].coordenada[1]<(
          (0.75*this.discretizacion.
              nodo[iNodo].coordenada
              [0])+30))

         {
          mat.asignaValor(0, 0, 5.4);
          mat.asignaValor(1, 1, 5.4);
```

```
                mat.asignaValor(0, 1, 0.0);
                mat.asignaValor(1, 0, 0.0);
               }

           //Fuera de la capa inclinada;
               pero aún en la capa superior
           else
               {
                mat.asignaValor(0, 0, 2.5);
                mat.asignaValor(1, 1, 2.5);
                mat.asignaValor(0, 1, 0.0);
                mat.asignaValor(1, 0, 0.0);
               }
       }

       return mat;
   }

   /**PERMEABILIDAD
    * 3 zonas con diferente permeabilidad:
       inferior, superior, cap.inclinada
    *
    * Obtiene el valor del tensor simétrico
       de permeabilidad
    * anisotrópica en un  nodo de la
       discretización.
    * @param iNodo Índice del nodo de la
       discretización
    * @return      Una matriz con el valor
       del tensor simétrico de
    *              permeabilidad anisotró
       pica.
    */
   public Matriz getKAst(int iNodo)
   {
       //Se simula la obtención del valor
           del tensor.
       MatrizDensa mat = new MatrizDensa
           (2,2);

       //Capa inferior
       if(this.discretizacion.nodo[iNodo].
           coordenada[1]<=0.50*this.getH())
       {
           //Capa Inclinada
           if(this.discretizacion.nodo[iNodo
               ].coordenada[1]>=
               ((0.75*this.discretizacion.
                   nodo[iNodo].coordenada
                   [0])+10)
           && this.discretizacion.nodo[iNodo
               ].coordenada[1]<(
                (0.75*this.discretizacion.
                    nodo[iNodo].coordenada
                    [0])+30))

               {
                mat.asignaValor(0, 0,
                    0.00000000001);
                mat.asignaValor(1, 1,
                    0.000000000001);
                mat.asignaValor(0, 1, 0.0);
                mat.asignaValor(1, 0, 0.0);
               }

           //Fuera de la capa inclinada,
               pero aún en la capa inferior
           else
               {
                mat.asignaValor(0, 0,
                    0.00000000000000011);
                mat.asignaValor(1, 1,
                    0.00000000000000011);
                mat.asignaValor(0, 1, 0.0);
                mat.asignaValor(1, 0, 0.0);
               }
       }

       //Capa superior
       else
           {

            //Capa Inclinada
            if(this.discretizacion.nodo[
                iNodo].coordenada[1]>=
                ((0.75*this.discretizacion.
                    nodo[iNodo].coordenada
                    [0])+10)
            && this.discretizacion.nodo[
                iNodo].coordenada[1]<(
```

```
                (0.75*this.discretizacion.
                    nodo[iNodo].coordenada
                    [0])+30))

                {
                 mat.asignaValor(0, 0,
                     0.00000000001);
                 mat.asignaValor(1, 1,
                     0.000000000001);
                 mat.asignaValor(0, 1, 0.0);
                 mat.asignaValor(1, 0, 0.0);
                }

            //Fuera de la capa inclinada,
                pero aún en la capa superior
            else
                {
                 mat.asignaValor(0, 0,
                     0.000000000000011);
                 mat.asignaValor(1, 1,
                     0.000000000000011);
                 mat.asignaValor(0, 1, 0.0);
                 mat.asignaValor(1, 0, 0.0);
                }
        }

    return mat;
}

/**
 * Obtiene el valor de la producción de
     calor en un  nodo
 * de la discretización.
 * @param iNodo Índice del nodo de la
     discretización
 * @return      El valor de la producción
      de calor en el nodo.
 */
public double getAAst(int iNodo)
{
    //En este caso particular no hay
        producción de calor.
    return 0.0;
}

/**
 * Obtiene el valor de la capacidad calor
     ífica del fluido.
 * @return
 */
public double getRhoc()
{
    return 4180000;
}

/**
 * Obtiene el valor de la densidad media
     del fluido.
 * @return
 */
public double getRho()
{
    return 968;
}

/**
 * Obtiene el valor de la viscosidad diná
     mica del fluido a la temperatura
 * de referencia.
 * @return
 */
public double getNeta()
{
    return 0.00018;
}

/**
 * Obtiene el valor del coeficiente de
     dilatación térmica del fluido.
 * @return
 */
public double getBeta()
{
    return 0.0008275;
}

/**
 * Obtiene el valor de la aceleración de
     la gravedad.
 * @return
```

```
     */
    public double getG()
    {
        return 9.81;
    }

    /**
     * Obtiene el valor del coeficiente de
         dependencia de la viscosidad
     * dinámica del fluido con la temperatura
         .
     * @return
     */
    public double getGamma()
    {
        return 0.0;
    }

    /**
     * Obtiene el valor de la diferencia má
         xima que debe de existir entre dos
     * valores consecutivos de la solución de
          Temperatura dentro del ciclo
     * iterativo de acoplamiento con Stream.
     * @return
     */
    public double getDifTempMax()
    {
        return 0.001;
    }

    /**
     * Obtiene el valor de la diferencia má
         xima que debe de existir entre dos
     * valores consecutivos de la solución de
          Stream dentro del ciclo
     * iterativo de acoplamiento con
         Temperatura.
     * @return
     */
    public double getDifStreamMax()
    {
        return 0.1;
    }

    /**
     * Obtiene el número máximo de
         iteraciones permitido dentro del
         ciclo
     * usado para acoplarlas soluciones de
         Stream y Temperatura.
     * @return
     */
    public int getMaxIt()
    {
        return 100;
    }

    /**
     * Obtiene la FuncionExpresión que define
          las condiciones iniciales de
     * temperatura del problema de conveccion
         .
     * @return  Una FuncionExpresion que
         define las condiciones iniciales de
     *          temperatura del problema de
         conveccion.
     */
    public FuncionExpresion getTempCondIni()
    {
        //Inicializa y asigna la función
            correspondiente a las C.I.
        FuncionExpresion condIniArg =

                new FuncionExpresion("-cos(pi
                    *z)/2*cos(x*pi)");

        return condIniArg;
    }

    /**
     * Obtiene las fronteras que definen el
         modelo de flujo de calor en un
     * problema de convección.
     * @return  Un arreglo de Frontera que
         contiene las fronteras que definen
     *          el modelo de flujo de calor.
     */
    public Frontera[] getTempFront()
    {
```

```
        //Variable usada para definir el id
            de la frontera
        int idFrontCont = 0;

        //Inicializa el arreglo de Fronteras,
             las cuales en este caso serán
        //cuatro.
        Frontera[] fronteraArg = new Frontera
            [4];

        //Incializa el Vector que servirá
            para definir la normal a cada
        //frontera
        AlgebraLineal.Vector normFront =  new
             AlgebraLineal.Vector(2);

        //Frontera 0
        //Frontera Dirichlet correspondiente
            al borde inferior del dominio
        normFront.AsignaValor(0, 0);
        normFront.AsignaValor(1, -1);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                String.valueOf(this.
                    temRealtoAdim(this.getT2
                    ())), normFront);

        //Frontera 1
        //Frontera Neumann correspondiente al
             borde derecho del dominio
        normFront.AsignaValor(0, 1);
        normFront.AsignaValor(1, 0);
        fronteraArg[idFrontCont] = new
            Frontera(false, idFrontCont++,
                "0", normFront);

        //Frontera 2
        //Frontera Dirichlet correspondiente
            al borde superior del dominio
        normFront.AsignaValor(0, 0);
        normFront.AsignaValor(1, 1);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                String.valueOf(this.
                    temRealtoAdim(this.getT1
                    ())), normFront);

        //Frontera 3
        //Frontera Neumann correspondiente al
             borde izquierdo del dominio
        normFront.AsignaValor(0, -1);
        normFront.AsignaValor(1, 0);
        fronteraArg[idFrontCont] = new
            Frontera(false, idFrontCont++,
                "0", normFront);

        return fronteraArg;
    }


    /**
     * Obtiene la asignación de los nodos
         borde que pertenecen a cada una
     * de las fronteras en particular del
         modelo de flujo de calor en un
     * problema de convección.
     * @return  Un arreglo de objetos List
         que contienen las listas de los
     *          índices de los nodos que
         pertenecen a cada una de las
     *          fronteras del problema.
     */
    public List<Integer> []
        getListTempNodFront()
    {
        ////////////////////////////////
        //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> [] nodFront = new
            ArrayList[getTempFront().length];

        //Inicializa las listas que contendrá
            n los índices de los nodos que
```

```
        //pertenecen a éstas
        for (int i= 0; i<getTempFront().
            length; i++)
        {
            nodFront[i]= new ArrayList <
                Integer >();
        }

        //En este ejemplo en particular se
            simula la lectura de los nodos
        //que pertenecen a cada frontera por
            medio de una rutina que define
        //y asigna las fronteras según los
            criterios particulares del
            ejemplo.

        //Recorre los nodos de la
            discretización para asignar a
            cada nodo borde
        //su correspondiente frontera
        for (Nodo nodoActual: this.
            discretizacion.nodo)
        {
            //Verifica que se trate de un
                nodo borde
            if(nodoActual instanceof
                NodoBorde)
            {


                //Se realiza una conversión
                    descendente al dominio
                //(De un Domino a un
                    DominioRectangular
                DominioRectangular dominio2d
                    =
                    (DominioRectangular) this.
                        discretizacion.dominio
                        ;

                //Se realiza una conversión
                    descendente al nodo
                    actual
                //(De un Nodo a un NodoBorde
                NodoBorde nodoBorde = (
                    NodoBorde)nodoActual;

                //Frontera 0
                //Asigna la frontera inferior
                if(nodoBorde.coordenada[1]==
                    dominio2d.getZ1())
                {
                    nodFront[0].add(
                        nodoActual.idNodo);
                }

                //Frontera 1
                //Asigna la frontera derecha
                if(nodoBorde.coordenada[0]==
                    dominio2d.getX2())
                {
                    nodFront[1].add(
                        nodoActual.idNodo);
                }

                //Frontera 2
                //Asigna la frontera superior
                if(nodoBorde.coordenada[1]==
                    dominio2d.getZ2())
                {
                    nodFront[2].add(
                        nodoActual.idNodo);
                }

                //Frontera 3
                //Asigna la frontera
                    izquierda
                if(nodoBorde.coordenada[0]==
                    dominio2d.getX1())
                {
                    nodFront[3].add(
                        nodoActual.idNodo);
                }
            }
        }
        return nodFront;
    }

    /**
```

```
     * Obtiene las fronteras que definen el
         modelo de flujo de masa en un
     * problema de convección.
     * @return  Un arreglo de Frontera que
         contiene las fronteras que definen
     *          el modelo de flujo de masa.
     */
    public Frontera[] getStreamFront()
    {
        //Variable usada para definir el id
            de la frontera
        int idFrontCont = 0;

        //Inicializa el arreglo de Fronteras,
             las cuales en este caso serán
        //cuatro.
        Frontera[] fronteraArg = new Frontera
            [4];

        //Incializa el Vector que servirá
            para definir la normal a cada
        //frontera
        AlgebraLineal.Vector normFront =  new
             AlgebraLineal.Vector(2);

        //Frontera 0
        //Frontera Dirichlet correspondiente
            al borde inferior del dominio
        normFront.AsignaValor(0, 0);
        normFront.AsignaValor(1, -1);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        //Frontera 1
        //Frontera Dirichlet correspondiente
            al borde derecho del dominio
        normFront.AsignaValor(0, 1);
        normFront.AsignaValor(1, 0);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        //Frontera 2
        //Frontera Dirichlet correspondiente
            al borde superior del dominio
        normFront.AsignaValor(0, 0);
        normFront.AsignaValor(1, 1);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        //Frontera 3
        //Frontera Dirichlet correspondiente
            al borde izquierdo del dominio
        normFront.AsignaValor(0, -1);
        normFront.AsignaValor(1, 0);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        return fronteraArg;
    }


    /**
     * Obtiene la asignación de los nodos
         borde que pertenecen a cada una
     * de las fronteras en particular del
         modelo de flujo de masa en un
     * problema de convección.
     * @return  Un arreglo de objetos List
         que contienen las listas de los
     *          índices de los nodos que
         pertenecen a cada una de las
     *          fronteras del problema.
     */
    public List <Integer> []
        getListStreamNodFront()
    {
        ////////////////////////////////
        //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> [] nodFront = new
    ArrayList[getTempFront().length];

//Inicializa las listas que contendrá
    n los índices de los nodos que
//pertenecen a éstas
for (int i= 0; i<getTempFront().
    length; i++)
{
    nodFront[i]= new ArrayList <
        Integer >();
}

//En este ejemplo en particular se
    simula la lectura de los nodos
//que pertenecen a cada frontera por
    medio de una rutina que define
//y asigna las fronteras según los
    criterios particulares del
    ejemplo.

//Recorre los nodos de la
    discretización para asignar a
    cada nodo borde
//su correspondiente frontera
for (Nodo nodoActual: this.
    discretizacion.nodo)
{
    //Verifica que se trate de un
        nodo borde
    if(nodoActual instanceof
        NodoBorde)
    {



        //Se realiza una conversión
            descendente al dominio
        //(De un Domino a un
            DominioRectangular
        DominioRectangular dominio2d
            =
           (DominioRectangular) this.
               discretizacion.dominio
               ;


        //Se realiza una conversión
            descendente al nodo
            actual
        //(De un Nodo a un NodoBorde
        NodoBorde nodoBorde = (
            NodoBorde)nodoActual;

        //Frontera 0
        //Asigna la frontera inferior
        if(nodoBorde.coordenada[1]==
            dominio2d.getZ1())
        {
            nodFront[0].add(
                nodoActual.idNodo);
        }

        //Frontera 1
        //Asigna la frontera derecha
        if(nodoBorde.coordenada[0]==
            dominio2d.getX2())
        {
            nodFront[1].add(
                nodoActual.idNodo);
        }

        //Frontera 2
        //Asigna la frontera superior
        if(nodoBorde.coordenada[1]==
            dominio2d.getZ2())
        {
            nodFront[2].add(
                nodoActual.idNodo);
        }

        //Frontera 3
        //Asigna la frontera
            izquierda
        if(nodoBorde.coordenada[0]==
            dominio2d.getX1())
        {
            nodFront[3].add(
                nodoActual.idNodo);
        }
```

```
            }
        }
        return nodFront;
    }

    /**
     * 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.
     */
    private double temRealtoAdim(double
        TRealArg)
    {
        //return (TRealArg-TRefReal)/(T2Real-
            T1Real);
        return (TRealArg-this.getTRef())/(
            this.getT2()-this.getT1());
    }

}
```

C.5 Capa4CapInc_AMN.java

```
package ModeloConveccion;

import AlgebraLineal.Matriz;
import AlgebraLineal.MatrizDensa;
import FEM.FuncionExpresion;
import FEM.Frontera;
import FEM.TriangulacionDominioRectangular;
import FEM.Nodo;
import FEM.NodoBorde;
import FEM.DominioRectangular;
import java.util.List;
import java.util.ArrayList;

/**
 * Clase usada para los ensayos con cuatro
     capas horizontales y una capa inclinada
 * @author Alma MN
 */
public class Capa4CapaInc_AMN extends
    DefinicionModelo
        implements
            DefinicionModeloConveccion2D {

    //Objeros usados para obtener la
        información que permite simular la
    //"lectura" de información de un modo
        externo
    private DominioRectangular dominio;
    private TriangulacionDominioRectangular
        discretizacion;

    public Capa4CapaInc_AMN()
    {
        //En este caso, como se está
            simulando la lectura de la
            información
        //de un medio externo, en esta rutina
             se crea una discretización igual
        //a la que se creo con la información
             de esta "interfaz" para poder
        //simular la lectura de la informació
            n en los nodos.

        //Define el dominio que se usará en
            el modelo. En este caso se trata
        //de un dominio rectangular orientaci
            ón vertical
        this.dominio = new DominioRectangular
            ();
        this.dominio.defineDominioRectangular
            (this.getX1(), this.getZ1(),
                this.getX2(), this.getZ2());

        //Define la la discretización usada
            para la ecuación de flujo de
            calor.
        this.discretizacion=new
          TriangulacionDominioRectangular(
              dominio,this.getNx(),this.getNz
              ());
```

```
    }

    /**
     * Obtiene el mínimo valor en la direcció
         n X del dominio rectangular.
     * @return
     */
    public double getX1()
    {
        return 0.0;
    }

    /**
     * Obtiene el máximo valor en la direcció
         n X del dominio rectangular.
     * @return
     */
    public double getX2()
    {
        return 50.0;
    }

    /**
     * Obtiene el mínimo valor en la direcció
         n Z del dominio rectangular.
     * @return
     */
    public double getZ1()
    {
        return 0.0;
    }

    /**
     * Obtiene el máximo valor en la direcció
         n Z del dominio rectangular.
     * @return
     */
    public double getZ2()
    {
        return 200.0;
    }

    /**
     * Obtiene la altura del dominio
         rectangular.
     * @return
     */
    public double getH()
    {
        return this.getZ2()-this.getZ1();
    }

    /**
     * Obtiene el ancho del dominio
         rectangular.
     * @return
     */
    public double getL()
    {
        return this.getX2()-this.getX1();
    }

    /**
     * Obtiene el 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.
     * @return
     */
    public int getNx()
    {
        return 30;
    }

    /**
     * Obtiene el 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.
     * @return
     */
    public int getNz()
    {
        return 120;
```

```
}

/**
 * Obtiene el coeficiente de relajación
     usado en Crank Nicholson.
 * @return
 */
public double getTheta()
{
    return 0.8;
}

/**
 * Obtiene el salto en el tiempo (Dt)
     usado en Crank Nicholson.
 * @return
 */
public double getDt()
{
    return 0.01;
}

/**
 * Obtiene el valor de la temperatura
     inferior.
 * @return
 */
public double getT1()
{
    return 100;
}

/**
 * Obtiene el valor de la temperatura
     superior.
 * @return
 */
public double getT2()
{
    return 250;
}

/**
 * Obtiene el valor de la temperatura de
     referencia.
 * @return
 */
public double getTRef()
{
    return (getT1()+getT2())/2.0;
}

/**
 * Obtiene el valor del tensor simétrico
     de conductividad térmica
 * anisotrópica en un  nodo de la
     triangulación.
 * @param iNodo Índice del nodo de la
     triangulación.
 * @return      Una matriz con el valor
     del tensor simétrico de
 *              conductividad térmica
     anisotrópica.
 */
public Matriz getLambAst(int iNodo)
{
    //Se simula la obtención del valor
        del tensor.
    MatrizDensa mat = new MatrizDensa
        (2,2);

    //Capa 1 (Inferior)
    if(this.discretizacion.nodo[iNodo].
        coordenada[1]<=this.getH()/4.0)
    {
       //Capa Inclinada
        if(this.discretizacion.nodo[iNodo
            ].coordenada[1]>=
            ((4*this.discretizacion.nodo[
                iNodo].coordenada[0])+10)
        && this.discretizacion.nodo[iNodo
            ].coordenada[1]<(
             (4*this.discretizacion.nodo[
                 iNodo].coordenada[0])
                 +40))

            {
             mat.asignaValor(0, 0, 2.7);
             mat.asignaValor(1, 1, 2.7);
             mat.asignaValor(0, 1, 0.0);
```

```
                mat.asignaValor(1, 0, 0.0);
               }

          // fuera de la capa inclinada;
               pero aún en el estrato
               inferior
          else
               {
                mat.asignaValor(0, 0, 5.4);
                mat.asignaValor(1, 1, 5.4);
                mat.asignaValor(0, 1, 0.0);
                mat.asignaValor(1, 0, 0.0);
               }
     }

     //Capa 2
   else if (this.discretizacion.nodo[iNodo
        ].coordenada[1]<2.0*this.getH()/4.0
       && this.discretizacion.nodo[iNodo].
            coordenada[1]>=1.0*this.getH()
            /4.0)
     {
          //Capa Inclinada
          if(this.discretizacion.nodo[iNodo
               ].coordenada[1]>=
               ((4*this.discretizacion.nodo[
                    iNodo].coordenada[0])+10)
            && this.discretizacion.nodo[iNodo
               ].coordenada[1]<(
                (4*this.discretizacion.nodo[
                     iNodo].coordenada[0])
                     +40))

               {
                mat.asignaValor(0, 0, 2.7);
                mat.asignaValor(1, 1, 2.7);
                mat.asignaValor(0, 1, 0.0);
                mat.asignaValor(1, 0, 0.0);
               }

          // fuera de la capa inclinada;
               pero aún en el estrato 2
          else
               {
                mat.asignaValor(0, 0, 5.1);
                mat.asignaValor(1, 1, 5.1);
                mat.asignaValor(0, 1, 0.0);
                mat.asignaValor(1, 0, 0.0);
               }
     }
     //Capa 3
   else if (this.discretizacion.nodo[iNodo
        ].coordenada[1]<3.0*this.getH()/4.0
       && this.discretizacion.nodo[iNodo].
            coordenada[1]>=2.0*this.getH()
            /4.0)
     {
        //Capa Inclinada
         if(this.discretizacion.nodo[iNodo
               ].coordenada[1]>=
               ((4*this.discretizacion.nodo[
                    iNodo].coordenada[0])+10)
            && this.discretizacion.nodo[iNodo
               ].coordenada[1]<(
                (4*this.discretizacion.nodo[
                     iNodo].coordenada[0])
                     +40))

               {
                mat.asignaValor(0, 0, 2.7);
                mat.asignaValor(1, 1, 2.7);
                mat.asignaValor(0, 1, 0.0);
                mat.asignaValor(1, 0, 0.0);
               }

          // fuera de la capa inclinada;
               pero aún en el estrato 3
          else
               {
                mat.asignaValor(0, 0, 5.4);
                mat.asignaValor(1, 1, 5.4);
                mat.asignaValor(0, 1, 0.0);
                mat.asignaValor(1, 0, 0.0);
               }
     }

      //Capa 4 (Superior)
     else
     {
        //Capa Inclinada
```

```
            if(this.discretizacion.nodo[iNodo
                ].coordenada[1]>=
                ((4*this.discretizacion.nodo[
                    iNodo].coordenada[0])+10)
               && this.discretizacion.nodo[iNodo
                ].coordenada[1]<(
                 (4*this.discretizacion.nodo[
                     iNodo].coordenada[0])
                     +40))

                {
                 mat.asignaValor(0, 0, 2.7);
                 mat.asignaValor(1, 1, 2.7);
                 mat.asignaValor(0, 1, 0.0);
                 mat.asignaValor(1, 0, 0.0);
                }

            // fuera de la capa inclinada;
                pero aún en el estrato
                superior
            else
                {
                 mat.asignaValor(0, 0, 5.1);
                 mat.asignaValor(1, 1, 5.1);
                 mat.asignaValor(0, 1, 0.0);
                 mat.asignaValor(1, 0, 0.0);
                }
        }

        return mat;
    }

    /**
     * Obtiene el valor del tensor simétrico
         de permeabilidad
     * anisotrópica en un  nodo de la
         discretización.
     * @param iNodo Índice del nodo de la
         discretización
     * @return      Una matriz con el valor
         del tensor simétrico de
     *              permeabilidad anisotró
         pica.
     */
    public Matriz getKAst(int iNodo)
    {
        //Se simula la obtención del valor
            del tensor.
        MatrizDensa mat = new MatrizDensa
            (2,2);

        //Capa 1 (inferior)
        if(this.discretizacion.nodo[iNodo].
            coordenada[1]<=this.getH()/4.0)
        {
            //Capa Inclinada
            if(this.discretizacion.nodo[iNodo
                ].coordenada[1]>=
                ((4*this.discretizacion.nodo[
                    iNodo].coordenada[0])+10)
               && this.discretizacion.nodo[iNodo
                ].coordenada[1]<(
                 (4*this.discretizacion.nodo[
                     iNodo].coordenada[0])
                     +40))

                {
                 mat.asignaValor(0, 0,
                     0.000000000001);
                 mat.asignaValor(1, 1,
                     0.0000000000001);
                 mat.asignaValor(0, 1, 0.0);
                 mat.asignaValor(1, 0, 0.0);
                }

            //Fuera de la capa inclinada,
                pero aún en la capa inferior
            else
                {
                 mat.asignaValor(0, 0,
                     0.0000000000000011);
                 mat.asignaValor(1, 1,
                     0.0000000000000011);
                 mat.asignaValor(0, 1, 0.0);
                 mat.asignaValor(1, 0, 0.0);
                }
        }

        //Capa 2
```

```
else if (this.discretizacion.nodo[iNodo
    ].coordenada[1]<2.0*this.getH()/4.0
    && this.discretizacion.nodo[iNodo].
        coordenada[1]>=1.0*this.getH()
        /4.0)
  {
      //Capa Inclinada
      if(this.discretizacion.nodo[iNodo
          ].coordenada[1]>=
          ((4*this.discretizacion.nodo[
              iNodo].coordenada[0])+10)
      && this.discretizacion.nodo[iNodo
          ].coordenada[1]<(
           (4*this.discretizacion.nodo[
               iNodo].coordenada[0])
               +40))

          {
           mat.asignaValor(0, 0,
               0.000000000001);
           mat.asignaValor(1, 1,
               0.0000000000001);
           mat.asignaValor(0, 1, 0.0);
           mat.asignaValor(1, 0, 0.0);
          }

      //Fuera de la capa inclinada,
          pero aún en la capa 2
      else
          {
           mat.asignaValor(0, 0,
               0.0000000000011);
           mat.asignaValor(1, 1,
               0.0000000000011);
           mat.asignaValor(0, 1, 0.0);
           mat.asignaValor(1, 0, 0.0);
          }
  }

//Capa 3
else if (this.discretizacion.nodo[iNodo
    ].coordenada[1]<3.0*this.getH()/4.0
    && this.discretizacion.nodo[iNodo].
        coordenada[1]>=2.0*this.getH()
        /4.0)
  {
      //Capa Inclinada
      if(this.discretizacion.nodo[iNodo
          ].coordenada[1]>=
          ((4*this.discretizacion.nodo[
              iNodo].coordenada[0])+10)
      && this.discretizacion.nodo[iNodo
          ].coordenada[1]<(
           (4*this.discretizacion.nodo[
               iNodo].coordenada[0])
               +40))

          {
           mat.asignaValor(0, 0,
               0.000000000001);
           mat.asignaValor(1, 1,
               0.0000000000001);
           mat.asignaValor(0, 1, 0.0);
           mat.asignaValor(1, 0, 0.0);
          }

      //Fuera de la capa inclinada,
          pero aún en la capa 3
      else
          {
           mat.asignaValor(0, 0,
               0.0000000000000011);
           mat.asignaValor(1, 1,
               0.0000000000000011);
           mat.asignaValor(0, 1, 0.0);
           mat.asignaValor(1, 0, 0.0);
          }
  }

//Capa 4 (superior)
else
{
    //Capa Inclinada
    if(this.discretizacion.nodo[iNodo
        ].coordenada[1]>=
        ((4*this.discretizacion.nodo[
            iNodo].coordenada[0])+10)
    && this.discretizacion.nodo[iNodo
        ].coordenada[1]<(
```

```
                (4*this.discretizacion.nodo[
                    iNodo].coordenada[0])
                    +40))

                {
                 mat.asignaValor(0, 0,
                     0.000000000001);
                 mat.asignaValor(1, 1,
                     0.0000000000001);
                 mat.asignaValor(0, 1, 0.0);
                 mat.asignaValor(1, 0, 0.0);
                }

            //Fuera de la capa inclinada,
                pero aún en la capa superior
            else
                {
                 mat.asignaValor(0, 0,
                     0.0000000000011);
                 mat.asignaValor(1, 1,
                     0.0000000000011);
                 mat.asignaValor(0, 1, 0.0);
                 mat.asignaValor(1, 0, 0.0);
                }
        }

        return mat;
    }

    /**
     * Obtiene el valor de la producción de
         calor en un  nodo
     * de la discretización.
     * @param iNodo Índice del nodo de la
         discretización
     * @return      El valor de la producción
          de calor en el nodo.
     */
    public double getAAst(int iNodo)
    {
        //En este caso particular no hay
            producción de calor.
        return 0.0;
    }

    /**
     * Obtiene el valor de la capacidad calor
         ífica del fluido.
     * @return
     */
    public double getRhoc()
    {
        return 4180000;
    }

    /**
     * Obtiene el valor de la densidad media
         del fluido.
     * @return
     */
    public double getRho()
    {
        return 968;
    }

    /**
     * Obtiene el valor de la viscosidad diná
         mica del fluido a la temperatura
     * de referencia.
     * @return
     */
    public double getNeta()
    {
        return 0.00018;
    }

    /**
     * Obtiene el valor del coeficiente de
         dilatación térmica del fluido.
     * @return
     */
    public double getBeta()
    {
        return 0.0008275;
    }

    /**
     * Obtiene el valor de la aceleración de
         la gravedad.
     * @return
```

```
     */
    public double getG()
    {
        return 9.81;
    }

    /**
     * Obtiene el valor del coeficiente de
         dependencia de la viscosidad
     * dinámica del fluido con la temperatura
         .
     * @return
     */
    public double getGamma()
    {
        return 0.0;
    }

    /**
     * Obtiene el valor de la diferencia má
         xima que debe de existir entre dos
     * valores consecutivos de la solución de
          Temperatura dentro del ciclo
     * iterativo de acoplamiento con Stream.
     * @return
     */
    public double getDifTempMax()
    {
        return 0.001;
    }

    /**
     * Obtiene el valor de la diferencia má
         xima que debe de existir entre dos
     * valores consecutivos de la solución de
          Stream dentro del ciclo
     * iterativo de acoplamiento con
         Temperatura.
     * @return
     */
    public double getDifStreamMax()
    {
        return 0.1;
    }

    /**
     * Obtiene el número máximo de
         iteraciones permitido dentro del
         ciclo
     * usado para acoplarlas soluciones de
         Stream y Temperatura.
     * @return
     */
    public int getMaxIt()
    {
        return 100;
    }

    /**
     * Obtiene la FuncionExpresión que define
          las condiciones iniciales de
     * temperatura del problema de conveccion
         .
     * @return  Una FuncionExpresion que
         define las condiciones iniciales de
     *          temperatura del problema de
         conveccion.
     */
    public FuncionExpresion getTempCondIni()
    {
        //Inicializa y asigna la función
            correspondiente a las C.I.
        FuncionExpresion condIniArg =

                //new FuncionExpresion("-cos(
                    pi*z)/2*cos(x*pi)");
                new FuncionExpresion("-sin(z*
                    pi-pi/2)/2");

        return condIniArg;
    }

    /**
     * Obtiene las fronteras que definen el
         modelo de flujo de calor en un
     * problema de convección.
     * @return  Un arreglo de Frontera que
         contiene las fronteras que definen
     *          el modelo de flujo de calor.
     */
```

```
public Frontera[] getTempFront()
{
    //Variable usada para definir el id
        de la frontera
    int idFrontCont = 0;

    //Inicializa el arreglo de Fronteras,
         las cuales en este caso serán
    //cuatro.
    Frontera[] fronteraArg = new Frontera
        [4];

    //Incializa el Vector que servirá
        para definir la normal a cada
    //frontera
    AlgebraLineal.Vector normFront =  new
         AlgebraLineal.Vector(2);

    //Frontera 0
    //Frontera Dirichlet correspondiente
        al borde inferior del dominio
    normFront.AsignaValor(0, 0);
    normFront.AsignaValor(1, -1);
    //fronteraArg[idFrontCont] = new
        Frontera(true, idFrontCont++,
    //         String.valueOf(this.
        temRealtoAdim(this.getT2())),
        normFront);
    fronteraArg[idFrontCont] = new
        Frontera(false, idFrontCont++,
            "0.3", normFront);

    //Frontera 1
    //Frontera Neumann correspondiente al
         borde derecho del dominio
    normFront.AsignaValor(0, 1);
    normFront.AsignaValor(1, 0);
    fronteraArg[idFrontCont] = new
        Frontera(false, idFrontCont++,
            "0", normFront);

    //Frontera 2
    //Frontera Dirichlet correspondiente
        al borde superior del dominio
    normFront.AsignaValor(0, 0);
    normFront.AsignaValor(1, 1);
    fronteraArg[idFrontCont] = new
        Frontera(true, idFrontCont++,
            String.valueOf(this.
                temRealtoAdim(this.getT1
                ())), normFront);

    //Frontera 3
    //Frontera Neumann correspondiente al
         borde izquierdo del dominio
    normFront.AsignaValor(0, -1);
    normFront.AsignaValor(1, 0);
    fronteraArg[idFrontCont] = new
        Frontera(false, idFrontCont++,
            "0", normFront);

    return fronteraArg;
}


/**
 * Obtiene la asignación de los nodos
     borde que pertenecen a cada una
 * de las fronteras en particular del
     modelo de flujo de calor en un
 * problema de convección.
 * @return  Un arreglo de objetos List
     que contienen las listas de los
 *          índices de los nodos que
     pertenecen a cada una de las
 *          fronteras del problema.
 */
public List<Integer> []
    getListTempNodFront()
{
    ////////////////////////////////
    //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> [] nodFront = new
    ArrayList[getTempFront().length];

//Inicializa las listas que contendrá
    n los índices de los nodos que
//pertenecen a éstas
for (int i= 0; i<getTempFront().
    length; i++)
{
    nodFront[i]= new ArrayList <
        Integer >();
}

//En este ejemplo en particular se
    simula la lectura de los nodos
//que pertenecen a cada frontera por
    medio de una rutina que define
//y asigna las fronteras según los
    criterios particulares del
    ejemplo.

//Recorre los nodos de la
    discretización para asignar a
    cada nodo borde
//su correspondiente frontera
for (Nodo nodoActual: this.
    discretizacion.nodo)
{
    //Verifica que se trate de un
        nodo borde
    if(nodoActual instanceof
        NodoBorde)
    {


        //Se realiza una conversión
            descendente al dominio
        //(De un Domino a un
            DominioRectangular
        DominioRectangular dominio2d
            =
           (DominioRectangular) this.
               discretizacion.dominio
               ;

        //Se realiza una conversión
            descendente al nodo
            actual
        //(De un Nodo a un NodoBorde
        NodoBorde nodoBorde = (
            NodoBorde)nodoActual;

        //Frontera 0
        //Asigna la frontera inferior
        if(nodoBorde.coordenada[1]==
            dominio2d.getZ1())
        {
            nodFront[0].add(
                nodoActual.idNodo);
        }

        //Frontera 1
        //Asigna la frontera derecha
        if(nodoBorde.coordenada[0]==
            dominio2d.getX2())
        {
            nodFront[1].add(
                nodoActual.idNodo);
        }

        //Frontera 2
        //Asigna la frontera superior
        if(nodoBorde.coordenada[1]==
            dominio2d.getZ2())
        {
            nodFront[2].add(
                nodoActual.idNodo);
        }

        //Frontera 3
        //Asigna la frontera
            izquierda
        if(nodoBorde.coordenada[0]==
            dominio2d.getX1())
        {
            nodFront[3].add(
                nodoActual.idNodo);
        }
    }
}
```

```
        return nodFront;
    }

    /**
     * Obtiene las fronteras que definen el
         modelo de flujo de masa en un
     * problema de convección.
     * @return  Un arreglo de Frontera que
         contiene las fronteras que definen
     *          el modelo de flujo de masa.
     */
    public Frontera[] getStreamFront()
    {
        //Variable usada para definir el id
            de la frontera
        int idFrontCont = 0;

        //Inicializa el arreglo de Fronteras,
             las cuales en este caso serán
        //cuatro.
        Frontera[] fronteraArg = new Frontera
            [4];

        //Incializa el Vector que servirá
            para definir la normal a cada
        //frontera
        AlgebraLineal.Vector normFront =  new
             AlgebraLineal.Vector(2);

        //Frontera 0
        //Frontera Dirichlet correspondiente
            al borde inferior del dominio
        normFront.AsignaValor(0, 0);
        normFront.AsignaValor(1, -1);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        //Frontera 1
        //Frontera Dirichlet correspondiente
            al borde derecho del dominio
        normFront.AsignaValor(0, 1);
        normFront.AsignaValor(1, 0);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        //Frontera 2
        //Frontera Dirichlet correspondiente
            al borde superior del dominio
        normFront.AsignaValor(0, 0);
        normFront.AsignaValor(1, 1);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        //Frontera 3
        //Frontera Dirichlet correspondiente
            al borde izquierdo del dominio
        normFront.AsignaValor(0, -1);
        normFront.AsignaValor(1, 0);
        fronteraArg[idFrontCont] = new
            Frontera(true, idFrontCont++,
                "0", normFront);

        return fronteraArg;
    }


    /**
     * Obtiene la asignación de los nodos
         borde que pertenecen a cada una
     * de las fronteras en particular del
         modelo de flujo de masa en un
     * problema de convección.
     * @return  Un arreglo de objetos List
         que contienen las listas de los
     *          índices de los nodos que
         pertenecen a cada una de las
     *          fronteras del problema.
     */
    public List <Integer> []
        getListStreamNodFront()
    {
        ////////////////////////////////
        //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> [] nodFront = new
    ArrayList[getTempFront().length];

//Inicializa las listas que contendrá
    n los índices de los nodos que
//pertenecen a éstas
for (int i= 0; i<getTempFront().
    length; i++)
{
    nodFront[i]= new ArrayList <
        Integer >();
}

//En este ejemplo en particular se
    simula la lectura de los nodos
//que pertenecen a cada frontera por
    medio de una rutina que define
//y asigna las fronteras según los
    criterios particulares del
    ejemplo.

//Recorre los nodos de la
    discretización para asignar a
    cada nodo borde
//su correspondiente frontera
for (Nodo nodoActual: this.
    discretizacion.nodo)
{
    //Verifica que se trate de un
        nodo borde
    if(nodoActual instanceof
        NodoBorde)
    {


        //Se realiza una conversión
            descendente al dominio
        //(De un Domino a un
            DominioRectangular
        DominioRectangular dominio2d
            =
           (DominioRectangular) this.
               discretizacion.dominio
               ;

        //Se realiza una conversión
            descendente al nodo
            actual
        //(De un Nodo a un NodoBorde
        NodoBorde nodoBorde = (
            NodoBorde)nodoActual;

        //Frontera 0
        //Asigna la frontera inferior
        if(nodoBorde.coordenada[1]==
            dominio2d.getZ1())
        {
            nodFront[0].add(
                nodoActual.idNodo);
        }

        //Frontera 1
        //Asigna la frontera derecha
        if(nodoBorde.coordenada[0]==
            dominio2d.getX2())
        {
            nodFront[1].add(
                nodoActual.idNodo);
        }

        //Frontera 2
        //Asigna la frontera superior
        if(nodoBorde.coordenada[1]==
            dominio2d.getZ2())
        {
            nodFront[2].add(
                nodoActual.idNodo);
        }

        //Frontera 3
        //Asigna la frontera
            izquierda
        if(nodoBorde.coordenada[0]==
            dominio2d.getX1())
```

```
                {
                    nodFront[3].add(
                        nodoActual.idNodo);
                }
            }
        }
        return nodFront;
    }

    /**
     * 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.
     */
    private double temRealtoAdim(double
        TRealArg)
    {
        //return (TRealArg-TRefReal)/(T2Real-
            T1Real);
        return (TRealArg-this.getTRef())/(
            this.getT2()-this.getT1());
    }

}
```

C.6 Graficas_Salidas.m

```
   clc, clear close all

%Cargar el archivo.txt de los resultados Java
     y asignarlos a la matriz A:

%L/getNx = ix ; H/getNz = iy  ; getNx y getNz
     son las particiones en Java

%% Pruebas

%% Dominio homogéneo e isotrópico, lambda
    =2.7, K=1e-12
% load salidaprueba_Capa1HL1_AMN_16mar.txt
% A = salidaprueba_Capa1HL1_AMN_16mar;
% ix=2.0; iy=2.0; H=100; L=100;

%% Dominio homogéneo e isotrópico: lambda
    =2.7, K=1e-16.
%% Capa inclinada homogénea e isotrópica:
    lambda=2.7, K=1.27e-12
% load salidaprueba_Capa1HL1CapaInc1SinFlujo_
    AMN_17mar.txt
% A = salidaprueba_Capa1HL1CapaInc1SinFlujo_
    AMN_17mar;
% ix=2.0; iy=2.0; H=100; L=100;

%% Dominio homogéneo e isotrópico: lambda
    =2.7, K=1e-16.
%% Capa inclinada homogénea e isotrópica:
    lambda=2.7, K=1.27e-12
% load salidaprueba_Capa1HL033CapaInc1Sin
    Flujo_AMN_18mar.txt
% A = salidaprueba_Capa1HL033CapaInc1Sin
    Flujo_AMN_18mar;
% ix=6.0; iy=2.0; H=100; L=300;

%% Dominio homogéneo e isotrópico: lambda
    =5.4, K=1e-16. FlujoCalor 0.3
%% Capa Inc homogénea y anisotrópica: lambda
    =2.7, kh=1e-12, kv=1e-13
% load salidaprueba_Capa1HL1CapaInc2Flujo
    Calor_AMN_17mar.txt
% A = salidaprueba_Capa1HL1CapaInc2Flujo
    Calor_AMN_17mar;
% ix=2.0; iy=2.0; H=100; L=100;

%% Dominio homogéneo e isotrópico: lambda
    =5.4, K=1e-16. FlujoCalor 0.3
%% CapaInc(0.03) hom-anisotrópica: lambda
    =2.7, kh=1e-12, kv=1e-13
% load salidaprueba_Capa1HL033CapaInc2Flujo
    Calor_AMN_17mar.txt
% A = salidaprueba_Capa1HL033CapaInc2Flujo
    Calor_AMN_17mar;
% ix=6.0; iy=2.0; H=100; L=300;
```

```
%% Dominio homogéneo e isotrópico: lambda
    =5.4, K=1e-16. FlujoCalor 0.3
%% CapaInc(0.15) hom-anisotrópica: lambda
    =2.7, kh=1e-12, kv=1e-13
% load salidaprueba_Capa1HL033CapaInc2Flujo
    Calorv2_AMN_17mar.txt  %v3
% A = salidaprueba_Capa1HL033CapaInc2Flujo
    Calorv2_AMN_17mar;
% ix=6.0; iy=2.0; H=100; L=300;

%% Dominio homogéneo e isotrópico: lambda
    =5.4, K=1e-16. FlujoCalor 0.3
%% CapaInc(6.33) hom-anisotrópica: lambda
    =2.7, kh=1e-12, kv=1e-13
% load salidaprueba_Capa1HL4CapaInc2Flujo
    Calor_AMN_17mar.txt
% A = salidaprueba_Capa1HL4CapaInc2Flujo
    Calor_AMN_17mar;
% ix=1.66; iy=1.66; H=200; L=50;

%% Dominio homogéneo e isotrópico: lambda
    =5.4, K=1e-16. FlujoCalor 0.3
%% CapaInc(19) hom-anisotrópica: lambda=2.7,
     kh=1e-12, kv=1e-13
% load salidaprueba_Capa1HL4CapaInc2Flujo
    Calorv2_AMN_17mar.txt
% A = salidaprueba_Capa1HL4CapaInc2Flujo
    Calorv2_AMN_17mar;
% ix=1.66; iy=1.66; H=200; L=50;

%% Dominio homogéneo e isotrópico: lambda
    =5.4, K=1e-16 0.3 *** SIN FLUJO
%% CapaInc(19) hom-anisotrópica: lambda=2.7,
     kh=1e-12, kv=1e-13
% load salidaprueba_Capa1HL4CapaInc2Flujo
    Calorv2sf_AMN_17mar.txt
% A = salidaprueba_Capa1HL4CapaInc2Flujo
    Calorv2sf_AMN_17mar;
% ix=1.66; iy=1.66; H=200; L=50;

%% Dominio heterogéneo e isotrópico de 2
    capas: lambda1=2.8, lambda2=2.5
%% K1=1.1e-16, K2=1.1e-14, Sin flujo de
    calor
%% CapaInc(4) hom-anisotrópica: lambda=5.4,
    kh=1e-11, kv=1e-12
% load salidaprueba_Capa2HL4CapaInc1Sin
    Flujo_AMN_18mar.txt
% A = salidaprueba_Capa2HL4CapaInc1Sin
    Flujo_AMN_18mar;
% ix=1.66; iy=1.66; H=200; L=50;

%% Dominio heterogéneo e isotrópico de 2
    capas: lambda1=2.8, lambda2=2.5
%% K1=1.1e-16, K2=1.1e-14, Sin flujo de
    calor
%% CapaInc(0.75) hom-anisotrópica: lambda
    =5.4, kh=1e-11, kv=1e-12
% load salidaprueba_Capa2HL033CapaInc1
    SinFlujo_AMN_18mar.txt
% A = salidaprueba_Capa2HL033CapaInc1
    SinFlujo_AMN_18mar;
% ix=6.0; iy=2.0; H=100; L=300;

%% 1a. Sin flujo de calor
% load salidaprueba_Capa4CapaIncHL4_capinc
    an_01mar.txt    %CapIncAn-4CapHor
% A = salidaprueba_Capa4CapaIncHL4_capinc
    an_01mar;          %[B,B] = [31,121]
% ix=1.66; iy=1.66; H=200; L=50;

%% 1b. Con flujo de calor desde el fondo
% load salidaprueba_Capa4CapaIncHL4_capinc
    an_fcinf_02mar.txt    %CapIncAn-4CapHor
% A = salidaprueba_Capa4CapaIncHL4_capinc
    an_fcinf_02mar;          %[B,B] = [31,121]
% ix=1.66; iy=1.66; H=200; L=50;

% 2a. Sin flujo de calor
load salidaprueba_Capa4difespCapaIncHL4_ 10
    mar.txt    %CapIncAn-4CapHor
A = salidaprueba_Capa4difespCapaIncHL4_ 10mar
    ;          %[B,B] = [31,121]
ix=1.66; iy=1.66; H=200; L=50;

%%

% Asignar valores de ejes, temperatura y
    stream:
```

```
XX = A(:,1); % Vector de valores en el eje X
YY = A(:,2); % Vector de valores en el eje Y
VT = A(:,3); % Vector de Temperatura
VS = A(:,4); % Vector de Stream

% Redimensionar de vector a matriz:
B = round(sqrt(size(A,1))); % vector de long
    A ---> matriz [B,B]
VTm = reshape(VT,[31,121]); % Matriz de
    Temperatura
VSm = reshape(VS,[31,121]); % Matriz de
    Stream

% reshape(x,N,[]) redimensiona x en matriz
    tamaño (N,longx/N)

% Transponer la matriz porque está al revés:
VTmr = VTm'; % Matriz reordenada Temperatura
VSmr = VSm'; % Matriz reordenada Stream

% Crear la malla XY para graficar (espacio de
     trabajo)
[x,y] = meshgrid(0.0:ix:L, 0.0:iy:H)

% TEMPERATURA
% Graficar los datos de Temperatura:
figure
subplot(1,2,1) % (r,c,p) posición puede
    abarcar uno o varios 1:2 1:4 etc
[CT,h] = contourf(x, y, VTmr, 20);

% Poner etiquetas a los contornos
clabel(CT, h, 'manual', 'FontSize',12,'Color'
    ,'white');
colormap
colorbar
title('Temperaturas')

hold on

% STREAM
% Graficar los datos de Stream:
subplot(1,2,2)
[CS,h] = contourf(x, y, VSmr, 20);
% Poner etiquetas a los contornos
clabel(CS, h, 'manual', 'FontSize',12,'Color'
    ,'white');
colormap
colorbar
title('Stream')
```