



**UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN**

SIMULACIÓN DE FLUJO EN MEDIOS POROSOS USANDO CVFE

**T E S I S
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN**

P R E S E N T A:
M A R I O A R T U R O N I E T O B U T R Ó N

Director de Tesis:
DR. LUIS MIGUEL DE LA CRUZ SALAS
Instituto de Geofísica, UNAM

Dedicatoria

A Beatriz una madre ejemplar.

A mis hermanos.

A mis abuelos.

Agradecimientos

Agradezco a mi tutor el Dr. Luis Miguel de la Cruz Salas por su tiempo, sus consejos, su paciencia, por aquellas pláticas de motivación, por su confianza, pero sobre todo por transmitirme una pequeña parte de su saber.

Agradezco a mis sinodales María Guadalupe Elena Ibargüengoitia González, Miguel Ángel Padilla Castañeda, Graciela del Socorro Herrera Zamarrón y Norberto Carmen Vera Guzmán, por el tiempo que dedicaron en revisar mi trabajo y por todos sus consejos.

A mi familia, especialmente a Beatriz por ser una madre dedicada que siempre me apoya en cada una de mis decisiones, por sus consejos y sobretodo por su amor, a mis hermanos Jacob y Lía por esos momentos de felicidad.

A Ernesto, por esas charlas y sobretodo por su colaboración y su amistad. A mis amigos Aitor, Alex, Aurelio, Heriberto, Juan Carlos, Karen, Marco, Octavio y Paulo por que en más de una ocasión me guiaron, me enseñaron, me aconsejaron pero sobretodo por brindarme su amistad, gracias amigos les debo mucho.

A Lulú, Amalia y Cecilia por su apoyo sincero durante todo este tiempo. Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por haberme otorgado una beca para realizar mis estudios.

Finalmente al Posgrado en Ciencia e Ingeniería de la Computación y a mi *alma mater*, la Universidad Nacional Autónoma de México por creer en mi y darme una educación de calidad.

¡Gracias a todos!

Índice general

Índice de figuras	X
Índice de tablas	XI
1. Introducción	1
1.1. Antecedentes	2
1.1.1. Proceso de Software Científico	2
1.1.2. La entropía y la crisis del software	3
1.1.3. Ingeniería de Software	4
1.2. Objetivos y metas	5
1.3. Organización de la tesis	6
2. Marco Teórico	7
2.1. Flujo en medios porosos	7
2.2. Ecuación de Balance de Masa	8
2.3. Flujo monofásico	9
2.4. CVFEM	10
2.4.1. Discretización usando CVFEM	10
2.4.2. Discretización del dominio y construcción de volúmenes de control(VC)	11
2.4.3. Integración de la ecuación diferencial en cada VC	12
2.4.4. Funciones de interpolación	14
2.4.5. Término Fuente	17
2.4.6. Condiciones de Frontera	17
2.5. Triangulación de Delaunay	18
3. Desarrollo del Sistema	21
3.1. Proceso de Administración del Proyecto	22

ÍNDICE GENERAL

3.2.	Proceso de Implementación de Software	32
3.2.1.	Antecedentes de Software	33
3.2.2.	Análisis de Modelos.	33
3.2.3.	Arquitectura y Diseño Detallado de Software	33
3.2.4.	Construcción del Modelo Computacional	33
3.2.5.	Pruebas de Integración	34
3.2.6.	Validación	35
3.2.7.	Entrega de Productos	35
4.	Validación	70
4.1.	Resultados preliminares	70
4.2.	Laplace en dos dimensiones	71
4.2.1.	Descripción de la prueba	71
4.2.2.	Descripción de Resultados	73
4.2.3.	Observaciones	74
4.3.	Laplace en un semicírculo	78
4.3.1.	Descripción	78
4.3.2.	Resultados	79
4.3.3.	Observaciones	80
4.4.	Poisson en dos dimensiones	83
4.4.1.	Descripción	83
4.4.2.	Resultados	84
4.4.3.	Observaciones	85
4.5.	Flujo monofásico	87
4.5.1.	Descripción	87
4.5.2.	Resultados	88
4.5.3.	Observaciones	90
4.6.	Entrega	95
5.	Conclusiones	96
5.1.	Crítica a los procesos de Software Científico	98
5.2.	Aportación	99
5.3.	Trabajo Futuro	99
Apéndices		101

A. ISO/IEC 29110 Perfil Básico	102
A.0.1. Administración del proyecto	102
A.0.2. Implementación de Software	103
B. Gmsh	107
B.1. Módulo Geométrico	107
B.1.1. Entidades geométricas	107
B.1.2. Entidades Físicas	108
B.1.3. Scripts	108
B.1.4. Módulo de Mallado	109
B.2. Archivo .msh	111
C. Lenguaje de Programación C++	113
C.1. Programación Genérica(PG)	113
C.1.1. Templates	113
C.1.2. Metaprogramación	114
D. SYSG	115
D.1. Código	115
D.2. Documentación Doxygen	117
D.3. Documentación ISO	117
D.4. Tesis	117
Referencias	117

Índice de figuras

1.1.	Esquema de avance del hardware en comparación con el desarrollo de software en áreas científicas. Tomada de <i>Supercomputing in plain english, Henry Neeman, OU Supercomputing Center for Education and Research, Univ. of Oklahoma, 2009.</i>	3
1.2.	Proceso de desarrollo de software en computo científico	4
2.1.	Dominio discretizado en elementos triangulares	11
2.2.	En la imagen se distinguen los dos tipos de volúmenes de control, alrededor de un nodo interno(verde) y alrededor de un nodo frontera(azul).	12
2.3.	Elemento triangular etiquetado en sentido contrario a las manecillas del reloj, con las letras i, j y k. Se muestra la unión del baricentro(mc) con las medianas ma,mb y md, como resultado, el elemento triangular queda dividido en tres partes iguales.	13
2.4.	Cada triángulo en una Triangulación de Delaunay tiene un disco abierto vacío circunscrito.	19
2.5.	Cada arista en la frontera de una triangulación convexa es fuertemente Delaunay, debido a que siempre es posible encontrar un disco vacío que contiene los puntos finales y ningún otro vértice.	20
3.1.	Actividades del proceso de Administración del Proyecto. Tomada de “ <i>Implementación del Estándar ISO/IEC 29110 Perfil Básico para el desarrollo de software del tipo científico” [4].</i> . .	22
3.2.	Actividades del proceso de Implementación de Software. Tomada de “ <i>Implementación del Estándar ISO/IEC 29110 Perfil Básico para el desarrollo de software del tipo científico” [4].</i> . .	32

3.3.	Diagrama de clases de la arquitectura para SYSG.	42
3.4.	Clase Mesh.	43
3.5.	Clase CVFEM.	44
3.6.	Diagrama de clases para COO y CSR.	46
3.7.	Clase Vector.	46
3.8.	Diagrama de clases para COO y CSR.	47
3.9.	Clase Resolve.	48
3.10.	Matriz dispersa. Tomada de <i>Método de volumen finito para flujo en una fase</i> [22].	55
3.11.	Se muestra la matriz almacenada en el formato CSR	57
3.12.	Sistema de entrada para probar los métodos numéricos. Tomada de <i>Método de volumen finito para flujo en una fase</i> [22]. . . .	58
3.13.	Comparación de tiempo para el algoritmo gradiente conjugado, usando dos implementaciones diferentes. Donde CGA(verde) representa la implementación utilizando sobrecarga de operadores y CGE(azul) representa la implementación utilizando programación genérica y técnicas de metaprogramación. . . .	61
3.14.	Comparación de tiempo para el algoritmo gradiente conjugado precondicionado con ILU, usando dos implementaciones diferentes. Donde CGAILU(verde) representa la implementación utilizando sobrecarga de operadores y CGEILU(azul) representa la implementación utilizando programación genérica y técnicas de metaprogramación. . . .	62
3.15.	Comparación de tiempo para el algoritmo gradiente conjugado precondicionado con MILU, usando dos implementaciones diferentes. Donde CGAMILU(verde) representa la implementación utilizando sobrecarga de operadores y CGEMILU(azul) representa la implementación utilizando programación genérica y técnicas de metaprogramación. . . .	63
3.16.	En la gráfica se muestra el gradiente conjugado(azul), el gradiente conjugado precondicionado con ILU(verde) y el gradiente conjugado precondicionado con MILU(rojo), para la implementación más eficiente.	64
3.17.	Comparación de tiempo para el algoritmo BICGSTAB, usando dos implementaciones diferentes. Donde BICGSTABA(verde) representa la implementación utilizando sobrecarga de operadores y BICGSTABE(azul) representa la implementación utilizando programación genérica y técnicas de metaprogramación.	65

3.18. Comparación de tiempo para el algoritmo BICGSTAB precondicionado con ILU, usando dos implementaciones diferentes.Donde BICGSTABAILU(verde) representa la implementación utilizando sobrecarga de operadores y BICGSTABEILU(azul) representa la implementación utilizando programación genérica y técnicas de metaprogramación.	66
3.19. Comparación de tiempo para el algoritmo BICGSTAB precondicionado con MILU, usando dos implementaciones diferentes.Donde BICGSTABAMILU(verde) representa la implementación utilizando sobrecarga de operadores y BICGSTABEMILU(azul) representa la implementación utilizando programación genérica y técnicas de metaprogramación.	67
3.20. En la gráfica se muestra el gradiente conjugado estabilizado(azul), el gradiente conjugado estabilizado precondicionado con ILU(verde) y el gradiente conjugado estabilizado precondicionado con MILU(rojo), para la implementación más eficiente.	68
4.1. Dominio de interés para la ecuación de Laplace	72
4.2. Solución para la ecuación de Laplace.	73
4.3. Malla para la ecuación de Laplace, generada con la herramienta gmsh.	74
4.4. La gráfica muestra el residual $\log r_n $ vs las iteraciones n, para el método del gradiente conjugado(azul) y el método del gradiente conjugado precondicionado con ILU(verde).	77
4.5. La gráfica muestra el residual $\log r_n $ vs las iteraciones n, para el método del gradiente conjugado estabilizado (azul) y el método del gradiente conjugado estabilizado precondicionado con ILU(verde).	77
4.6. Dominio acotado por dos semicírculos para la ecuación de Laplace.	78
4.7. Solución para la ecuación de Laplace en un semicírculo.	79
4.8. Malla para la ecuación de Laplace en un semicírculo.	80
4.9. La gráfica muestra el residual para el problema de Laplace acotado entre dos semicírculos $\log r_n $ vs las iteraciones n, para el método del gradiente conjugado(azul) y el método del gradiente conjugado precondicionado con ILU(verde).	82

4.10. La gráfica muestra el residual para el problema de Laplace acotado entre dos semicírculos $\log r_n $ vs n, para el método del gradiente conjugado estabilizado(azul) y el método del gradiente conjugado estabilizado precondicionado con ILU(verde).	82
4.11. Dominio para la ecuación de Poisson	83
4.12. Solución de la ecuación de Poisson	84
4.13. Discretización del dominio curvo para la ecuación de Poisson.	85
4.14. La gráfica muestra el residual para el problema de Poisson acotado entre dos semicírculos $\log r_n $ vs las iteraciones n, para el método del gradiente conjugado(azul) y el método del gradiente conjugado precondicionado con ILU(verde).	86
4.15. Dominio para la ecuación de presión con permeabilidad aleatoria, con una región de alta permeabilidad(azul) y una de baja permeabilidad (amarilla).	88
4.16. Solución para el flujo de medios porosos monofásico.	89
4.17. Malla para el flujo de medios porosos monofásico.	90
4.18. Componente k11 de la matriz de permeabilidad para cada nodo de la malla.	91
4.19. Componente k22 de la matriz de permeabilidad para cada nodo de la malla.	92
4.20. Componente k11 de la matriz de permeabilidad para cada nodo de la malla.	93
4.21. Componente k22 de la matriz de permeabilidad para cada nodo de la malla.	94
A.1. Procesos del Perfil Básico ISO/IEC 29110 Perfil Básico, Tomada de Norma Técnica NTP-RT-ISO/IEC TR 29110-5-1-2,INDECOP,2012.	103
A.2. Flujo entre las actividades del Proceso Administración del Proyecto, incluyendo los productos de trabajo más relevantes y la relación existente entre ellos. Tomada de Norma Técnica NTP-RT-ISO/IEC TR 29110-5-1-2,INDECOP,2012.	105
A.3. Flujo de trabajo del Proceso de Implementación de Software, compuesto por seis actividades, las cuales están descritas por 41 tareas. Tomada de Norma Técnica NTP-RT-ISO/IEC TR 29110-5-1-2,INDECOP,2012.	106
B.1. Geometría definida en gmsh.	109

B.2. Menú Options de gmsh.	109
B.3. Mesh - 2D, para discretizar el dominio en elementos triangulares.	110
B.4. Define la triangulación de Delaunay sobre el dominio usando gmsh.	110

Índice de cuadros

3.1.	Objetivos de la iteración.	24
3.2.	Tareas a ser realizadas en la iteración.	24
3.3.	Cambios a realizar durante la iteración.	25
3.4.	Riesgos encontrados.	25
3.5.	Estado actual del proyecto.	26
3.6.	Avance de tareas.	27
3.7.	Productos.	27
3.8.	Cambios.	28
3.9.	Riesgos encontrados.	29
3.10.	Resumen del Reporte de Seguimiento.	29
3.11.	Resumen del Cierre del Plan del Proyecto.	30
3.12.	Reporte de Productos.	30
3.13.	Ejecuciones para el almacenamiento de una matriz en COO y su conversión en CSR.	57
3.14.	Ejecuciones para el almacenamiento de una matriz en COO y su conversión en CSR.	60

Capítulo 1

Introducción

La ciencia es la herramienta del ser humano para dar respuesta a distintas preguntas. Inicialmente era sostenida por dos grandes pilares: la teoría y la experimentación, en ocasiones la pregunta era tan complicada que no podía ser contestada y era considerada como un problema intratable.

Por otro lado, hace algunas décadas nació el cómputo científico, que desde sus inicios ha sido utilizado ampliamente por científicos e ingenieros en la solución de problemas que requieren de cálculos exhaustivos. El cómputo científico ha permitido que la ciencia sea más flexible, de tal forma que se pueda hacer frente a los problemas intratables. Con esto no resulta extraño que ahora se considere al cómputo científico como el *tercer pilar de la ciencia* [1].

Existe toda una colección de herramientas, técnicas y teorías requeridas para resolver problemas en ciencia e ingeniería (véase [2]). El científico utiliza el cómputo como una simple herramienta para el desarrollo de programas que lo ayuden a responder una pregunta en específico, como resultado, sus programas no evolucionan, se estancan y finalmente quedan en el olvido. Una forma de solucionar esto, es por medio de normas que guíen al científico en el desarrollo de software, sin embargo, muchas de éstas fueron desarrolladas para grandes empresas y su adopción en un ambiente científico resulta complicado y quizás en la mayoría de los casos poco significativo.

En este trabajo se estudia el movimiento de fluidos en medios porosos mediante la solución numérica de ecuaciones diferenciales parciales. Para tratar las ecuaciones se utiliza el método numérico conocido como CVFEM (Control Volume Finite Element Method), véase [3]. El desarrollo computacional presentado en esta tesis se hizo siguiendo un proceso de desarrollo de software

propuesto en el trabajo: “Implementación del Estándar ISO/IEC 29110 Perfil Básico para el desarrollo de software del tipo científico”, véase [4], realizado por Ernesto Albarrán estudiante del Posgrado en Ciencia e Ingeniería de la Computación, especializado en ingeniería de software.

1.1. Antecedentes

El estudio del flujo en medios porosos, ha sido tratado desde hace varias décadas, debido a su importancia en múltiples problemas de ciencia e ingeniería, por ejemplo; en agricultura, hidrología, geotermia y recuperación de petróleo. Para describir este tipo de flujos en medios porosos, son utilizados modelos, que en su mayoría son matemáticos, su objetivo es describir el comportamiento del fluido que ocupan los poros. Estos modelos generalmente no se pueden resolver de forma analítica, por lo tanto se requieren métodos numéricos, que permitan una buena aproximación de la solución. Los más usados son: diferencias finitas(MDF) [5], elemento finito(MEF) [5] y volumen finito(MVF) [6]. El objetivo principal de éstos, es transformar la ecuación diferencial parcial en una serie de ecuaciones algebraicas lineales, cuya solución sea una buena aproximación de la ecuación original. Sin embargo, para emplearlos se deben almacenar y procesar una gran cantidad de datos. Por lo tanto se requieren considerables recursos de hardware y software.

El hardware progresó día con día, por ejemplo actualmente(noviembre 2016) la supercomputadora más poderosa es Sunway TaihuLight, que cuenta con 10,649,600 núcleos de procesamiento y un rendimiento máximo de 93,014.6 TFlop/s [7]. A pesar de este crecimiento en hardware, no se observa un crecimiento igual en el software científico(Ver Fig. 1.1).

1.1.1. Proceso de Software Científico

Desarrollar software científico es distinto a desarrollar software comercial [8], debido a la gran cantidad de conocimientos previos necesarios, por lo que el científico, debe ser el desarrollador. En este caso se identificó que el proceso que siguen la mayoría de los científicos; es llamado *Modelación matemática y computacional(MMC)*. Consiste en la generación de modelos conceptuales del problema investigado: posteriormente se transforma en un modelo matemático que indica la o las ecuaciones gobernantes del fenómeno: y debido a la complejidad de su solución son usadas técnicas numéricas para aproximar la

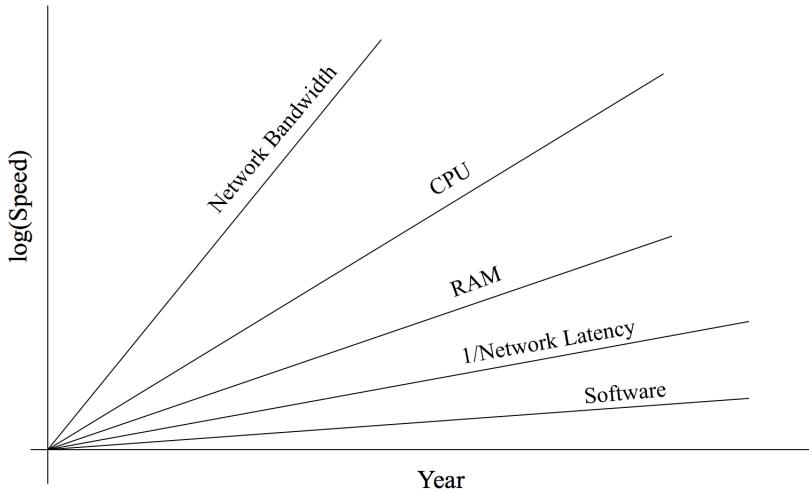


Figura 1.1: Esquema de avance del hardware en comparación con el desarrollo de software en áreas científicas. Tomada de *Supercomputing in plain english, Henry Neeman, OU Supercomputing Center for Education and Research, Univ. of Oklahoma, 2009*.

solución de dichas ecuaciones: los algoritmos numéricos son posteriormente implementados en programas de cómputo, lo cual se conoce como *Modelo computacional*. La figura 1.2 muestra un esquema de este proceso.

1.1.2. La entropía y la crisis del software

La entropía, nos indica el grado de desorden de un sistema termodinámico: podemos usar este mismo concepto durante la construcción de sistemas computacionales(véase [9]). En un sistema computacional la entropía siempre crece, a medida que éste sea modificado. Algunas ideas similares establecidas por Lehman [9] indican que:

- Un programa que es usado, se modificará.
- Cuando un programa es modificado su entropía crecerá.

La entropía provoca que el entendimiento y mantenimiento de un sistema sea un proceso complejo, además, ésta crece de manera exponencial. Por lo tanto, autores como; Grady Booch [10] indican que la crisis del software es la

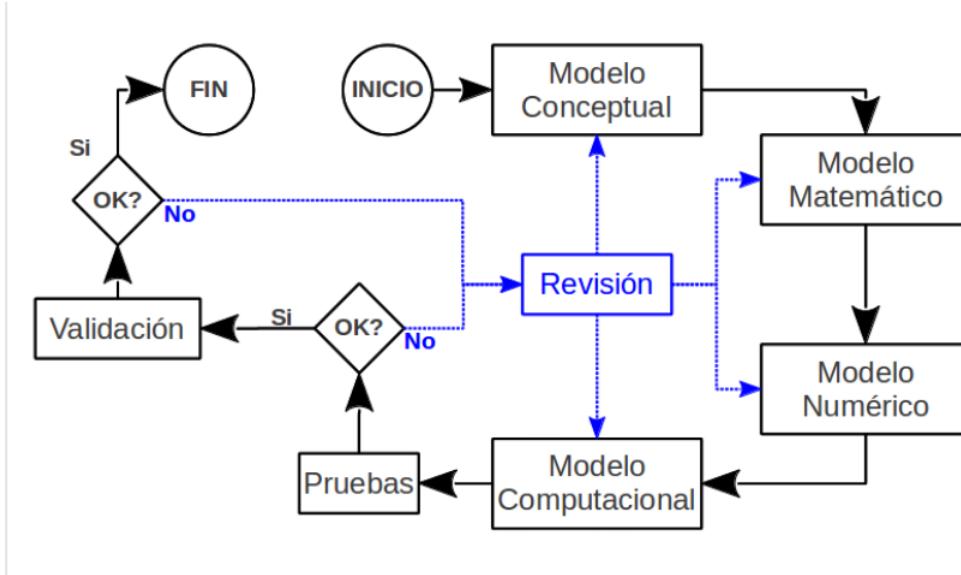


Figura 1.2: Proceso de desarrollo de software en computo científico

imposibilidad de desarrollar sistemas que sean simples de entender, mantener y de extender.

Ambas, la crisis del software y su entropía hacen que el avance del software sea un proceso inconsistente y con respecto a otras tecnologías, el software crezca muy poco. Resulta necesario entonces establecer estrategias que sean capaces de mantener controlados estos problemas.

1.1.3. Ingeniería de Software

Encontramos el software en toda infraestructura cibernetica, utilizada para la ciencia y la ingeniería. Es común definir al software como cualquier programa que está en ejecución en un dispositivo de procesamiento. Considerando la definición dada por Sommerville [11], en éste trabajo indicamos al software como un conjunto formado por: un sistema, documentación y la configuración asociada que permite que opere de manera correcta. En general, la construcción de software útil con bajos costos y tiempos cortos es complicado.

La ingeniería de software es una herramienta para la solución de problemas durante la producción de software [9], permite establecer un proceso de desarrollo ordenado y bien documentado, de tal manera que sea factible

su crecimiento futuro en el uso de nuevas tecnologías o de su reutilización, además permite la adquisición de conocimientos que pueden ser usados en proyectos subsecuentes.

Para llevar a cabo el proceso de desarrollo, la ingeniería de software hace uso de modelos de procesos, donde su propósito es facilitar la construcción de software, que éste sea de calidad y de bajo costo. Algunos de ellos son: cascada, evolutivo y en componentes [11].

1.2. Objetivos y metas

El objetivo principal de este trabajo es el diseño y la construcción de software científico para simular problemas de flujo en medios porosos mediante el método CVFE, usando buenas prácticas que aseguren calidad.

Las metas que nos hemos propuesto son:

- Realizar la construcción del sistema mediante una metodología de desarrollo de software basada en la norma ISO/IEC 29110 perfil básico.
- Implementar un modelo de proceso de desarrollo de software.
- Obtener los requerimientos funcionales y no funcionales del sistema, del problema y del método numérico de solución seleccionado.
- Proponer y diseñar una arquitectura para el sistema.
- Implementar el sistema con base en la arquitectura diseñada, con el fin de cumplir los requerimientos.
- Realizar pruebas de funcionalidad, de paquete y de tipo *benckmark*, para calibrar numéricamente el sistema.
- Resolver problemas de flujo en medios porosos.
- Entregar la documentación generada del sistema final.

1.3. Organización de la tesis

La tesis está organizada de la siguiente manera:

- Capítulo 2. Se describe el modelo matemático general de flujo en medios porosos. Además, se estudia el CVFEM, el cual fué utilizado para encontrar la solución a las ecuaciones gobernantes, mostrando el esquema de discretización y del tratamiento de las condiciones de frontera.
- Capítulo 3. Se describe la metodología de software utilizada para la construcción de la herramienta SYSG.
- Capítulo 4. Se resuelven los casos de estudio particulares de este trabajo, y se reportan los resultados.
- Capítulo 5. Se presentan las conclusiones finales.

Capítulo 2

Marco Teórico

En este trabajo estudiamos el flujo en medios porosos de una fase desde el punto de vista de la mecánica de medio continuo, es decir, desde un punto de vista macroscópico, en donde las discontinuidades a nivel atómico no son de interés. Para describir el comportamiento de un medio poroso, en este trabajo, se usan modelos matemáticos, los cuales en su mayoría vienen en forma de ecuaciones diferenciales ordinarias o parciales (*ecuaciones gobernantes*). Un modelo es capaz de aproximar el comportamiento de un sistema [12], debido a ésto, los usan ampliamente científicos e ingenieros.

Determinar la solución exacta que satisface una ecuación diferencial parcial, en un dominio dado, resulta en ocasiones una tarea complicada o imposible, debido a la falta de herramientas matemáticas, en estos casos, son utilizados métodos numéricos.

2.1. Flujo en medios porosos

Desde un punto de vista macroscópico, un medio poroso es un sistema continuo, el cual está formado por un material sólido y huecos [12] que pueden o no, estar conectados entre sí[13]. Normalmente al material sólido se le llama matriz sólida, a los huecos, poros y al volumen ocupado por los poros se le llama porosidad y se denota con la letra ϕ [12]. Dos clases de porosidad son usadas: la porosidad total y la porosidad efectiva, en donde la primera se refiere al volumen que ocupan los poros en el espacio y la segunda al volumen que ocupan solo los poros interconectados[13].

Cuando la matriz sólida contiene poros interconectados entre sí, los cuales

están completamente ocupados por un fluido, puede existir un movimiento del fluido. Predecir como se comporta este movimiento, ha sido un problema estudiado desde hace varias décadas.

Para estudiar el problema de flujo en medios porosos, en este trabajo se toman en cuenta las siguientes hipótesis [12]:

- La matriz sólida está saturada con el fluido.
- La matriz sólida es ligeramente elástica.
- El fluido es ligeramente compresible.
- La velocidad del fluido cumple la Ley de Darcy.
- No hay procesos de difusión del fluido.

2.2. Ecuación de Balance de Masa

Las ecuaciones gobernantes del fenómeno de flujo en medios porosos en una fase, se obtienen con ayuda de la formulación axiomática presentada por Herrera [12]. Como primer paso se identifica una familia de propiedades extensivas, en éste caso con ayuda de la porosidad ϕ y de la densidad ρ , se puede describir la masa del fluido definida como:

$$M = \int_{B(t)} \phi \rho dx$$

El segundo paso del método axiomático consiste en aplicar las ecuaciones de balance local(véase [12]) a cada propiedad extensiva con lo cual:

$$\frac{\partial(\phi\rho)}{\partial t} + \nabla \cdot (\rho \underline{u}) = q \quad (2.2.1)$$

donde se ha sustituido la velocidad de Darcy definida por $\underline{u} = \phi \underline{v}$ y se han tomado en consideración las hipótesis presentadas en la sección 2.1.

2.3. Flujo monofásico

Si suponemos que la matriz sólida es ligeramente compresible, podemos establecer la compresibilidad de la roca como:

$$c_R = \frac{1}{\phi} \frac{d\phi}{dp} \Rightarrow \phi \approx \phi^0 [1 + c_R (p - p^0)] \Rightarrow \frac{d\phi}{dp} \approx \phi^0 c_R$$

y para la compresibilidad del fluido se tiene:

$$c_f = \frac{1}{\rho} \frac{d\rho}{dp}$$

expandiendo la derivada temporal de la ecuación 2.2.1:

$$\frac{\partial(\phi\rho)}{\partial t} = \phi \frac{\partial\rho}{\partial p} \frac{\partial p}{\partial t} + \rho \frac{\partial\phi}{\partial p} \frac{\partial p}{\partial t}$$

con esto:

$$\frac{\partial(\phi\rho)}{\partial t} = \rho\phi c_f \frac{\partial p}{\partial t} + \rho p^0 c_R \frac{\partial p}{\partial t} = \rho\phi c_T \frac{\partial p}{\partial t} \quad (2.3.1)$$

en donde $c_T = \left(c_f + \frac{\phi^0}{\phi} c_R\right)$ representa la compresibilidad total, además la Ley de Darcy para flujo en una fase dice:

$$u = -\frac{1}{\mu} \underline{k} (\nabla p - \rho \underline{g}) \quad (2.3.2)$$

donde μ es la viscosidad, \underline{k} es el tensor de permeabilidad y \underline{g} es la fuerza de gravedad.

Finalmente tomado todo lo anterior en consideración se tiene que:

$$\phi\rho c_T \frac{\partial(p)}{\partial t} - \nabla \cdot \left(\rho \left[\frac{1}{\mu} \underline{k} (\nabla p - \rho \underline{g}) \right] \right) = q \quad (2.3.3)$$

La ecuación 2.3.3 es la ecuación de balance de masa para flujo en una fase en términos de presión, la cual será objeto de estudio en este trabajo.

2.4. CVFEM

El CVFEM(*Control Volume Finite Element Method*) [3] es un método de elemento finito que usa volúmenes de control, usado principalmente en problemas de transporte [14]. Su objetivo principal es la transformación de ecuaciones diferenciales parciales, en un sistema de ecuaciones lineales que al resolverse aproxime la solución de la ecuación original. Cuenta con dos grandes ventajas. Primero, es un método conservativo, es decir, cada volumen de control satisface el principio de conservación, segundo, permite el tratamiento de dominios con geometrías complicadas. Consiste en una serie de pasos bien establecidos:

- Discretización del dominio y construcción de volúmenes de control(VC).
- Integración de la ecuación diferencial en cada VC.
- Aproximación de las integrales para obtener una expresión discreta.
- Formulación del sistema lineal de ecuaciones aplicando las aproximaciones a cada VC, así como las condiciones de frontera.

2.4.1. Discretización usando CVFEM

Consideremos la ecuación 2.3.3, con las siguientes suposiciones:

- No se consideran efectos de la fuerza de gravedad, es decir $\underline{g} = 0$.
- El fluido y la matriz porosa son incompresibles, es decir ρ y ϕ son constantes.
- La viscosidad del fluido es constante.

con lo cual tenemos:

$$-\nabla \cdot (\underline{\underline{\Gamma}}_p \nabla p) = q(x, y) \quad (2.4.1)$$

donde: $\underline{\underline{\Gamma}}_p = \frac{\rho}{\mu} * \begin{bmatrix} k_{11} & 0 \\ 0 & k_{22} \end{bmatrix}$.

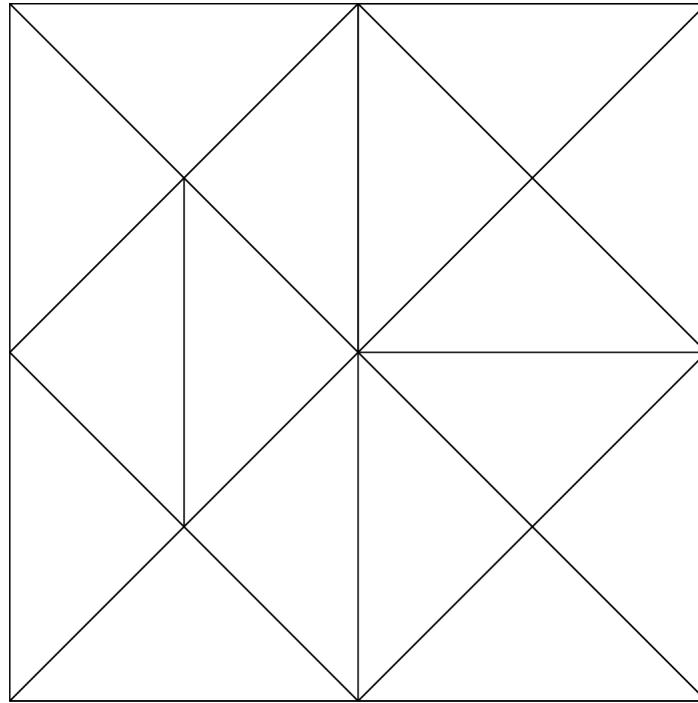


Figura 2.1: Dominio discretizado en elementos triangulares

2.4.2. Discretización del dominio y construcción de volúmenes de control(VC)

El primer paso es la discretización del dominio en elementos triangulares de tres nodos(Ver Fig. 2.1), para éste propósito se utiliza normalmente el algoritmo de Delaunay[15]. Para construir los VC, el baricentro de cada elemento triangular K se une con los puntos medios de sus aristas, de tal forma que queda dividido en tres áreas iguales (Ver Fig. 2.3).

Alrededor de cada vértice se forman secciones poligonales las cuales son los VC. Los volúmenes de control no se traslapan y son contiguos de tal manera que encierran a uno y sólo un vértice. Pueden distinguirse dos tipos, aquellos alrededor de un nodo interno y aquellos alrededor de un nodo de frontera(Ver Fig. 2.2).

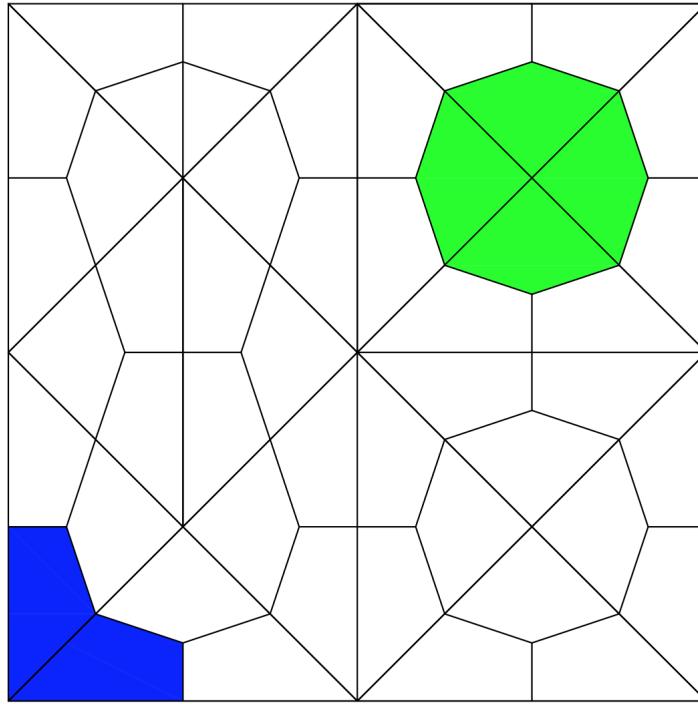


Figura 2.2: En la imagen se distinguen los dos tipos de volúmenes de control, alrededor de un nodo interno(verde) y alrededor de un nodo frontera(azul).

2.4.3. Integración de la ecuación diferencial en cada VC

Integramos la ecuación 2.4.1 con respecto a cada volumen de control(v_i), además p se debe cambiar sólo por la contribución del VC lo cual se denota como p_h , de lo cual:

$$-\int_{v_i} \nabla \cdot (\underline{\underline{\Gamma}}_p \nabla p_h) dV = \int_{v_i} q(x, y) dV \quad (2.4.2)$$

usando el teorema de la divergencia:

$$-\int_{\partial v_i} \underline{\underline{\Gamma}}_p \nabla p_h \cdot \underline{n} dS = \int_{v_i} q(x, y) dV$$

para resolver la integral, se trabaja sobre la contribución de cada elemento triangular (Ver figura. 2.3), de lo cual:

$$-\int_{mamc+mcmd} \underline{\underline{\Gamma_p}} \nabla p_h \cdot \underline{n} dS = \int_{v_i} q(x, y) dV$$

de esto:

$$-\int_{m_a}^{m_c} \left(\underline{\underline{\Gamma_p}} \nabla p_h \right) \cdot \underline{n} dS - \int_{m_c}^{m_d} \left(\underline{\underline{\Gamma_p}} \nabla p_h \right) \cdot \underline{n} dS = \int_{v_i} q(x, y) dV$$

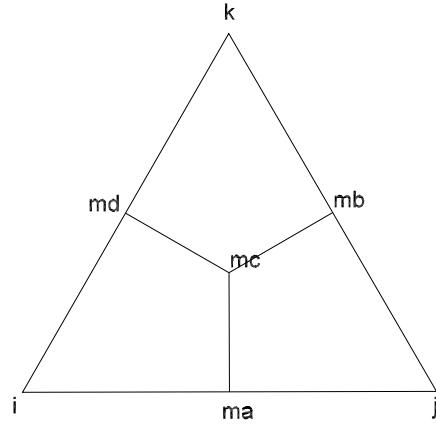


Figura 2.3: Elemento triangular etiquetado en sentido contrario a las manecillas del reloj, con las letras i, j y k. Se muestra la unión del baricentro(mc) con las medianas ma,mb y md, como resultado, el elemento triangular queda dividido en tres partes iguales.

trabajando sobre:

$$-\int_{m_a}^{m_c} \left(\underline{\underline{\Gamma_p}} \nabla p_h \right) \cdot \underline{n} dS$$

se tiene que:

$$-\int_{m_a}^{m_c} n_1 \frac{\partial p_h}{\partial x_1} dS - \int_{m_a}^{m_c} n_2 \frac{\partial p_h}{\partial x_2} dS$$

notemos que la integral es de línea, por lo cual el resultado es: la distancia que hay entre m_c y m_a , si la resolvemos, entonces se tiene:

$$-n_1 \frac{\partial p_h}{\partial x_1} (m_c - m_a) - n_2 \frac{\partial p_h}{\partial x_2} (m_c - m_a)$$

además:

$$n_1 = \frac{m_{c,2} - m_{a,2}}{|m_a m_c|}, \quad n_2 = \frac{m_{a,1} - m_{c,1}}{|m_a m_c|}$$

con lo cual:

$$-\frac{\partial p_h}{\partial x_1} (m_{c,2} - m_{a,2}) - \frac{\partial p_h}{\partial x_2} (m_{a,1} - m_{c,1})$$

de la misma forma, se tiene:

$$-\int_{m_c}^{m_d} \left(\underline{\underline{\Gamma}}_p \nabla p \right) \cdot \underline{n} dS$$

es igual a:

$$-\frac{\partial p_h}{\partial x_1} (m_{a,2} - m_{d,2}) - \frac{\partial p_h}{\partial x_2} (m_{d,1} - m_{a,1})$$

sumando y simplificando:

$$\frac{\partial p_h}{\partial x_1} (m_{a,2} - m_{d,2}) + \frac{\partial p_h}{\partial x_2} (m_{d,1} - m_{a,1}) \quad (2.4.3)$$

Notemos las derivadas parciales que acompañan a la expresión 2.4.3 para calcularlas debemos hacer alguna suposición acerca de la variable dependiente p_h .

2.4.4. Funciones de interpolación

Para aproximar la variable dependiente p_h en cada triángulo K hacemos uso de funciones de interpolación, en este trabajo se usan como funciones de interpolación, funciones lineales (véase [3]).

Interpolación Lineal

Se aproxima la variable dependiente con la siguiente aproximación lineal 2.4.4

$$p_h = p_i \lambda_i + p_j \lambda_j + p_k \lambda_k \quad (2.4.4)$$

Donde las λ 's son funciones base o de forma, las cuales tienen el propósito de aproximar el valor de la variable dependiente a trozos sobre cada elemento, se definen en función de los vértices de cada elemento y deben cumplir dos restricciones, las cuales son:

- La evaluación de la función base sobre el vértice para el cual es definido, es igual a uno, en cualquier otro vértice es cero.
- La suma de las funciones base es igual a 1.

Pueden ser calculadas como sigue:

$$\begin{pmatrix} \lambda_i \\ \lambda_j \\ \lambda_k \end{pmatrix} = \frac{1}{2|K|} \begin{pmatrix} c_i & a_i & b_i \\ c_j & a_j & b_j \\ c_k & a_k & b_k \end{pmatrix} \begin{pmatrix} 1 \\ x_1 \\ x_2 \end{pmatrix} \quad (2.4.5)$$

en donde $|K|$ es el área del triángulo K y los coeficientes son calculados con 2.4.6.

$$a_i = y_j - y_k, \quad b_i = -(x_j - x_k), \quad c_i = x_j y_k - x_k y_j \quad (2.4.6)$$

son permutados en orden natural para encontrar los restantes, con todo ésto podemos aproximar el gradiente de la siguiente manera 2.4.7.

$$\frac{\partial \lambda_l}{\partial x_1} = \frac{a_l}{2|K|}, \quad \frac{\partial \lambda_l}{\partial x_2} = \frac{b_l}{2|K|}, \quad l = i, j, k \quad (2.4.7)$$

Siguiendo con la ecuación 2.4.3, y con ayuda de la expresión para calcular medianas, se tiene:

$$m_{a,2} - m_{d,2} = \frac{a_i}{2} = |K| \frac{\partial \lambda_i}{\partial x_1}, \quad m_{d,1} - m_{a,1} = \frac{b_i}{2} = |K| \frac{\partial \lambda_i}{\partial x_2}$$

con lo cual:

$$\frac{\partial p_h}{\partial x_1} (m_{a,2} - m_{d,2}) + \frac{\partial p_h}{\partial x_2} (m_{d,1} - m_{a,1}) = |K| \frac{\partial p_h}{\partial x_1} \frac{\partial \lambda_i}{\partial x_1} + |K| \frac{\partial p_h}{\partial x_2} \frac{\partial \lambda_i}{\partial x_2}$$

aplicando las derivadas parciales:

$$|K| \frac{\partial p_h}{\partial x_1} \frac{\partial \lambda_i}{\partial x_1} + |K| \frac{\partial p_h}{\partial x_2} \frac{\partial \lambda_i}{\partial x_2} = |K| p_i \frac{\partial \lambda_i}{\partial x_1} \frac{\partial \lambda_i}{\partial x_1} + \\ |K| p_j \frac{\partial \lambda_j}{\partial x_1} \frac{\partial \lambda_i}{\partial x_1} + |K| p_k \frac{\partial \lambda_k}{\partial x_1} \frac{\partial \lambda_i}{\partial x_1} + |K| p_i \frac{\partial \lambda_i}{\partial x_2} \frac{\partial \lambda_i}{\partial x_2} + |K| p_j \frac{\partial \lambda_j}{\partial x_2} \frac{\partial \lambda_i}{\partial x_2} + |K| p_k \frac{\partial \lambda_k}{\partial x_2} \frac{\partial \lambda_i}{\partial x_2}$$

podemos reagrupar los términos comunes, por ejemplo:

$$|K| p_i \frac{\partial \lambda_i}{\partial x_1} \frac{\partial \lambda_i}{\partial x_1} + |K| p_i \frac{\partial \lambda_i}{\partial x_2} \frac{\partial \lambda_i}{\partial x_2}$$

de ésto:

$$|K| \left(\frac{\partial \lambda_i}{\partial x_1}, \frac{\partial \lambda_i}{\partial x_2} \right) \cdot \left(\frac{\partial \lambda_i}{\partial x_1}, \frac{\partial \lambda_i}{\partial x_2} \right)$$

de la misma forma se hace para los términos restantes y finalmente tenemos:

$$|K| p_i \frac{\partial \lambda_i}{\partial x_1} \frac{\partial \lambda_i}{\partial x_1} + |K| p_j \frac{\partial \lambda_j}{\partial x_1} \frac{\partial \lambda_i}{\partial x_1} + |K| p_k \frac{\partial \lambda_k}{\partial x_1} \frac{\partial \lambda_i}{\partial x_1} + |K| p_i \frac{\partial \lambda_i}{\partial x_2} \frac{\partial \lambda_i}{\partial x_2} + \\ |K| p_j \frac{\partial \lambda_j}{\partial x_2} \frac{\partial \lambda_i}{\partial x_2} + |K| p_k \frac{\partial \lambda_k}{\partial x_2} \frac{\partial \lambda_i}{\partial x_2} = |K| \nabla \lambda_i \cdot \nabla \lambda_i p_i + |K| \nabla \lambda_j \cdot \nabla \lambda_i p_j + |K| \nabla \lambda_k \cdot \nabla \lambda_i p_k \quad (2.4.8)$$

la suma de las funciones de forma deben sumar uno, lo cual implica:

$$\lambda_i + \lambda_j + \lambda_k = 1$$

aplicando el gradiente y despejando $\nabla \lambda_i$, se tiene:

$$\nabla \lambda_i = \nabla \lambda_j - \nabla \lambda_k \quad (2.4.9)$$

sustituyendo 2.4.9 en 2.4.4 y simplificando se tiene finalmente:

$$- T_{ij} (p_j - p_i) - T_{ik} (p_k - p_i) \quad (2.4.10)$$

Donde T_{ij} y T_{ik} son llamados coeficientes de transmisibilidad y están dados por:

$$T_{ij} = -|K| \nabla \lambda_j \cdot \nabla \lambda_i, T_{ik} = -|K| \nabla \lambda_k \cdot \nabla \lambda_i$$

La expresión 2.4.10 debe ser aplicada a cada elemento triangular.

2.4.5. Término Fuente

Tomemos el término que representa una fuente:

$$\int_{v_i} q(x, y) dV \quad (2.4.11)$$

Es necesario resolver una integral sobre los volúmenes de control. Para dar solución a esta integral, es posible utilizar algún método numérico de aproximación como cuadratura Gaussiana. En el CVFEM es posible calcular la contribución de la fuente en cada elemento triangular, por medio de la aproximación [16]:

$$\int_{v_{ma-md}} q(x, y) dV = q(x_i, y_i) \Delta V = q(x_i, y_i) \frac{|K|}{3} \quad (2.4.12)$$

donde x_i, y_i corresponden a las coordenadas del nodo que está encerrado por el VC y $|K|$ es el área del elemento triangular. Por lo tanto:

$$\int_{v_i} q(x, y) dV = q(x_i, y_i) \sum_{j=1}^n \frac{|K_j|}{3} \quad (2.4.13)$$

donde $|K_j|$ son los elementos de triangulares que rodean al nodo i.

2.4.6. Condiciones de Frontera

Para encontrar una solución única de una ecuación diferencial parcial, es necesario contar con condiciones de frontera y con condiciones iniciales, en éste caso se dice que el problema fue bien planteado [12]. Las condiciones de frontera o de contorno especifican el valor que tendrá la variable dependiente en la frontera. Para problemas con variables en el espacio, se tienen tres tipos de condiciones:

- Dirichlet.

- Neumann.
- Robin.

Para este trabajo solo nos interesan las condiciones tipo Dirichlet y tipo Neumann.

Condiciones tipo Dirichlet

Un problema con condiciones tipo Dirichlet se escribe como sigue:

$$\Delta u = f_{\Omega} \quad \text{en } \Omega \quad (2.4.14)$$

$$u = \nu(\underline{x}, \underline{y}) \quad \text{sobre } \partial\Omega \quad (2.4.15)$$

En el CVFEM, los nodos que se encuentran sobre una frontera que tiene condiciones tipo Dirichlet no necesitan ser calculados, basta con sustituir el valor de la frontera en las ecuaciones en donde aparezca como variable dicho nodo.

Condiciones tipo Neumann

Las condiciones tipo Neumann se definen como sigue:

$$\frac{\partial u}{\partial n} = \nu(\underline{x}, \underline{y}) \quad (2.4.16)$$

En el CVFEM, los nodos que se encuentran en una frontera Neumann, deberán ser incluidos en el sistema de ecuaciones para ser resueltos simultáneamente con los nodos interiores.

2.5. Triangulación de Delaunay

Muchas de las áreas que están relacionadas con resolver ecuaciones diferenciales parciales utilizan triangulaciones o mallas como apoyo en su solución. La triangulación de Delaunay es una estructura geométrica, en dos dimensiones, tiene ventaja sobre otras posibles triangulaciones de puntos fijos. La triangulación de Delaunay maximiza el ángulo mínimo. También optimiza otros criterios geométricos relacionados con la precisión de interpolación. Las

triangulaciones de Delaunay se han estudiado minuciosamente, en particular, para la geometría computacional y se disponen de algoritmos excelentes para construirlos.

La triangulación de *Delaunay* de un conjunto de puntos S , introducido por Boris Nikolaevich Delaunay en 1934, indica que ningún punto en S está en el interior de cualquier triángulo circunscrito en un disco.

Definición 2.1 *Delaunay*. En el contexto de un conjunto finito de puntos S , un triángulo es *Delaunay* si sus vértices están en S y su disco abierto circunscrito es vacío. i.e., no contiene puntos de S .

Note que cualquier número de puntos en S puede estar en un triángulo de Delaunay circunscrito. Una arista es *Delaunay* si sus vértices están en S y si tiene a los más una circunferencia.

Una **Triangulación de Delaunay** de S , denotado por $\text{Del}S$, es una triangulación de S en la cual cada triángulo es de Delaunay, como se muestra en la figura 2.4.

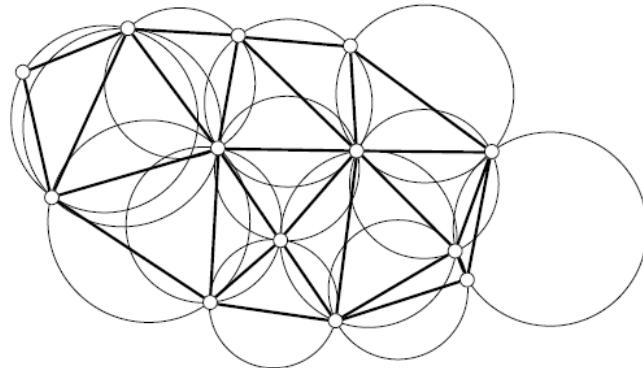


Figura 2.4: Cada triángulo en una Triangulación de Delaunay tiene un disco abierto vacío circunscrito.

Definición 2.2 *Fuertemente Delaunay* En el contexto de un conjunto finito de puntos S , un triángulo τ es fuertemente Delaunay si sus vértices están en S y su disco circunscrito cerrado no contiene puntos en S excepto los

vértices de τ . Una arista e es fuertemente Delaunay si sus vértices están en S y si tiene a lo más un disco circunscrito cerrado que no contiene puntos en S excepto los vértices de e . Cada punto en S es vértice fuertemente Delaunay.

Considere dos ejemplos de aristas fuertemente Delaunay. Primero, cada arista en la frontera de una triangulación de S es fuertemente Delaunay. La figura 2.5 muestra la razón de ésto. Segundo, la arista que conecta un punto $v \in S$ con su vecino más cercano $w \in S$ es fuertemente Delaunay, por que el disco cerrado más pequeño que contiene a v y w no contiene ningún otro punto en S . Cada triangulación de Delaunay conecta cada vértice con su vecino más cercano.

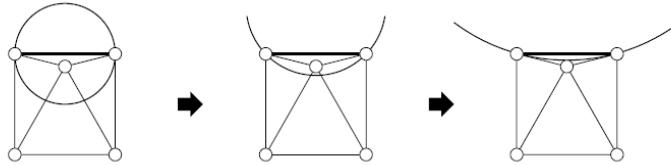


Figura 2.5: Cada arista en la frontera de una triangulación convexa es fuertemente Delaunay, debido a que siempre es posible encontrar un disco vacío que contiene los puntos finales y ningún otro vértice.

Las triangulaciones de Delaunay son valiosas por que optimizan varios criterios geométricos: el ángulo más pequeño, el disco circunscrito más grande y el disco de contención mínimo más grande, todos éstos elementos aquí presentados son demostrados y estudiados de manera más profunda en [15] y [17].

En este trabajo utilizamos el software Gmsh [18] el cual está basado en triangulaciones de Delaunay para construir mallas en 2D y 3D sobre dominios irregulares.

Capítulo 3

Desarrollo del Sistema

En muchas ocasiones, los científicos que desarrollan sistemas para dar solución a problemas específicos, no le dan importancia al uso de buenas prácticas de desarrollo de software. Como consecuencia sus sistemas no tienen la capacidad de evolucionar y de ser reutilizados por otros científicos, además su mantenimiento puede llegar a ser una tarea muy complicada.

El objetivo principal de este trabajo es la construcción de software científico para la simulación de flujo en medios porosos. Para conseguirlo la norma ISO/IEC 29110 Perfil Básico (véase el apéndice A), fué adaptada para su uso en la construcción de software científico. Esta adaptación está descrita en la tesis de maestría: “Implementación del Estándar ISO/IEC 29110 Perfil Básico para el desarrollo de software del tipo científico” [4].

En nuestro caso, la construcción de un software para la simulación de flujo en medios porosos tiene las siguientes características:

- Cambios constantes en sus requerimientos.
- Recursos limitados.
- La experiencia en el CVFEM es limitada.
- El sistema será de menos de 500,000 líneas de código.

La implementación de la norma en este trabajo se realizó de forma evolutiva. Para trabajar en forma evolutiva se planearon 9 iteraciones de desarrollo con duración de un mes cada una, cada iteración está bien documentada. Se presenta exclusivamente la documentación generada en la última iteración, para acceder a la documentación completa, véase el Apéndice D.

3.1. Proceso de Administración del Proyecto

La norma ISO/IEC 29110 contempla dos procesos: la administración y la implementación de software. La administración se encarga de establecer y asignar tareas, tal que la construcción de software científico sea un proceso ordenado y documentado. En la adaptación utilizada, la administración cuenta con cuatro actividades, las cuales se pueden ver en la figura 3.1.

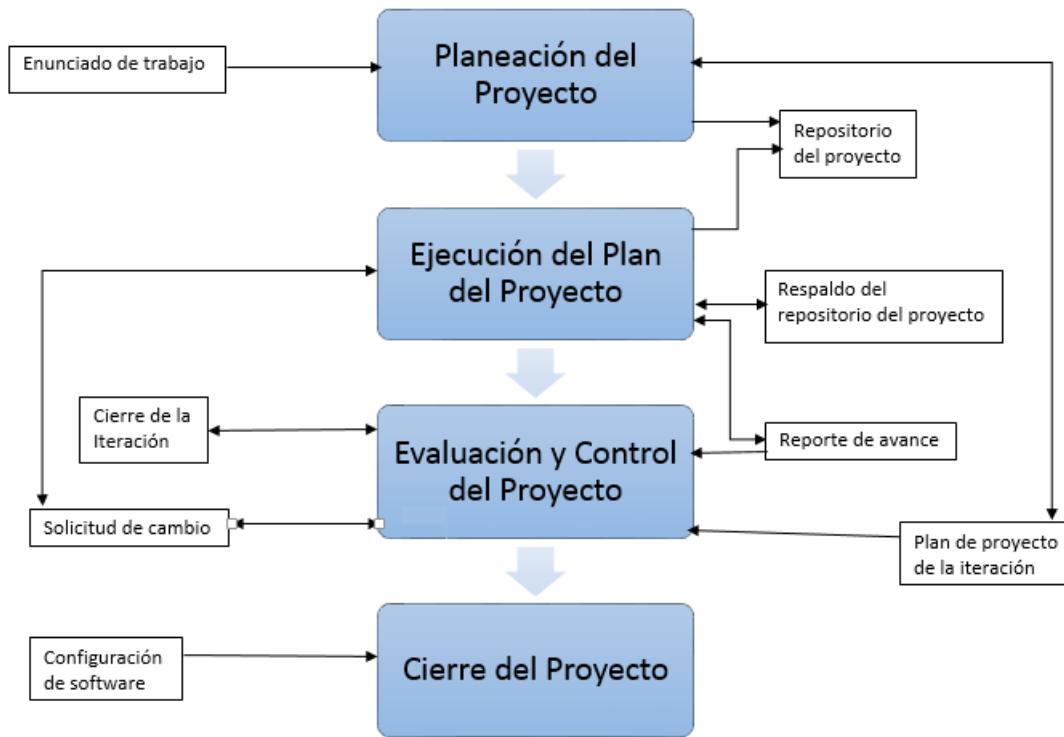


Figura 3.1: Actividades del proceso de Administración del Proyecto. Toma-dada de “*Implementación del Estándar ISO/IEC 29110 Perfil Básico para el desarrollo de software del tipo científico*” [4].

Cada actividad tiene las responsabilidades siguientes:

- **Planeación del proyecto.** Generar un Plan del Proyecto, encargado de la definición de objetivos, de establecer y asignar tareas, del tratamiento de los cambios y de la identificación de riesgos.
- **Ejecución del Plan del Proyecto.** Verificar el estado de las tareas, solicitudes de cambios e identificación de nuevos riesgos.
- **Evaluación y Control del proyecto.** Examinar las tareas realizadas, con el fin de tomar decisiones y acciones correctivas ante problemas que afecten al proyecto.
- **Cierre del proyecto.** Entregar un producto que cumpla con los objetivos.

Los documentos generados para cada actividad en el proceso de administración en nuestro trabajo fueron:

- **Plan del Proyecto.** Este documento esta asociado a la primera actividad, es decir aquí se definen: los objetivos, las tareas con sus responsables, cambios y riesgos de cada iteración.
- **Reporte de Seguimiento.** Esta asociado con la segunda y tercera actividad, su fin es establecer una herramienta que permita al equipo de trabajo determinar: el estado actual del proyecto, desviaciones de los objetivos, cambios y riesgos.
- **Cierre del Plan del Proyecto.** Esta asociado con la última actividad del proceso de administración, se encarga de dar una visión completa del avance del proyecto en cada iteración de forma resumida.

Se incluyen los documentos generados durante la última iteración para el proceso de administración.

Plan del Proyecto

1. Objetivos

ID	Objetivos
1	Realizar pruebas de tipo Benchmark.
2	Realizar cambios necesarios.
3	Validación del sistema.
4	Revisar documentación del sistema.

Cuadro 3.1: Objetivos de la iteración.

2. Tareas

ID	Funcionalidad o Actividad	Responsable
1	<i>Definir pruebas Benchmark</i>	
	1.1 Obtener problemas con solución analítica	Administrador del proyecto.
	1.2 Entender las pruebas a realizar	Desarrollador.
2	<i>Realizar Cambios</i>	
	2.1 Realizar los cambios necesarios al sistema para poder iniciar y aplicar las pruebas Benchmark	Desarrollador.
3	<i>Validación</i>	
	3.1 Aplicar el sistema a la pregunta inicial a resolver	Administrador, Desarrollador.
4	<i>Documentación</i>	
	4.1 Verificar que la documentación esté completa y disponible	Desarrollador.

Cuadro 3.2: Tareas a ser realizadas en la iteración.

3.Cambios

ID	Producto	Descripción	Solicitante
1	CVFEM	Modificación de la funcionalidad que construye la matriz y la del vector b de la ecuación $\underline{A}\underline{x} = \underline{b}$.	Desarrollador
2	Visualizador	Construcción de una función para la graficación de una superficie.	Administrador, Desarrollador
3	CVFEM	Cálculo de la funcionalidad de permeabilidad aleatoria.	Desarrollador

Cuadro 3.3: Cambios a realizar durante la iteración.

4.Riesgos

Nombre	Descripción	Plan de contingencia	Impacto	Estado del riesgo
Fecha de entrega	La fecha de conclusión del proyecto está próxima y aún falta mucho por realizar.	Solicitar una extensión de tiempo.	Alto	Identificado
Reuniones	Reuniones con el Administrador y experto del tema, deben ser más frecuentes.	Constante comunicación por otros medios: E-mail, Teléfono, etc.	Alto	Controlado

Cuadro 3.4: Riesgos encontrados.

Reporte de Seguimiento

Proyecto: *[Simulación de flujo en medios porosos usando CVFE]*

Periodo a Reportar: 1-11-2016.] al [30-11-2016]

1. Reporte de Actividades

Integrante	Actividad u Objetivo	Tiempo Total	Estado	Observaciones
Desarrollador	Pruebas Benchmark	45 hr	Finalizado	La obtención de las pruebas de parte del administrador tomó dos horas, el entendimiento y realización de las pruebas tomó mucho tiempo, fueron 3 pruebas a realizar, y en cada prueba se realizaron los cambios necesarios para el buen funcionamiento del sistema.
Desarrollador	Cambios	40 hr	Finalizado	Los cambios realizados fueron aplicados en conjunto con las pruebas Benchmark, se realizaron cambios en el paquete del malla, CVFEM.
Desarrollador	Validación	40 hr	Pendiente	La validación se encuentra pendiente, debido a los cambios que se deben realizar.
Desarrollador	Documentación	16 hr	Pendiente	La documentación está siendo revisada y actualizada, aún falta por definir completamente ciertos documentos.
Total de Horas trabajadas		141		

Cuadro 3.5: Estado actual del proyecto.

2.Tareas

Actividad y/o tarea	Integrante	Fecha de entrega	Fecha real
Obtener la definición del problema benchmark	Desarrollador	[2-11-2016]	[2-11-2016]
Entendimiento de las pruebas	Desarrollador	[4-11-2016]	[16-11-2016]
Realizar cambios	Desarrollador	[11-11-2016]	[16-11-2016]
Validación	Desarrollador	[22-11-2016]	Pendiente
Documentación	Equipo de trabajo	[22-11-2016]	Pendiente

Cuadro 3.6: Avance de tareas.

3.Productos

ID	Producto	Estado
1	Tratamiento del Dominio	Terminado
2	CVFEM	Terminado
3	Almacenamiento de la Matriz <u>A</u>	Terminado
4	Cálculo del vector <u>x</u>	Terminado
5	Almacenamiento del vector <u>x</u> y de <u>b</u>	Terminado
6	Visualizador	Iniciado

Cuadro 3.7: Productos.

4.Cambios

ID	Producto	Descripción	Solicitante	Estado
1	Tratamiento del Dominio	<p>Se realizaron los siguientes cambios:</p> <ul style="list-style-type: none"> ■ Reordenamiento de nodos. ■ Distinción entre nodos frontera e interiores. ■ Tratamiento de nodos Neumann y Dirichlet. 	Desarrollador	Realizado
2	CVFEM	<p>Se realizaron los siguientes cambios:</p> <ul style="list-style-type: none"> ■ Análisis de permeabilidad. ■ Distinción de subdominios. ■ Realización de tipos de matrices para nodos tipo Neumann y Dirichlet. 	Desarrollador	Realizado

Cuadro 3.8: Cambios.

5.Riesgos

Nombre	Descripción	Plan de contingencia	Impacto	Estado del riesgo
Tiempo para realizar cambios	El tiempo de entrega y el tiempo para la realización de los cambios es insuficiente.	Solicitar una extensión de tiempo.	Alto	Identificado
Cambios que puedan comprometer el funcionamiento del sistema	Al momento de realizar los cambios, el sistema pueda dejar de funcionar.	Tener las versiones del código disponibles, en caso de que no se halle la solución al problema de manera inmediata.	Alto	Controlado

Cuadro 3.9: Riesgos encontrados.

6.Resumen

Tareas	Cambios	Riesgos
A tiempo: 4	Solicitados: 5	Encontrados 2
Retrasadas: 2	Rechazados: 0	Resueltos 1
Adelantadas: 0	Realizados: 5	Postergados 1
Postergadas: 2	Postergados: 0	

Cuadro 3.10: Resumen del Reporte de Seguimiento.

Cierre del Plan del Proyecto

1. Resumen

Productos	Cambios
A tiempo: 5	Solicitados: 0
Retrasados: 0	Rechazados: 0
Adelantados: 0	Realizados: 7
Postergados: 1	Postergados: 0

Riesgos	Actividades u Objetivos
Encontrados: 2	Postergados: 2
Resueltos: 2	Alcanzados: 2

Cuadro 3.11: Resumen del Cierre del Plan del Proyecto.

2. Reporte de productos.

Nombre del producto	Cambios		Estado del producto
	Encontrados	Corregidos	
Tratamiento del Dominio	3	3	/Terminado
CVFEM	3	3	Terminado
Almacenamiento de la matriz <u>A</u>	0	0	/Terminado
Cálculo del vector <u>x</u>	0	0	/Terminado
Almacenamiento del vector <u>x</u> y del vector <u>b</u>	0	0	Terminado

Cuadro 3.12: Reporte de Productos.

3.Retroalimentación.

- Terminar la validación del sistema.
- Iniciar el cierre del proyecto.

4.Comentarios

Se tiene el sistema en un estado próximo a finalizar, faltan ciertas modificaciones al sistema relacionadas con la validación, una vez realizadas deben de ser aprobadas por el Administrador. El trabajo es complejo y en ésta parte del proyecto se necesita de la experiencia y apoyo del experto en el área.

Aún no se tiene la certeza del tiempo que se necesita para terminar estas modificaciones pero se trabaja para que todo el sistema esté listo antes de diciembre del 2016, esperando presentar el trabajo para evaluación en el mes de enero del 2017.

3.2. Proceso de Implementación de Software

Su propósito es la construcción de un sistema ejecutable. Cuenta con 7 actividades las cuales se pueden ver en la figura 3.2.

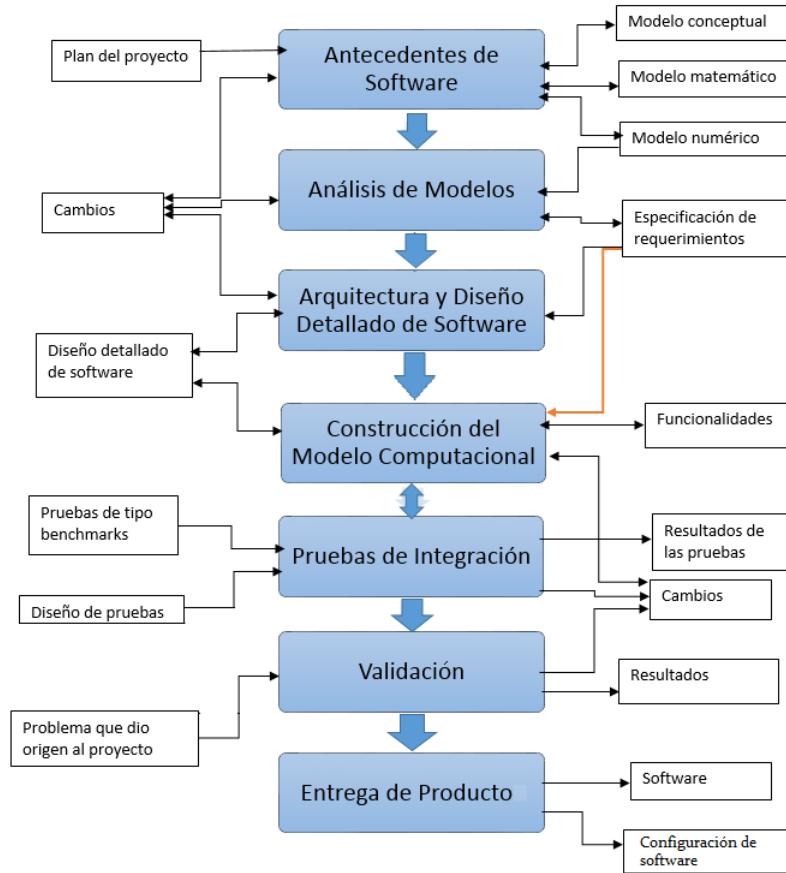


Figura 3.2: Actividades del proceso de Implementación de Software. Toma- da de “*Implementación del Estándar ISO/IEC 29110 Perfil Básico para el desarrollo de software del tipo científico*” [4].

3.2.1. Antecedentes de Software

Su principal objetivo es normalizar el conocimiento del problema entre todos los miembros del equipo de trabajo. Se hace un estudio minucioso del problema a resolver, se determina: el modelo conceptual, el modelo matemático y el modelo computacional.

3.2.2. Análisis de Modelos.

Esta actividad es la encargada de la toma de los requerimientos. Los requerimientos determinan los objetivos de un sistema, es decir, indican: qué debe hacer y qué restricciones tendrá. Normalmente los requerimientos se clasifican en dos tipos: funcionales y no funcionales [11]. Los requerimientos funcionales, determinan que debe hacer un sistema a una entrada dada, mientras que los no funcionales se refieren a las propiedades del sistema.

3.2.3. Arquitectura y Diseño Detallado de Software

Para satisfacer las necesidades identificadas por los requerimientos, normalmente los sistemas se descomponen en subsistemas que se comunicarán entre sí para resolver el problema. Este conjunto de subsistemas y la manera en que interactúan es llamada la arquitectura del sistema [11]. Los científicos normalmente requieren resolver problemas que cuentan con requerimientos semejantes. Por lo tanto, resulta indispensable elegir con cuidado una arquitectura de software que sea fácilmente escalable y robusta, de tal manera que pueda ser reutilizada en proyectos subsecuentes.

3.2.4. Construcción del Modelo Computacional

Consiste en la construcción del código y su integración. La integración del sistema, es el proceso de comunicar todos los módulos que fueron desarrollados de manera independiente, según lo indicado en la arquitectura. En este trabajo se utilizó la programación orientada a objetos (POO) [19]. En la POO se crean objetos los cuales interactúan para resolver un problema. Cada objeto tiene una identidad que lo diferencia de otros. Cada objeto tiene un estado interno (variables) que puede cambiar con la interacción con otros objetos. Cada objeto tiene un comportamiento (métodos), que permite

realizar algunas tareas específicas. Los objetos se agrupan en clases, las cuales definen las variables y métodos. Por ejemplo, podemos definir una clase de nombre Mesh, la cual puede definir variables como: número de vértices y de elementos; y métodos cómo: construir la malla, proporcionar el número de vértices, etc. En general, cada objeto puede solicitar información de otro objeto y utilizar los servicios de los demás objetos.

3.2.5. Pruebas de Integración

Para garantizar el funcionamiento correcto del software se usan pruebas. El procedimiento general de las pruebas es, ir de lo particular a lo general. En este trabajo fueron identificadas tres tipos de pruebas:

- Por funcionalidad.
- Por paquete.
- De Benchmark.

El objetivo de las primeras dos es probar partes pequeñas del sistema, de forma tal que se descubran defectos individuales [11], mientras las últimas tratan al sistema por completo.

Por Funcionalidad

Cada función codificada debe ser sujeta a un proceso de validación. Normalmente estas pruebas se dejan como responsabilidad del desarrollador y no son documentadas, debido a que en grandes sistemas se tienen muchas funciones y la documentación de éstas puede ser un proceso no significativo.

Por paquete

Es necesario hacer pruebas de caja negra sobre cada subsistema construido, es decir, alimentamos al subsistema con una entrada conocida y esperamos obtener también una salida conocida. En este tipo de pruebas, se debe establecer un documento, que permita a otros usuarios revisar con detalle el procedimiento de las pruebas.

Benchmark

Una vez que el sistema esta completo, se prueba por medio de problemas para los cuales se conoce de manera exacta la solución, se debe establecer un documento, con las especificaciones de la prueba.

3.2.6. Validación

Una vez que se alcanzó la suficiente confianza en el sistema gracias a las pruebas, se procede a resolver el problema que dio origen al proyecto.

3.2.7. Entrega de Productos

Se entrega el software científico final.

Los documentos generados para cada actividad en el proceso de implementación en nuestro trabajo fueron:

- **Marco Teórico.** Corresponde a la primera actividad del proceso de implementación. Contiene el problema a resolver y una normalización de conceptos para el equipo de trabajo.
- **Especificación de Requerimientos.** Contiene todos los requerimientos identificados del proyecto.
- **Arquitectura y Diseño Detallado de Software.** Especifica una normatividad que deberá seguirse durante la construcción del sistema, la arquitectura seleccionada y el detalle de cada subsistema.
- **Pruebas de Integración.** Muestra de forma detallada las pruebas para que puedan ser reproducidas.
- **Validación.** Muestra la solución al problema que originó el proyecto.

Se incluyen los documentos generados durante la última iteración para los requerimientos, la arquitectura y las pruebas de cada paquete. Se dejan de forma exclusiva los documentos relacionados a las pruebas de Benchmark y a la validación para el capítulo 4.

Especificación de Requerimientos

1. Descripción general del problema

El sistema debe ser capaz de leer un dominio en dos dimensiones, construido con la herramienta gmsh según la triangulación de Delaunay. Las especificaciones del archivo se encuentran en el apéndice B. Con lo cual debe ser capaz de almacenar los nodos y los elementos triangulares, los nodos deben ser ordenados respecto a su coordenada y y después a su coordenada x .

Debido a la discretización del problema (véase el Marco Teórico 2), es necesaria la construcción de un sistema de ecuaciones donde la matriz generada será para cada nodo incógnita y debe considerar condiciones de frontera. Dado que el tipo de matriz obtenido con este método es dispersa no estructurada y con el fin de optimizar el uso de recursos computacionales se debe evitar el almacenamiento de datos no significativos. Existe la necesidad de resolver el sistema de ecuaciones lineales rápidamente, por lo cual métodos directos, como la factorización LU, no son utilizados, debido a que son ineficientes en términos de computación [20], en este caso son usados métodos iterativos básicos y métodos del subespacio de Krilov.

El sistema de ecuaciones es resuelto con métodos numéricos de álgebra lineal. Para visualizar la solución obtenida se utiliza una herramienta externa, por lo tanto esta debe ser almacenada en un archivo con el siguiente formato, coordenada x , coordenada y y solución en ese punto.

2. Requerimientos funcionales

2.1. Tratamiento del Dominio de interés

- Leer el contenido de un archivo *.msh.
- Almacenar en memoria las coordenadas x, y de cada nodo de la discretización.
- Almacenar en memoria los nodos soporte de cada elemento triangular; es decir, los nodos que conforman a cada elemento triangular.

- Reordenar los nodos, primero por su coordenada y y después por su coordenada x , por ejemplo:

Coordenada x	Coordenada y
0	0
0.5	0
1	0
0.2	0.1
0.5	0.4
0.6	0.5
0.3	0.7

- Reordenar los nodos soporte de cada elemento triangular según la nueva numeración de los nodos.
- Distinguir los nodos que están en el contorno del dominio y aquellos que están en el interior.
- Distinguir los nodos de contorno en dos tipos:
 - Neumann.
 - Dirichlet
- Almacenar el valor de cada nodo de frontera.
- Identificar para cada nodo los elementos que son su soporte; es decir, que elementos contienen a ese nodo como uno de sus vértices.
- Identificar los nodos vecinos de cada nodo.
- Calcular el área de cada elemento triangular.
- Identificación de regiones dentro de la geometría.
- Determinar los nodos incógnita.
- Mostrar en pantalla la información de la malla.

2.2. CVFEM

- Debe ser capaz de construir un sistema de ecuaciones de la forma $\underline{\underline{A}}\underline{x}=\underline{b}$.
- La matriz $\underline{\underline{A}}$ debe ser construida a partir de los coeficientes de transmisibilidad según la discretización.
- El vector \underline{b} se debe construir a partir de la teoría y la discretización del vector.
- Debe de ser capaz de calcular la permeabilidad en distintos subdominios según lo descrito en el problema.

2.3. Almacenamiento de un vector.

- Debe almacenar en memoria los componentes de un vector en números de tipo flotante de doble precisión (double).
- Se definen las operaciones vector-vector:
 - Suma.
 - Resta.
 - Producto punto.
- Se define la operación matriz por vector para el esquema de almacenamiento.
- Cálculo de una norma euclíadiana $\|V\| = \sqrt{v_1^2 + v_2^2 \dots + v_n^2}$
- Cálculo del error medio cuadrático rms: $V_{rms} = \frac{\sqrt{v_1^2 + v_2^2 \dots + v_n^2}}{n}$

2.4. Almacenamiento de una matriz.

- Almacenamiento de los componentes de una matriz.
- Inicializar una matriz con ceros.
- Acceso a los valores de la matriz.
- Redimensionar una matriz.

2.5. Álgebra lineal.

- Implementar los métodos para sistemas de ecuaciones lineales.
 - Jacobi.
 - Gauss.
 - SOR.
 - Gradiente Conjugado.
 - BICGSTAB.
- Implementar precondicionadores de la matriz.
 - Jacobi.
 - MILU.
 - ILU.

2.6. Solve

- Devuelve la solución aproximada en un archivo de texto con el formato: coordenadasx coordenadasy aproximación.
- Almacena la solución analítica de cada benchmark.
- Calcula la distribución del error para los problemas de tipo benchmark usando $|error| = |exacta_{i,j} - calculada_{i,j}|$. El resultado es almacenado en un archivo de texto con el formato: coordenadasx coordenadasy $|error|$.

2.7. Visualizador.

- Graficación de la solución final usando la biblioteca matplotlib.

3. Requerimientos no funcionales

- El rendimiento es un factor importante.

- Debe permitir un mantenimiento sencillo.
- Minimización del uso de memoria en el almacenamiento de la matriz.

4. Restricciones de construcción

Debe ser usado el paradigma orientado a objetos utilizando el lenguaje de programación C++.

Arquitectura y Diseño Detallado de Software

1. Normatividad Interna

Con la finalidad de hacer más legible el código para el equipo de trabajo y para otros usuarios se tienen las siguientes reglas en la construcción del sistema:

- Los atributos inician con *(Palabra(s)clave(s))*.
- Palabra clave inicia con minúscula, si son 2 palabras, la primera con minúscula y la segunda con mayúscula.
- Nombre de las clases con mayúscula.
- Nombre de la clase = Nombre del archivo = Palabra identificadora.
- Toda clase se escribe en 2 archivos: hpp y cpp a menos que sean clases Template.
- Toda clase es documentada con ayuda de la herramienta Doxygen.
- Los comentarios se hacen por línea y se deja un espacio en blanco antes de iniciar la escritura de los comentarios.
- Al inicio de cada archivo se describe generalmente lo que hace la clase.
- Nombre del autor.
- Descripción del programa de manera general.
- Los métodos inician con minúscula y las palabras clave terminan con P mayúscula.
- Toda clase debe tener un constructor vacío y el constructor copia.
- Los parámetros variables de los métodos inician con minúsculas.

2. Arquitectura de Software

En la construcción del sistema se debe aplicar el POO, el rendimiento y el mantenimiento son requerimientos importantes para satisfacerlos, la arquitectura diseñada consta de 6 clases que pueden ser modificadas con sencillez debido a que cada una de éstas tiene una tarea específica, además existe una poca comunicación entre ellas. Se presenta una descripción abstracta del diagrama de clases (Ver Fig. 3.3) de la arquitectura propuesta.

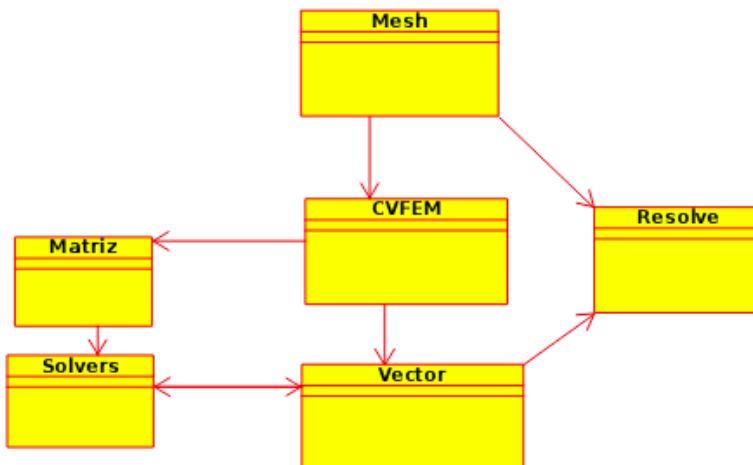


Figura 3.3: Diagrama de clases de la arquitectura para SYSG.

Cada clase se comunica con los demás por medio de una interfaz. Por lo tanto si se llega a la conclusión que una clase no satisface los requerimientos o que puede ser optimizada de alguna manera, puede ser reemplazada por otra fácilmente, siempre y cuando la interfaz se mantenga. A continuación se presenta de manera más detallada cada una de las clases.

3. Descripción de Componentes de Software

Mesh

La clase Mesh(Ver figura 3.4), tiene como objetivo el tratamiento del dominio y sus funcionalidades principales son:

- Almacenar los nodos y los elementos de la malla.
- Reordenar los nodos en forma ascendente.
- Identificar los nodos frontera y los nodos interiores.
- Identificar los elementos soporte de cada nodo.
- Dado un nodo, identificar los nodos vecinos.
- Calcular el área de cada elemento triangular.
- Almacenamiento de las condiciones de frontera de cada nodo.



Figura 3.4: Clase Mesh.

CVFEM

La clase CVFEM(Ver figura 3.5), tiene por objetivo principal formar el sistema de ecuaciones $\underline{Ax} = \underline{b}$, aplicando la expresión discreta a cada vértice de la malla, tomando en cuenta las condiciones de frontera. Sus funcionalidades más importantes son:

- Cálculo de los coeficientes de las funciones de forma.
- Cálculo de los coeficientes de transmisibilidad.
- Construcción de la matriz \underline{A} .
- Cálculo de la permeabilidad.
- Cálculo del vector b.

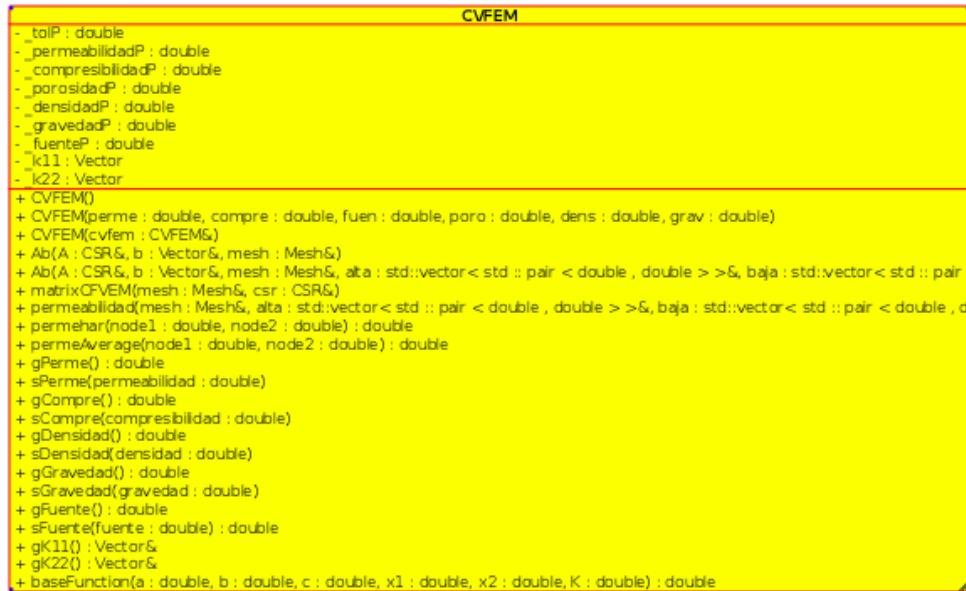


Figura 3.5: Clase CVFEM.

Matriz

La matriz que se construye con el CVFEM (Ver figura 3.6), es dispersa y no estructurada, por lo tanto muchos de sus componentes son cero y aquellos que no son cero se presentan en posiciones irregulares. Los componentes que son cero no son significativos para obtener la aproximación de la solución final, por lo tanto, almacenarlos constituye una pérdida de memoria, conviene entonces usar estrategias de almacenamiento que permitan almacenar y operar sólo la información relevante. Existen distintos esquemas de almacenamiento para matrices dispersas, en este trabajo se eligieron COO y CSR(véase [21]).

- **COO.** Ésta estrategia es la más natural, almacena el valor y las coordenadas i y j de cada componente no cero en tres arreglos de nombre: data, row y col. Su ventaja con relación a otros es que la inserción es constante, sin embargo, la búsqueda de un elemento es una operación ineficiente cuya complejidad es $O(nnz)$, donde nnz es el número de elementos no cero, debido a que los elementos pueden estar desordenados.
- **CSR.** Ésta estrategia, almacena los componentes por renglón en tres arreglos: data, col e irow, en estos: se almacenan los datos, el índice de columna y los índices de inicio y fin de cada renglón. Su ventaja principal es que la complejidad de buscar un elemento dentro de la matriz es $O(\log_2 n)$, donde n es el número máximo de elementos en un renglón.

Sabiendo todo esto y con el fin de minimizar: el tiempo de procesamiento y el almacenamiento, es utilizado el esquema COO para la construcción de la matriz y una vez formado se convierte al esquema CSR.

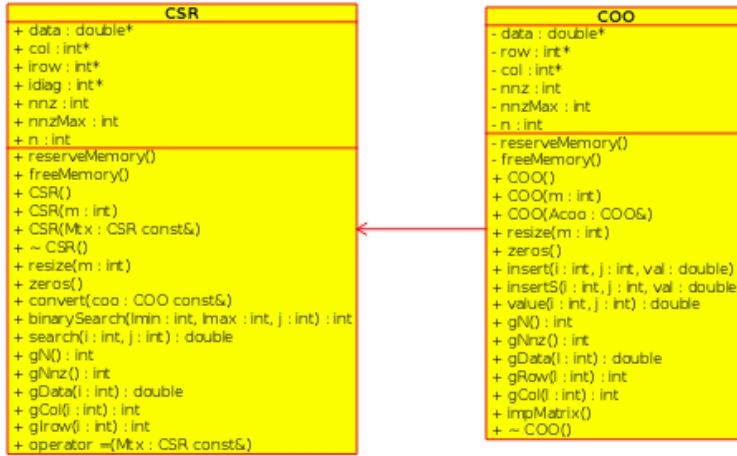


Figura 3.6: Diagrama de clases para COO y CSR.

Almacenamiento de Vectores

Se transforma en la clase Vector(Ver figura 3.7), su objetivo principal, es almacenar y operar la información de los vectores definidos por CVFEM. Sus operaciones principales son:

- Almacenamiento de un vector dado.
- Cálculo de la norma euclidiana.
- Cálculo de la media cuadrática rms.

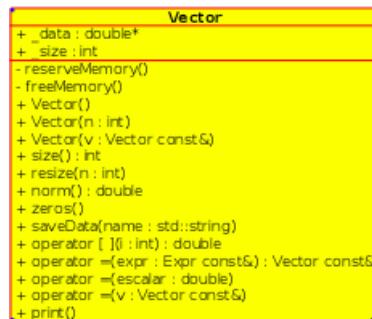


Figura 3.7: Clase Vector.

Solvers

Se implementa una librería de métodos iterativos, los cuales son: Jacobi, Gauss Seidel, SOR, Gradiente conjugado(CG) y BICGSTAB. Además, con el fin de mejorar la matriz son implementados precondicionadores los cuales son: Jacobi, ILU y MILU. Pueden ser usados por CG y BICGSTAB. (Ver figura 3.8).



Figura 3.8: Diagrama de clases para COO y CSR.

Con el fin de mejorar la eficiencia de los métodos numéricos, no se usa en ningún caso polimorfismo dinámico.

Resolve

La clase Resolve (Ver figura 3.9) tiene como tareas:

- Construcción de un archivo con la solución aproximada del problema para ser leída por un visualizador externo.
- Construcción de la solución exacta para los problemas benchmark.

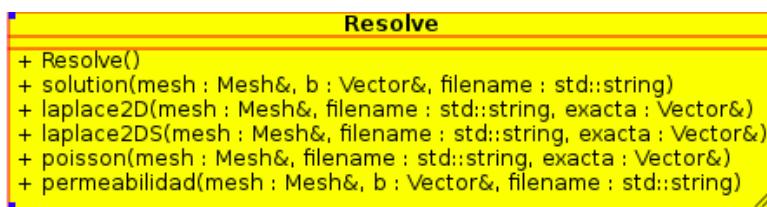


Figura 3.9: Clase Resolve.

Visualizador de datos externos

Se eligió como visualizador externo la biblioteca Matplotlib, para la visualización de los datos, para tal motivo se desarrolla un programa en python que permita su uso.

Comunicación

Para la comunicación de las clases se deben tomar en cuenta las siguientes consideraciones:

- **Mesh.** Esta no requiere de utilizar ningún servicio de otra clase, para funcionar requiere únicamente del archivo generado con gmsh.
- **CVFEM.** Esta clase construye el sistema $\underline{\underline{A}}x = \underline{b}$, por lo tanto requiere de la información de la malla, requiere contenedores para almacenar a la matriz $\underline{\underline{A}}$ y al vector \underline{b} .
- **COO y CSR.** Contenedores para almacenar la matriz generada por la clase CVFEM en formatos optimizados.
- **Vector.** Contenedor, para almacenar el vector b y la solución final.

- **Solvers** Sólo requieren la matriz $\underline{\underline{A}}$, al vector \underline{b} y al vector \underline{x} para calcular la solución del sistema.
- **Solve.** Requiere la información de la malla y la solución calculada con los métodos de álgebra lineal.
- **OperVect.** Incluye todas las operaciones vectoriales y la multiplicación matriz vector con expression templates.

Para estudiar con mayor detalle los métodos de cada clase verifique lo indicado en el apéndice D.

Pruebas de Integración

Descripción de la prueba

La clase a probar es Mesh, la cual es la encargada del tratamiento del dominio que previamente debió ser construido y discretizado con la herramienta gmsh, para este trabajo se consideran sólo geometrías rectangulares y geometrías formadas por dos semi-círculos.

Datos de entrada

- Archivo *.msh con la discretización del dominio.
- Coordenadas de los nodos esquina de una geometría rectangular.

Código de Prueba

Se deben definir los 4 vértices que forman el dominio:

```
// Definimos los vertices del dominio
std :: vector<double> vertices ;
vertices.push_back(0.0) ;
vertices.push_back(0.0) ;
vertices.push_back(1.0) ;
vertices.push_back(2.0) ;
```

Se construye un objeto de tipo Mesh el cual es alimentado con el archivo que contiene la geometría discretizada y los vértices esquina:

```
// Objeto para el tratamiento de la malla
Mesh mesh("./GMSH/Laplace/mesh10.msh", vertices) ;
```

Los nodos que se encuentran en la frontera pueden ser Dirichlet o Neumann, por lo tanto debe indicarse para cada nodo su tipo y su valor:

```
// Cargamos las condiciones de frontera
std::vector<int> border = mesh.bordeNode();
for (int i = 0; i < border.size(); ++i)
{
    if (mesh.coordY(border[i]) == 0
        && (mesh.coordX(border[i]) != 0
            && mesh.coordX(border[i]) != 1))
    {
        mesh.boundaryCondition(0,100);
    }
    else
    {
        mesh.boundaryCondition(0,0);
    }
}
```

Se deben obtener los nodos que son incógnitas, los cuales son nodos interiores o nodos Neumann:

```
// Es necesario determinar los nodos incognita
mesh.nodeIncongnite();
mesh.showTotal();
```

Resultado esperado

- Lectura y almacenamiento en memoria de: nodos y elementos.
- Ordenamiento de nodos y elementos según lo indicado en los requerimientos.
- Distinción entre nodos interiores y nodos frontera.
- Identificación de nodos incógnita.
- Cálculo del área de cada elemento triangular.

Resultado obtenido

La comprobación fué realizada con una impresión de pantalla y una comparación obtenida a mano, para distintos archivos, los cuales son incluidos en el código en la carpeta gmsh, los resultados fueron exitosos.

Defectos encontrados

Sin defectos encontrados

Pruebas de Integración

Descripción de la prueba

La clase a probar es CVFEM la cual es la encargada de construir la matriz $\underline{\underline{A}}$ y el vector \underline{b} , para que posteriormente sea resuelto.

Datos de entrada

- Objeto tipo Mesh que contiene la información del dominio de interés.
- Contenedor para el almacenamiento de la matriz $\underline{\underline{A}}$.
- Contenedor para el almacenamiento del vector \underline{b} .

Código de prueba

Contenedores para la matriz $\underline{\underline{A}}$ y el vector \underline{b} :

```
// Definimos las variables necesarias
CSR A(mesh.nNodeIncongnite()); // Matriz CVFEM
Vector b(mesh.nNodeIncongnite()); // Vector b
```

Objeto de tipo CVFEM y se llama a la función encargada de la construcción de la matriz y del vector la cual se alimenta con los contenedores y la información de la malla:

```
CVFEM cvfem;
cvfem.Ab(A, b, mesh);
```

Resultado esperado

- Que las propiedades de las funciones base sean satisfechas.
- Matriz dispersa no estructurada, para los nodos incógnita.
- Vector b, para los nodos incógnita.

Resultado obtenido

Para distintos tamaños de malla, se cumplió lo especificado por la teoría, para las funciones base (véase la sección 2.4.4). Se obtuvo una matriz dispersa y un vector, sin embargo, debido a la gran cantidad de operaciones no se verificó la validez de la matriz. La clase fue verificada por completo durante las pruebas de benchmark.

Defectos encontrados

Sin defectos encontrados.

Pruebas de Integración

Descripción de la prueba

Se prueba la clase COO Y CSR que implementan esquemas útiles de almacenamiento.

Datos de entrada

Para probar los esquemas de almacenamiento es necesaria una matriz dispersa no estructurada, en este caso, con el fin de simplificar la prueba se eligió una matriz dispersa estructurada, la cual es obtenida de la discretización de la ecuación de Poisson $\nabla T(x, y) = 1$, usando el método de diferencias finitas cuya forma puede observarse en la figura 3.10.

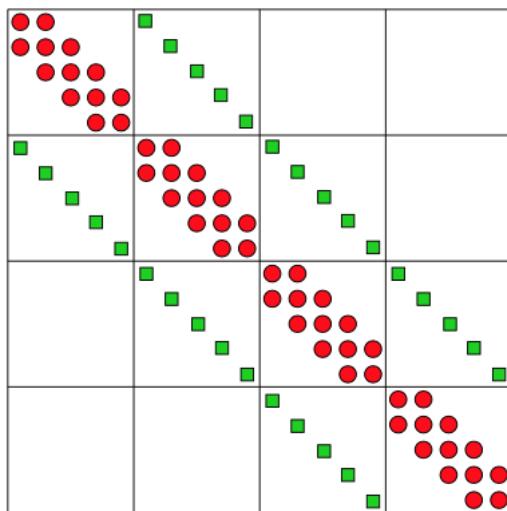


Figura 3.10: Matriz dispersa. Tomada de *Método de volumen finito para flujo en una fase*[22].

Código de prueba

Se define el tamaño de los contenedores:

```
int nx=1001;
int ny=nx;
int n = (nx - 1)*(ny - 1);
//almacenamiento
CSR A(n); //matriz en formato CSR
COO Acoo(n); //matriz temporal
```

Se almacena las entradas distintas de cero:

```
for (int j=1;j<ny;++j)
{
    for (int i=1;i<nx;++i)
    {
        if(j>1)
            Acoo.insert(1, 1-(nx-1), -1.);
        if(i>1)
            Acoo.insert(1, 1-1,-1.);
            Acoo.insert(1, 1,4.);
        if(i<nx-1)
            Acoo.insert(1, 1+1,-1.);
        if(j<ny-1)
            Acoo.insert(1, 1+(ny-1),-1.);
        ++l;
    }
}
```

Se transforma una matriz de COO a CSR y se imprime en pantalla:

```
A.convert(Acoo); //convierte de COO a formato CSR
Acoo.impMatrix();
```

Resultado esperado

- Almacenamiento eficiente de una matriz dispersa.
- Conversión de la matriz COO a CSR.

Resultado obtenido

Para diferentes tamaños de la matriz la prueba fue exitosa, en este caso se presenta la impresión de pantalla(Ver figura 3.11) resultado de usar una matriz de 9x9 , cabe mencionar que los ceros no fueron almacenados, sin embargo, se imprimen para dar legibilidad. La tabla muestra el tiempo utilizado para la construcción de matrices de distinto tamaño en formato COO y su conversión a CSR.

4	-1	0	-1	0	0	0	0	0
-1	4	-1	0	-1	0	0	0	0
0	-1	4	0	0	-1	0	0	0
-1	0	0	4	-1	0	-1	0	0
0	-1	0	-1	4	-1	0	-1	0
0	0	-1	0	-1	4	0	0	-1
0	0	0	-1	0	0	4	-1	0
0	0	0	0	-1	0	-1	4	-1
0	0	0	0	0	-1	0	-1	4

Figura 3.11: Se muestra la matriz almacenada en el formato CSR

Tamaño	Inserción COO [ms]	Conversión CSR [ms]
100x100	0.002	0.002
10000x10000	0.12	0.199
40000x40000	0.584	1.126
90000x90000	1.146	2.018
160000x160000	2.745	3.862
250000x250000	5.185	6.504
360000x360000	6.733	9.288
640000x640000	8.05	16.027
810000x810000	11.155	19.669
1000000x1000000	13.785	28.496

Cuadro 3.13: Ejecuciones para el almacenamiento de una matriz en COO y su conversión en CSR.

Defectos encontrados

Sin defectos encontrados

Pruebas de Integración

Descripción de la prueba

Prueba para los métodos numéricos: Jacobi, Gauss-Seidel, SOR, Gradiente Conjugado (CG), Gradiente Conjugado Precondicionado (CGP), gradiente conjugado estabilizado (BICGSTAB) y Gradiente Conjugado Estabilizado Precondicionado (BICGSTABP). Los precondicionadores usados son: Jacobi, ILU y MILU. Son usadas las clases COO y CSR para el almacenamiento del sistema de ecuaciones (véase [21]).

Uno de los requerimientos no funcionales, indica que la eficiencia es un factor importante. Con éste objetivo en mente, se desarrollaron dos implementaciones diferentes. La primera usando sobrecarga de operadores y la segunda utilizando programación genérica y técnicas de metaprogramación.

Datos de entrada

Se requiere un sistema de ecuaciones, para este propósito fue elegido el formado por el método de diferencias finitas en la ecuación de Poisson $\nabla T(x, y) = 1$, cuya estructura puede ser observada en la figura 3.12.

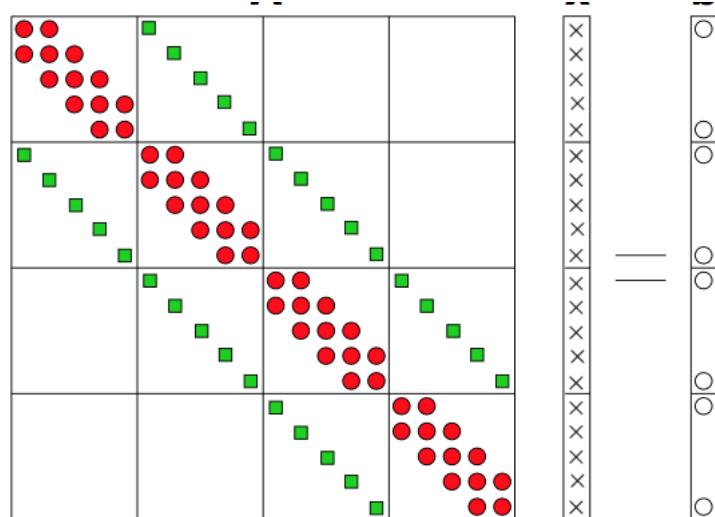


Figura 3.12: Sistema de entrada para probar los métodos numéricos. Tomada de *Método de volumen finito para flujo en una fase* [22].

Código de prueba

Se define el sistema de ecuaciones a ser resuelto:

```

int nx = 11; // Nodos en el eje x
int ny=nx; // Nodos en el eje y
double dx = 1./nx; // Paso en x
double dy = 1./ny; // Paso en y
int n = (nx - 1)*(ny - 1); // Numero de nodos totales
//almacenamiento
CSR A(n); //matriz en formato CSR
Vector x(n),b(n); // Vectores para el sistema

{
    COO Acoo(n); //matriz temporal
    for(int j=1;j<ny;++j)
    {
        for(int i=1;i<nx;++i)
        {
            if(j>1)
                Acoo.insert(1, 1-(nx-1),-1.);
            if(i>1)
                Acoo.insert(1, 1-1,-1.);
                Acoo.insert(1, 1,4.);
            if(i<nx-1)
                Acoo.insert(1, 1+1,-1.);
            if(j<ny-1)
                Acoo.insert(1, 1+(ny-1),-1.);
                ++1;
        }
    }
    A.convert(Acoo); //convierte de COO a formato CSR
}
b = 1.*dx*dx; // se llena el vector b

```

Se utiliza un método para la solución del sistema de ecuaciones, por ejemplo:

```

BICGSTAB bicg; // Metodo utilizado
bicg.solve(A,x,b); // Solver
std :: cout << std :: endl;
bicg.report(); // Imprime un reporte

```

Resultado esperado

Se espera que:

- Los métodos numéricos converjan.
- A medida que crezca el sistema de ecuaciones, se requiera más tiempo en alcanzar su solución.
- La implementación más eficiente sea la que incluye programación genérica y técnicas de metaprogramación.

Resultado obtenido

Fueron usados diferentes tamaños de matrices para el problema los cuales pueden ser vistos en el cuadro 3.14.

Tamaño	Inserción COO [ms]	Conversión CSR [ms]
100x100	0.002	0.002
10000x10000	0.12	0.199
40000x40000	0.584	1.126
90000x90000	1.146	2.018
160000x160000	2.745	3.862
250000x250000	5.185	6.504
360000x360000	6.733	9.288
5504100x5504100	8.05	16.027
810000x810000	11.155	19.669
1000000x1000000	13.785	28.496

Cuadro 3.14: Ejecuciones para el almacenamiento de una matriz en COO y su conversión en CSR.

Los resultados obtenidos fueron verificados por medio del residuo usando $r = \|b - Ax\|$, entre más grande es la matriz r se aproxima más a cero. Además la implementación usando técnicas de programación genérica y metaprogramación fué superior en tiempo a la implementación con sobre-carga de operadores. A continuación se presentan comparaciones entre las implementaciones, para los métodos de solución.

Comparación entre las dos implementaciones del gradiente conjugado

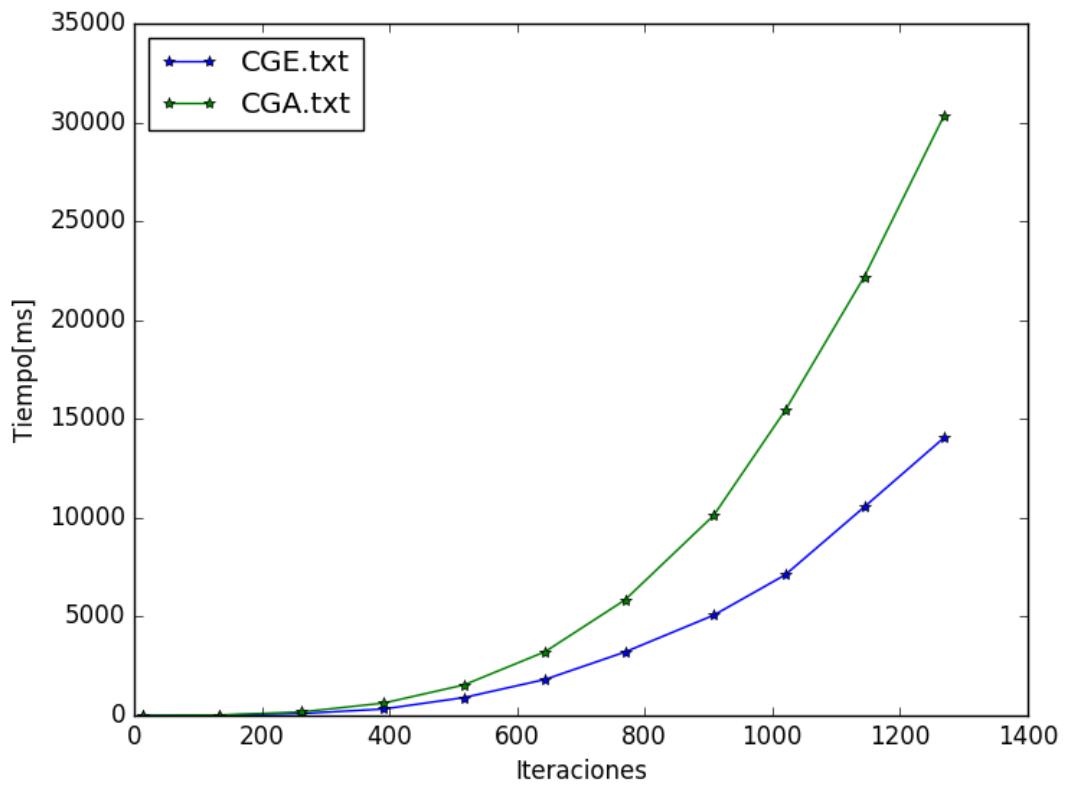


Figura 3.13: Comparación de tiempo para el algoritmo gradiente conjugado, usando dos implementaciones diferentes. Donde CGA(verde) representa la implementación utilizando sobrecarga de operadores y CGE(azul) representa la implementación utilizando programación genérica y técnicas de metaprogramación.

Comparación entre las dos implementaciones del gradiente conjugado precondicionado con ILU

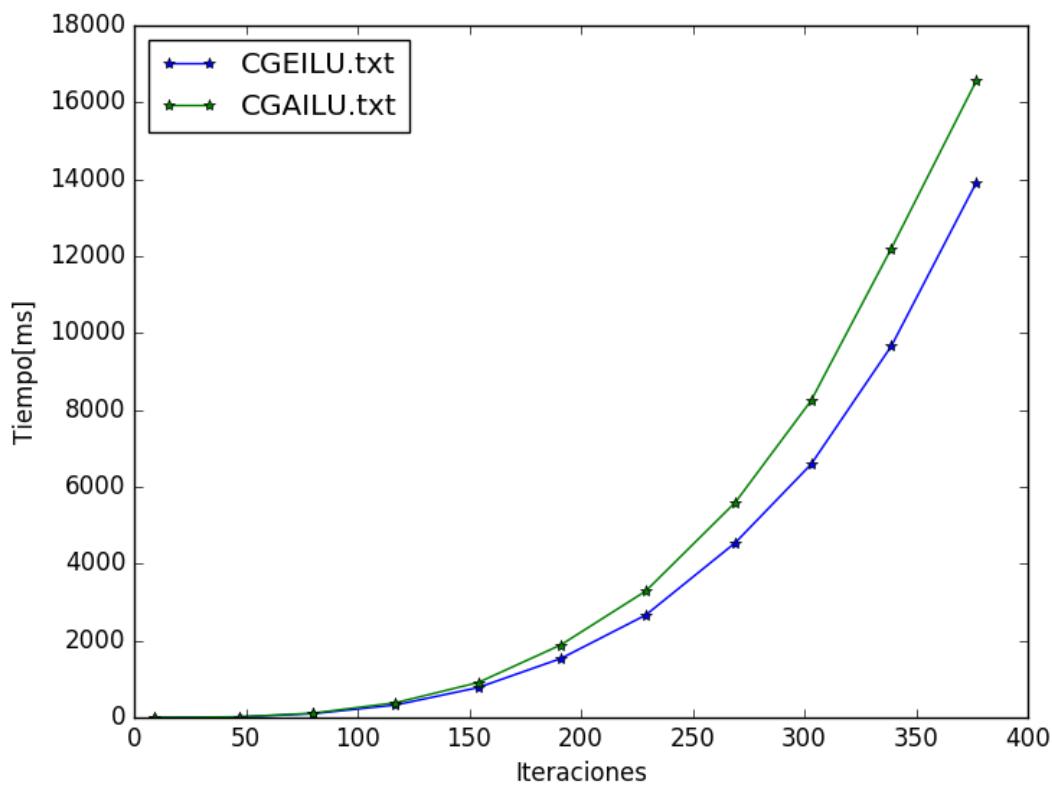


Figura 3.14: Comparación de tiempo para el algoritmo gradiente conjugado precondicionado con ILU, usando dos implementaciones diferentes. Donde CGAILU(verde) representa la implementación utilizando sobrecarga de operadores y CGEILU(azul) representa la implementación utilizando programación genérica y técnicas de metaprogramación.

Comparación entre las dos implementaciones del gradiente conjugado precondicionado con MILU

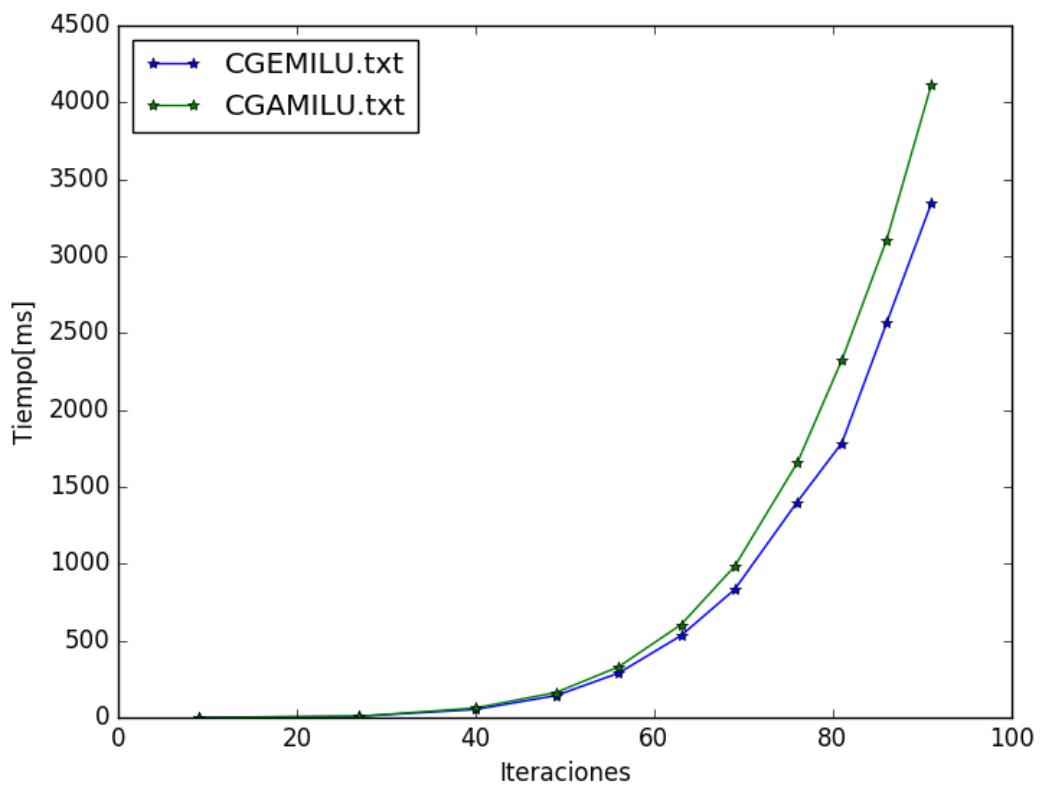


Figura 3.15: Comparación de tiempo para el algoritmo gradiente conjugado precondicionado con MILU, usando dos implementaciones diferentes. Donde CGAMILU(verde) representa la implementación utilizando sobre carga de operadores y CGEMILU(azul) representa la implementación utilizando programación genérica y técnicas de metaprogramación.

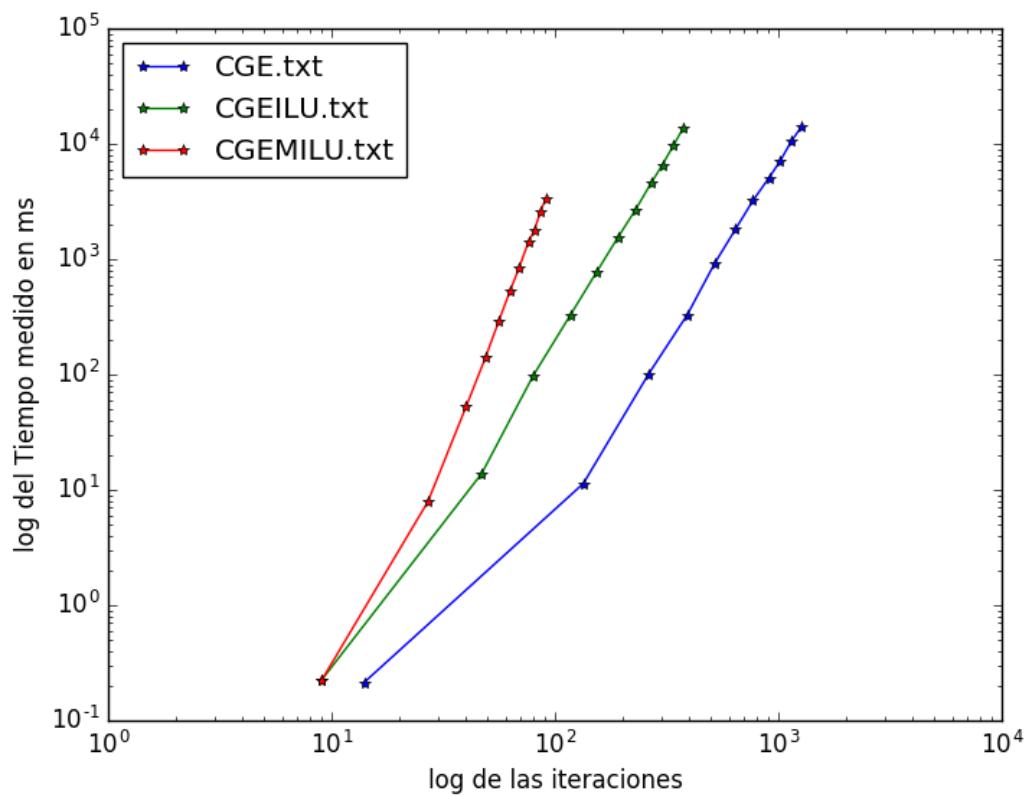
Comparación del gradiente conjugado y sus precondicionadores

Figura 3.16: En la gráfica se muestra el gradiente conjugado(azul), el gradiente conjugado precondicionado con ILU(verde) y el gradiente conjugado precondicionado con MILU(rojo), para la implementación más eficiente.

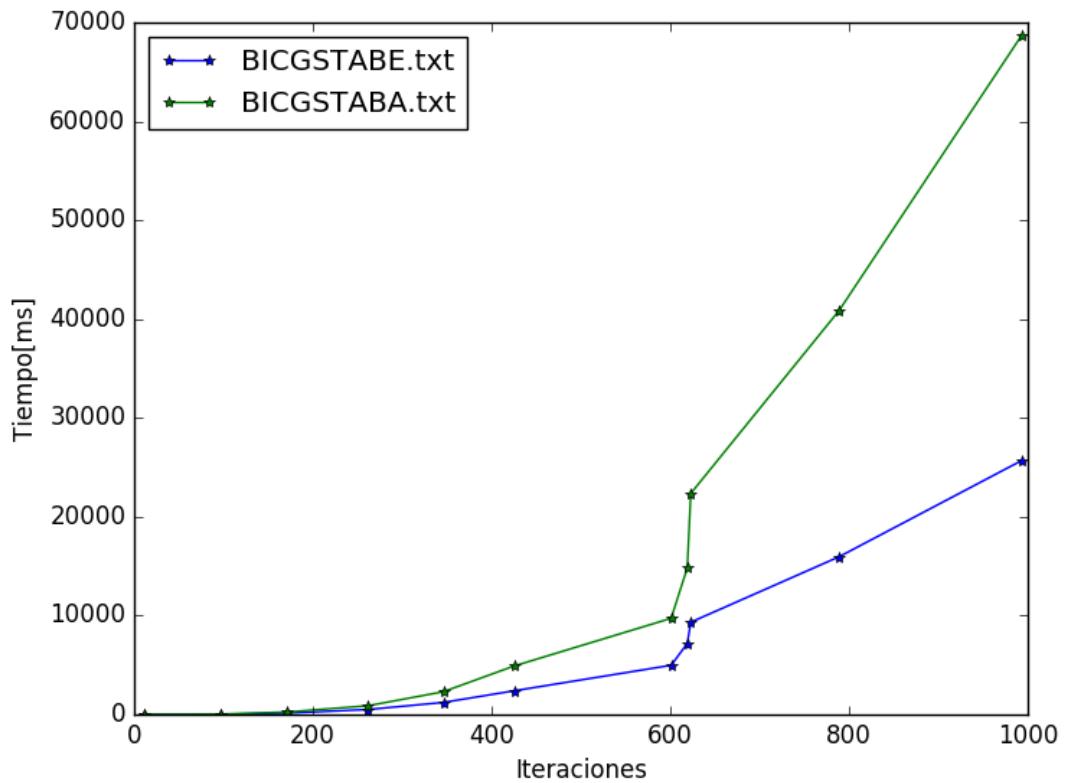
Comparación entre las dos implementaciones del gradiente BICGSTAB

Figura 3.17: Comparación de tiempo para el algoritmo BICGSTAB, usando dos implementaciones diferentes. Donde BICGSTABA(verde) representa la implementación utilizando sobre carga de operadores y BICGSTABE(azul) representa la implementación utilizando programación genérica y técnicas de metaprogramación.

Comparación entre las dos implementaciones de BICGSTAB precondicionado con ILU

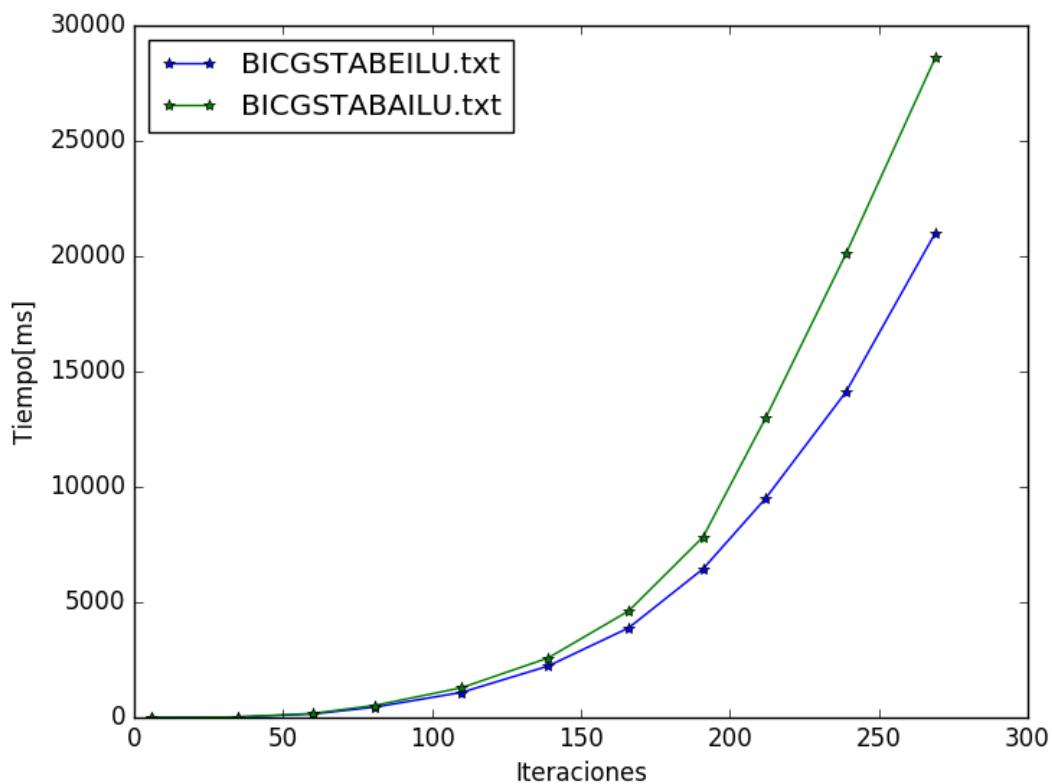


Figura 3.18: Comparación de tiempo para el algoritmo BICGSTAB precondicionado con ILU, usando dos implementaciones diferentes. Donde BICGSTABAILU(verde) representa la implementación utilizando sobre carga de operadores y BICGSTABEILU(azul) representa la implementación utilizando programación genérica y técnicas de metaprogramación.

Comparación entre las dos implementaciones de BICGSTAB precondicionado con MILU

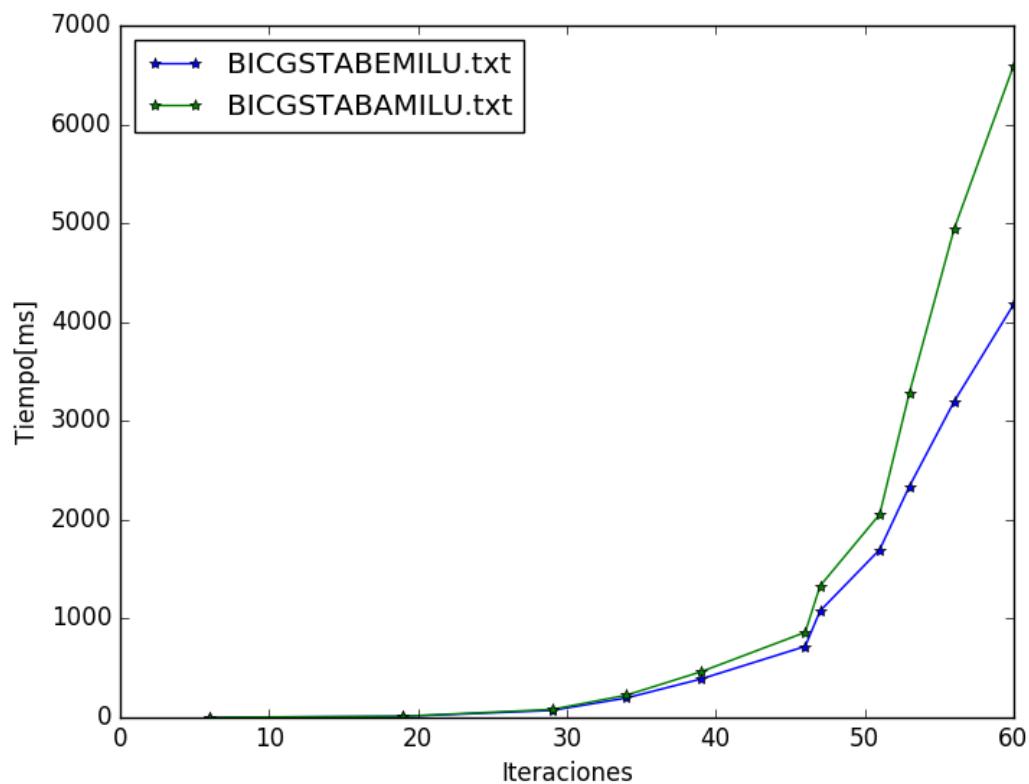


Figura 3.19: Comparación de tiempo para el algoritmo BICGSTAB precondicionado con MILU, usando dos implementaciones diferentes. Donde BICGSTABAMILU(verde) representa la implementación utilizando sobrecarga de operadores y BICGSTABEMILU(azul) representa la implementación utilizando programación genérica y técnicas de metaprogramación.

Comparación de BICGSTAB y sus precondicinadores

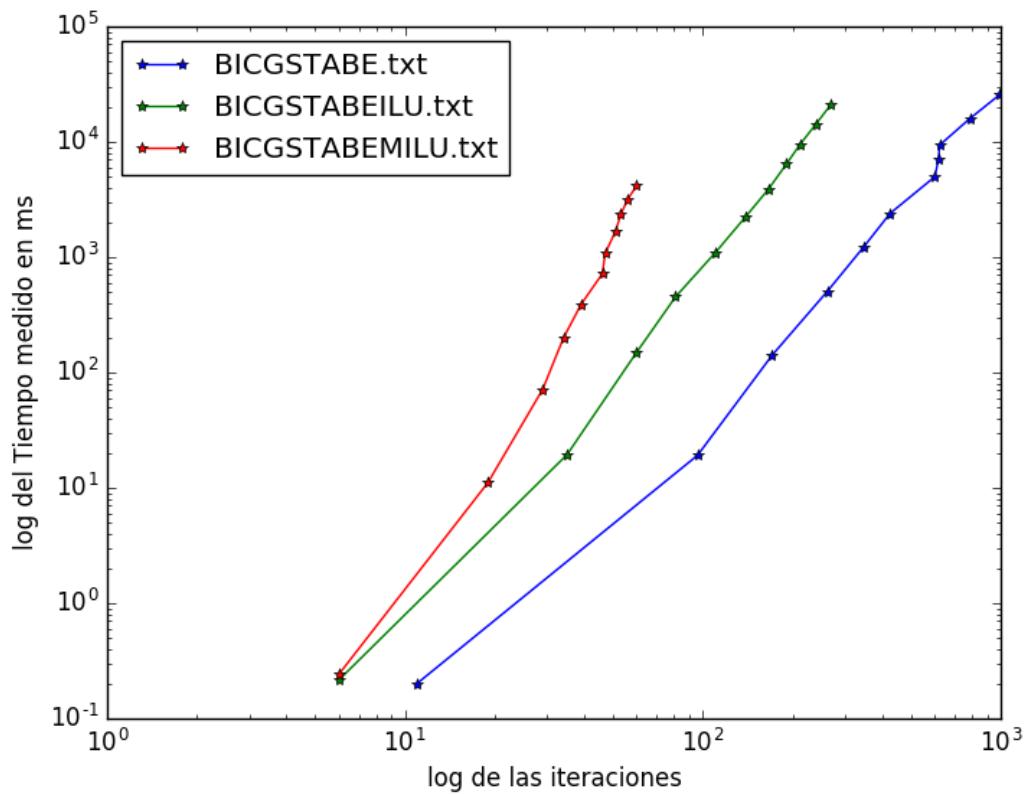


Figura 3.20: En la gráfica se muestra el gradiente conjugado estabilizado(azul), el gradiente conjugado estabilizado precondicionado con ILU(verde) y el gradiente conjugado estabilizado precondicionado con MILU(rojo), para la implementación más eficiente.

Se concluye por lo tanto que el uso de técnicas de programación genérica y de metaprogramación generan un código más flexible y óptimo.

Defectos encontrados

- Los métodos Jacobi, Gauss-Seidel y SOR no convergen con la tolerancia y el número de iteraciones permitidas.
- No hay ninguna reducción en el número de iteraciones necesarias para la convergencia utilizando el precondicionador de Jacobi.

Capítulo 4

Validación

Una vez que esta completo el sistema en cada iteración, se debe determinar si éste cumple con los requerimientos, es decir si es capaz de determinar la solución al problema inicial. Sin embargo, los científicos no conocen la solución al problema que tratan de resolver, entonces, ¿Cómo se puede determinar que la aproximación obtenida por sus códigos es correcta? Una forma, es utilizando pruebas de validación.

Las pruebas de validación permiten a los científicos aumentar la confianza en sus sistemas, pues éstas establecen un conjunto de pruebas, las cuales son llamadas puntos de referencia (*del inglés benchmark*). Estos puntos de referencia son problemas bien estudiados en la literatura científica, para los cuales se conoce la solución exacta y pueden ser usados para calibrar el código. También permiten detectar todos los errores que no fueron encontrados por las dos primeras etapas de pruebas.

Se presentan tres pruebas de validación y un problema interesante en cuyo dominio se tienen dos zonas con diferente permeabilidad.

4.1. Resultados preliminares

SYGS puede ser fácilmente utilizado para resolver problemas numéricos de flujo en medios porosos debido a la documentación generada. En ésta versión sólo se consideran dominios rectangulares y dominios formados por dos semicircunferencias. La ventaja de nuestro desarrollo es que para todos los problemas presentados, la codificación de la solución se realiza siguiendo los mismos pasos con pequeñas diferencias, los cuales son:

1. Incluir encabezados de las clases necesarias.
2. Declarar un objeto para el tratamiento del dominio de interés.
3. Declarar un objeto para la construcción del sistema de ecuaciones lineales.
4. Definir las condiciones de frontera.
5. Declarar y definir las variables necesarias para la solución.
6. Declarar y generar el sistema de ecuaciones lineales.
7. Resolver el sistema de ecuaciones lineales.
8. Calcular la solución analítica en cada nodo en caso de contar con ella.
9. Calcular la distribución del error.
10. Almacenar la solución aproximada, según el formato de un graficador externo.
11. Graficar la solución aproximada.

En las siguientes secciones se presentan las pruebas de validación realizadas. Además, la codificación de estos problemas son incluidos en el proyecto para un estudio más profundo, véase el apéndice D. Para la solución del sistema de ecuaciones en las pruebas de validación realizadas es elegido BICGSTAB, debido a que se observó en las pruebas de integración que éste trabaja de manera óptima.

4.2. Laplace en dos dimensiones

4.2.1. Descripción de la prueba

Nuestro primer problema consiste en determinar la dirección de flujo en un medio homogéneo, este fenómeno se describe por medio de la ecuación de Laplace:

$$-\Delta T(x, y) = 0 \quad (4.2.1)$$

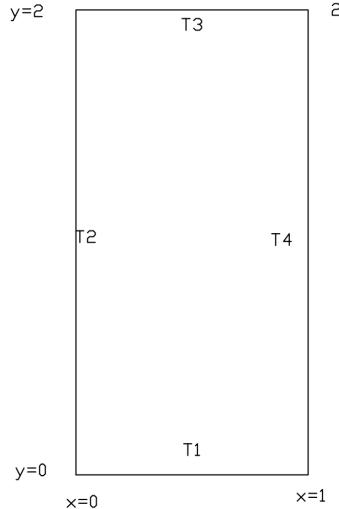


Figura 4.1: Dominio de interés para la ecuación de Laplace

El dominio $\Omega = (0, 1) \times (0, 2)$, es un rectángulo(Ver Fig. 4.1), las condiciones de frontera están dadas por:

$$T_1 = 100, T_2 = T_3 = T_4 = 0$$

La solución analítica del problema es:

$$T(\underline{x}) = T_1 \left[2 \sum_{n=1}^{\infty} \frac{1 - (-1)^n}{n\pi} \frac{\sinh\left(\frac{n\pi(H-y)}{L}\right)}{\sinh\left(\frac{n\pi H}{L}\right)} \sin\left(\frac{n\pi x}{L}\right) \right]$$

4.2.2. Descripción de Resultados

Se genera una circulación del flujo en el interior del dominio de interés. Según el valor de la condición inicial. En la figura 4.2, se muestra la solución exacta, la solución aproximada y la distribución del error.

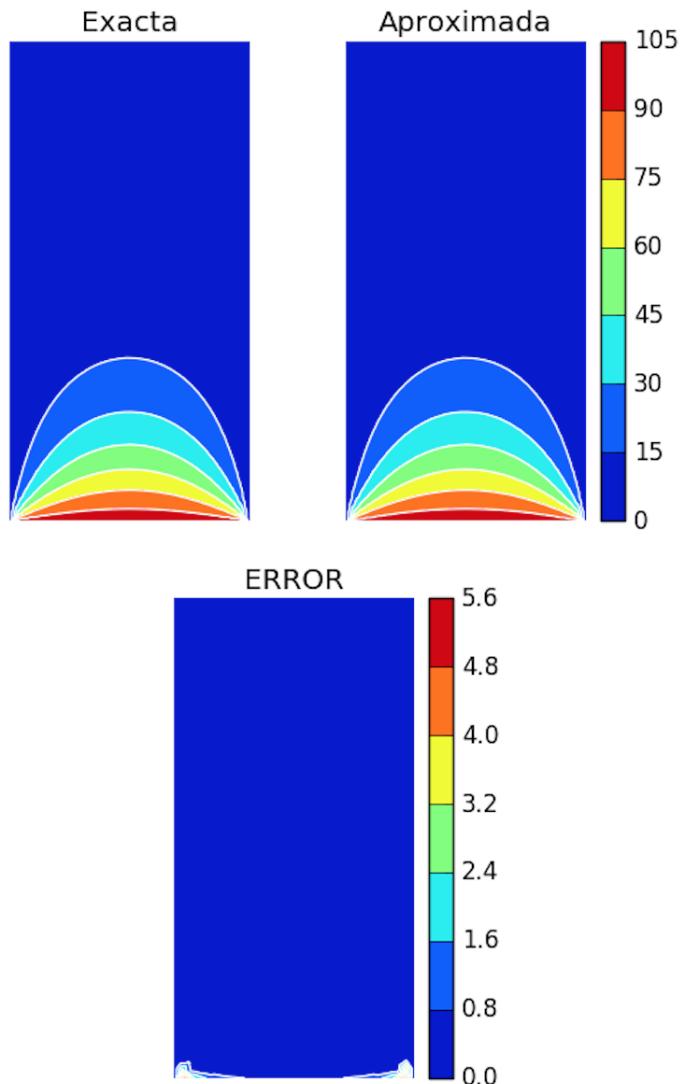


Figura 4.2: Solución para la ecuación de Laplace.

4.2.3. Observaciones

La malla utilizada en esta prueba puede observarse en la figura 4.3, la cual esta compuesta por 4380 nodos y 8758 elementos.

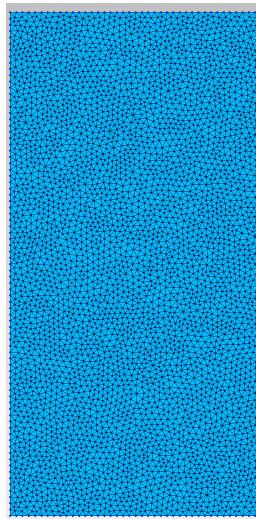


Figura 4.3: Malla para la ecuación de Laplace, generada con la herramienta gmsh.

Para codificar la solución a la ecuación de Laplace usando SYGS, se requiere darle un tratamiento al dominio de interés, para éste fin se declara un objeto de la clase Mesh, el cual recibe un archivo con la discretización del dominio y para identificar fácilmente los nodos frontera de los nodos interiores en un dominio rectangular se requieren sus vértices, en código:

```
// Definimos los vertices del dominio
std :: vector<double> vertices ;
vertices.push_back(0.0) ;
vertices.push_back(0.0) ;
vertices.push_back(1.0) ;
vertices.push_back(2.0) ;
// Definimos un objeto para el tratamiento de la malla
Mesh mesh("./GMSH/Laplace/mesh10.msh", vertices);
```

Una vez que se genera el objeto mesh tiene como tareas: la lectura del archivo, el reordenamiento de los nodos, la identificación de los nodos frontera, identificación de los elementos soporte para un nodo y el cálculo del área

de cada elemento triangular. Es necesario indicar para los nodos frontera si son: Dirichlet o Neumann y qué valor tienen. Se utiliza el método boundaryCondition(int type, double val), si type es 0 entonces tratamos con un nodo tipo Dirichlet y si es 1 con un nodo tipo Neumann. En éste caso se llaman a los nodos de frontera y se recorren uno a uno, asignando su valor y su tipo según las restricciones del problema, en código:

```
// Cargamos las condiciones de frontera
std :: vector<int> border = mesh.bordeNode(); // Nodos frontera
for (int i = 0; i < border.size(); ++i)
{
    if (mesh.coordY(border[i]) == 0 && (mesh.coordX(border[i]) != 0 && mesh.coordX(border[i]) != 1))
    {
        mesh.boundaryCondition(0,100);
    }
    else
    {
        mesh.boundaryCondition(0,0);
    }
}
```

Según la teoría, en los nodos tipo Dirichlet conocemos la solución exacta del problema y podemos no incorporarlos como incógnitas en la matriz de transmisibilidad. Por otro lado los nodos tipo Neumann son lugares en donde no conocemos con precisión la solución del problema, por lo tanto deben ser considerados en la matriz de transmisibilidad. Por lo cual, un paso importante es determinar, cuántos y cuáles nodos incógnita tendremos, para asignar los recursos de memoria suficientes, en código:

```
mesh.nodeIncongnite(); // Nodos Incognita
CSR A(mesh.nnodeIncongnite()); // Matriz A del sistema
Vector b(mesh.nnodeIncongnite()); // Vector b del sistema
Vector x(mesh.nnodeIncongnite()); // Vector x del sistema
```

Se construye el sistema de ecuaciones lineales, según el método CVFEM:

```
// Objeto para crear el sistema de ecuaciones
CVFEM cvfem;
cvfem.Ab(A,b, mesh); // Construye A y b
```

Definimos un objeto de la clase BICGSTAB, para hacer uso del método del gradiente conjugado estabilizado, el cual recibe el sistema de ecuaciones a resolver:

```
// Resolvemos el Sistema de ecuaciones
BICGSTAB bicg;
bicg.solve(A,x,b); // BICGSTAB
bicg.report(); // Muestra un reporte
```

Se calcula la solución exacta, la solución aproximada total y la distribución del error, las cuales se almacenen en un archivo de texto:

Se almacena la solución aproximada y la exacta (en caso de contar con ella) en un archivo de texto:

```
// Calculamos la solucion exacta
Vector exacta(mesh.nNode()); // Solucion exacta
solver.laplace2D(mesh, "laplace2DE.txt", exacta);
Vector x1(mesh.nNode()); // Vector b para la solucion final
Vector error(mesh.nNode()); // Almacena la distribucion del error
// Almacenamos la solucion aproximada con CVFEM
solver.solution(mesh,x,x1);
solver.storage(mesh,x1,"laplace2DA.txt");
// Calculamos la distribucion del error
error = exacta - x1;
solver.storageE(mesh,error,"error.txt");
```

En la gráfica 4.4 se muestra el cálculo del residuo durante cada iteración realizada por el gradiente conjugado y el gradiente conjugado precondicionado con ILU. En la gráfica 4.5 se muestra el cálculo del residuo durante cada iteración realizada por el gradiente conjugado estabilizado y el gradiente conjugado estabilizado precondicionado con ILU.

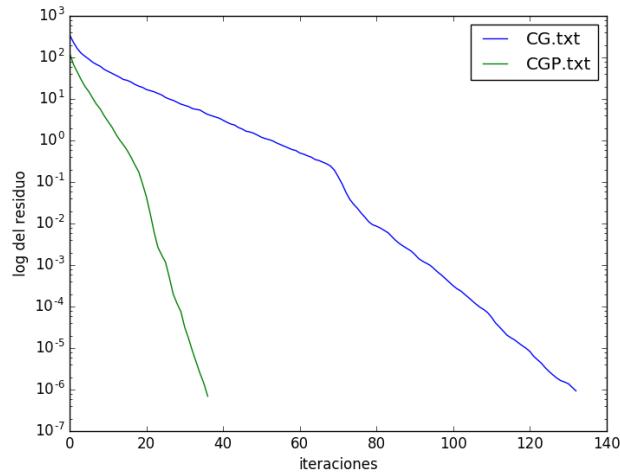


Figura 4.4: La gráfica muestra el residual $\log |r_n|$ vs las iteraciones n, para el método del gradiente conjugado(azul) y el método del gradiente conjugado precondicionado con ILU(verde).

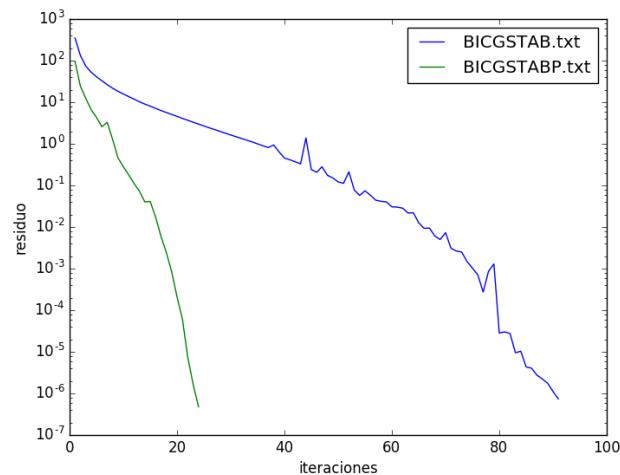


Figura 4.5: La gráfica muestra el residual $\log |r_n|$ vs las iteraciones n, para el método del gradiente conjugado estabilizado (azul) y el método del gradiente conjugado estabilizado precondicionado con ILU(verde).

4.3. Laplace en un semicírculo

4.3.1. Descripción

El CFVEM es utilizado en la solución de problemas con dominios complicados, es decir dominios que presentan una geometría no rectangular. En éste contexto determinamos la dirección de flujo en un medio homogéneo en un dominio acotado por dos semicírculos, la ecuación de Laplace (ec. 4.2.1) describe este fenómeno.

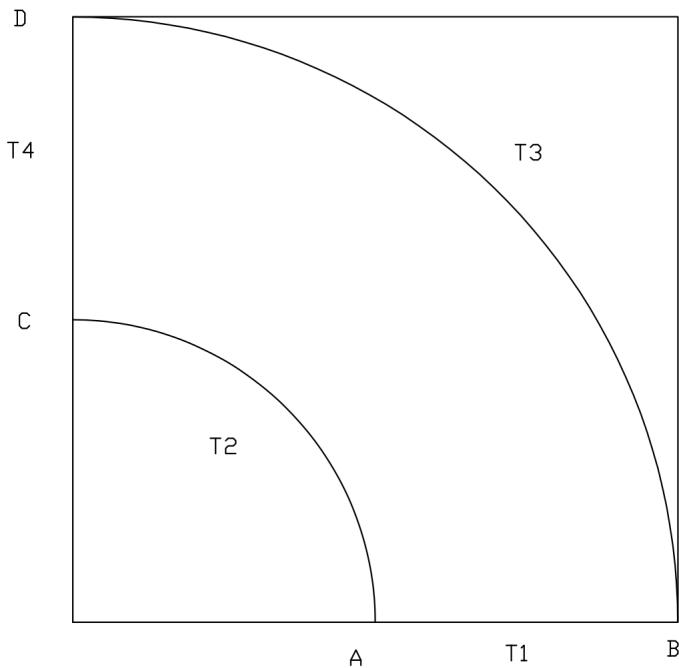


Figura 4.6: Dominio acotado por dos semicírculos para la ecuación de Laplace.

Condiciones de frontera:

$$T_1 = 0, T_2 = \frac{\sin(\theta)}{R_{0A}}, T_3 = \frac{\sin(\theta)}{R_{0B}}, T_4 = \frac{1}{r}$$

Solución analítica:

$$T = \frac{\sin(\theta)}{r}$$

4.3.2. Resultados

Se genera una circulación del flujo en el interior del dominio de interés, según la geometría del problema. En la figura 4.7, se muestra la solución exacta, la aproximada y la distribución del error.

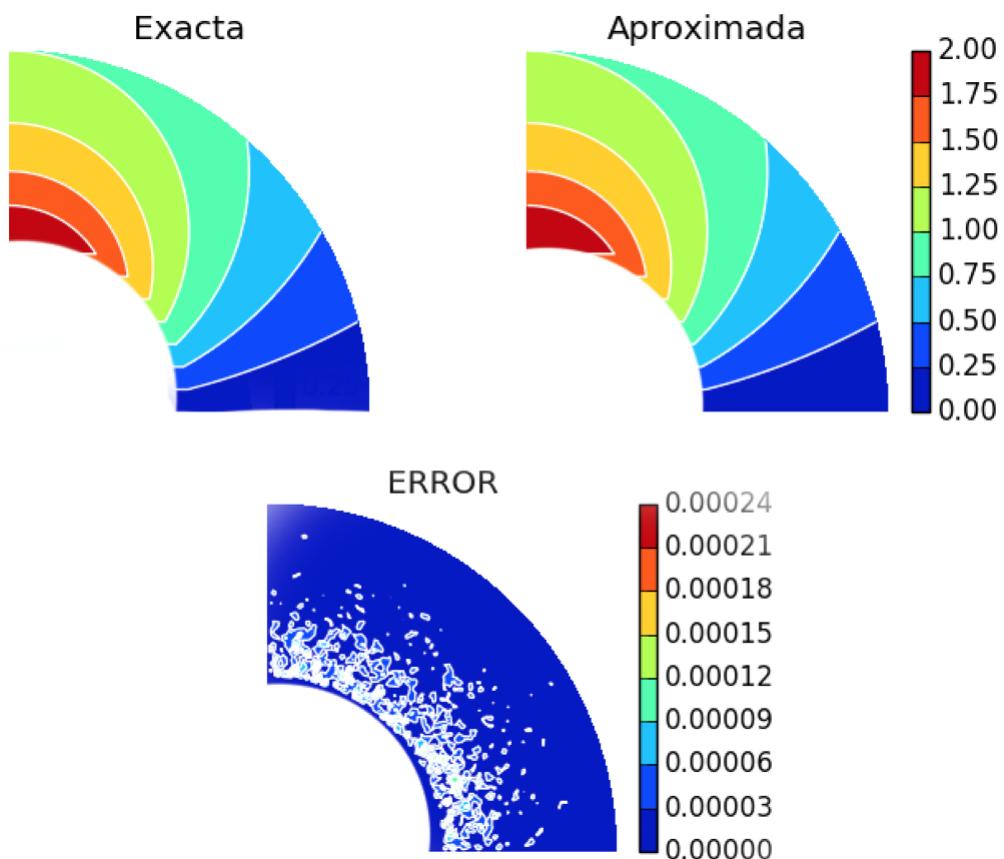


Figura 4.7: Solución para la ecuación de Laplace en un semicírculo.

4.3.3. Observaciones

La malla utilizada en esta prueba puede observarse en la figura 4.8, la cual tiene 4176 nodos y 8108 elementos.

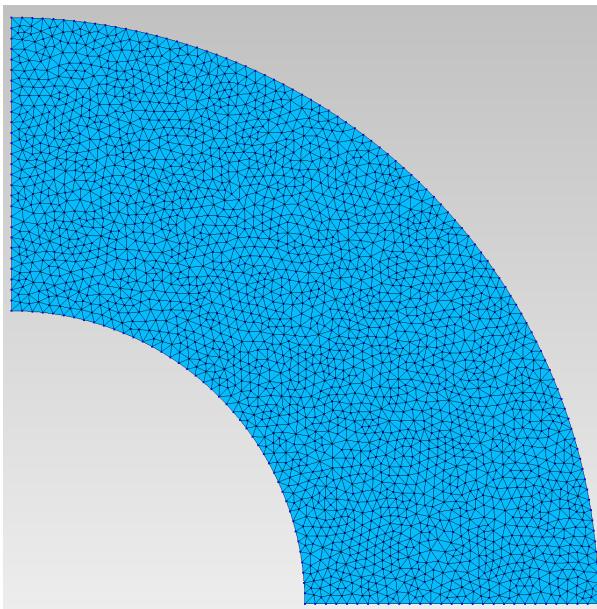


Figura 4.8: Malla para la ecuación de Laplace en un semicírculo.

La codificación resulta ser muy similar al problema anterior, llamamos otro constructor que permita el tratamiento del semicírculo, de esto:

```
// Definimos los radios del los semicirculos
double ra = 0.5; // Radio externo
double rb = 1.0; // Radio interno
// Definimos un objeto para el tratamiento de la malla
Mesh mesh("./GMSH/Laplace/mesh10.msh",ra,rb);
```

Además las condiciones de frontera cambian, de lo cual:

```
// Cargamos las condiciones de frontera
std::vector<int> border = mesh.bordeNode();
for (int i = 0; i < border.size(); ++i)
{
    double r = sqrt(pow(mesh.coordX(border[i]), 2) + pow(mesh.
        coordY(border[i]), 2));
    double theta = atan(mesh.coordY(border[i]) / mesh.coordX(
        border[i]));
    if (mesh.coordX([i]) == 0)
    {
        mesh.boundaryCondition(0, 1/r);
    }
    if (mesh.coordY(border[i]) == 0)
    {
        mesh.boundaryCondition(0, 0);
    }
    if (fabs(ra-r) <= tol)
    {
        mesh.boundaryCondition(i, 0, sin(theta)/ra);
    }
    if (fabs(rb-r) <= tol)
    {
        mesh.boundaryCondition(0, sin(theta)/rb);
    }
}
// Mostramos el valor de las condiciones de frontera
mesh.showBoundaryCondition();
```

La construcción del sistema y su solución son codificados exactamente de la misma manera que el ejemplo anterior. En la gráfica 4.9 se muestra el cálculo del residuo durante cada iteración realizada por el gradiente conjugado y el gradiente conjugado precondicionado con ILU y en la gráfica 4.10 se muestra el cálculo del residuo durante cada iteración realizada por el gradiente conjugado estabilizado y el gradiente conjugado estabilizado precondicionado con ILU.

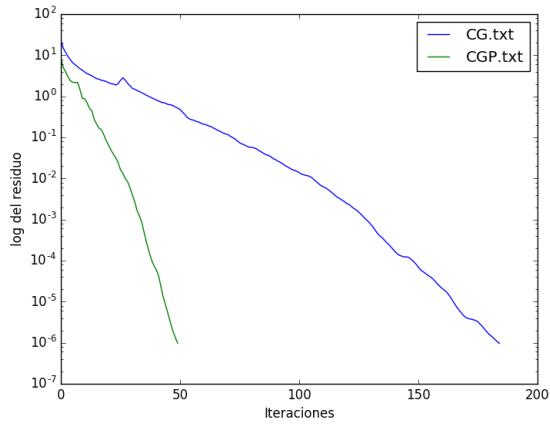


Figura 4.9: La gráfica muestra el residual para el problema de Laplace acotado entre dos semicírculos $\log |r_n|$ vs las iteraciones n, para el método del gradiente conjugado(azul) y el método del gradiente conjugado precondicionado con ILU(verde).

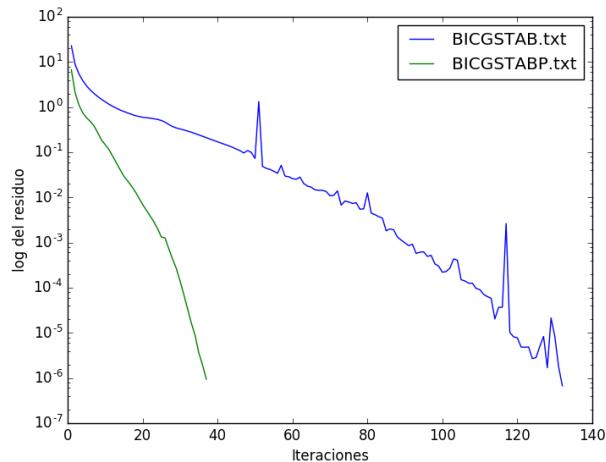


Figura 4.10: La gráfica muestra el residual para el problema de Laplace acotado entre dos semicírculos $\log |r_n|$ vs n, para el método del gradiente conjugado estabilizado(azul) y el método del gradiente conjugado estabilizado precondicionado con ILU(verde).

4.4. Poisson en dos dimensiones

4.4.1. Descripción

El flujo en medios porosos puede ser expresado en términos de la presión como se vió en la sección 2.3. Este problema consiste en determinar la distribución de la presión en un medio homogéneo, este fenómeno puede ser representado mediante la ecuación:

$$-\nabla \cdot (\Gamma \nabla p) = f(x_1, x_2) \quad (4.4.1)$$

El dominio $\Omega = (0, 1) \times (0, 1)$, es un cuadrado unitario, Γ es el tensor identidad y la función queda definida por:

$$f(x_1, x_2) = 2\pi^2 \cos(\pi x_1) \cos(\pi x_2)$$

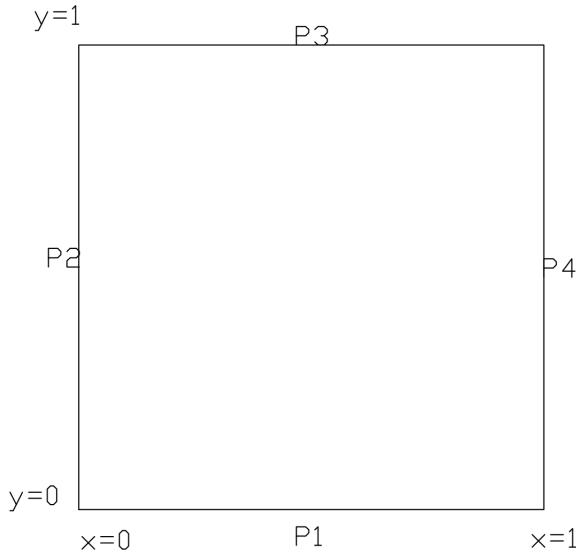


Figura 4.11: Dominio para la ecuación de Poisson

y las condiciones iniciales:

$$\begin{aligned} \nabla p \cdot \underline{v} &= 0 & x_1 &= 0 \quad \text{and} \quad x_1 &= 1, & x_2 & \in (0, 1) \\ p &= \cos(\pi x_1) & x_1 &\in (0, 1), & x_2 &= 0 \\ p &= -\cos(\pi x_1) & x_1 &\in (0, 1), & x_2 &= 1 \end{aligned}$$

Solución analítica:

$$p = \cos(\Pi x_1) \cos(\Pi x_2)$$

Debido a las características del problema la ec. 4.4.1, puede transformarse en la ecuación de Poisson $-\Delta p = f(x_1, x_2)$.

4.4.2. Resultados

En las esquinas $(0,0)$ y $(1,1)$ se aplica mayor cantidad de presión. Esto genera una distribución en el dominio de interés de manera diagonal. En la figura 4.12 se muestra la solución exacta, la solución aproximada y la distribución del error.

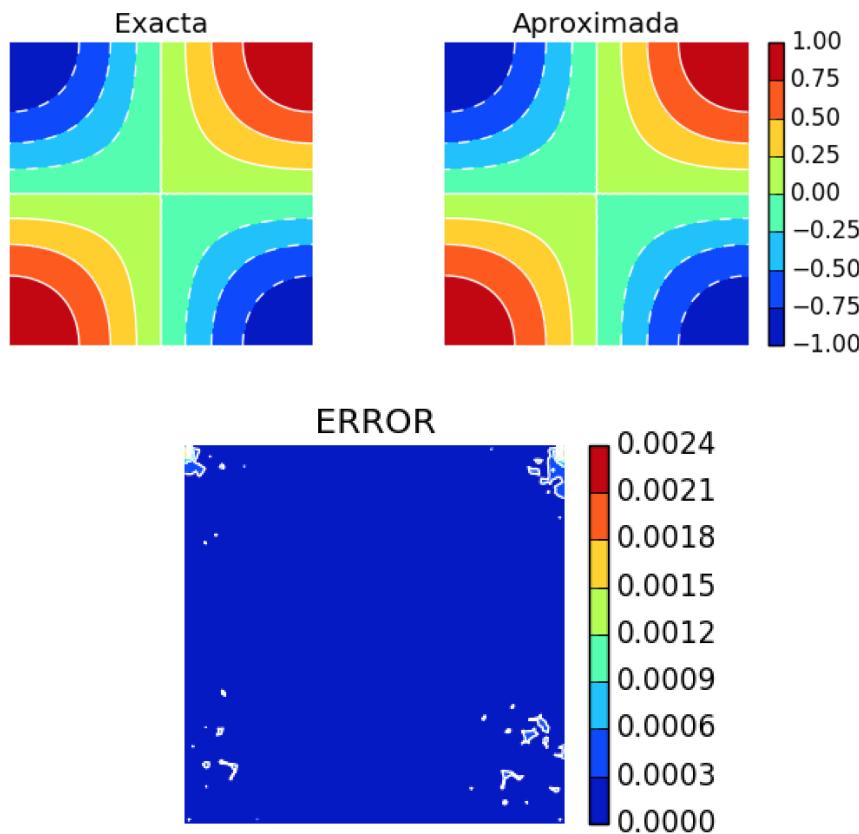


Figura 4.12: Solución de la ecuación de Poisson

4.4.3. Observaciones

Para éste ejemplo se usó la malla observada en la figura 4.13, tiene 3436 nodos y 6670 elementos triangulares.

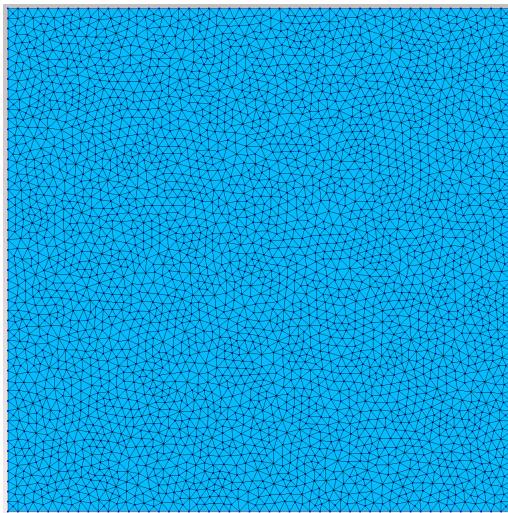


Figura 4.13: Discretización del dominio curvo para la ecuación de Poisson.

Para ver con detalle la codificación del problema, véase lo indicado en el apéndice D. En la gráfica 4.14 se muestra el cálculo del residuo durante cada iteración realizada por el gradiente conjugado estabilizado y el gradiente conjugado estabilizado precondicionado con ILU.

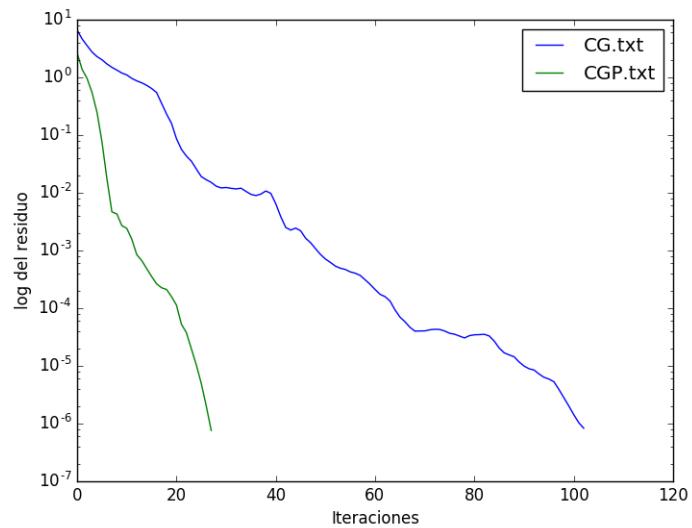


Figura 4.14: La gráfica muestra el residual para el problema de Poisson acotado entre dos semicírculos $\log |r_n|$ vs las iteraciones n, para el método del gradiente conjugado(azul) y el método del gradiente conjugado precondicionado con ILU(verde).

4.5. Flujo monofásico

4.5.1. Descripción

Normalmente la permeabilidad de una región del espacio estudiado no es homogénea. En estos casos el estudio de flujo en medios porosos debe incluir un perfil de permeabilidad. Este perfil creará subdominios en donde el fluido puede avanzar con facilidad y algunos en donde se le dificulte. En este caso se trabaja con la ec. 2.4.1, con las siguientes características:

- El dominio $\Omega = (0, 100) \times (0, 100)$ (Ver Fig. 4.15).
- Una región de alta permeabilidad.
- Una región de baja permeabilidad.
- La densidad $\rho = 1$.
- La viscosidad $\mu = 0,5$.
- Una fuente $q(x, y) = 0$.
- Las condiciones de frontera están dadas por $P_2 = 10$, $P_4 = 1$ y $P_1, P_3 = 0$.

La permeabilidad, matemáticamente es un tensor que esta representado por:

$$\underline{\underline{k}} = \begin{bmatrix} k_{11} & 0 \\ 0 & k_{22} \end{bmatrix} \quad (4.5.1)$$

donde sus componentes indican la permeabilidad en cada punto del dominio, el cual es tomado generalmente constante. Una manera de crear un perfil de permeabilidad es sustituyendo cada componente del tensor por una función que evaluada en cada punto del dominio devuelva un valor distinto. En este caso cada componente es sustituida por una función pseudoaleatoria.

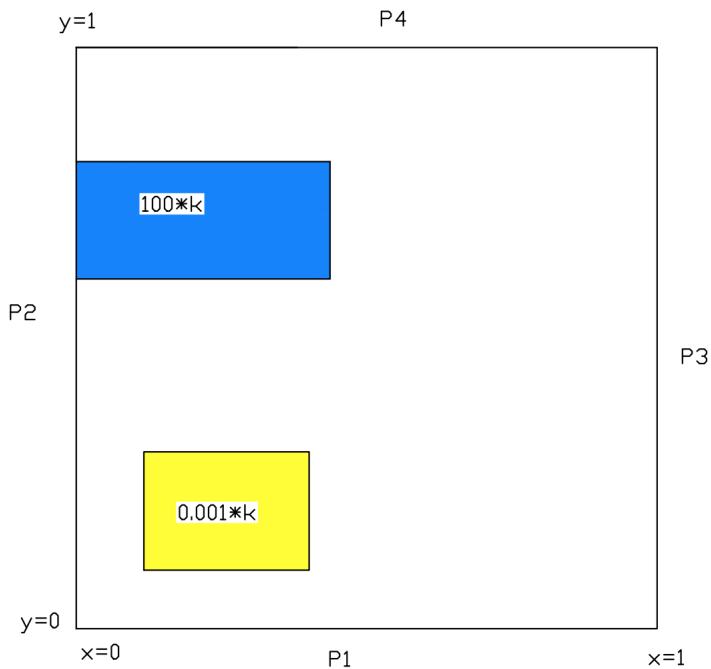


Figura 4.15: Dominio para la ecuación de presión con permeabilidad aleatoria, con una región de alta permeabilidad(azul) y una de baja permeabilidad (amarilla).

4.5.2. Resultados

En la figura 4.16, se muestra la solución calculada por SYSG. Nótese que el fluido va de izquierda a derecha y existe una zona en donde el fluido avanza con facilidad, ésta corresponde a la región de alta permeabilidad, observe que también en una región el fluido avanza con dificultad, ésta corresponde con la zona de baja permeabilidad.

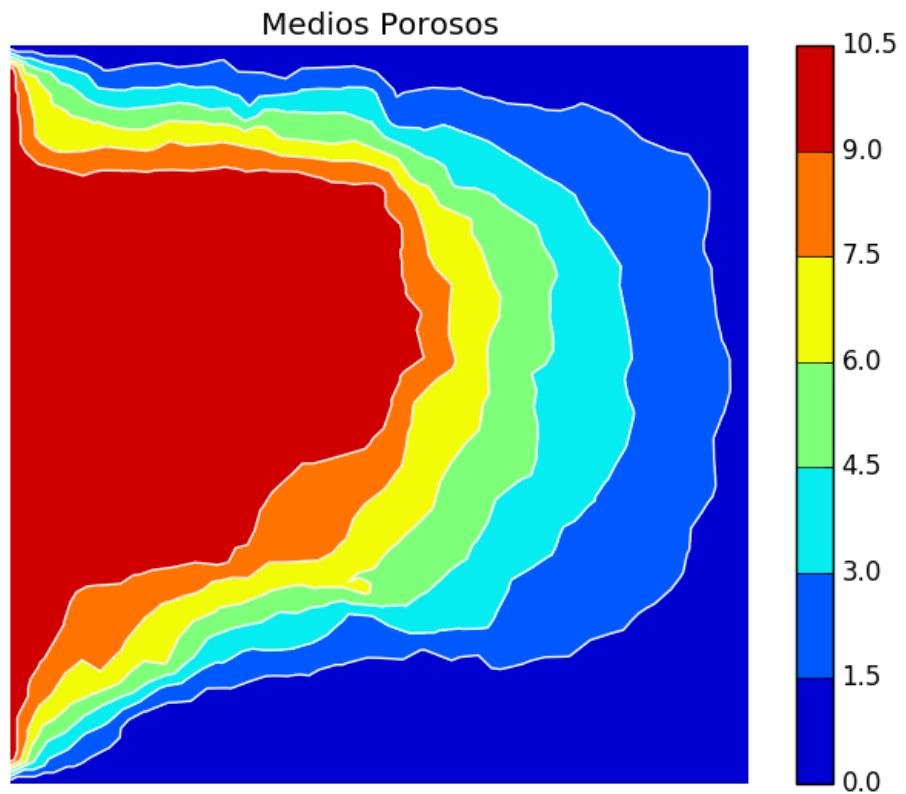


Figura 4.16: Solución para el flujo de medios porosos monofásico.

4.5.3. Observaciones

Para éste ejemplo se uso la malla que puede ser observada en la figura 4.17, tiene 942 nodos y 1954 elementos.

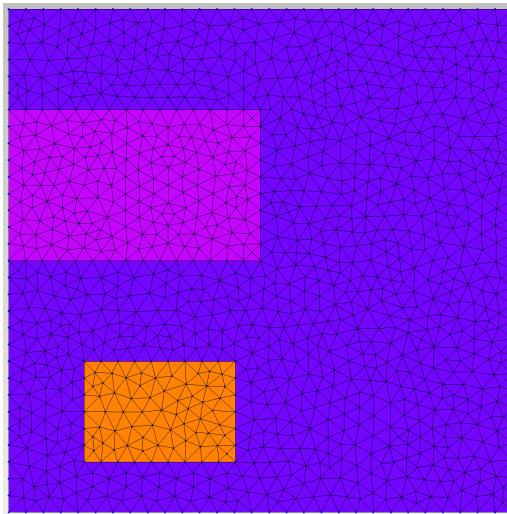


Figura 4.17: Malla para el flujo de medios porosos monofásico.

Para solucionar el problema debe considerarse:

- Cálculo de la permeabilidad en cada punto de la malla.
- La identificación de las regiones de alta y baja permeabilidad.
- Cálculo de la permeabilidad en un elemento triangular.

Cálculo de la permeabilidad en cada punto de la malla

Para el cálculo de la permeabilidad de forma aleatoria en cada punto de la malla se usó:

```
_k11[ i ] = _permeabilidad * ((double)rand() / RAND_MAX);
_k22[ i ] = _permeabilidad * ((double)rand() / RAND_MAX);
```

donde el valor de `_permeabilidad` puede ser especificado por el usuario, por defecto, tiene un valor de uno. Si se gráfica la componente $k11$ (Ver Fig.

4.18) y la componente k_{22} (Ver Fig. 4.19), se puede observar el perfil de la permeabilidad pseudoaleatoria.

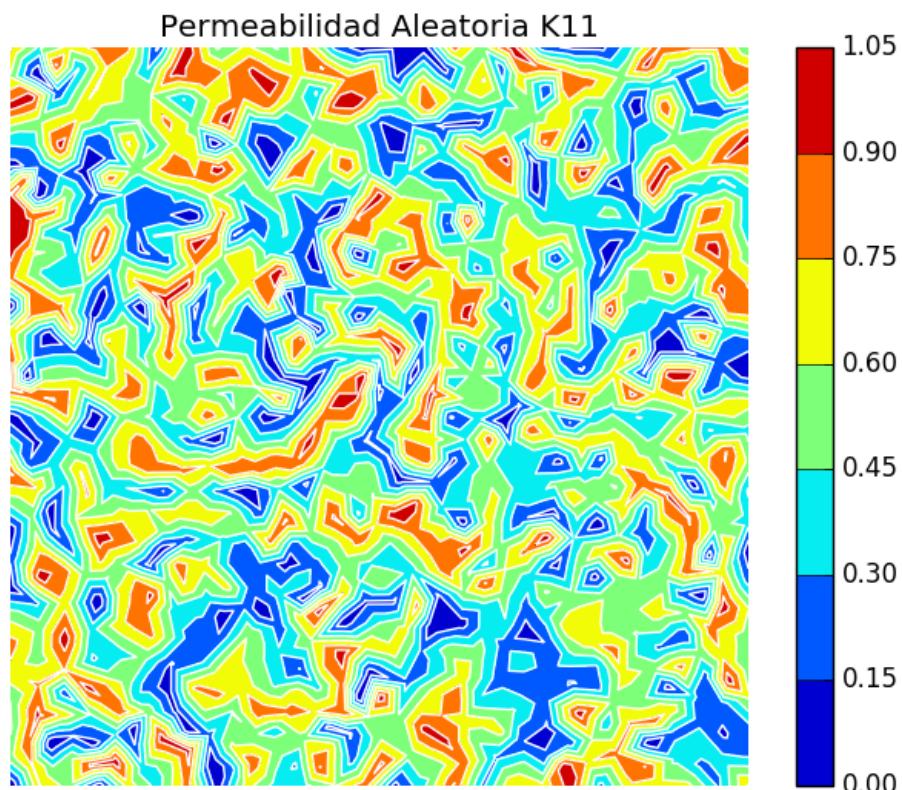


Figura 4.18: Componente k_{11} de la matriz de permeabilidad para cada nodo de la malla.

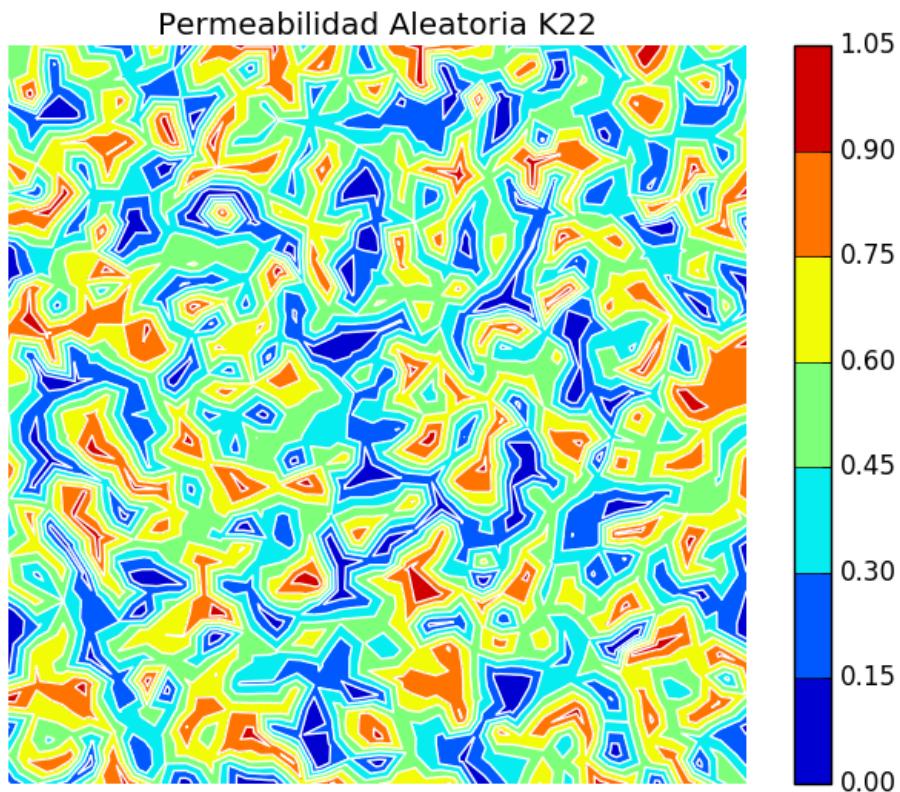


Figura 4.19: Componente k_{22} de la matriz de permeabilidad para cada nodo de la malla.

Identificación de las regiones de alta y baja permeabilidad.

El dominio de interés contiene dos regiones donde la permeabilidad es distinta, estas zonas deben ser identificadas. Posteriormente para cada nodo, que pertenezca a una zona de distinta permeabilidad sus componentes k_{11} y k_{22} deben ser multiplicados, por un valor constante. En éste caso los nodos que aparecen en la zona de alta permeabilidad son multiplicados por 100 y los que aparecen en la zona de baja permeabilidad son multiplicados por 0.001.

Se muestra en las siguientes gráficas el resultado de multiplicar las zonas de distinta permeabilidad.

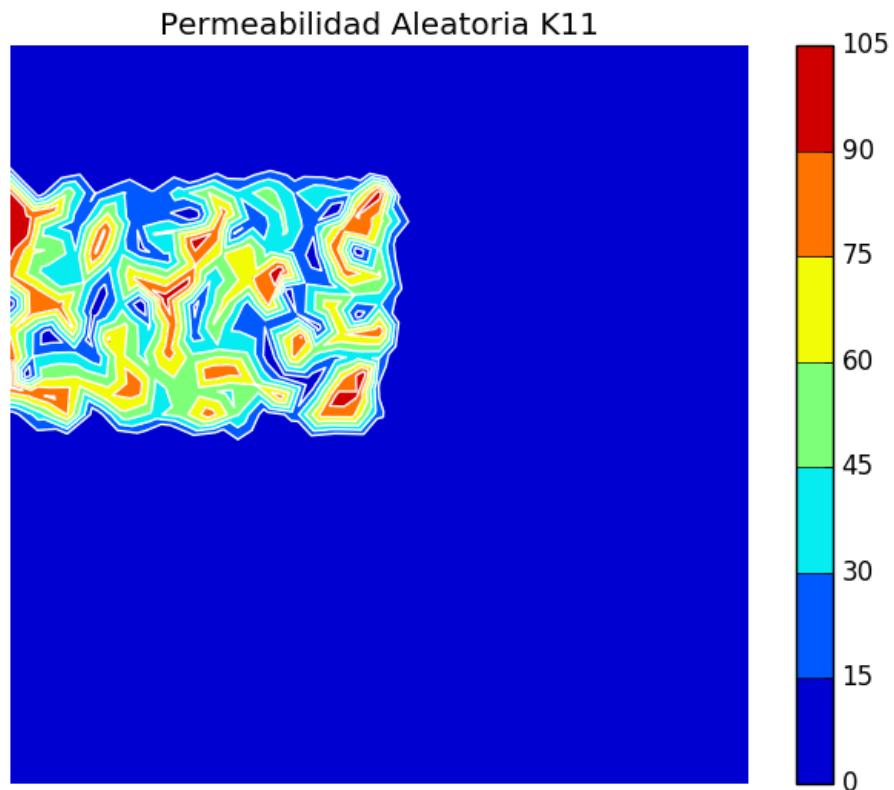


Figura 4.20: Componente k11 de la matriz de permeabilidad para cada nodo de la malla.

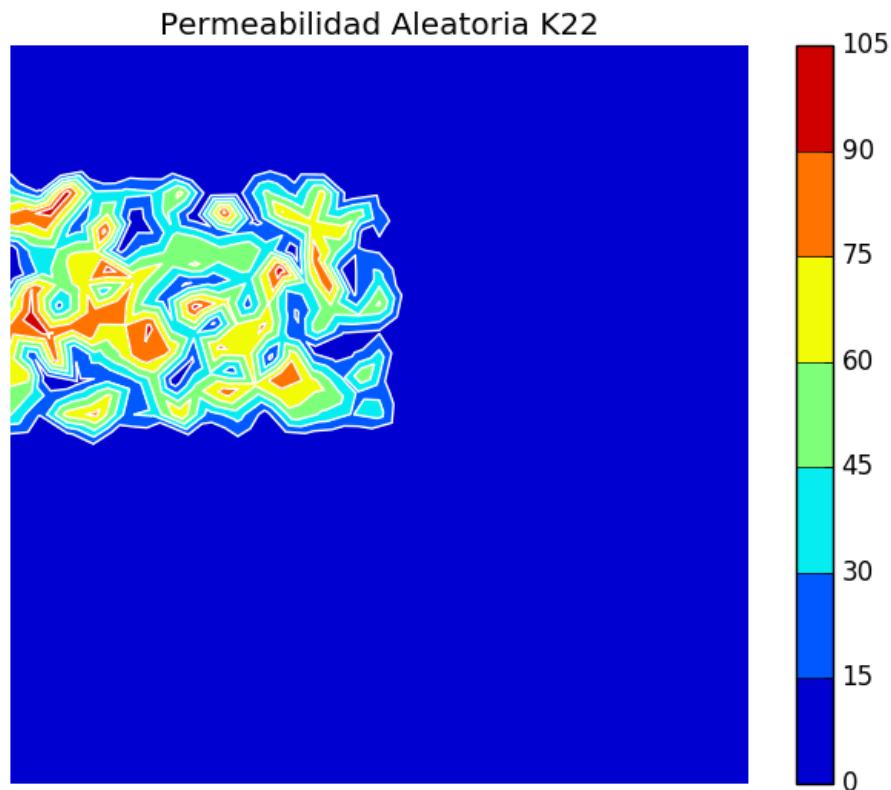


Figura 4.21: Componente k22 de la matriz de permeabilidad para cada nodo de la malla.

Cálculo de la permeabilidad en un elemento triangular

Hasta ahora, cuando se construía la matriz del sistema de ecuaciones, se tenía una permeabilidad constante en todo el dominio de interés. En éste caso, cada nodo tiene su propio tensor de permeabilidad, por lo tanto para calcular los coeficientes: T_{ij} y T_{ik} (véase la sección 2.4), debe tomarse una consideración adicional, la cual consiste en tomar el promedio armónico , para T_{ij} :

$$k_{ij} = \frac{n}{\frac{1}{k_i} + \frac{1}{k_j}}$$

y para T_{ik} :

$$k_{ik} = \frac{n}{\frac{1}{k_i} + \frac{1}{k_k}}$$

4.6. Entrega

Para que la comunidad universitaria tenga acceso completo al software, la entrega del proyecto se hace por medio de una carpeta que puede ser obtenida vía web según lo descrito en el apéndice D.

Capítulo 5

Conclusiones

En este trabajo se presentaron los modelos matemáticos de flujo en medios porosos usando la formulación axiomática presentada por Herrera [12]. Se estudió el método CVFE para el tratamiento de ecuaciones diferenciales parciales. Se construyó la herramienta SYSG cuyo objetivo principal fue:

El diseño y la construcción de software científico para simular problemas de flujo en medios porosos mediante el método CVFE, usando buenas prácticas que aseguren calidad.

Las metas propuestas fueron:

- Realizar la construcción del sistema mediante una metodología de desarrollo de software basada en la norma ISO/IEC 29110 perfil básico.
- Implementar un modelo de proceso de desarrollo de software.
- Obtener los requerimientos funcionales y no funcionales del sistema, del problema y del método numérico de solución seleccionado.
- Proponer y diseñar una arquitectura para el sistema.
- Implementar el sistema con base en la arquitectura diseñada, con el fin de cumplir los requerimientos.
- Realizar pruebas de funcionalidad, de paquete y de tipo *benckmark*, para calibrar numéricamente el sistema.
- Resolver problemas de flujo en medios porosos.
- Entregar la documentación generada del sistema final.

Este trabajo presenta de forma sencilla el método CVFE básico, realizando una recopilación de la literatura.

Las pruebas mostradas en el capítulo 3 muestran que el uso de paradigmas de programación modernos; programación orientada a objetos, programación genérica y técnicas de metaprogramación producen códigos más flexibles, más poderosos y que conviene utilizarlos en la construcción de software científico. Uno de sus beneficios es que para problemas similares, la codificación de la solución es similar a la codificación de los problemas de benchmark. Con el fin de tener un grado de confianza en SYGS fueron utilizados tres problemas para su calibración.

La simulación de la ecuación de Laplace que sirve para determinar el flujo en un medio homogéneo, demostró que el método CVFE no requiere de mallas tan finas para alcanzar una buena aproximación de la solución final, además la mayor cantidad de problemas se presentan en las esquinas del dominio, pues en estos puntos la derivada tiene problemas, una posible solución es hacer un refinamiento en estos lugares.

La simulación de la ecuación de Laplace limitada por dos semicírculos demostró la versatilidad de la herramienta SYGS y del método CVFE en el tratamiento de dominios no rectangulares.

La simulación de la ecuación de Poisson muestra que la aproximación utilizada para tratar el término fuente en cada volumen de control es correcta.

En los problemas benchmark se mostró una gráfica del residuo contra las iteraciones, cuando se utilizó el gradiente conjugado (CG) se observó la disminución del error en cada iteración y cuando se utilizó el gradiente conjugado precondicionado (CGP) se mostró que la tendencia mejoraba, por lo tanto concluimos que el cálculo del residuo en el CG no siempre disminuye y esto está relacionado directamente con el condicionamiento de la matriz. En el caso de BICGSTAB se muestra en cada gráfica obtenida, una convergencia irregular, el análisis de esta convergencia sale de los objetivos de este trabajo, sin embargo con el fin de demostrar que el método fue implementado correctamente se verificaron otras implementaciones (véase [23], [20]).

Se muestra que una manera de investigar el flujo en medios porosos es

utilizando perfiles de permeabilidad construidos por medio de funciones pseudoaleatorias.

Los objetivos y las metas fueron alcanzadas satisfactoriamente. De manera general se concluye que la construcción de software científico es una tarea complicada debido a todos los conocimientos especializados que se requieren.

5.1. Crítica a los procesos de Software Científico

Los científicos que se dedican a la investigación desarrollan sistemas destinados a la solución de problemas específicos. Normalmente estos sistemas son muy eficientes, sin embargo, éstos pocas veces trascienden a la solución de problemas más grandes a pesar de tener requerimientos similares, esto es debido al proceso que utilizan en la construcción de software.

La construcción de software involucra una gran cantidad de variables y en general es un proceso complicado, en este sentido el proceso de Modelación matemática y computacional (véase la sección 1.1.1) es una herramienta útil para la construcción de software científico académico, pero resulta ser limitado y poco flexible. No ofrece ninguna herramienta que permita su evolución, su mantenimiento y su reutilización. No implementa planes de mitigación de riesgos, los problemas de construcción son resueltos conforme éstos surgen.

La adaptación al estándar ISO/IEC 29110 Perfil Básico desarrollado por Ernesto Albarrán [4], toma el proceso de MMC y lo amplia con herramientas útiles que permiten la construcción de software de manera ordenada. Proporciona un medio para almacenar información de los proyectos, es decir; acumula información útil de administración que puede ser utilizada por los científicos en proyectos subsecuentes para dar estimaciones de recursos. La adaptación es sencilla para personas que no tienen un amplio conocimiento de ingeniería de software, es flexible, pues podemos tomar sólo lo necesario para nuestro proyecto. Sin embargo, esta adaptación excluye información sobre el alcance del proyecto, lo cual no nos parece que sea correcto, por que el alcance viene dado por el problema inicial. Al igual que MMC no se incluyen etapas de optimización para mejorar el rendimiento del sistema. Tampoco

incorpora herramientas para el almacenamiento de información de interés científico; por ejemplo, si durante la construcción de software nos encontramos con un problema matemático complicado no hay ninguna herramienta que permita almacenar el procedimiento utilizado en su solución. Se concluye que:

- Su uso en la construcción de software científico siempre es conveniente, pues garantiza un proceso ordenado y bien documentado, con lo cual puede ser extendido o reutilizado fácilmente.
- Los estándares deben adaptarse al científico y no el científico a los estándares.

5.2. Aportación

Las aportaciones principales de este trabajo son:

- Una crítica al proceso utilizado en la construcción de software científico.
- Dejar un código computacional libre escrito en C++ y bien documentado que queda a disposición de la comunidad universitaria para mejorarlo, reutilizarlo y compartirlo.
- La implementación de un proceso de construcción de software que asegure calidad.

5.3. Trabajo Futuro

La herramienta SYGS permite desarrollar distintos tipos de problemas. Debido a la documentación es posible adicionar nuevas funcionalidades a la herramienta de forma sencilla. Algunas características que pueden ser trabajadas a corto plazo son:

- Numeración de caras y aristas de cada elemento triangular.
- Uso de perfiles de permeabilidad que representen de manera más exacta condiciones reales.
- Manejo de términos advectivos.

- Manejo de términos temporales.
- Tratamiento de problemas de transporte.
- Inclusión de tecnologías de procesamiento paralelo.

A mediano y largo plazo:

- Construcción del dominio de interés.
- Desarrollo del algoritmo de Delaunay para la discretización del dominio.
- Desarrollo de sistemas acoplados.
- Aplicaciones de interés para la comunidad científica como: aguas subterráneas, geotermia, recuperación de hidrocarburos, etc.

Apéndices

Apéndice A

ISO/IEC 29110 Perfil Básico

Adaptar las normas utilizadas por grandes empresas en la construcción de software en el cómputo científico, resulta muy complicado y en la mayoría de los casos poco significativo para el científico que se desempeña en la academia. Conviene utilizar una norma que pueda ser utilizada con bajos recursos de tiempo, dinero y de personas.

La ISO/IEC 29110 [24], es una norma internacional para el ciclo de vida de un proyecto de software. Está dirigida a pequeñas empresas (a lo más 25 personas). Define roles específicos para el equipo de trabajo, de tal manera que cada miembro conozca sus responsabilidades. Propone dos procesos: la administración y la implementación de software(Ver Fig. A.1).

A.0.1. Administración del proyecto

Es un proceso presente durante todo el ciclo de vida del proyecto. Comienza antes de la construcción de cualquier cosa, planifica y da seguimiento, establece un conjunto de tareas que permitan su realización exitosa.

En la norma ISO/IEC 29110 éste proceso esta compuesto de cuatro actividades, las cuales son divididas en tareas, en la primera se realizan 15, en la segunda 6, en la tercera 3 y finalmente en la última actividad se realizan 2, dando un resultado total de 26(véase Fig A.2).

Cuenta con 7 objetivos fundamentales, éstos son:

1. Desarrollar un plan de proyecto, de acuerdo a las necesidades, debe ser revisado y aceptado por el cliente. Se debe hacer una estimación de las tareas y de los recursos necesarios.

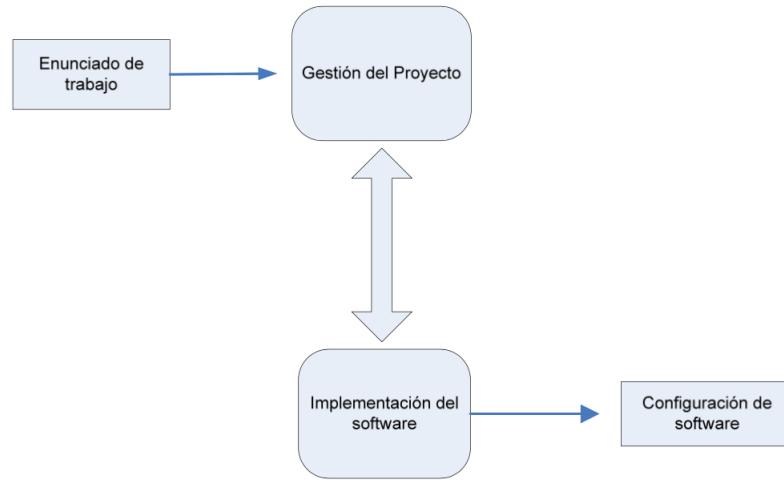


Figura A.1: Procesos del Perfil Básico ISO/IEC 29110 Perfil Básico, Tomada de Norma Técnica NTP-RT-ISO/IEC TR 29110-5-1-2,INDECOPI,2012.

2. Revisar y documentar el avance del proyecto, utilizando el plan previamente desarrollado.
3. Atender los cambios que puedan ser solicitados.
4. Llevar a cabo reuniones con el equipo de trabajo y con el cliente.
5. Identificar riesgos.
6. Establecer una estrategia de control de versiones.
7. Verificar la calidad del software, es decir garantizar que se cumplió con las necesidades del cliente.

A.0.2. Implementación de Software

Es un proceso, cuyo objetivo es transformar las especificaciones del software en un sistema ejecutable [11]. Resulta de realizar en orden las actividades, de análisis, diseño, construcción, integración y pruebas (Ver Fig. A.2).

La norma ISO/IEC 29110, permite la utilización de cualquier modelo de procesos de software. Éste proceso esta compuesto de 6 actividades, las cuales

son divididas en tareas, en la primera se realizan 2, en la segunda 7, en la tercera 8, en la cuarta 7, en la quinta 11 y finalmente en la última actividad se realizan 6, dando un resultado total de 41(véase Fig A.3).

Cuenta con 6 objetivos fundamentales, éstos son:

1. Realizar las tareas de las actividades cumpliendo el plan del proyecto.
2. Definir y analizar los requerimientos, los cuales tienen que ser aprobados por el cliente.
3. Definir la arquitectura y el diseño del sistema.
4. Implementar los módulos que conforman el sistema y definir sus pruebas.
5. Llevar a cabo la integración de los módulos y utilizar casos de prueba para verificar que cumpla con los requisitos.
6. La documentación del software, para el usuario, operación y mantenimiento.

Ventajas

Según la ISO [24], los estándares internacionales hacen que: *las cosas realmente funcionen*, pues garantizan la calidad, seguridad y la eficiencia, de los productos y de los servicios, lo cual facilita entre muchas cosas el comercio entre países.

Para el estándar ISO/IEC 29110, las ventajas son:

1. Definir correctamente los requerimientos y los productos que deben ser entregados al cliente.
2. Proporcionar visibilidad y acciones correctivas sobre los problemas y desviación del proyecto.
3. La utilización de un proceso sistemático que satisfaga las necesidades del cliente.



Figura A.2: Flujo entre las actividades del Proceso Administración del Proyecto, incluyendo los productos de trabajo más relevantes y la relación existente entre ellos. Tomada de Norma Técnica NTP-RT-ISO/IEC TR 29110-5-1-2,INDECOPI,2012.



Figura A.3: Flujo de trabajo del Proceso de Implementación de Software, compuesto por seis actividades, las cuales están descritas por 41 tareas. Tomada de Norma Técnica NTP-RT-ISO/IEC TR 29110-5-1-2, INDECOP, 2012.

Apéndice B

Gmsh

Gmsh[18] es una herramienta multiplataforma(GNU/Linux, Mac OS, Windows), puede ser obtenida desde <http://gmsh.info/> actualmente se trabaja con la versión 2.13.2, se distribuye bajo la licencia GPL (*del inglés General Public License*)de GNU. Es una herramienta para generar mallas 3D y 2D sobre dominios geométricos, cuenta con cuatro módulos:

- Geométrico.
- Mallado.
- Solucionadores.
- Pos-procesado.

En este trabajo sólo nos interesan los dos primeros módulos, los cuales son descritos en las siguientes secciones.

B.1. Módulo Geométrico

Gmsh incluye una interfaz gráfica y un lenguaje propio, ambos pueden ser usados en la construcción del dominio de interés.

B.1.1. Entidades geométricas

Las entidades geométricas, son las encargadas de la construcción del dominio, las usadas en este trabajo fueron:

- **Point.** Crea un punto según las coordenadas dadas.
- **Line.** Crea una línea entre dos puntos.
- **Circle.** Define una semicircunferencia.

B.1.2. Entidades Físicas

En ocasiones, para la construcción del dominio de interés, es necesario el uso de entidades geométricas auxiliares, las cuales no se requieren en la discretización. En tales casos gmsh permite seleccionar en una entidad física aquellas entidades geométricas que formen su dominio. Las entidades físicas usadas en este trabajo fueron:

- **Physical Line.** Identifica líneas como entidades físicas, son útiles para la identificación de los nodos que están en las fronteras.
- **Physical Surface.** Identifica las superficies que serán discretizadas.

B.1.3. Scripts

A continuación se presenta el dominio para la ecuación de Laplace en un semicírculo(véase la sección 4.3).

```

1 lc = 0.2
2 Point(1) = {0.5, 0, 0, lc}; // Punto del dominio 1
3 Point(2) = {1.0, 0.0, 0, lc}; // Punto del dominio 2
4 Point(3) = {0, 1, 0, lc}; // Punto del dominio 3
5 Point(4) = {0, 0.5, 0, lc}; // Punto del dominio 4
6 Point(5) = {0, 0, 0, lc}; // Punto auxiliar
7 Line(1) = {1, 2}; // Línea entre el punto 1 y 2
8 Circle(2) = {2, 5, 3}; // Semicircunferencia
9 Line(3) = {3, 4}; // Línea entre el punto 3 y 4
10 Circle(4) = {1, 5, 4}; // Semicircunferencia
11 Physical Line(5) = {3, 4, 1, 2}; // Fronteras del dominio
12 Line Loop(6) = {3, -4, 1, 2}; // Define una superficie
13 Plane Surface(7) = {6}; // Entidad geométrica
14 Physical Surface(8) = {7}; // Dominio a ser mallado

```

En la figura B.1 puede observarse el dominio generado por medio del script anterior.

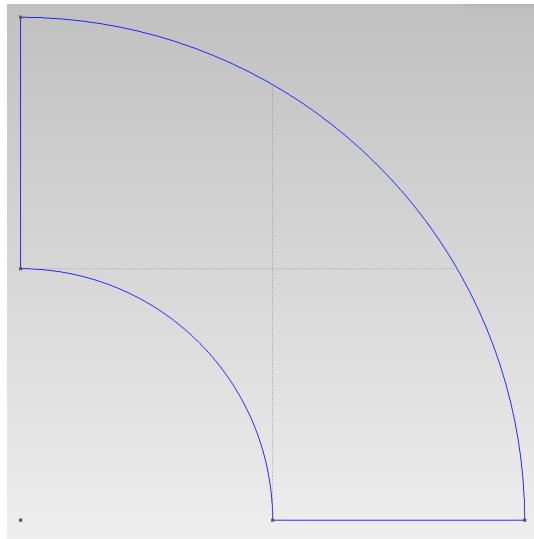


Figura B.1: Geometría definida en gmsh.

B.1.4. Módulo de Mallado

Discretiza el dominio definido con el módulo geométrico. El proceso consiste en dividir la figura en triángulos o cuadrángulos para 2D y para 3D en tetraedros, hexaedros, prismas y pirámides. Cuenta con varios algoritmos de discretización en el caso de 2D se tienen: MeshAdapt, Delaunay y Frontal. Para mallar vamos a gmsh y elegimos el algoritmo a utilizar, para esto vamos a Tools - Options - Mesh - General (Fig. B.2).

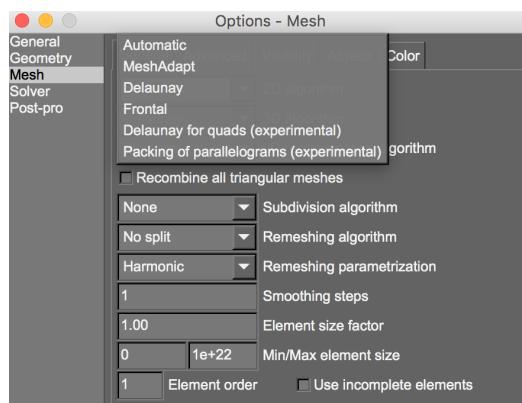


Figura B.2: Menú Options de gmsh.

El paso final es ir a Modules - Mesh - 2D (Ver Fig. B.3).

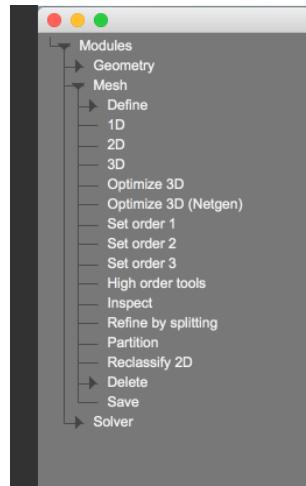


Figura B.3: Mesh - 2D, para discretizar el dominio en elementos triangulares.

El resultado se puede observar en la Fig. B.4.

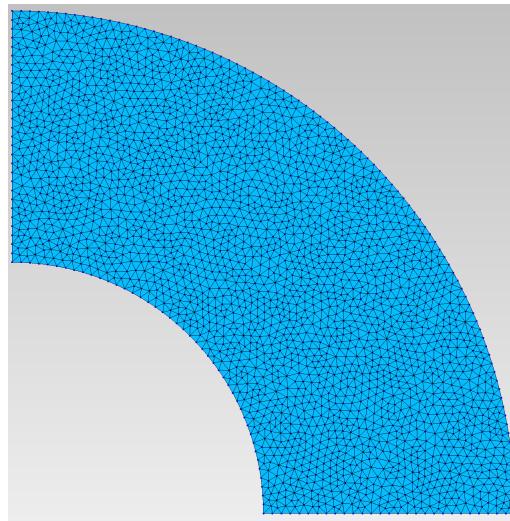


Figura B.4: Define la triangulación de Delaunay sobre el dominio usando gmsh.

Guardamos la malla en File - Save Mesh.

B.2. Archivo .msh

La discretización del dominio en una malla queda almacenada en un archivo de texto con extensión ".msh", cuyo contenido es:

```
1 $MeshFormat
2 2.2 0 8
3 $EndMeshFormat
4 $Nodes
5 2519
6 1 0.5 0 0
7 2 1 0 0
8 3 0 1 0
9 4 0 0.5 0
10 5 0.5178571428571428 0 0
11 6 0.5357142857142857 0 0
12 7 0.5535714285714285 0 0
13 8 0.5714285714285714 0 0
14 9 0.5892857142857142 0 0
15 .
16 .
17 .
18 $EndNodes
19 $Elements
20 5036
21 1 1 2 5 1 1 5
22 2 1 2 5 1 5 6
23 3 1 2 5 1 6 7
24 4 1 2 5 1 7 8
25 .
26 .
27 .
28 189 2 2 8 7 996 2499 629
29 190 2 2 8 7 2506 1302 267
30 191 2 2 8 7 629 2499 862
31 192 2 2 8 7 2264 1751 899
32 193 2 2 8 7 209 1477 733
33 194 2 2 8 7 246 2264 899
34 .
35 .
36 .
37 $EndElements
```

se divide en tres bloques delimitados por las palabras:

- **MeshFormat-EndMeshFormat.** Indica la Versión utilizada, el tipo del archivo y el tamaño del archivo.
- **Nodes-EndNodes.** Indica primero el número total de nodos, además lista los nodos con su ubicación por medio de coordenadas.
- **Elements-EndElements** Indica primero el número total de elementos presentes en el archivo, además lista los elementos, su tipo, número de etiquetas, la primera etiqueta que indica la entidad física a la que pertenece, la segunda etiqueta indica la entidad geométrica a la que pertenece y el identificador del nodo que lo conforma.

Así pues el archivo indica:

- **Línea 5.** Indica el número de nodos totales en el archivo.
- **Línea 6.** Identifica a cada nodo e indica su posición espacial con las coordenadas x, y, z.
- **Línea 20.** Indica el número de elementos totales en el archivo.
- **Línea 21.** Indica un identificador de elemento, un tipo de elemento en este caso es una línea de dos nodos (véase [18]), el número de etiquetas, las etiquetas y finalmente los dos nodos que forman el elemento.

La descripción de este archivo es muy importante para SYSG pues en nuestro caso consideramos a los elementos tipo 1 como aquellos que se encuentran sobre la frontera.

Apéndice C

Lenguaje de Programación C++

Es un lenguaje de programación multiparadigma, desarrollado por Bjarne Stroustrup a mediados del los 80, como extensión del lenguaje C. Actualmente es usado ampliamente por científicos e ingenieros en la construcción de sistemas que permitan resolver problemas.

C.1. Programación Genérica(PG)

Es una técnica de programación que permite desarrollar códigos eficientes y organizados que permitan su reutilización de forma sencilla[25]. Definen una sola implementación para un amplio número de casos, por lo tanto pueden reducir el número de líneas de código necesarias para la construcción de un proyecto. Pueden utilizar polimorfismo estático. Debido a lo anterior es usualmente utilizada en la construcción de bibliotecas.

En C++ son usados los templates para hacer uso de la PG.

C.1.1. Templates

Es un mecanismo de reutilización de código de C++. Hay dos tipos de templates: funciones template y clases template, en ambos casos proveen un comportamiento para diferentes tipos de datos. Representan una familia de funciones o clases según sea el caso. La definición de una clase o una función template es prácticamente igual que su definición ordinaria a excepción que se agrega un renglón extra que indicara su parametrización. Por ejemplo:

```

1 template <typename T>
2 class Vector
3 {
4     private:
5         T *data;
6         int size;
7     public:
8         Vector() {};
9         ~Vector();
10 };

```

En este caso se utiliza la palabra clave `template` y el parámetro `T`, para indicar la familia de clases, note que el parámetro `T` se utiliza para definir el tipo de dato. Dependiendo de las necesidades del usuario `data` puede ser en principio cualquier tipo de dato primitivo o un objeto. Los templates son definidos en tiempo de compilación por lo tanto es posible tener varias clases con tipos diferentes con una sola implementación. Sin embargo, en ocasiones algunas operaciones no estarán optimizadas para algún tipo de dato, en ese momento se tiene la oportunidad de hacer una especialización la cual permite implementar la función para el tipo de dato.

C.1.2. Metaprogramación

Erwin Unruh en 1994 descubrió que los templates podían ser ocupados para realizar cálculos en tiempo de compilación. Esta característica fue llamada *template instantiation* y esto es conocido como metaprogramación.

Expression Templates

Es una técnica de metaprogramación, que permite evaluar datos solo cuando sean necesarios lo que permite tener códigos más eficientes.

Apéndice D

SYSG

El software científico SYSG, está disponible para su estudio en la liga <https://bitbucket.org/m0nT3cR1s70/syssg>. Cuenta con una archivo read-me que contiene información importante del proyecto y cuatro carpetas:

- Código
- DocumentaciónDoxygen
- DocumentaciónISO
- Tesis

Se explica el contenido de cada carpeta en las siguientes secciones.

D.1. Código

Dentro de esta carpeta se incluye el código completo del proyecto, las pruebas tipo benchmark, las mallas utilizadas en la carpeta Gmsh y un archivo makefile.

El archivo main.cpp cuyo contenido es:

```

1 int main(int argc, char** argv)
2 {
3     laplace(); // Laplace
4     //laplaceS(); // Laplace en un semicirculo
5     //poisson(); // Poisson
6     //weird(); // Validamos el sistema SYSG
7     return 0;
8 }
```

en donde es posible elegir, qué ejemplo, quiere ser ejecutado, para realizar la simulación, en éste caso la ecuación de laplace. Para revisar algún otro basta con comentar y descomentar lo requerido.

Para compilar el código basta con el uso de la instrucción:

make

para su ejecución:

make run

como resultado tendrá varios archivos de texto generados:

- **sol.txt**. Sólo disponible para las pruebas de benchmark y contiene la solución exacta del problema en los nodos de la malla.
- **solx.txt**. Contiene la solución aproximada de el problema usando el método CVFEM.
- **error.txt**. Solo disponible para las pruebas de benchmark y contiene la resta de la solución exacta con la aproximada.

Para su visualización se incluye un programa python, para su uso:

python visualize.py

D.2. Documentación Doxygen

Doxygen [26] es una herramienta, para generar un API del código fuente de un proyecto. En este caso es incluido el API en forma de una página web y de un documento en latex.

- **API en formato html.** Para acceder a esta documentación abrir el archivo index.html en cualquier navegador web, el cual ésta contenido en la carpeta html.
- **API en formato Latex.** Para acceder a este documento, basta con ejecutar el archivo makefile, que contiene la carpeta latex.

D.3. Documentación ISO

Se presenta toda la documentación obtenida después de aplicar el están-dar ISO/IEC 29110 con un enfoque evolutivo, se presentan las 9 iteraciones realizadas, las cuales tuvieron una duración de un mes.

D.4. Tesis

Con el fin de centralizar toda la información del proyecto, se incluyen las dos tesis desarrolladas como producto de esta investigación, las cuales son:

- Implementación del el Estándar ISO/IEC 29110 Perfil Básico para el desarrollo de software del tipo científico.
- Simulación de flujo en medios porosos usando CVFE.

En donde la primera es la encargada de revisar con detalle la norma ISO y de la adaptación.

Bibliografía

- [1] Computational Science Subcommitee. Computational science: Ensuring america's competitiveness. Technical report, Report to the President, 2005.
- [2] Gene H. Golub and James M. Ortega (Auth.). *Scientific Computing. An Introduction with Parallel Computing*. Elsevier Inc, Academic Press.
- [3] Zhangxin Chen. *Computational Methods for Multiphase Flows in Porous Media*. Society for Industrial and Applied Mathematics, 2006.
- [4] Ernesto Rodríguez Albarrán. Implementación del estandar iso/iec 29110 perfil básico para el desarrollo de software del tipo científico, tesis de maestría, 2017.
- [5] Yuri Skiba. *Métodos y esquemas numéricos un análisis computacional*. Universidad Nacional Autónoma De México, 2005.
- [6] Versteeg H.and Malalasekra W. *An introduction to computational fluid dynamics*. Pearson Ed., 2ed. edition, 2007.
- [7] Top500 supercomputer sites visitado el 7 de noviembre de 2016.
- [8] Judith Segal and Chris Morris. Developing scientific software. *IEEE*, 2008.
- [9] Ivar Jacobson. *Object Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley Professional, 1992.
- [10] Grady Booch. *Object-oriented Analysis and Design with Applications*. Object Technology Series. Addison-Wesley, 2007.

- [11] Ian Sommerville and Pete Sawyer. *Requirements Engineering - A Good Practice Guide*. John Wiley and Sons, 1997.
- [12] Ismael Herrera and George F. Pinder. *Mathematical Modeling in Science and Engineering An Axiomatic Approach*. WILEY, 2012.
- [13] Royal Eugene Collins. *Flow of fluids through porous materials*. Penwell Publishing Co., 1961.
- [14] Alex E. Larreteguy y Jose Converti. Algoritmo para resolver la ecuación de transporte basado en volúmenes de control. 2011.
- [15] Tamal K Dey Siu-Wing Cheng and Jonathan Shewchuk. *Delaunay mesh generation*. CRC Press, 2013.
- [16] Jesús Xamán y Miguel Angel Gijón. *Dinámica De Fluidos Computacional Para Ingenieros*. Palibrio, 2015.
- [17] Paul L. George and Houman Borouchaki. Delaunay triangulation and meshing. 1998.
- [18] Gmsh visitado del 25 de agosto de 2016 <http://gmsh.info/>.
- [19] Paul Deitel and Harvey M. Deitel. *C++ How to Program*. Prentice Hall, 7 edition, 2009.
- [20] Shao-Liang Zhang. A class of product-type krylov-subspace methods for solving nonsymmetric linear systems. *Journal of Computational and Applied Mathematics*, 2001.
- [21] Yousef Saad. *Iterative methods for sparse linear systems*. Society for Industrial and Applied Mathematics, 2 edition, 2003.
- [22] Luis Miguel de la Cruz Salas. Método de volumen finito para flujo de una fase. Technical report, Instituto de Geofísica, UNAM, 2011.
- [23] Morten M.T. Wang and Tony W.H. Sheu. An element-by-element bics-tab iterative method for three-dimensional stady navier-stokes equations. *Journal of Computational and Applied Mathematics*, 1995.
- [24] Iso visitado el 7 de noviembre de 2016.

- [25] David Vandevoorde and Nicolai M. Josuttis. *C++ Templates: The Complete Guide*. Addison-Wesley Professional, 2002.
- [26] Dimitri van Heesch. *Doxygen 1.3.8 manual*, 2004.
- [27] Christophe Geuzaine and Jean-Francois Remacle. *Gmsh Reference Manual*, 9 October 2016.
- [28] Luis Miguel de la Cruz Salas. Flujo en una y dos fases en medios porosos: Modelos matemáticos, numéricos y computacionales. Technical report, Instituto de Geofísica, UNAM, 2012.
- [29] Juan Luis Vazquez. *The porous medium equation: mathematical theory*. Oxford mathematical monographs. Clarendon, 1 edition, 2007.
- [30] Giuseppe Toscani. *A central limit theorem for solutions of the porous medium equation*. 2005.
- [31] Jorge A. Ososrno Manzo y Rafael Rodriguez Nieto Agustin V. Mejía Diaz, Ricardo Gomez Saavedra. *Apuntes de Flujo de Fluidos en Medios Porosos*. Oxford mathematical monographs. Facultad de Ingeniería UNAM, 1 edition, 2007.
- [32] Maria Giulia Di Giuseppe Anna Tramelli Claudia Troise Renato Somma Giuseppe De Natale Stefano Carlino, Antonio Troiano. Exploitation of geothermal energy in active volcanic areas: A numerical modelling applied to high temperature mofete geothermal field, at campi flegrei caldera(southern italy). *ELSEVIER*, 2015.
- [33] Alexandre Lamoureux and Bantwal R. Baliga. Improved formulations of the discretized pressure equation and boundary treatments in collocated equal-order control-volume finite-element methods for incompressible fluid flow. *Taylor and Francis*, 2011.
- [34] Joao Flávio Viera de Vasconcellos and Clovis R. Maliska. A finite method based on voronoi discretization for fluid flow problems. *Taylor and Francis*, 2010.
- [35] Hans Petter Langtangen. *Computational Partial Differential Equations*. Springer, 1999.

- [36] Richard L. Burden and Douglas J. Faires. *Análisis numérico*. Thomson Learning, 2002.
- [37] Ephraim M. Sparrow W. J. Minkowycz and Jayathi Y. Murthy. *Handbook of Numerical Heat Transfer*. Wiley, 2006.
- [38] Ian Sommerville. *Ingeniería del Software - 7Ed Spanish*. Pearson Educación, 2005.
- [39] Jose Antonio Cerrado Somolinos. *Introducción a la ingeniería del software*. Centro de estudios Ramon Areces - UNED, 2006.
- [40] Diane F. Kelly. A software chasm: Software engineering and scientific computing. *IEEE*, 2007.
- [41] Bjarne Stroustrup. *Programming: Principles and Practice Using C++*. 2nd Edition. Addison-Wesley Professional, 2 edition, 2014.
- [42] Bjarne Stroustrup. *The C++ Programming Language, 4th Edition*. Addison-Wesley Professional, 4 edition, 2013.
- [43] Eduardo Raffo Lecca. Programación genérica en c++, usando metaprogramación. *Industrial Data*, 10(1):080–087, 2007.
- [44] Hayo Thielecke. An introduction to c++ template programming. 2014.
- [45] Nicolai Josuttis. C++ templates: the complete guide, 2003.
- [46] Eduardo Raffo Lecca. El poder del turbo c++ ver. 3.0. *Lima, Perú*, 1995.
- [47] David Abrahams and Aleksey Gurtovoy. *C++ template metaprogramming: concepts, tools, and techniques from Boost and beyond*. Pearson Education, 2004.