UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

FACULTAD 9

Título: Desarrollo de Biblioteca de Métodos Numéricos (BMN), referente a Sistemas de Ecuaciones Lineales, Sistemas de Gran Dimensión y Poco Densos e Integración Numérica.

TRABAJO DE DIPLOMA PARA OPTAR POR EL TÍTULO DE INGENIERO EN CIENCIAS INFORMÁTICAS

Autor(es): Carlos Luis Serrano Rosales
Solangel Rodríguez Vázquez

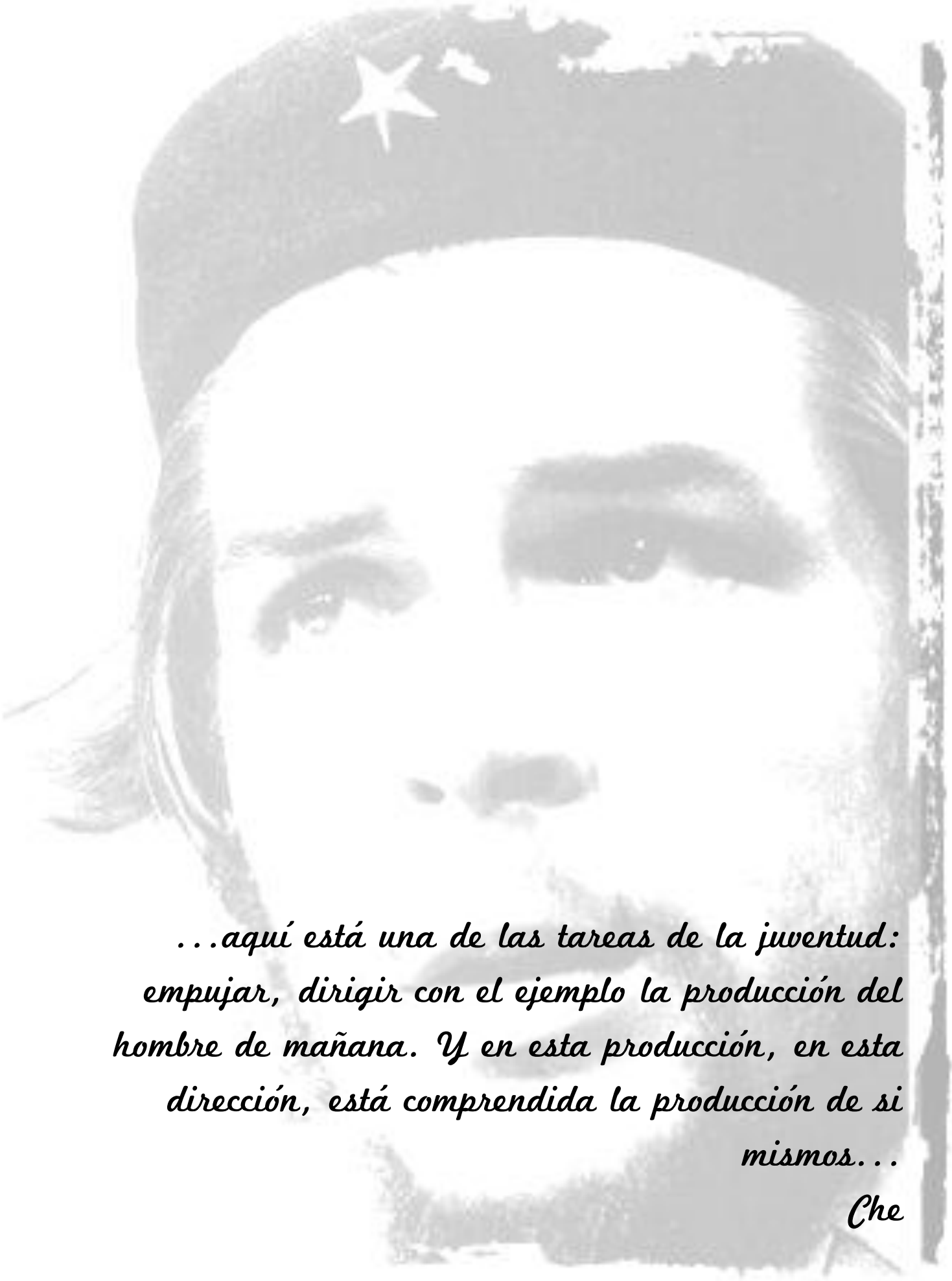
Tutor: Lic. José Ángel Lago Graverán

Co-tutor: Ing. Alexey Díaz Domínguez
Lic. Yusnier Valle Martínez

Consultor: Dr. Carlos Jesús Morón Álvarez

Ciudad de la Habana, 23 de junio del 2008

"Año 50 de la Revolución"



...aquí está una de las tareas de la juventud: empujar, dirigir con el ejemplo la producción del hombre de mañana. Y en esta producción, en esta dirección, está comprendida la producción de si mismos...

Che

A la Revolución, a Fidel y a la UCI por darnos la oportunidad de estudiar y ser mejores personas cada día.

A nuestros padres por haber dedicado parte de su vida a nuestra formación, por confiar plenamente en nosotros y no haber escatimado ni un momento para ver realizado este sueño que es de todos.

A nuestro tutor por pensar en nosotros para este tema de tesis y por apoyarnos en el transcurso de toda esta trayectoria.

Agradecimientos

Carlos

A mis padres por ser mi ejemplo y guía en todo momento, por todo el apoyo que me han dado toda la vida, por tanto amor, sacrificio, consejos, preocupación constante y dedicación.

A mis tíos y primos, por tener seguridad de que lograría este éxito, en especial a la memoria de mi tío Dagoberto, se que estaría muy orgulloso de mí.

A toda mi familia que de una forma u otra me han ayudado a llegar a donde estoy y confiaron siempre en mí.

A mi Soly por ser la personita mas linda de este mundo, con la que he compartido todos estos años y que siempre creyó en mi, parte de este logro es gracias a ella, gracias por todo tu amor.

A mis amigos del Pre – Universitario por todas las cosas lindas que pasamos juntos, en especial a Biguin, Luis, Arismel, Linet, Beatriz, Jatiana, Violeta e Indira.

A mis viejos y nuevos amigos de la UCI, por acompañarme en todo este difícil recorrido y haberme permitido estar junto a ellos.

A mi tutor por todo el apoyo y dedicación a este trabajo, ha sido un honor trabajar con él.

A todos mis profes por haber aportado cada uno su granito de arena a este gran logro, en especial a Onelia.

A mis vecinos por todas sus muestras de afecto y preocupación.

A todas las personas que no menciono y que de una forma u otra me ayudaron a llegar adonde estoy.

Solangel:

A mis padres por apoyarme en todo, por guiarme por la senda correcta, por quererme como nadie. Por ser las personas más grandes de este mundo, por no dejarme caer nunca y por estar siempre ahí para mí.

A mi hermana que me ha enseñado la realidad y la verdad de la vida y que siempre me ha apoyado.

A mis sobrinos que son los niños más lindos del mundo.

A mi Abuela y Abuelo que no se encuentran hoy entre nosotros pero sé que están orgullosos de mí donde quiera que estén.

A mi Abuela Guille la más linda de las abuelas y a mi abuelo Godofredo por darme los padres que tengo.

A mi tía Marlen y mi tío Juan Ramón que siempre me han apoyado y me han ayudado en todo lo que ha estado a su alcance y lo que no.

A mis primos que son mis hermanos varones, los que espero poder ver graduarse algún día como yo hoy.

A mi familia en general, que forman parte de lo que soy porque gracias a ellos también sobrevivo.

A mi novio, quien es la persona con la que he compartido estos 5 años de la universidad y que siempre me ha seguido en mis locuras y en mis acciones sin importar las consecuencias.

A mi amiga Nayay que me ha apoyado siempre y que ha estado ahí cuando la he necesitado, aun cuando me vuelve loca de vez en cuando.

A mis amigos Natacha, David, Laito, Onel, Frank que siempre han sido lo máximo.

A Silvano que ha sido un amigo entrañable con el que he podido contar y conversar abiertamente de lo que sea y que me ayudó en innumerables ocasiones.

A todos aquellos que no menciono.

Declaración de Autoría

Declaramos que somos los únicos autores de este trabajo y autorizamos a la Facultad 9 de la Universidad de las Ciencias Informáticas a hacer uso del mismo en su beneficio.

Para que así conste firmamos la presente a los 23 días del mes de junio del año 2008.

Solangel Rodríguez Vázquez **Carlos Luis Serrano Rosales**

_____ _____

José Ángel Lago Graverán

_____

Resumen

El siguiente trabajo describe el desarrollo de una Biblioteca de Métodos Numéricos, la cual contiene la implementación de métodos numéricos que dan solución a Sistemas de Ecuaciones Lineales, a casos especiales de Sistemas de Gran Dimensión y Poco Densos e Integración Numérica.

El uso de métodos numéricos es una tendencia actual y una necesidad en el desarrollo de muchas aplicaciones informáticas. Su estudio e implementación puede ser una tarea ardua, al igual que encontrar una herramienta libre que permita el trabajo con estos métodos. Este trabajo propone, pues, una solución a dichos problemas.

Los usuarios finales tendrán en su poder entonces una herramienta que los ayudará en el desarrollo de soluciones que requieran el uso de los métodos numéricos antes mencionados. La solución está desarrollada en Software Libre lo cual indica que su uso no esta restringido a ningún ámbito por lo que puede ser usada y modificada según las necesidades individuales.

Se desarrolló además junto con la biblioteca una documentación en la cual se abordan detalles de cada uno de los métodos numéricos implementados. Apoyará fuertemente al usuario final y servirá como base para la utilización de la Biblioteca.

Palabras Claves: biblioteca, sistemas de ecuaciones lineales, sistemas de gran dimensión y poco densos, integración numérica, software libre.

Índice

Introducción **1**
CAPÍTULO 1: Fundamentación Teórica **5**
1.1. Introducción 5
1.2. Conceptos asociados al dominio del problema 5
1.2.1. Matemática Numérica 5
1.2.2. Métodos Numéricos 5
1.2.2.1. Métodos Numéricos Directos 6
1.2.2.2. Métodos Numéricos Iterativos 6
1.2.3. Biblioteca 6
1.2.4. Biblioteca de Métodos Numéricos 6
1.2.5. Matriz 6
1.2.6. Sistemas de Ecuaciones Lineales 7
1.2.7. Sistemas de gran dimensión y poco densos 7
1.2.8. Integración Numérica 8
1.2.9. Función 8
1.3. Objeto de Estudio 8
1.3.1. Descripción General 8
1.3.2. MN para Sistemas de Ecuaciones Lineales 10
1.3.2.1. Eliminación Gaussiana 10
1.3.2.2. Gauss Jordan 12
1.3.2.3. Método de Jacobi 13
1.3.2.4. Gauss – Seidel 14
1.3.3. MN para Sistemas de Gran Dimensión y Poco Densos 15
1.3.3.1. Método de Thomas para Matrices Tridiagonales 15
1.3.3.2. Gauss para Matrices Tridiagonales 16
1.3.4. MN para Sistemas de Integración Numérica 18
1.3.4.1. Regla Rectangular 18
1.3.4.2. Regla Trapezoidal 20
1.3.4.3. Método de Simpson 23
1.3.4.4. Método de Romberg 27
1.4. Estado del Arte 29
1.4.1. Herramientas Matemáticas 29
1.4.1.1. Derive 29
1.4.1.2. Matlab 30

1.4.1.3. Mathematica Software 30
1.4.2. Bibliotecas 31
1.4.2.1. Blas 31
1.4.2.2. CBlas 32
1.4.2.3. Lapack 32
1.4.2.4. Atlas 32
1.4.2.5. ScaLapack 33
1.4.2.6. SLEPc 33
1.4.2.7. PETSc 34
1.4.2.8. PSPASES 34
1.5. Conclusiones parciales 35
CAPÍTULO 2: Tendencias y Tecnologías actuales a desarrollar. 36
2.1. Introducción 36
2.2. Metodología de Desarrollo de Software 36
2.2.1. Rational Unified Process (RUP) 36
2.2.2. Extreme Programming (XP) 38
2.2.3. Microsoft Solution Framework (MSF) 39
2.2.4. ¿Por qué RUP? 40
2.3. Lenguaje de Modelado: UML 41
2.4. Herramientas CASE 42
2.4.1. Visual Paradigm 42
2.4.2. Rational Rose 43
2.4.3. ¿Por qué Visual Paradigm? 44
2.5. Lenguaje de Programación 44
2.5.1. C++ 44
2.5.2. Java 45
2.5.3. Python 46
2.5.4. ¿Por qué C++? 47
2.6. IDE de Desarrollo 47
2.6.1. KDevelop 47
2.6.2. Eclipse 48
2.6.3. Code::Blocks 49
2.6.4. ¿Por qué Code::Blocks? 49
2.7. Conclusiones Parciales 50
CAPÍTULO 3: Presentación de la solución propuesta 51
3.1. Introducción 51
3.2. Modelo de Dominio 51

3.2.1. ¿Cuándo se aplica un modelo de dominio? 51
3.2.2. Diagrama de clases del Modelo de Dominio 52
3.2.3. Conceptos principales del Modelo de Dominio 52
3.3. Requerimientos 53
3.3.1. Requerimientos Funcionales 54
3.3.2. Requerimientos No Funcionales 54
3.4. Descripción del Sistema. Modelos de Casos de Uso del Sistema 55
3.4.1. Determinación y justificación de los actores del sistema 55
3.4.2. Casos de Uso del Sistema 55
3.4.3. Diagrama de Casos de Uso del Sistema 55
3.4.4. Expansión de los Casos de Uso 56
3.5. Conclusiones Parciales 59
CAPÍTULO 4: Construcción de la solución propuesta. 60
4.1. Introducción 60
4.2. Patrones 60
4.2.1. Patrones de Diseño 60
4.2.2. Patrones de Arquitectura 61
4.2.2.1. Arquitectura en Capas 61
4.2.2.2. Arquitectura Orientada a Objetos 61
4.2.2.3. Arquitectura Final 62
4.3. Diagramas de clase del diseño 62
4.3.1. Diagrama de Clases Subsistema de Clases Generales 63
4.3.2. Diagrama de Clases Subsistema de SEL 64
4.3.3. Diagrama de Clases Subsistema de SGDPD 65
4.3.4. Diagrama de Clases Subsistema de IN 66
4.4. Diagramas de Interacción 66
4.4.1 Diagramas de Secuencia 67
4.4.2 Diagramas de Colaboración 67
4.5. Modelo de Implementación 68
4.5.1. Diagramas de Componentes 68
4.6. Prueba del Sistema Propuesto 69
4.7. Conclusiones Parciales 72
CAPÍTULO 5: Estudio de Factibilidad. 73
5.1. Introducción 73
5.2. Planificación 73
5.2.1. Objetivos de la Planificación 73
5.3. Estimación de los Costos 74

5.4. Beneficios 79
5.4.1. Beneficios tangibles 79
5.4.2. Beneficios Intangibles 79
5.5. Análisis de costos y beneficios 80
5.6. Conclusiones Parciales 80
Conclusiones Generales 81
Recomendaciones 82
Referencias Bibliográficas 83
Bibliografía Consultada 85
Anexos 86
Glosario 98

Índice de Tablas

Tabla 1: Requisitos Funcionales 54
Tabla 2: Actor del Sistema y su funcionalidad. 55
Tabla 3: Prioridad de los Casos de Uso 55
Tabla 4: Descripción Textual Caso de Uso Calcular Solución SEL. 56
Tabla 5: Descripción Textual Caso de Uso Calcular Solución SGDPD 57
Tabla 6: Descripción Textual Casos de Uso Calcular Integral Numérica 58
Tabla 7: Caso de Prueba Hallar solución de un SEL 69
Tabla 8: Caso de prueba Hallar solución de un SGDPD 71
Tabla 9: Caso de Prueba Calcular la IN de una función. 71
Tabla 10: Factor de peso de los actores sin ajustar 75
Tabla 11: Factor de peso de los casos de uso sin ajustar 75
Tabla 12: Factor de Complejidad Técnica 76
Tabla 13: Factor de Ambiente 77
Tabla 14: Distribución del esfuerzo por actividades 78
Tabla 15: Descripción de CC_Matriz 86
Tabla 16: Descripción de CC_SEL_Directo 87
Tabla 17: Descripción de CC_E_Gaussiana_PivT 87
Tabla 18: Descripción de CC_E_Gaussiana_PivP 88
Tabla 19: Descripción de CC_E_GaussJordan 88
Tabla 20: Descripción de CC_SEL_Iterativo 88
Tabla 21: Descripción de CC_Jacobi 88
Tabla 22: Descripción de CC_Seidel 89
Tabla 23: Descripción de CC_SGDPD 89
Tabla 24: Descripción de CC_Thomas 89
Tabla 25: Descripción de CC_GaussT 89
Tabla 26: Descripción de CC_Funcion 89
Tabla 27: Descripción de CC_Integral 90
Tabla 28: Descripción de CC_Rectangular 90
Tabla 29: Descripción de CC_Trapecios 90
Tabla 30: Descripción de CC_Simpson 90
Tabla 31: Descripción de CC_Romberg 91

Índice de Figuras

Figura 1: Modelo Rectangular 19
Figura 2: Modelo Rectangular con intervalos 20
Figura 3: Método Trapecio. 21
Figura 4: Método Trapecios. Representación del Área. 21
Figura 5: Sub-áreas de Simpson 23
Figura 6: Notaciones de Simpson 24
Figura 7: Tabla resultado iteraciones de Romberg 29
Figura 8: Diagrama de Clases del Dominio 52
Figura 9: Diagrama de Casos de Uso del Sistema 56
Figura 10: Diagrama de Subsistemas del Diseño 63
Figura 11: Diagrama de Clases Subsistema de Clases Generales 63
Figura 12: Diagrama de Clases Subsistema de SEL 64
Figura 13: Diagrama de Clases Subsistema de SGDPD 65
Figura 14: Diagrama de Clases Subsistema de IN 66
Figura 15: Diagrama de Componentes de Código Fuente 68
Figura 16: Diagrama de Componentes de Código Ejecutable 69
Figura 17: Diagrama de Secuencia SEL Escenario Métodos Directos 92
Figura 18: Diagrama de Secuencia SEL Escenario Métodos Iterativos 92
Figura 19: Diagrama de Secuencia SGDPD 93
Figura 20: Diagrama de Secuencia IN Escenario Rectangular, Trapecios y Simpson 93
Figura 21: Diagrama de Secuencia IN Escenario Romberg 94
Figura 22: Diagrama de Colaboración SEL Escenario Métodos Directos 95
Figura 23: Diagrama de Colaboración SEL Escenario Métodos Iterativos 95
Figura 24: Diagrama de Colaboración SGDPD 96
Figura 25: Diagrama de Colaboración IN Escenario Rectangular, Trapecios y Simpson 96
Figura 26: Diagrama de Colaboración IN Escenario Romberg 97

Introducción

Existe en la actualidad una tecnología que nos desafía diariamente con sus avances y nuevos enfoques: La Tecnología de la Información y las Comunicaciones (TIC). Es innegable la gran importancia que esta posee hoy día en el desenvolvimiento de la sociedad. La revolución acaecida con el surgimiento de las computadoras y el uso de estas en diferentes esferas, se ha visto acrecentada a partir de la aparición del fenómeno llamado Internet. A medida que pasa el tiempo se van descubriendo nuevas aplicaciones e implicaciones de la enorme disponibilidad y acceso a la información que existe desde los lugares más remotos de la geografía mundial.

Desde hace más de medio siglo nuestro país ha sido sometido a un brutal bloqueo a manos del gobierno de los Estados Unidos de América. Estas acciones dirigidas contra el pueblo han propiciado que muchas veces no se obtengan los beneficios que acarrean las tecnologías de la información.

El gobierno cubano en un esfuerzo por elevar el nivel de acceso a la información y el conocimiento, ha llevado a cabo la informatización de nuestro territorio nacional y facilitado que hasta en los lugares más recónditos de nuestra nación exista el acceso a las computadoras. La Universidad de las Ciencias Informáticas, surgida al albor de la Batalla de Ideas, constituye un peldaño importante en el esfuerzo que lleva a cabo el Comandante en Jefe junto a los demás dirigentes de la Revolución con el objetivo de hacer de las tecnologías una herramienta de acceso popular.

En nuestra universidad se han llevado a cabo varios proyectos de informatización tanto para nuestra institución y el país como para el extranjero. En el trabajo diario en estos proyectos surgen problemáticas y necesidades que impiden el buen desarrollo y funcionamiento del personal que trabaja en los mismos. La poca disponibilidad de soluciones informáticas en software libre que permitan el trabajo con métodos numéricos es una de las tantas problemáticas que pueden surgir.

Los métodos numéricos son utilizados para resolver computacionalmente problemas matemáticos. Aunque hay muchos tipos de métodos numéricos, todos comparten una característica común: llevan a cabo un gran número de tediosos cálculos aritméticos. Es por ello que la computadora es una herramienta que facilita el uso y el perfeccionamiento de la implementación de estos métodos.

En la actualidad, gracias a la gran evolución que han tenido los métodos numéricos y su implementación en potentes computadoras, es posible, por ejemplo, modelar el choque de un vehículo o hacer el análisis aerodinámico estructural de un avión, resolviendo en cada caso sistemas algebraicos de ecuaciones con varios cientos de miles (a veces de millones) de incógnitas. Además se puede apreciar su uso en varias aplicaciones como software de música, software de diseño, simuladores, etc.

Existen situaciones en las cuales el programador tiene pocos conocimientos de matemática y se tiene que enfrentar a la ardua tarea de implementar métodos numéricos que le son necesarios para resolver el problema al cual se enfrenta, en este momento debe investigar todo lo relacionado a los métodos que le son más eficientes para realizar su trabajo, conocer todo respecto a su funcionamiento para después hacer una implementación correcta del mismo.

En muchos casos no existe una completa documentación relacionada con los métodos a implementar que satisfaga las necesidades del programador.

Todo esto conlleva a la **situación problemática**, la cual se ve reflejada en los siguientes aspectos.

- ✓ Aplicaciones cuyo funcionamiento requieren del trabajo con métodos numéricos.
- ✓ Poca disponibilidad en software libre de herramientas informáticas que den la posibilidad del trabajo con métodos numéricos.
- ✓ Poca documentación que aborde el tema de los métodos numéricos para un programador.
- ✓ Falta de experiencia por parte del personal informático para seleccionar el método numérico adecuado para dar solución a los problemas que se enfrenta.
- ✓ Pérdida de tiempo a la hora del estudio de métodos numéricos y de su implementación.

El **problema científico** viene dado entonces por la poca disponibilidad que existe en software libre de bibliotecas de métodos numéricos para el trabajo en aplicaciones que utilizan estos métodos para su funcionamiento y de una documentación que sirve de ayuda para el uso de estos métodos.

El **objeto de estudio** es la Matemática Numérica.

Nuestro **campo de acción** son los métodos numéricos referentes a Sistemas de Ecuaciones Lineales (SEL), Sistemas de Gran Dimensión y Poco Densos (SGDPD) e Integración Numérica (IN).

La **idea a defender** parte de la premisa siguiente:

Con el desarrollo de la biblioteca de métodos numéricos aumentará la disponibilidad de bibliotecas de clases en software libre y de documentación asociada al uso de los métodos implementados en estas bibliotecas.

Todo esto conlleva al **objetivo general** de este trabajo, desarrollar en software libre una biblioteca de métodos numéricos para ser utilizada en el desarrollo de aplicaciones cuyo funcionamiento requiera del uso de estos métodos.

Para la culminación satisfactoria del trabajo y dar cumplimiento al objetivo antes expuesto se han trazado varias **tareas**:

1. Estudiar el estado del arte.

2. Escoger herramientas para el desarrollo de la biblioteca.
3. Realizar el diseño de la biblioteca.
4. Realizar la implementación de la biblioteca.
5. Realizar pruebas a la biblioteca.
6. Elaborar documentación de la biblioteca.

Para una correcta comprensión y realización del trabajo se hace uso de los siguientes métodos científicos:

1. Método Teórico
   1.1. Lógico.
      1.1.1. Modelación: para crear un modelo (abstracción) que explique el resultado final de nuestro trabajo.
   1.2. Históricos: para el estudio de antecedentes que existen respecto a bibliotecas de este tipo y programas o software que trabajen con los diferentes métodos numéricos a tratar.
2. Métodos Empíricos
   2.1. Entrevistas: al tutor, cotutor, consultor u otras personas especializados en el tema de la matemática numérica para profundizar en el tema de los métodos numéricos, sus algoritmos y funcionamiento.
   2.2. Otros: revisión de bibliografía para apoyar el estudio de los diferentes métodos numéricos.

El Trabajo de Diploma ha sido conformado de la siguiente manera:

Capítulo 1_Fundamentación Teórica: Este capítulo contiene los conceptos asociados al dominio del problema, se explican cada unos de los métodos numéricos que serán tratados en la biblioteca, además de un análisis del estado del arte referente a las bibliotecas y aplicaciones que permiten el trabajo con métodos numéricos.

Capítulo 2_Tendencias y tecnologías actuales a desarrollar: En este capítulo se describe lo correspondiente a las tecnologías actuales que se utilizan en el desarrollo de bibliotecas haciendo énfasis en las que se apoya la propuesta a desarrollar, dejando argumentado la decisión de usar RUP como metodología de desarrollo, UML para la modelación de la solución, Visual Paradigm como herramienta CASE, C++ como lenguaje de programación y Code::Blocks como entorno de desarrollo.

Capítulo 3_Presentación de la solución propuesta: Se propone una solución al problema planteado a través del modelo de dominio y la identificación y descripción de los casos de uso del sistema. También se describen los conceptos fundamentales que se manejan en el sistema y se proponen todos los requisitos funcionales (RF) y no funcionales (RNF) con los cuales debe cumplir el sistema.

Capítulo 4_Construcción de la solución propuesta: Se construye de forma completa la solución propuesta. Se efectúa el diseño completo de la aplicación, tratando aspectos fundamentales como la arquitectura, los diagramas de interacción y los diagramas de componentes.

Capítulo 5_Estudio de Factibilidad: Se especifican los costos y beneficios tangibles e intangibles, la planificación, la gestión de proyectos, el tipo de estimación realizada en el trabajo por punto de casos de uso, etc.

CAPÍTULO 1: Fundamentación Teórica

1.1. Introducción

En el presente capítulo se abordan los conceptos asociados al dominio del problema que resultan de relevancia para el tema abordado en este trabajo y que permitirán obtener un mejor entendimiento de lo que se quiere tratar, además de abordar cada uno de los métodos numéricos a desarrollar, se muestran algunas herramientas y bibliotecas que de una forma u otra permiten el trabajo con métodos numéricos.

1.2. Conceptos asociados al dominio del problema

1.2.1. Matemática Numérica

La Matemática Numérica es la rama de las Matemáticas que propone, desarrolla, analiza y aplica algoritmos y métodos numéricos para obtener soluciones aproximadas de problemas matemáticos. **(1)** Trefethen **(2)** diserta sobre varias definiciones del Análisis Numérico. En su opinión la mejor es la que dice que: El Análisis Numérico es el estudio de algoritmos para los problemas de las matemáticas continuas, después los algoritmos son implementados en computadoras. Se trata por tanto de una disciplina matemática totalmente aplicada. Sanz-Serna **(3)** dice: “El Cálculo Numérico además de una disciplina académica que se estudia en las aulas, es parte de un esfuerzo humano global desarrollado a lo largo de la historia para resolver problemas científicos y técnicos”.

La Matemática Numérica puede definirse entonces como la teoría y la práctica del cálculo eficiente y la estimación del error de la solución aproximada. En la Matemática Numérica no basta con solucionar el problema, sino que interesa también el tiempo que se necesita para obtener la solución y la estimación del error de la aproximación, de ahí que uno de sus objetivos es la elección del método más adecuado para la solución del problema.

1.2.2. Métodos Numéricos

Los métodos numéricos son técnicas mediante las cuales es posible formular problemas matemáticos de tal forma que puedan resolverse usando operaciones aritméticas. Hay muchos tipos de métodos numéricos, y comparten una característica común: invariablemente se deben realizar un buen número de tediosos cálculos aritméticos. **(4)**

Son un medio para reforzar la comprensión de las matemáticas, porque profundizan en los temas que de otro modo resultarían oscuros, esto aumenta su capacidad de comprensión y entendimiento en la materia.

1.2.2.1. Métodos Numéricos Directos

Los métodos numéricos directos son los que, si se puede evitar todos los errores por redondeo, se obtiene la respuesta en un número finito de pasos.

1.2.2.2. Métodos Numéricos Iterativos

Los métodos numéricos iterativos son todos aquellos que producen una sucesión infinita de respuestas aproximadas, sucesión que, bajo ciertas condiciones, converge hacia la solución exacta del problema.

1.2.3. Biblioteca

También conocida en el ámbito de la UCI como librería, no son más que trozos de código que contienen alguna funcionalidad pre – construida que puede ser utilizada por un ejecutable. Las bibliotecas contienen en su interior variables y funciones, las cuales en ocasiones están encapsuladas en forma de clases, permiten tratar las colecciones de estas funciones como una sola unidad, y representan una forma muy conveniente para el manejo y desarrollo de aplicaciones grandes.

1.2.4. Biblioteca de Métodos Numéricos

Con el concepto brindado anteriormente de biblioteca se puede decir entonces que Biblioteca de Métodos Numéricos (BMN) sería una biblioteca cuyas funcionalidades serían los métodos numéricos y su algoritmo de resolución.

1.2.5. Matriz

En matemáticas, una matriz es un cuadrado o tabla de números ordenados en forma de filas y columnas. Se llama matriz de dimensión mxn a un conjunto de números reales dispuestos en m filas y n columnas de la siguiente forma.

$$\begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{pmatrix}$$

Una matriz con una sola columna o una sola fila se denomina vector. Una matriz $1 \times n$ (una fila y n columnas) se denomina vector fila, y una matriz $m \times 1$ (una columna y m filas) se denomina vector columna.

1.2.6. Sistemas de Ecuaciones Lineales

Cuando se plantea la resolución de varias ecuaciones a la vez con varias incógnitas, se evidencia la existencia de un sistema de ecuaciones, y en el caso más sencillo, donde todas las ecuaciones sean lineales, se llama sistema de ecuaciones lineales.

Estos sistemas se pueden escribir de forma tradicional así:

$$\left.\begin{array}{c} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + ex_3 + \cdots + a_{2n}x_n = b_2 \\ \cdots \quad \cdots \quad \cdots \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 + \cdots + a_{3n}x_n = b_3 \end{array}\right\}$$

Un sistema así expresado tiene "m" ecuaciones y "n" incógnitas, donde a_{ij} son números reales, llamados coeficientes del sistema, los valores b_m son números reales, llamados términos independientes del sistema, las incógnitas x_j son las variables del sistema, y la solución del sistema es un conjunto ordenado de números reales $(s_1, s_2, \ldots, s_n)$ tales que al sustituir las incógnitas $x_1, x_2, \ldots, x_n$ por los valores $s_1, s_2, \ldots, s_n$ se verifican *a la vez* las "m" ecuaciones del sistema. (5)

Este mismo *sistema* de ecuaciones lineales en notación matricial tiene esta forma:

$$\mathrm{Ax} = \mathrm{b}$$

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \cdots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} \end{pmatrix} \cdot \begin{pmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{pmatrix}$$

Matriz de coeficiente — Matriz de incógnitas — Matriz de términos independientes

Se llama matriz ampliada de dimensión $m \times (n+1)$ a la matriz que se obtiene al añadir a la matriz del sistema la columna de los términos independientes, y se denota A^*, es decir:

$$\left(\begin{array}{ccccc|c} a_{11} & a_{12} & a_{13} & \cdots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \cdots & a_{2n} & b_2 \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ a_{m1} & a_{m2} & a_{m3} & \cdots & a_{mn} & b_m \end{array}\right)$$

1.2.7. Sistemas de gran dimensión y poco densos

Los sistemas de ecuaciones se consideran de gran dimensión cuando se trabaja con una cantidad considerable de incógnitas y los costos de computación son elevados, y se dice entonces que son poco densos cuando la mayoría de los elementos son ceros.

1.2.8. Integración Numérica

La integración numérica constituye una amplia gama de algoritmos para calcular el valor numérico de una integral definida.

El problema básico considerado por la integración numérica es calcular una solución aproximada a la integral definida:

$$\int_a^b f(x)dx$$

Los métodos de integración numérica pueden ser descritos generalmente como combinación de evaluaciones del integrando para obtener una aproximación a la integral. Una parte importante del análisis de cualquier método de integración numérica y que se cumple para cualquier método de la matemática numérica es estudiar el comportamiento del error de aproximación como una función del número de evaluaciones del integrando. Un método que produce un pequeño error para un pequeño número de evaluaciones es normalmente considerado superior.

1.2.9. Función

Dados dos conjuntos X e Y , una función de X en Y es una correspondencia matemática denotada como:

$$f: X \to Y$$

Que cumple con las siguientes condiciones:

- ✓ Condición de existencia: todos los elementos de X están relacionados con elementos de y, es decir, $\forall x \in X, \exists y \in Y \setminus (x, y) \in f$.
- ✓ Condición de unicidad: cada elemento de X esta relacionado con un único elemento de Y, es decir, si $(x, y_1) \in f \wedge (x, y_2) \in f \Longrightarrow y_1 = y_2$.

Una función es un caso particular de relación y de correspondencia matemática. Cada relación o correspondencia de un elemento $x \in X$ con un (y sólo uno) $y \in Y$ se denota $f(x) = y$, en lugar de $(x, y) \in f$.

1.3. Objeto de Estudio

1.3.1. Descripción General

En la realidad, muchas veces se pueden obtener respuestas a problemas las cuales vienen expresadas en forma de sistema de ecuaciones lineales de la forma $Ax = b$. Cuando la matriz es pequeña no hay mucho problema, porque se puede proceder de forma manual para solucionar dichos sistemas. Pero se presentarán problemas para dar solución a sistemas los cuales se expresan por

matrices en orden $m\ x\ n$ siendo m y n muy grandes. La forma manual es tediosa, engorrosa, propensa a errores de cálculo, y en una gran parte de los casos impracticable. Es en estos casos en que se requiere la ayuda de la computadora para que pueda (por medio de ciertas técnicas) dar solución a estos sistemas de ecuaciones. Aquí aparecen entonces los distintos métodos numéricos directos e iterativos desarrollados para resolver sistemas de ecuaciones lineales.

Otro problema que surge en la ingeniería se presenta con frecuencia con la necesidad de integrar una función que sería, en general, de una de las tres formas siguientes:

- ✓ Una función simple y continua tal como un polinomio, una función exponencial o una función trigonométrica.
- ✓ Una función complicada y continua que es difícil o imposible de integrar directamente.
- ✓ Una función tabulada en donde los valores de x y $f(x)$ se dan en un conjunto de puntos discretos, como es el caso a menudo, de datos experimentales.

En el primer caso, la integral simplemente es una función que se puede evaluar fácilmente usando métodos analíticos aprendidos en el cálculo. En los dos últimos casos, sin embargo, se deben emplear métodos aproximados.

Las fórmulas de integración de Newton-Cotes son los esquemas más comunes dentro de la integración numérica. Se basan en la estrategia de reemplazar una función complicada o un conjunto de datos tabulares con alguna función aproximada que sea más fácil de integrar.

La integral se puede aproximar usando una serie de polinomios aplicados por partes a la función o a los datos sobre intervalos de longitud constante.

Se dispone de las formas abierta y cerrada de las fórmulas de Newton-Cotes. Las formas cerradas son aquellas en donde los puntos al principio y al final de los límites de integración se conocen. Las fórmulas abiertas tienen los límites de integración extendidos más allá del rango de los datos.

Se puede llegar a la conclusión de que existen varias razones para llevar a cabo la integración numérica. La principal puede ser la imposibilidad de realizar la integración de forma analítica. Es decir, integrales que requerirían un gran conocimiento y manejo de matemática avanzada pueden ser resueltas de una manera más sencilla mediante métodos numéricos. La solución analítica de una integral nos arrojaría una solución exacta mientras que la solución numérica nos daría una solución aproximada. El error de la aproximación, que depende del método que se utilice y de qué tan fino sea, puede llegar a ser tan pequeño que es posible obtener la misma precisión de la solución analítica en las primeras cifras decimales.

1.3.2. MN para Sistemas de Ecuaciones Lineales

1.3.2.1. Eliminación Gaussiana

La notación matricial da en la matemática un modo compacto de expresar los sistemas de ecuaciones lineales.

La resolución de cualquier sistema de ecuaciones es computacionalmente muy costosa y se adoptan métodos indirectos que simplifican su tratamiento al representar la matriz original en formatos que son abordables de forma mucho más simple para su inversión (por ejemplo, matrices triangulares). Para hacer la transformación suelen utilizarse diferentes métodos de factorización uno de los cuales es la Eliminación Gaussiana.

Para comenzar, es necesario decir que sólo nos interesan los sistemas cuadrados, o sea, igual número de ecuaciones e incógnitas. Se supone además que son sistemas determinados o lo que es lo mismo de solución única.

Sea el sistema a resolver

$$\begin{array}{ccccccc} a_{11}x_1 & + & a_{12}x_2 & +\cdots+ & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & +\cdots+ & a_{2n}x_n & = & b_2 \\ & & & & & \vdots & \\ a_{n1}x_1 & + & a_{n2}x_2 & +\cdots+ & a_{nn}x_n & = & b_n \end{array}$$

Cada uno de sus coeficientes y términos independientes son números reales, este sistema se puede representar de forma matricial

$$\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & & \vdots \\ a_{n1} & a_{n2} & \cdots & a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{bmatrix}$$

Esta matriz representa el sistema en su forma reducida a la ecuación $Ax = b$ donde A seria la matriz de los coeficientes, x el vector incógnita y b como el vector de términos independientes.

Como el sistema es determinado A es no singular o lo que es lo mismo decir, su determinante es igual a cero.

El algoritmo de Eliminación Gaussiana se divide en dos etapas, el proceso directo y el proceso inverso.

Proceso Directo: (6)

Este proceso consiste en aplicar en la matriz transformaciones de dos tipos, intercambiar dos ecuaciones del sistema y sumar a una ecuación, miembro a miembro, otra ecuación pivote del sistema multiplicada por un número real cualquiera.

La primera transformación no cambia la solución del sistema, aunque si afecta el determinante de

A, ya que equivale a la permutación de dos de sus filas, lo cual cambia el signo del determinante. La segunda transformación no altera ni la solución del sistema ni el valor del determinante. Resumiendo, se puede decir que estas transformaciones tienen como objetivo transformar el sistema a la forma triangular:

$$\begin{array}{rcrcrcl} c_{11}x_1 & + & c_{12}x_2 & +\cdots+ & c_{1n}x_n & = & d_1 \\ & & c_{22}x_2 & +\cdots+ & c_{2n}x_n & = & d_2 \\ & & & & \vdots & & \\ & & & & c_{nn}x_n & = & d_n \end{array}$$

Como el algoritmo se trabaja de forma computacional el sistema se escribe en forma de una matriz de n por $n+1$ que agrupa los elementos de A y los de b y que se conoce como matriz ampliada del sistema.

$$A|b = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & b_n \end{bmatrix}$$

A la que una vez aplicado el proceso directo toma forma escalonada.

$$\begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1n} & d_1 \\ 0 & c_{22} & \cdots & c_{2n} & d_c \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & c_{nn} & d_n \end{bmatrix}$$

Para que la matriz tome forma escalonada, primero se efectúan $n-1$ transformaciones elementales del segundo tipo, utilizando la fila 1 como pivote y afectando a las demás filas de modo que se hagan ceros todos los elementos de la primera columna que están debajo de la diagonal, después $n-2$ transformaciones del segundo tipo con la fila 2 como pivote, para modificar a las filas debajo de ella de manera que se hagan ceros todos los elementos de la segunda columna debajo de la diagonal; este proceso continúa hasta llegar a la fila $n-1$ como pivote para eliminar el coeficiente de la columna $n-1$ debajo de la diagonal.

En el paso i del proceso directo, en el cual la fila enésima actúa como pivote, la fila $k(k>i)$ se cambia de acuerdo con la fórmula:

$$fila\ k := fila\ k - \frac{a'_{ki}}{a'_{ii}}(fila\ i)\ \ k = i+1, i+2, \dots, n$$

Donde los coeficientes con apóstrofe indican que no son necesariamente los originales.

Proceso Inverso. (6)

Una vez que se obtiene el sistema en forma triangular se puede calcular x_n de la última ecuación, después x_{n-1} a partir de la penúltima ecuación y así sucesivamente hasta encontrar x_1 de la primera ecuación. La forma para obtener la solución de abajo para arriba le da nombre a este proceso.

$$x_n = \frac{1}{c_{nn}} d_n$$

$$x_{n-1} = \frac{1}{c_{n-1,n-1}} \left(d_{n-1} - c_{n-1,n} x_n\right)$$

$$\vdots$$

$$x_1 = \frac{1}{c_{11}} (d_1 - c_{12}x_2 - c_{13}x_3 - \cdots - c_{1n}x_n)$$

Estrategias de pivote

La estrategia de pivote es el criterio que se utiliza para seleccionar la fila que se ha de utilizar como pivote en cada paso del proceso directo. Existen tres estrategias de pivote, la elemental, la parcial y la total, en el presente caso, se utilizaron la parcial y la total.

Estrategia parcial

Al realizar el paso i del proceso directo se analizan todas las filas desde la enésima hasta la última para seleccionar aquella que posea el elemento en posición i con el mayor valor absoluto y esta fila se intercambia con la enésima fila. Esta estrategia garantiza que el factor que se utiliza para multiplicar a la fila pivote sea siempre menor que 1 con lo cual los errores acumulados en la fila enésima, lejos de ampliarse disminuyen.

Estrategia total

Al realizar el paso i del proceso directo se busca el elemento de mayor valor absoluto de la sub-matriz que aún no se encuentra escalonada y se intercambian dos filas y dos columnas de manera que ese elemento pase a ocupar la posición de a'_{ii}. Esta estrategia da mejores resultados en cuanto a la propagación de errores.

1.3.2.2. Gauss Jordan

El método de Gauss – Jordan es similar a la Eliminación Gaussiana, con la diferencia de que primero hace el pivote igual a 1, y luego hace ceros en toda la columna del pivote. La solución al sistema de ecuaciones queda en la última columna de la matriz aumentada y mediante proceso inverso se obtiene la solución del sistema.

1.3.2.3. Método de Jacobi

Sea $Ax = b$ un sistema de n ecuaciones lineales. Cumpliéndose la condición de que este sistema es cuadrado y de solución única, su forma desarrollada es la siguiente:

$$\begin{array}{rcrcrcl} a_{11}x_1 & + & a_{12}x_2 & +\cdots+ & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_2 & +\cdots+ & a_{2n}x_n & = & b_2 \\ & & & & & \vdots & \\ a_{n1}x_1 & + & a_{n2}x_2 & +\cdots+ & a_{nn}x_n & = & b_n \end{array}$$

Si todos los elementos $a_{ii}(i = 1,2, \dots, n)$ de la diagonal son no nulos, el sistema se puede escribir así:

$$\begin{array}{rcccccccc} x_1 & = & \frac{b_1}{a_{11}} & - & 0x_1 & - & \frac{a_{12}}{a_{11}}x_2 & -\cdots- & \frac{a_{1n}}{a_{11}}x_n \\ x_2 & = & \frac{b_2}{a_{22}} & - & \frac{a_{21}}{a_{22}}x_1 & - & 0x_2 & -\cdots- & \frac{a_{2n}}{a_{22}}x_n \\ & \vdots & & & & & & & \\ x_n & = & \frac{b_n}{a_{nn}} & - & \frac{a_{n1}}{a_{nn}}x_1 & - & \frac{a_{n2}}{a_{nn}}x_2 & -\cdots- & 0x_n \end{array}$$

Donde, se aprecia, en la enésima ecuación $(i = 1, 2, \dots, n)$ se ha despejado x_i. Utilizando la forma matricial, el sistema puede escribirse:

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & \dots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & \dots & -\frac{a_{2n}}{a_{22}} \\ \vdots & \vdots & & \vdots \\ \frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \dots & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} + \begin{bmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \vdots \\ \frac{b_n}{a_{nn}} \end{bmatrix}$$

Llamando M y c a las matrices:

$$M = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & \dots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & \dots & -\frac{a_{2n}}{a_{22}} \\ \vdots & \vdots & & \vdots \\ \frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \dots & 0 \end{bmatrix} \quad c = \begin{bmatrix} \frac{b_1}{a_{11}} \\ \frac{b_2}{a_{22}} \\ \vdots \\ \frac{b_n}{a_{nn}} \end{bmatrix}$$

El sistema toma la forma: $x = Mx + c$

A partir de la ecuación anterior se puede generar un proceso iterativo matricial, partiendo de un vector inicial $x^{(0)}$ y generando una sucesión de vectores $x^{(1)}, x^{(2)}, x^{(3)}, \dots$ mediante la ecuación recursiva:

$$x^{(k)} = Mx^{(k-1)} + c \qquad k = 1,2,3, \dots$$

La sucesión generada puede o no converger, para que esto ocurra la matriz A tiene que ser de diagonal predominante, esto es, que para cada fila $i = 1,2,3, \dots, n$, el elemento de la diagonal sea, en valor absoluto, mayor que la suma de los valores absolutos de los otros elementos. **(6)**

1.3.2.4. Gauss – Seidel

El método de Gauss-Seidel es iterativo, es decir, a partir de una solución inicial para el vector de las variables incógnitas se obtiene una nueva aproximación en cada pasada del algoritmo, hasta que se disponga de una aproximación a la solución suficientemente válida. El problema de los métodos iterativos es asegurar la convergencia hacia la solución, cada método tiene su criterio de convergencia que debe ser verificado previamente.

El método de Seidel es una variación del método anteriormente descrito.

En el método de Jacobi, al calcular la variable x_i se utilizan los valores de las demás variables que se obtuvieron en la iteración anterior, sin embargo, en ese momento ya se han calculado los nuevos valores de $x_1, x_2, \dots, x_{i-1}$ que son, por lo general, mejores aproximaciones que los obtenidos en la iteración anterior. En el método de Seidel, una vez que se obtiene el nuevo valor de una variable, éste se utiliza para actualizar los valores de las variables que siguen; de esta forma, no se necesita guardar los valores de la iteración anterior, lo cual simplifica el algoritmo y ahorra memoria y, por otra parte, la velocidad de la convergencia mejora sustancialmente.

En términos más precisos, sea $Ax = b$ un sistema de n ecuaciones lineales, que a su vez es cuadrado y de diagonal predominante escrito en su forma desarrollada.

$$\begin{array}{rcrcrcl} a_{11}x_1 & + & a_{12}x_1 & +\cdots+ & a_{1n}x_n & = & b_1 \\ a_{21}x_1 & + & a_{22}x_1 & +\cdots+ & a_{2n}x_n & = & b_2 \\ & & & & & \vdots & \\ a_{n1}x_1 & + & a_{n2}x_1 & +\cdots+ & a_{nn}x_n & = & b_n \end{array}$$

Como los elementos $a_{ii} = (i = 1,2,..,n)$ de la diagonal son no nulos, el sistema, igual que antes, se puede escribir así:

$$\begin{array}{rcccccccc} x_1 & = & \frac{b_1}{a_{11}} & - & 0x_1 & - & \frac{a_{12}}{a_{11}}x_2 & -\cdots- & \frac{a_{1n}}{a_{11}}x_n \\ x_2 & = & \frac{b_2}{a_{22}} & - & \frac{a_{21}}{a_{22}}x_1 & - & 0x_2 & -\cdots- & \frac{a_{2n}}{a_{22}}x_n \\ & \vdots & & & & & & & \\ x_n & = & \frac{b_n}{a_{nn}} & - & \frac{a_{n1}}{a_{nn}}x_1 & - & \frac{a_{n2}}{a_{nn}}x_2 & -\cdots- & 0x_n \end{array}$$

El proceso iterativo de Seidel queda entonces definido de la siguiente manera:

$$
\begin{aligned}
x_1^{(k)} &= \frac{b_1}{a_{11}} - \frac{a_{12}}{a_{11}}x_2^{(k-1)} - \frac{a_{13}}{a_{11}}x_3^{(k-1)} - \cdots - \frac{a_{1n}}{a_{11}}x_n^{(k-1)} \\
x_2^{(k)} &= \frac{b_2}{a_{22}} - \frac{a_{21}}{a_{22}}x_1^{(k)} - \frac{a_{23}}{a_{22}}x_3^{(k-1)} - \cdots - \frac{a_{2n}}{a_{nn}}x_n^{(k-1)} \\
x_3^{(k)} &= \frac{b_3}{a_{33}} - \frac{a_{31}}{a_{32}}x_1^{(k)} - \frac{a_{32}}{a_{33}}x_2^{(k)} - \cdots - \frac{a_{3n}}{a_{nn}}x_n^{(k-1)} \\
&\vdots \\
x_n^{(k)} &= \frac{b_n}{a_{nn}} - \frac{a_{n1}}{a_{nn}}x_1^{(k)} - \frac{a_{n2}}{a_{nn}}x_2^{(k)} - \cdots - \frac{a_{n,n-1}}{a_{nn}}x_{n-1}^{(k)}
\end{aligned}
$$

Aunque el método de Seidel se puede expresar de forma matricial, no se gana mucho en este sentido, pues las matrices que aparecen ya no son tan simples. **(6)**

1.3.3. MN para Sistemas de Gran Dimensión y Poco Densos

1.3.3.1. Método de Thomas para Matrices Tridiagonales

El método de Thomas es una forma simplificada del método de Eliminación Gaussiana que puede ser usado para resolver sistemas que generan matrices tridiagonales. **(7)**

Un sistema Tridiagonales puede ser escrito de la siguiente manera.

$$a_i x_{i-1} + b_i x_i + c_i x_{i+1} = d_i$$

Donde $a_1 = 0$ y $c_n = 0$

En forma de matriz el sistema puede ser escrito así.

$$
\begin{bmatrix}
b_1 & c_1 & & & \\
a_2 & b_2 & c_2 & & \\
 & a_3 & b_3 & \ddots & \\
 & & \ddots & \ddots & c_{n-1} \\
 & & & a_n & b_n
\end{bmatrix}
\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_n \end{bmatrix}
=
\begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_n \end{bmatrix}
$$

La modificación de los coeficientes se hace como sigue a continuación denotando los coeficientes modificados como primos. **(7)**

$a_i' = 0$

$b_i' = 1$

$$
c_i' = \begin{cases} \dfrac{c_1}{b_1} & ; i = 1 \\ \dfrac{c_i}{b_i - c_{i-1}' a_i} & ; i = 2,3,\ldots,n-1 \end{cases}
$$

$$
d_i' = \begin{cases} \dfrac{d_1}{b_1} & ; i = 1 \\ \dfrac{d_i - d_{i-1}' a_i}{b_i - c_{i-1}' a_i} & ; i = 2,3,\ldots,n \end{cases}
$$

Luego la solución se tendría por sustitución inversa o proceso inverso.

$x_n = d'_n$

$x_{n-1} = d'_{i-1} - c'_{i-1}x_i \quad ; i = n, n-1, \dots, 2$

1.3.3.2. Gauss para Matrices Tridiagonales

En muchos problemas importantes aparecen sistemas de ecuaciones muy grandes con la característica especial de que la matriz del sistema es tridiagonal, esto es, sólo la diagonal principal y las dos adyacentes, no son nulas. Para sistemas de este tipo se puede obtener un algoritmo de Gauss especializado sumamente eficiente. Ante todo, es obvio que en lugar de guardar todos los elementos de la matriz del sistema (la mayoría de los cuales son ceros) es preferible, solamente guardar las tres diagonales no nulas y el vector de términos independientes. El sistema por tanto se expresa matricialmente como: **(6)**

$$\begin{bmatrix} a_1 & c_1 & & & & \\ b_2 & a_2 & c_2 & & & \\ & b_3 & a_3 & c_3 & & \\ & & b_4 & \ddots & \ddots & \\ & & & \ddots & a_{n-1} & c_{n-1} \\ & & & & b_n & a_n \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} d_1 \\ d_2 \\ d_3 \\ \vdots \\ d_{n-1} \\ d_n \end{bmatrix}$$

Donde se ha tomado el convenio de no escribir los términos (todos ellos ceros) fuera de las tres diagonales. Para aplicar el método de Gauss al sistema, primero debe construirse la matriz ampliada:

$$\begin{bmatrix} a_1 & c_1 & & & & & d_1 \\ b_2 & a_2 & c_2 & & & & d_2 \\ & b_3 & a_3 & c_3 & & & d_3 \\ & & b_4 & \ddots & \ddots & & \vdots \\ & & & \ddots & a_{n-1} & c_{n-1} & d_{n-1} \\ & & & & b_n & a_n & d_n \end{bmatrix}$$

En el desarrollo que sigue se supondrá que los términos que se utilizan como denominadores no son ceros. Posteriormente se darán condiciones que garantizan que ello se cumpla. Dividiendo por a_1 toda la primera fila queda:

$$\begin{bmatrix} 1 & p_1 & & & & & q_1 \\ b_2 & a_2 & c_2 & & & & d_2 \\ & b_3 & a_3 & c_3 & & & d_3 \\ & & b_4 & \ddots & \ddots & & \vdots \\ & & & \ddots & a_{n-1} & c_{n-1} & d_{n-1} \\ & & & & b_n & a_n & d_n \end{bmatrix}$$

Donde: $p_1 = {}^{c_1}/_{a_1}$ y $q_1 = {}^{d_1}/_{a_1}$

Ahora se multiplicara la fila por $-b_2$ para sumársela a la segunda fila, de modo que se anule el segundo elemento de la columna 1 (que es único de esa columna por debajo de la diagonal que ya no era cero). La matriz quedará entonces:

$$\begin{bmatrix} 1 & p_1 & & & & & q_1 \\ 0 & r_2 & c_2 & & & & d_2 - b_2 q_1 \\ & b_3 & a_3 & c_3 & & & d_3 \\ & & b_4 & \ddots & \ddots & & \vdots \\ & & & \ddots & a_{n-1} & c_{n-1} & d_{n-1} \\ & & & & b_n & a_n & d_n \end{bmatrix}$$

Donde $r_2 = a_2 - b_2 p_1$

Ahora se divide toda la segunda fila por r_2 con el objetivo de hacer 1 el elemento de la diagonal principal. La matriz ampliada queda de la forma:

$$\begin{bmatrix} 1 & p_1 & & & & & q_1 \\ 0 & 1 & p_2 & & & & q_2 \\ & b_3 & a_3 & c_3 & & & d_3 \\ & & b_4 & \ddots & \ddots & & \vdots \\ & & & \ddots & a_{n-1} & c_{n-1} & d_{n-1} \\ & & & & b_n & a_n & d_n \end{bmatrix}$$

Donde $p_2 = {}^{c_2}/_{r_2}$ y $q_2 = {}^{d_2 - b_2 q_1}/_{r_2}$

Si ahora se usa la segunda fila como pivote para colocar un cero en lugar de b_3 y se definen r_3, p_3 y q_3 de manera análoga al paso anterior, la matriz se transforma en:

$$\begin{bmatrix} 1 & p_1 & & & & & q_1 \\ 0 & 1 & p_2 & & & & q_2 \\ & 0 & 1 & p_3 & & & q_3 \\ & & b_4 & \ddots & \ddots & & \vdots \\ & & & \ddots & a_{n-1} & c_{n-1} & d_{n-1} \\ & & & & b_n & a_n & d_n \end{bmatrix}$$

Donde $r_3 = a_3 - b_3 p_2$, $p_3 = {}^{c_3}/_{r_3}$ y $q_3 = {}^{d_3 - b_3 q_2}/_{r_3}$

Procediendo de esta manera, la matriz ampliada se transforma en una matriz equivalente con todos los elementos nulos por debajo de la diagonal.

$$\begin{bmatrix} 1 & p_1 & & & & & q_1 \\ 0 & 1 & p_2 & & & & q_2 \\ & 0 & 1 & p_3 & & & q_3 \\ & & 0 & \ddots & \ddots & & \vdots \\ & & & 0 & 1 & p_{n-1} & q_{n-1} \\ & & & & 0 & 1 & q_n \end{bmatrix}$$

Donde $p_1 = {}^{c_1}/_{a_1}$, $q_1 = {}^{d_1}/_{a_1}$

Y $r_i = a_i - b_i p_{i-1}$, $p_i = {}^{c_i}/_{r_i}$ y $q_i = \frac{d_i - b_i q_{i-1}}{r_i}$ para $i = 2,3,\dots,n$

El sistema representado por esta matriz tiene la forma:

$$\begin{bmatrix} 1 & p_1 & & & & \\ 0 & 1 & p_2 & & & \\ & 0 & 1 & p_3 & & \\ & & 0 & \ddots & \ddots & \\ & & & 0 & 1 & p_{n-1} \\ & & & & 0 & 1 \end{bmatrix} * \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ \vdots \\ q_{n-1} \\ q_n \end{bmatrix}$$

El proceso inverso se simplifica notablemente por el hecho de que en cada ecuación solamente aparecen dos incógnitas, salvo en la última que permite encontrar x_n. Despejando sucesivamente las incógnitas de abajo hacia arriba, se obtiene:

$$x_n = q_n$$
$$x_{n-1} = q_{n-1} - p_{n-1}x_n$$
$$x_{n-2} = q_{n-2} - p_{n-2}x_{n-1}$$
$$\vdots$$
$$x_2 = q_2 - p_2 x_3$$
$$x_1 = q_1 - p_1 x_2$$

1.3.4. MN para Sistemas de Integración Numérica

1.3.4.1. Regla Rectangular

La regla rectangular para integrales consiste en dividir el área que se desea encontrar en n sub-áreas en forma de rectángulos.

Para la explicación que se muestra a continuación se toman como referencia las siguientes variables:
n: Número de sub-áreas en las cuales se divide el área a calcular
Δ_x ó dx: Ancho o base de cada sub-área
li ó a: Límite inferior definido para el cálculo del área
ls ó b: Límite superior definido para el cálculo del área.

La integral definida entre los puntos a y b de una función continua y acotada $f(x)$ representa el área comprendida debajo de esa función. En ocasiones es necesario calcular integrales (áreas) de modo

numérico, es decir, sin conocer la integral explícita de la función $f(x)$. Existen varios posibles métodos para calcular esta área. Quizás el más sencillo sea sustituir el área por un conjunto de n sub-áreas donde cada sub-área semeja un pequeño rectángulo elemental de base $dx = (b-a)/n$ y altura h, El área sería:

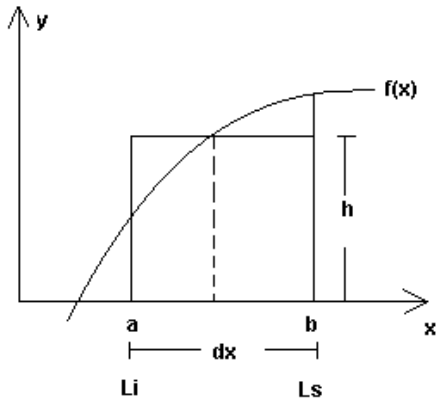


Figura 1: Modelo Rectangular

$\text{área} = \text{h} * \Delta_x$ (Fig 1). Donde h es el valor de la función calculada en el punto medio del área, ó sea $f(a + dx/2)$ y Δx ó dx es el ancho definido para dicha sub-área. Nótese que entre más grande es dx entonces mayor será el área que se quita y pone al área real que se desea calcular (Área colocada entre la función y la línea azul y área quitada entre la función y la línea roja). Si se toma li como límite inferior definido para el cálculo de la integral entonces el punto sobre el eje x para el cálculo de h será: $li + \Delta x/2$. Teniendo en cuenta lo anterior el área será: $área = \Delta x * f(li + dx/2)$

Si el área que se desea calcular se divide entre n sub-áreas, donde cada una de ellas representa un pequeño rectángulo, entonces el área total será:

Área del primer rectángulo:
$A_1 = h_1 * dx = h_1 * \Delta x$, donde h_1 será la función evaluada en la mitad de la sección del primer rectángulo, se podría decir entonces en términos generales que h_1 es igual a la función evaluada en x_1, $h_1 = f(x_1)$. Teniendo en cuenta lo anterior se deduce que $x_1 = (li + dx/2)$ y por lo tanto el área de ese primer rectángulo será:

$$A_1 = \Delta x * f(x_1) = \Delta x * f(li + \frac{\Delta x}{2})$$

Del mismo modo se puede decir que el área del segundo rectángulo es:

$$A_2 = \Delta x * f(x_2) = \Delta x * f(li + 3 * \frac{\Delta x}{2})$$

Área del tercer rectángulo es:

$$A_3 = \Delta x * f(x_3) = \Delta x * f(li + 5 * \frac{\Delta x}{2})$$

Área del enésimo rectángulo:

$$A_i = \Delta x * f(x_i) = \Delta x * f(li + (2 * i - 1)\frac{\Delta x}{2})$$

Área total que será la sumatoria de todas las áreas parciales y quedará así:

$$A_T = \int_{li}^{ls} f(x)dx = \Delta x * \sum_{i=1}^{n} f(li + (2 * i - 1)\frac{\Delta x}{2})$$

La representación gráfica de esta forma de aproximar la integral se presenta en la Figura 2.

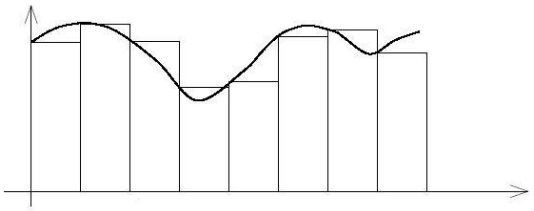


Figura 2: Modelo Rectangular con intervalos

1.3.4.2. Regla Trapezoidal

La Regla Trapezoidal es parte de las fórmulas de integración de Newton-Cotes, las cuales se basan en el reemplazo de una función complicada de resolver de forma manual o datos tabulados con una función aproximada que sea difícil de resolver.

Esta regla para integrales consistente en dividir el área que se desea encontrar en n sub-áreas en forma de trapecios.

El área de un trapecio viene dado por la fórmula $\frac{1}{2}(y_1 + y_2) * h$, donde h es la altura del trapecio mientras que y_1 y y_2 representan las bases del mismo.

La figura muestra como sería el cálculo del área bajo la curva de la función $f(x)$ entre los límites a y b si se dividiera dicha sub-área en un solo trapecio. En este caso el error que se cometería sería demasiado grande con respecto al área real que se desea obtener.

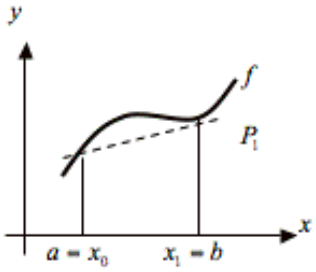


Figura 3: Método Trapecio.

La estrategia más simple y que evitaría menor error en el cálculo, consiste en subdividir el intervalo en n sub-intervalos de pequeño tamaño y aproximar el área como la suma de las áreas de cada uno de los trapecios que se forman.

$\Delta x = (b-a)/n$ sería el ancho de cada una de las sub-áreas. n sería el número de pequeñas sub-áreas en las que se divide el área total que se desea calcular.

Llamando a las coordenadas y_i $(i = 1,2,3,\dots,n)$, las áreas serían.

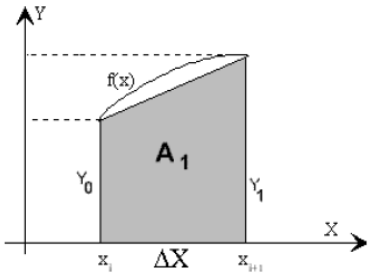


Figura 4: Método Trapecios. Representación del Área.

$$A_1 = \Delta x\left(\frac{y_0 + y_1}{2}\right) \ (Ecuación\ 1)$$

Área del primer trapecio:

$$A_1 = \frac{y_0 + y_1}{2} * \Delta x = \frac{\Delta x}{2} * (y_0 + y_1) = \frac{\Delta x}{2} * [f(li) + f(li + \Delta x)]$$

Esto si se tiene en cuenta $y_0 = f(li) = f(a)$ y $y_1 = f(li + dx) = f(a + dx) = f(a + \Delta x)$, si se toma a $a = li = limite\ inferior$.

Teniendo en cuenta lo anterior se puede decir entonces que el área del segundo trapecio es:

$$A_2 = \frac{y_1 + y_2}{2} * \Delta x = \frac{\Delta x}{2} * (y_1 + y_2) = \frac{\Delta x}{2} * [f(li + \Delta x) + f(li + 2 * \Delta x)]$$

Y por lo tanto el área del tercer trapecio sería:

$$A_3 = \frac{y_2 + y_3}{2} * \Delta x = \frac{\Delta x}{2} * (y_2 + y_3) = \frac{\Delta x}{2} * [f(li + 2 * \Delta x) + f(li + 3 * \Delta x)]$$

Y así sucesivamente hasta llegar a la enésima área que sería:

$$A_2 = \frac{y_{i-1} + y_i}{2} * \Delta x = \frac{\Delta x}{2} * (y_{i-1} + y_i) = \frac{\Delta x}{2} * [f(li + (i-1)\Delta x) + f(li + i * \Delta x)]$$

El área total comprendida entre $X = a$ y $X = b$ está dada por:

$$A = \int_a^b f(x)dx \cong A_1 + A_2 + A_3 + \cdots + A_n \quad (Ecuación\ 2)$$

Sustituyendo la ecuación 1 en la 2 se obtiene

$$A = \int_a^b f(x)dx \cong \frac{\Delta x}{2}(y_0 + 2y_1 + 2y_2 + \cdots + 2y_{n-1} + y_n)$$

La cual recibe el nombre de Fórmula Trapezoidal, y se puede expresar como:

$$A = \int_a^b f(x)dx \cong \frac{\Delta x}{2}\left(y_0 + y_n + 2\sum(y_2 + y_3 + \cdots + y_{n-1})\right)$$

$$A = \int_a^b f(x)dx \cong \frac{\Delta x}{2}\left(y_0 + y_n + 2\sum_{i=1}^{n-1} y_i\right)$$

Ahora se sabe que y_0 y y_n son valores de la evaluación de la función en cada uno de los límites, es decir y_0 es la función evaluada en el límite a y y_n es la función evaluada en el límite b.

$$A = \int_a^b f(x)dx \cong \frac{\Delta x}{2}\left(f(a) + f(b) + 2\sum_{i=1}^{n-1} y_i\right)$$

Ahora, y_i sería la evaluación en cada uno de los puntos sobre el eje x de base común a cada una de las sub-áreas.

$$y_1 = f(a + 1dx)$$

$$y_2 = f(a + 2dx)$$

$$y_3 = f(a + 3dx)$$

$$\vdots$$

$$y_i = f(a + idx)$$

Por tanto la ecuación general para el cálculo de la integral por el método trapezoidal será:

$$A = \int_a^b f(x)dx \cong \frac{\Delta x}{2}\left(f(a) + f(b) + 2\sum_{i=1}^{n-1} f(a + i * dx)\right)$$

Que también se puede expresar como:

$$A = \int_a^b f(x)dx \cong \frac{\Delta x}{2}\left(\frac{f(a) + f(b)}{2} + \sum_{i=1}^{n-1} f(a + i * dx)\right)$$

1.3.4.3. Método de Simpson

Otra manera de obtener una estimación aún más exacta de una integral, es la de usar polinomios de orden superior para conectar los puntos, en el caso particular del método que usa orden 2, es decir de la forma $ax^2 + bx + c$.

A las fórmulas resultantes de calcular la integral bajo estos polinomios se les conoce como Reglas de Simpson.

En este procedimiento se toma el intervalo de anchura $2h$, comprendido entre x_i y x_{i+2}, y se sustituye la función $f(x)$ por la parábola que pasa por tres puntos (x_i, y_i), (x_{i+1}, y_{i+1}), (x_{i+2}, y_{i+2}).

En el método de Simpson se asume cada sub-área como un pequeño arco de parábola de la forma $ax^2 + bx + c$ con límite inferior en $-h$ y límite superior en h por ende la mitad de la pequeña área se encontrará en el punto 0, tal como muestra la figura.

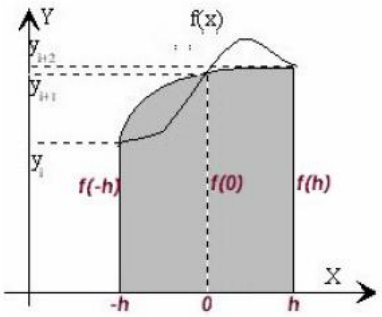


Figura 5: Sub-áreas de Simpson

Integrando este arco de parábola entre los límites descritos se tiene:

$$\int_{-h}^{h} (ax^2 + bx + c)dx = \frac{ax^3}{3} + \frac{ax^2}{2} + cx$$

Reemplazando cada uno de los límites se tiene:

$$\left[\frac{ah^3}{3}+\frac{bh^2}{2}+ch\right]-\left[-\frac{ah^3}{3}+\frac{bh^2}{2}-ch\right]$$

Ahora quitando los paréntesis se tendrá:

$$\frac{ah^3}{3}+\frac{bh^2}{2}+ch+\frac{ah^3}{3}-\frac{bh^2}{2}+ch=2\frac{ah}{3}+2ch$$

Simplificando un poco la ecuación se tendrá:

$$\int_{-h}^{h}(ax^2+bx+c)dx=\frac{h}{3}[2ah^2+6c]\quad (Ecuación\ 1)$$

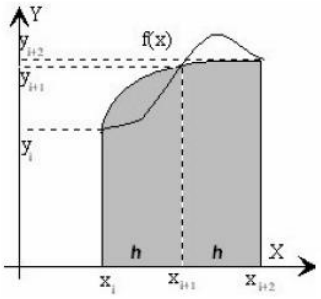


Figura 6: Notaciones de Simpson

Observando la figura, en lo que respecta a notaciones, se puede decir que $f(x_i)=y_i=f(-h)$, $f(x_{i+1})=y_{i+1}=f(0)$, $f(x_{i+2})=y_{i+2}=f(h)$, entonces se podría obtener el siguiente sistema de ecuaciones, evaluando la ecuación general de la parábola ax^2+bx+c en cada uno de los puntos de la pequeña sub-área $[-h, 0-h]$:

$f(-h)=ah^2-bh+c$, se puede tomar esta altura como $y_0=f(x_i)$.

$f(0)=c$, se toma esta altura como $y_1=f(x_{i+1})$.

$f(h)=ah^2+bh+c$, y esta altura como $y_2=f(x_{i+2})$.

De lo anterior se puede decir

$$y_0+y_2=2ah^2+2c\quad (Ecuación\ 2)$$

$$y_1=c\quad (Ecuación\ 3)$$

Retomando la ecuación (1) se puede expresar igualmente de la siguiente manera:

$$\int_{-h}^{h}(ax^2+bx+c)dx=\frac{h}{3}[2ah^2+2c+4c] \quad (Ecuación\ 4)$$

Reemplazando las ecuaciones 2 y 3 en la 4 se tiene que:

$$\int_{-h}^{h}(ax^2+bx+c)dx=\frac{h}{3}[y_0+4y_1+y_2]=A_1 \quad (Ecuación\ 5)$$

Interpretando la ecuación 5 con base en la sub-área seleccionada A_1 para desarrollar el Método de Simpson, se diría que el área del segmento es igual a la suma de la altura o función evaluada en el lado izquierdo más cuatro veces la función evaluada en la parte central de la sub-área más la función evaluada en el lado derecho de la sub-área, todo esto multiplicado por el ancho de la sub-área y divido por 3.

La simple inspección visual de esta figura y la que describe el procedimiento de los trapecios o los rectángulos, confirma que el Método de Simpson deberá ser mucho más exacto que los procedimientos mencionados.

Si a y b se denominan como x_0 y x_2, y $f_i(x_i)$ se representa mediante un polinomio de Lagrange de segundo orden, entonces la integral sería:

$$I=\left[\begin{array}{c}\frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}f(x_0)+\frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}f(x_1)+\\ \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}f(x_2)\end{array}\right]dx$$

Después de integrar y de reordenar los términos, resulta la siguiente ecuación:

$$I=(b-a)\frac{f(x_0)+4f(x_1)+f(x_2)}{6} \quad (5a)$$

Si se toma $(b-a)/6\approx h/3$, $f(x_0)=y_0$, $f(x_1)=y_1$ y $f(x_2)=y_2$, entonces se tiene como solución de la sub área $I=\frac{h}{3}(y_0+4y_1+y_2)$, que sería lo mismo mostrado en la ecuación 5.

Ahora se sabe que el área que se desea encontrar sería la sumatoria de todas las sub-áreas que se calculen. Al igual que los métodos de la Regla Trapezoidal y de la Regla Rectangular, entre más sub-áreas tenga la integral a calcular, más exacto será el valor encontrado. El área aproximada en el intervalo $[a,b]$ es:

$$\int_a^b f(x)dx = A_1 + A_2 + A_3 + \cdots + A_n$$

Ahora dejando esta ecuación en términos de la ecuación (5) se tendrá

$$\int_a^b f(x)dx = \frac{h}{3}(y_0 + 4y_1 + y_2) + \frac{h}{3}(y_2 + 4y_3 + y_4) + \cdots + \frac{h}{3}(y_{2n-2} + 4y_{2n-1} + y_{2n})$$

Simplificando $h/3$ y sumando los términos se tendrá:

$$\int_a^b f(x)dx = \frac{h}{3}(y_0 + 4y_1 + y_2 + 4y_3 + 2y_4 + 4y_5 + 2y_6 + \cdots + 4y_{2n-1} + y_{2n})$$

Donde n sería el número de sub-áreas en el cual se ha dividido el área que se desea calcular.

Todo esto se podría representar de la siguiente manera:

$$\int_a^b f(x)dx = \frac{h}{3}\left[y_0 - y_{2n} + 4\sum_{i=1}^{n} y_{2i-1} + 2\sum_{i=1}^{n-1} y_{2i}\right] \quad (Ecuación\ 6)$$

Donde el primer y segundo término del corchete contiene los valores de la evaluación de la función en los extremos, el tercero la suma de los términos de índice impar y el cuarto término la suma de los términos de índice par.

Ahora lo que se conoce en un momento determinado, cuando se desea calcular el valor de la integral definida, son los siguientes términos:

$a = Limite\ inferior$

$b = Limite\ superior$

$n = Número\ de\ sub\ áreas$

$f(x) Funcion\ sobre\ la\ cual\ se\ desea\ integrar$

Con los valores anteriores se puede calcular el valor dx así $dx = (b - a)/n$ y $h = dx/2$.

Es necesario entonces dejar la ecuación en términos de $f(x), a, b$ y dx ó h así:

Los primeros términos $y_0 = f(a)$ y $y_{2n} = f(b)$

Analizando ahora los términos impares: $y_1 = f(a + 1dx/2)$, $y_3 = f(a + 3dx/2)$, $y_5 = f(a + 5dx/2)$, por tanto se tendría de manera general:

$y_{2i-1} = f(a + (2i - 1)dx/2)$ ó $y_{2i-1} = f(a + (2i - 1)h)$

Analizando ahora los términos pares: $y_2 = f(a + 1dx)$, $y_4 = f(a + 2dx)$, $y_6 = f(a + 3dx)$ por tanto se tendría de manera general:

$y_{2i} = f(a + idx)$ ó $y_{2i} = f(a + 2ih$

Por todo lo antes planteado se tendrá en forma definitiva la solución:

$$\int_a^b f(x)dx = \frac{h}{3}\left[f(a) - f(b) + \sum_{i=1}^{n} 4f\left(a + \frac{(2i-1)dx}{2}\right) + 2f(a + idx)\right]$$

1.3.4.4. Método de Romberg

La integración de Romberg es un método numérico para obtener una estimación del valor de una integral definida con base en dos o más aplicaciones de una fórmula como la de los Trapecios (o Simpson) empleando diferentes tamaños de paso, pero que es mejorada al combinarse con el proceso de extrapolación de Richardson.

Este método aporta un procedimiento de cálculo numérico de integrales de aproximaciones sucesivas a través de la idea de estimar el error a través del doble cálculo.

La fórmula de extrapolación de Richardson:

$$I_h + \frac{I_h - I_{2h}}{2^p - 1} = \frac{2^p I_h - I_{2h}}{2^p - 1}$$

Es la base del método de Romberg, para un mejor entendimiento del mismo se realizaron algunos cambios en la notación.

I_0^0 el resultado obtenido mediante el método de los trapecios usando un número pequeño de n subintervalos.
I_1^0el resultado obtenido mediante el método de los trapecios con $2n$ subintervalos.
I_2^0el resultado obtenido mediante el método de los trapecios con $4n$ subintervalos.
I_k^0el resultado obtenido con $\left(2^k\right) * n\ subintevalos\ para\ k = 0,1,2, \dots, m$

Con estos valores iniciales, se pueden calcular mejores aproximaciones mediante la fórmula de Richardson, con la notación antes propuesta la fórmula de Richardson se transforma en:

$$I_k^1 = \frac{2^p I_k^0 - I_{k-1}^0}{2^p - 1} \quad k = 1,2,3, \ldots$$

Así mismo, para el método de los trapecios es $p = 2$

$$I_k^1 = \frac{2^p I_k^0 - I_{k-1}^0}{3} \quad k = 1,2,3, \ldots$$

Esta última fórmula permite, con una reducida cantidad de operaciones, mejorar las aproximaciones iniciales realizada por el método de los trapecios. Los resultados de la iteración 1 I_k^1 poseen un error de truncamiento de orden 4.

Este proceso puede continuarse. A partir de la iteración 1, se pueden obtener los de la iteración 2, para eso basta tomar en la fórmula propuesta $p + 2$, en lugar de p

$$I_k^2 = \frac{2^{p+2} I_k^1 - I_{k-1}^1}{2^{p+2} - 1} \quad k = 2,3,4, \ldots$$

En resumen se puede decir que la iteración m se obtiene a partir de la $m - 1$

$$I_k^m = \frac{2^{p+2(m-1)} I_k^{m-1} - I_{k-1}^{m-1}}{2^{p+2(m-1)} - 1}$$

Donde el error de truncamiento es $p + 2(m - 1)$, por lo que se puede afirmar que con cada nueva iteración el orden del error se incrementa en 2. Como $p = 2$ la fórmula anterior quedaría:

$$I_k^m = \frac{4^m I_k^{m-1} - I_{k-1}^{m-1}}{4^m - 1} \quad m = 1,2,3 \ldots \quad k = m, m+1, m+2$$

La siguiente tabla muestra la relación entre las diversas aproximaciones obtenidas en el proceso iterativo. Esta tabla se va construyendo por filas, el primero elemento de la fila se calcula mediante el método de los trapecios y los restantes por la fórmula de Richardson. En cada fila el error se acota mediante la diferencia en valor absoluto entre el último elemento de la fila y el último elemento de la fila precedente. **(6)**

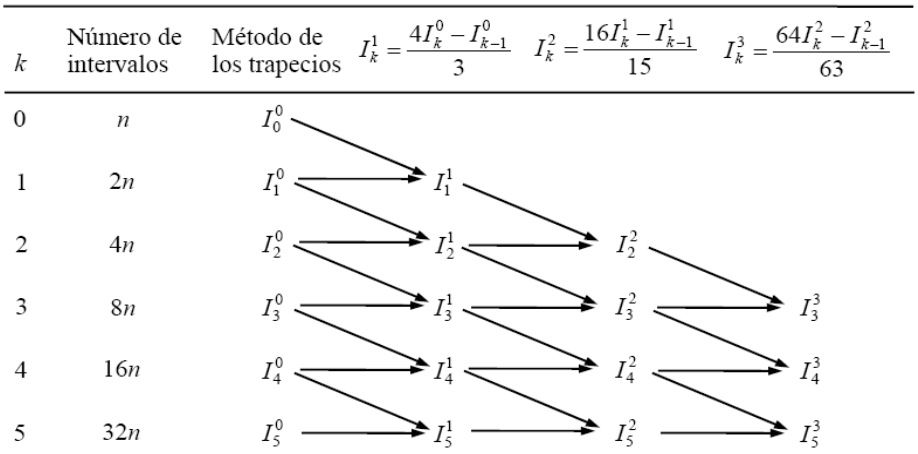


Figura 7: Tabla resultado iteraciones de Romberg

1.4. Estado del Arte

1.4.1. Herramientas Matemáticas

1.4.1.1. Derive

DERIVE es un paquete de software con capacidad para desarrollar cálculo simbólico, análisis gráfico y manipulación numérica. Se trata de un programa que se ejecuta en el entorno Windows y que, por lo tanto, presenta las características habituales que tienen dichas aplicaciones. **(8)**

Es un programa de álgebra computacional usado ampliamente con propósitos educativos. Además es uno de los llamados "Programas de Cálculo Simbólico", que se pueden definir como programas para ordenadores personales (PC) que sirven para trabajar con la matemática usando las notaciones propias (simbólicas) de esta ciencia.

Derive permite el trabajo con distintas áreas de la matemáticas, entre otras se pueden citar, el trabajo con vectores, matrices y determinantes, resolución de ecuaciones y de sistemas de ecuaciones, trabajo con derivadas, integrales (definidas e indefinidas), series, límites, polinomios de Taylor, representación gráfica de funciones en forma explícita, implícita, paramétrica y en coordenadas polares, representación gráfica de funciones de dos variables y operaciones con polinomios y fracciones algebraicas.

Además, es posible programar funciones que usen las distintas capacidades del programa, de modo que aumenta así sensiblemente el espectro de sus aplicaciones. Derive se suministra con varios ficheros de funciones para propósitos diversos como resolver ecuaciones diferenciales, trabajar en Álgebra Lineal, etc. **(8)**

Su uso esta restringido a la plataforma Windows en sus versiones 98, Me, 2000 y XP.

1.4.1.2. Matlab

MATLAB es un entorno de computación y desarrollo de aplicaciones totalmente integrado orientado a proyectos en donde se encuentran implicados elevados cálculos matemáticos y visualización gráfica de los mismos. MATLAB integra análisis numérico, cálculo matricial, proceso de señal y visualización gráfica en un entorno completo donde los problemas y sus soluciones son expresados del mismo modo en que se escribirían tradicionalmente, sin necesidad de hacer uso de la programación tradicional. **(9)**

El nombre de MATLAB proviene de la contracción de los términos MATrix LABoratory y fue inicialmente concebido para proporcionar fácil acceso a las librerías *LINPACK* y *EISPACK*, las cuales representan hoy en día dos de las librerías más importantes en computación y cálculo matricial.

MATLAB es un sistema de trabajo interactivo cuyo elemento básico de trabajo son las matrices. El programa permite realizar de un modo rápido la resolución numérica de problemas.

Los usos más característicos de la herramienta se encuentran en áreas de computación y cálculo numérico tradicional, prototipaje algorítmico, teoría de control automático, estadística, análisis de series temporales para el proceso digital de señal.

MATLAB dispone de un “integrador” simbólico.

Además también se dispone del programa Simulink que es un entorno gráfico interactivo con el que se puede analizar, modelar y simular la dinámica de sistemas no lineales.

1.4.1.3. Mathematica Software

Es una herramienta especializada en análisis numérico y cálculo simbólico, que incorpora un potente lenguaje de programación propio y una interfaz externa que permite salidas a C, Fortran y TEX, además de otras potentes comunicaciones con otros paquetes mediante MathLink. **(10)**

Ingenieros, científicos, analistas financieros, investigadores, profesores y estudiantes de enseñanza superior usan en todo el mundo Mathematica para desarrollar sus cálculos de precisión en proyectos críticos debido a que es una herramienta interactiva de cálculo y un versátil lenguaje de programación para una rápida y precisa solución a problemas técnicos.

Por sus amplias capacidades de cálculo, gráficos y sonidos, tiene interesantísimas aplicaciones en los siguientes campos:

- ✓ Ciencias Físicas: Física teórica y experimental, química, ciencias de los materiales y ciencias de la tierra.
- ✓ Ciencias de la Computación: Ingeniería informática, desarrollo de software, gráficos.

- ✓ Ciencias Matemáticas: Matemática pura, matemática aplicada, estadística, investigación operativa, etc.
- ✓ Negocios y Finanzas: Análisis financiero, economía, ciencias actuariales, gestión.
- ✓ Ciencias de la Salud: Investigación médica, biología, bioquímica, psicología y ciencias del entorno.

Posee varias características que lo ameritan como uno de los software de cálculos matemáticos más usados en las universidades y demás niveles en los cuales sea de gran ayuda. Estas son:

- ✓ Realización de cálculos y simulaciones de cualquier nivel de complejidad mediante el uso de la amplia biblioteca de funciones matemáticas y computacionales.
- ✓ Rápida y fácil importación y exportación de datos, que incluye imágenes y sonido, en más de veinte formatos.
- ✓ Generación de documentos interactivos, independientes de la plataforma, con textos, imágenes, expresiones matemáticas, botones e hyperlinks.
- ✓ Entrada de expresiones a través del teclado o de la paleta (programable) más adecuada.
- ✓ Construcción de complejas expresiones y fórmulas con formato automático y ruptura de líneas.
- ✓ Exportación de los "notebooks" a formato HTML para presentaciones web o LaTeX para publicaciones especiales.

1.4.2. Bibliotecas

1.4.2.1. Blas

Basic Linear Algebra Subroutines ó Subrutinas de Algebra Lineal Básica.

El paquete BLAS contiene una serie de funciones básicas para la resolución de problemas básicos del álgebra lineal, como la suma o producto de vectores y matrices. BLAS dispone de versiones especiales para matrices estructuradas (triangulares, banda, etc.). BLAS se encuentra organizado por niveles. El nivel uno corresponde a las funciones que realizan operaciones sobre vectores (como la suma de vectores). El nivel dos corresponde a las funciones que realizan operaciones sobre matrices y vectores (como el producto de matriz por vector) y el nivel tres corresponde a las funciones que realizan operaciones sobre matrices. Cuanto mayor sea el nivel de las operaciones, mejores resultados se obtienen. Todas las funciones BLAS mantienen una nomenclatura estándar. Asumiendo un nombre de función con siete letras agrupadas en tres bloques (TMMOOOO, tipo de datos, tipo de matriz, código de operación), las opciones son las siguientes: **(11)**

1.4.2.2. CBlas

Se corresponde con la biblioteca BLAS versión en “C”. CBLAS proporciona a la interfaz de C BLAS rutinas, que fueron originalmente escrito en FORTRAN. El CBLAS interfaz de la biblioteca fue compilado utilizando el compilador de C de IGP.

1.4.2.3. Lapack

Linear Algebra Package ó Paquete de Algebra Lineal.

Es una colección de subrutinas escritas en Fortran77 para resolver los problemas matemáticos más comunes que surgen a partir de la modelación y que se enmarcan en el campo del álgebra lineal numérica.

El paquete Lapack contiene una serie de funciones para la resolución de problemas fundamentales del álgebra lineal, como la resolución de sistemas de ecuaciones, la factorización en valores propios, numerosas descomposiciones matriciales, etc. Lapack dispone de versiones especiales para matrices estructuradas (triangulares, banda, etc.) y se encuentra en continuo desarrollo desde 1992. LAPACK utiliza BLAS para la implementación de las funciones. **(11)**

Las rutinas de Lapack se estructuran en tres niveles. En primer lugar, se definen las rutinas ‘driver’, que resuelven problemas completos (resolución de un sistema de ecuaciones, resolución de un problema de mínimos cuadrados, cálculo de los valores propios o singulares de una matriz). Las funciones ‘driver’ son las que se recomienda utilice el usuario. En segundo lugar se definen las funciones computacionales, que resuelven tareas computacionales completas (factorización LU, descomposición QR) que si bien pueden utilizarse directamente, están generalmente orientadas a resolver una parte de un problema completo que resuelve una rutina ‘driver’. Finalmente se encuentran las rutinas auxiliares, utilizadas por las rutinas ‘driver’ y computacionales para la resolución de los problemas. **(11)**

Incluye rutinas para resolver sistemas de ecuaciones lineales, sistemas de ecuaciones lineales por mínimos cuadrados, problema de valores propios y problemas de valores singulares.

1.4.2.4. Atlas

Automatically Tuned Linear Algebra Software ó Software de Álgebra Lineal Sintonizado Automáticamente.

ATLAS es un proyecto de investigación y un paquete de software. El propósito de ATLAS es proveer un software de álgebra lineal altamente portable. La versión más reciente provee una completa API

para C y Fortran77. ATLAS puede ser usado en cualquier rutina que necesite cálculo rápido de álgebra lineal. ATLAS es usado ampliamente por investigadores**.**

ATLAS es una implementación de un nuevo estilo de software de alto rendimiento de producción y el mantenimiento automatizado de la llamada Optimización Empírica de Software (AEOS).

En un AEOS habilitado para la biblioteca, hay diferentes formas de realizar un determinado núcleo de operaciones donde son ofrecidos, temporizadores y empíricamente se usan para determinar que aplicación es la mejor para una determinada plataforma de arquitectura. ATLAS utiliza dos técnicas para el suministro de diferentes implementaciones de operaciones del kernel: múltiple aplicación y generación de código.

En la generación de código, está escrito que pueden generar diferentes implementaciones del kernel para un parámetro altamente programable, un ejemplo de ello es el generador de código.

1.4.2.5. ScaLapack

Software Library for Linear Algebra Computations on Distributed-Memory Computers

ScaLapack incluye un subconjunto de rutinas Lapack rediseñadas para memorias distribuidas MIMD para computación paralela o distribuida. Está diseñado para cálculo heterogéneo y es portable sobre cualquier sistema que soporte MPI o PVM. Igual que Lapack, las rutinas de ScaLapack están basadas en algoritmos particionados en bloques para minimizar la frecuencia de movimiento de datos entre diferentes niveles de jerarquía de memoria. **(12)**

Scalapack puede resolver sistemas de ecuaciones lineales, problemas lineales de mínimos cuadrados, problemas de valor propio, y problemas de valor singular. ScaLapack también puede manejar muchos cálculos asociados, tales como la estimación de la factorización de una matriz.

1.4.2.6. SLEPc

Scalable Library for Eigenvalue Problem Computations.

Se puede utilizar para la solución de los problemas formulados ya sea estándar o en forma generalizada, así como otros relacionados con problemas como la descomposición de valor singular.

El énfasis del programa se produce en los métodos y técnicas apropiadas para los problemas en los que las matrices asociadas son escasas. Por lo tanto, la mayoría de los métodos ofrecidos por la biblioteca son los métodos de proyección u otros métodos con propiedades similares. Ejemplos de estos métodos son Arnoldi, y Lanczos Subspace Iteration, por nombrar algunos. SLEPc implementa estos métodos básicos, así como más sofisticados algoritmos.

También ofrece una función de apoyo para el espectro de transformaciones. SLEPc es una biblioteca general en el sentido de que abarca norma generalizada y problemas de valor propio, ya sea real o con complejo de la aritmética.

SLEPc se construye en la parte superior de PETSc, Se puede considerar una extensión de PETSc ya que proporciona todas las funcionalidades necesarias para la solución de los problemas de valor propio. Esto significa que PETSc debe ser instalado con anterioridad, a fin de utilizar SLEPc. Los usuarios de PETSc encontrarán SLEPc muy fácil de usar, ya que aplica el mismo paradigma de programación. (13)

1.4.2.7. PETSc

Portable, Extensible Toolkit for Scientific Computation ó Herramienta Portable Extensible de Computación Científica.

Biblioteca para la resolución numérica de aplicaciones científicas modeladas mediante ecuaciones en derivadas parciales.

PETSc se destina para el uso en gran escala de proyectos de aplicación, la ciencia computacional en curso. Muchos proyectos se desarrollarán en torno a las bibliotecas PETSc. Es fácil de utilizar para los principiantes, incluye un gran conjunto de paralelas de ecuaciones lineales y no lineales de resolver que son fácilmente utilizados en la aplicación de códigos escrito en C, C + +, Fortran y Python ahora.

Una característica de PETSc es la inclusión entre sus funciones de métodos de resolución de sistemas de ecuaciones lineales por métodos iterativos.

Incluye el Bigradiente Conjugado y el GMRES (Generalized Minimal Residual). Estos son los métodos que mejores resultados dieron cuando se utilizó la librería CXML (Compaq eXtended Mathematical Library). Esta última librería se descartó ya que no esta disponible para Linux y es comercial.

Otra característica de PETSc es que es capaz de procesar los sistemas de ecuaciones en paralelo, aprovechando así la capacidad de cómputo del clúster GIRMA-Lab1. Todo ello de forma muy transparente a la hora de programarlo, dispone de una buena documentación, muy bien estructurada y con numerosos ejemplos. Es portable ya que funciona tanto en Unix como en Windows.

1.4.2.8. PSPASES

Parallel SPArse Symmetric dirEct Solver

Librería para la resolución, por métodos directos, de sistemas de ecuaciones cuya matriz de coeficientes es dispersa, simétrica y definida positiva.

Es de un alto rendimiento, escalable, de forma paralela, basada en la biblioteca MPI, destinados a la resolución de sistemas de ecuaciones lineales de matrices esparcidas. La biblioteca ofrece diversas interfaces para resolver el sistema utilizando cuatro fases del método directo de la solución: llenar y calcular la reducción de pedidos, realizar factorización simbólica, el cálculo numérico de factorización, triangular y resolver sistemas de ecuaciones. La biblioteca pone en práctica de manera eficiente los algoritmos paralelos escalables desarrollado por los autores, para calcular cada una de las fases. (14)

1.5. Conclusiones parciales

El presente capítulo recogió el resultado de todo un análisis de los conceptos fundamentales relacionados con el trabajo y los conceptos fundamentales de los métodos numéricos a implementar en la biblioteca. Se hace referencia además a algunas de las bibliotecas existentes que implementan distintos métodos numéricos.

CAPÍTULO 2: Tendencias y Tecnologías actuales a desarrollar.

2.1. Introducción

En este capítulo se aborda lo correspondiente a las tendencias y tecnologías actuales que se utilizan en el desarrollo de bibliotecas haciendo énfasis en las que se apoya nuestra propuesta para su desarrollo. Se abordan temas relacionados con las herramientas, lenguaje de programación a utilizar, así como el lenguaje de modelado y la metodología a emplear.

2.2. Metodología de Desarrollo de Software

Todo desarrollo de software es riesgoso y difícil de controlar, en este punto es donde se hace necesario el uso de una metodología de software, la cual plantea un conjunto de pasos y procedimientos que deben seguirse para desarrollar un producto. Puede seguir uno o varios modelos de ciclo de vida los cuales indican que es lo que hay que obtener a lo largo del desarrollo del proyecto.

Es también objetivo de la metodología elevar la calidad del producto final trabajando en función de esta en cada una de las fases que define.

2.2.1. Rational Unified Process (RUP)

El Proceso Unificado es un proceso de desarrollo de Software.

El Proceso Unificado es un marco de trabajo genérico que puede especializarse para una gran variedad de sistemas de software, para diferentes áreas de aplicación, diferentes tipos de organización, diferentes niveles de aptitud y diferentes tamaños de proyecto.

El Proceso Unificado está basado en componentes, lo cual quiere decir que el sistema de software en construcción está formado por componentes de software interconectados a través de interfaces bien definidas. Es un proceso que define claramente quién, cuando, como y qué debe hacerse, y como su enfoque está basado en modelos, utiliza un lenguaje bien definido para tal fin, el UML.

RUP posee tres características esenciales *está dirigido por los Casos de Uso*: que orientan el proyecto a la importancia para el usuario y lo que este quiere, *está centrado en la arquitectura*: que relaciona la toma de decisiones que indican cómo tiene que ser construido el sistema y en qué orden, y es *iterativo*

e incremental: donde divide el proyecto en mini proyectos donde los casos de uso y la arquitectura cumplen sus objetivos de manera más depurada. **(15)**

La metodología RUP esta dividida en cuatro fases de desarrollo del producto:

- ✓ *Inicio:* El objetivo en esta etapa es determinar la visión del proyecto.
- ✓ *Elaboración:* En esta etapa el objetivo es determinar la arquitectura óptima.
- ✓ *Construcción:* En esta etapa el objetivo es llevar a obtener la capacidad operacional inicial, o sea crear el producto.
- ✓ *Transición:* El objetivo es llegar a obtener el release del proyecto.

RUP describe como:

- ✓ Obtener, organizar y documentar la funcionalidad y restricciones requeridas.
- ✓ Documentar y monitorear las alternativas y decisiones.
- ✓ Las nociones de Casos de Uso y de Escenarios utilizadas en RUP han demostrado ser una manera excelente de capturar los requerimientos funcionales y asegurarse que direccionan el diseño, la implementación y la prueba del sistema, logrando así que el sistema satisfaga las necesidades del usuario.

Utiliza arquitecturas basadas en componentes:

- ✓ El proceso de software debe focalizarse en el desarrollo temprano de una arquitectura robusta ejecutable, antes de comprometer recursos para el desarrollo en gran escala. RUP describe como diseñar una arquitectura flexible, que se acomode a los cambios, comprensible intuitivamente y promueve una más efectiva reutilización de software. *Soporta el desarrollo de software basado en componentes*: módulos no triviales que completan una función clara. RUP provee un enfoque sistemático para definir una arquitectura utilizando componentes nuevos y preexistentes.

Modelización del software visualmente:

- ✓ RUP muestra como hacer la modelación de un software visualmente para capturar la estructura y comportamiento de arquitecturas y componentes. Las abstracciones visuales ayudan a comunicar diferentes aspectos del software; comprender los requerimientos, ver como los elementos del sistema se relacionan entre sí, mantener la consistencia entre diseño e implementación y promover una comunicación precisa. El estándar UML (Lenguaje de Modelado Unificado), creado por Rational Software, es el cimiento para una modelización visual exitosa.

Verifica la calidad del software:

- ✓ Es necesario evaluar la calidad de un sistema respecto de sus requerimientos de funcionalidad, confiabilidad y performance. La actividad fundamental es el testing, que permite encontrar las

fallas antes de la puesta en producción. RUP asiste en el planeamiento, diseño, implementación, ejecución y evaluación de todos estos tipos de testing.

- ✓ El aseguramiento de la calidad se construye dentro del proceso, en todas las actividades, involucrando a todos los participantes, utilizando medidas y criterios objetivos, permitiendo así detectar e identificar los defectos en forma temprana.

Controla los cambios al software:

- ✓ La capacidad de administrar los cambios es esencial en ambientes en los cuales el cambio es inevitable. RUP describe como controlar, rastrear y monitorear los cambios para permitir un desarrollo iterativo exitoso. Es también una guía para establecer espacios de trabajo seguros para cada desarrollador, suministrando el aislamiento de los cambios hechos en otros espacios de trabajo y controlando los cambios de todos los elementos de software (modelos, código, documentos, etc.). Describe como automatizar la integración y administrar la conformación de releases.

2.2.2. Extreme Programming (XP)

Esta metodología se basa en la idea de que existen cuatro variables que guían el desarrollo de sistemas: Costo, Tiempo, Calidad y Alcance. La manera de encarar los desarrollos avalados por este modelo de desarrollo es permitir a las fuerzas externas (gerencia, clientes) manejar hasta tres de estas variables, quedando el control de la restante en manos del equipo de desarrollo. Este modelo hace visibles de manera más o menos continua estas cuatro variables. **(16)**

Características de XP

Es imposible prever todo antes de comenzar a programar; es imposible o si lo fuera es demasiado costoso e innecesario, ya que muchas veces se gasta demasiado tiempo y recursos en cambiar la documentación de la planificación para que se parezca al código. Para evitar esto, XP intenta implementar una forma de trabajo donde se adapte fácilmente a las circunstancias.

Básicamente consiste en trabajar estrechamente con el cliente, haciendo pequeñas iteraciones (mini-entregas), cada dos semanas, donde no existe más documentación que el código en sí; cada versión contiene las modificaciones necesarias según el cliente vaya retroalimentando el sistema (por eso es necesaria la disponibilidad del cliente durante todo el desarrollo).

Para suplir la falta de requisitos, casos de uso, y demás herramientas; XP utiliza *historias de usuarios*, la historia de usuario es una frase corta que representa alguna función que realizara el sistema. Cada historia de usuario no puede demorar en desarrollarse más de una semana, si así lo requiriera, debe segmentarse.

Es requisito para XP definir un *estándar en el tipo de codificación*, esto hace que los programadores tengan definido ya el estilo de programación y no que cada uno programe a su estilo.

El *testing en cada iteración* es más que importante; de eso se trata este paradigma de programación, corregir mientras se programa. De esta forma se van cubriendo todos los baches que cada versión padezca.

El código no es de nadie, todo el equipo puede manipular el código que existe, de esta forma cada pareja puede mejorar cada sección de código que utiliza, esto requiere de un testing del mismo y la re-implementación en el sistema general.

Cada dos semanas se entrega una versión al cliente, que lo verifica, realiza el feedback y se continúa el desarrollo; este ciclo continúa hasta que el sistema cumpla con las expectativas del cliente, acto que concluirá el proyecto.

No existe documentación del proyecto (para nosotros, el talón de Aquiles de este modelo) lo que más se acerca a la documentación son las historias de usuario, pero al concluir el proyecto se descartan. Inclusive se recomienda hacer dos secciones, una con todas las historias de usuario que faltan por desarrollar, y otra donde se archiven las concluidas, esto aproximará el estado de avance del proyecto.

2.2.3. Microsoft Solution Framework (MSF)

Esta es una metodología flexible e interrelacionada con una serie de conceptos, modelos y prácticas de uso, que controlan la planificación, el desarrollo y la gestión de proyectos tecnológicos. MSF se centra en los modelos de proceso y de equipo dejando en un segundo plano las elecciones tecnológicas. **(17)**

MSF proporciona prácticas comprobadas para planificar, crear y poner en marcha soluciones exitosas. En oposición a la metodología prescriptiva, MSF proporciona una estructura flexible y escalable para conocer las necesidades de una organización o equipo encargado de un proyecto de cualquier tamaño. La orientación de MSF consiste en proporcionar principios, modelos y disciplinas para manejar personas, procesos y elementos de tecnología con los que se encuentra la mayoría de los proyectos. **(18)**

Características de MSF

- ✓ *Adaptable:* es parecido a un compás, usado en cualquier parte como un mapa, del cual su uso es limitado a un específico lugar.
- ✓ *Escalable:* puede organizar equipos tan pequeños entre 3 o 4 personas, así como también, proyectos que requieren 50 personas o más.
- ✓ *Flexible:* es utilizada en el ambiente de desarrollo de cualquier cliente.

- ✓ *Tecnología Agnóstica:* porque puede ser usada para desarrollar soluciones basadas sobre cualquier tecnología.

MSF se compone de varios modelos encargados de planificar las diferentes partes implicadas en el desarrollo de un proyecto: Modelo de Arquitectura del Proyecto, Modelo de Equipo, Modelo de Proceso, Modelo de Gestión del Riesgo, Modelo de Diseño de Proceso y finalmente el Modelo de Aplicación.

- ✓ *Modelo de Arquitectura del Proyecto:* Diseñado para acortar la planificación del ciclo de vida. Este modelo define las pautas para construir proyectos empresariales a través del lanzamiento de versiones.
- ✓ *Modelo de Equipo:* Este modelo ha sido diseñado para mejorar el rendimiento del equipo de desarrollo. Proporciona una estructura flexible para organizar los equipos de un proyecto. Puede ser escalado dependiendo del tamaño del proyecto y del equipo de personas disponibles.
- ✓ *Modelo de Proceso:* Diseñado para mejorar el control del proyecto, minimizando el riesgo, y aumentar la calidad acortando el tiempo de entrega. Proporciona una estructura de pautas a seguir en el ciclo de vida del proyecto, describiendo las fases, las actividades, la liberación de versiones y explicando su relación con el Modelo de equipo.
- ✓ *Modelo de Gestión del Riesgo:* Diseñado para ayudar al equipo a identificar las prioridades, tomar las decisiones estratégicas correctas y controlar las emergencias que puedan surgir. Este modelo proporciona un entorno estructurado para la toma de decisiones y acciones valorando los riesgos que puedan provocar.
- ✓ *Modelo de Diseño del Proceso:* Diseñado para distinguir entre los objetivos empresariales y las necesidades del usuario. Proporciona un modelo centrado en el usuario para obtener un diseño eficiente y flexible a través de un enfoque iterativo. Las fases de diseño conceptual, lógico y físico proveen tres perspectivas diferentes para los tres tipos de roles: los usuarios, el equipo y los desarrolladores.
- ✓ *Modelo de Aplicación:* Diseñado para mejorar el desarrollo, el mantenimiento y el soporte, proporciona un modelo de tres niveles para diseñar y desarrollar aplicaciones software. Los servicios utilizados en este modelo son escalables, y pueden ser usados en un solo ordenador o incluso en varios servidores.

2.2.4. ¿Por qué RUP?

Una vez estudiado lo referente a estas metodologías de desarrollo expuestas en epígrafes anteriores se determinó, por las características y el alcance de la biblioteca, utilizar RUP debido a que aporta todos los elementos para desarrollar aplicaciones grandes y que requieren de mucha documentación, XP en cuanto a estos aspectos le resta importancia a la documentación y al lenguaje de modelado y

MSF le resta importancia a la selección de la tecnologías que en el presente trabajo tiene prioridad por estar destinado a realizarse en software libre.

2.3. Lenguaje de Modelado: UML

El lenguaje unificado de modelado o notación (UML) sirve para especificar, visualizar y documentar esquemas de sistemas de software orientado a objetos. UML no es un método de desarrollo, lo que significa que no sirve para determinar qué hacer en primer lugar o cómo diseñar el sistema, sino que simplemente le ayuda a visualizar el diseño y a hacerlo más accesible para otros. Es el estándar de descripción de esquemas de software.

UML está diseñado para su uso con software orientado a objetos, y tiene un uso limitado en otro tipo de cuestiones de programación. **(19)**

UML se compone de muchos elementos de esquematización que representan las diferentes partes de un sistema de software. Los elementos UML se utilizan para crear diagramas, que representa alguna parte o puntos de vista del sistema. Posee formas de modelar conceptos como por ejemplo las funciones del sistema, además de otras particularidades como la de escribir clases en un lenguaje determinado, esquemas de bases de datos y componentes de software reusables. Usa procesos de otras metodologías, aprovechando la experiencia de sus creadores.

UML se ha convertido en el lenguaje de modelado más conocido y utilizado en la actualidad; aún cuando todavía no es un estándar oficial, con las siguientes características:

- ✓ Permite modelar sistemas utilizando técnicas orientadas a objetos (OO).
- ✓ Permite especificar todas las decisiones de análisis, diseño e implementación, construyéndose así modelos precisos, no ambiguos y completos.
- ✓ Puede conectarse con lenguajes de programación (Ingeniería directa e inversa).
- ✓ Permite documentar todos los artefactos de un proceso de desarrollo (requisitos, arquitectura, pruebas, versiones, etc.).
- ✓ Cubre las cuestiones relacionadas con el tamaño, propias de los sistemas complejos y críticos.
- ✓ Existe un equilibrio entre expresividad y simplicidad, pues no es difícil de aprender ni de utilizar.
- ✓ UML es independiente del proceso, aunque para utilizarlo óptimamente se debería usar en un proceso que fuese dirigido por los casos de uso, centrado en la arquitectura, iterativo e incremental.

2.4. Herramientas CASE

Las Herramientas CASE son diversas aplicaciones informáticas destinadas a aumentar la productividad en el desarrollo de software reduciendo el coste de las mismas en términos de tiempo y de dinero. Estas herramientas nos pueden ayudar en todos los aspectos del ciclo de vida de desarrollo del software en tareas como el proceso de realizar un diseño del proyecto, cálculo de costes, implementación de parte del código automáticamente con el diseño dado, compilación automática, documentación o detección de errores entre otras.

El concepto de CASE es muy amplio; y una buena definición genérica, que pueda abarcar esa amplitud de conceptos, sería la de considerar a la Ingeniería de Software Asistida por Computación (CASE), como la aplicación de métodos y técnicas a través de las cuales se hacen útiles a las personas, comprender las capacidades de las computadoras, por medio de programas, de procedimientos y su respectiva documentación. **(20)**

2.4.1. Visual Paradigm

Visual Paradigm es un laureado producto que facilita a las organizaciones el diseño visual y el diagrama, integrar y desplegar sus aplicaciones empresariales de misión crítica y sus bases de datos subyacentes. La herramienta de desarrollo de software ayuda a su equipo a sobresalir todo el modelo de construcción y despliegue del software de proceso de desarrollo, y a aumentar al máximo la aceleración de ambos equipos y de los individuos.

Soporta un conjunto de lenguajes, tanto en la generación de código y como de ingeniería inversa sobre Java, C++, CORBA IDL, PHP, XML Schema, Ada y Python. Además, apoya la generación de código C#, VB. NET, código script de Flash, Delphi, Perl, Ruby, entre otros. Ingeniería Inversa también apoya clase Java, .NET, .dll y .exe, JDBC. **(21)**

Licencia: Gratuita y Comercial.

Características:

- ✓ Producto de calidad.
- ✓ Soporta aplicaciones web.
- ✓ Las imágenes y reportes generados, no son de muy buena calidad.
- ✓ Varios idiomas.
- ✓ Generación de código para Java y exportación como HTML.
- ✓ Fácil de instalar y actualizar.
- ✓ Compatibilidad entre ediciones.

Además Visual Paradigm ofrece:

- ✓ Entorno de creación de diagramas para UML 2.0
- ✓ Diseño centrado en casos de uso y enfocado al negocio que generan un software de mayor calidad.
- ✓ Uso de un lenguaje estándar común a todo el equipo de desarrollo que facilita la comunicación.
- ✓ Capacidades de ingeniería directa (versión profesional) e inversa.
- ✓ Modelo y código que permanece sincronizado en todo el ciclo de desarrollo
- ✓ Disponibilidad de múltiples versiones, para cada necesidad.
- ✓ Disponibilidad de integrarse en los principales IDEs.
- ✓ Disponibilidad en múltiples plataformas.

2.4.2. Rational Rose

Rational Rose es la herramienta CASE que comercializan los desarrolladores de UML y que soporta de forma completa la especificación del UML 1.1. Esta herramienta propone la utilización de cuatro tipos de modelo para realizar un diseño del sistema, utilizando una vista estática y otra dinámica de los modelos del sistema, uno lógico y otro físico. Permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software.

Rational Rose utiliza un proceso de desarrollo iterativo controlado, donde el desarrollo se lleva a cabo en una secuencia de iteraciones. Cada iteración comienza con una primera aproximación del análisis, diseño e implementación para identificar los riesgos del diseño, los cuales se utilizan para conducir la iteración, primero se identifican los riesgos y después se prueba la aplicación para que éstos se hagan mínimos.

Cuando la implementación pasa todas las pruebas que se determinan en el proceso, ésta se revisa y se añaden los elementos modificados al modelo de análisis y diseño. Una vez que la actualización del modelo se ha modificado, se realiza la siguiente iteración. **(22)**

Rose permite que haya varias personas trabajando a la vez en el proceso iterativo controlado, para ello posibilita que cada desarrollador opere en un espacio de trabajo privado que contiene el modelo completo y tenga un control exclusivo sobre la propagación de los cambios en ese espacio de trabajo.

También es posible descomponer el modelo en unidades controladas e integrarlas con un sistema para realizar el control de proyectos que permite mantener la integridad de dichas unidades.

Se pueden generar códigos en distintos lenguajes de programación a partir de un diseño en UML.

Rational Rose proporciona mecanismos para realizar la denominada Ingeniería Inversa, es decir, a partir del código de un programa, se puede obtener información sobre su diseño.

Costo de la Licencia: Alto

Cubre todo el ciclo de vida de un proyecto:

- ✓ Concepción y formalización de un proyecto.
- ✓ Construcción de los componentes.
- ✓ Transición a los usuarios y certificación de las distintas fases.

2.4.3. ¿Por qué Visual Paradigm?

Después de realizado un estudio de estas dos herramientas, se determina que la herramienta ideal para el trabajo con la Ingeniería de Software es Visual Paradigm, Rational y Paradigm son herramientas muy potentes para el propósito ingenieril pero esta ultima lleva una ventaja, posee versiones multiplataforma. Visual Paradigm es además completamente compatible con la metodología que se decidió utilizar brinda muchas facilidades en la generación de la documentación del software que se está desarrollando. Todo esto favorece un buen desarrollo del producto por lo cual se obtendrá una mayor calidad en el software.

2.5. Lenguaje de Programación

Las máquinas en general, y las computadoras en particular, necesitan de un lenguaje propio para poder interpretar las instrucciones que se les dan y para que los usuarios puedan controlar su comportamiento. Ese lenguaje que permite esta relación con las computadoras es el lenguaje de programación.

Un lenguaje de programación es aquel elemento dentro de la informática que nos permite crear programas mediante un conjunto de instrucciones, operadores y reglas de sintaxis; que pone a disposición del programador para que este pueda comunicarse con los dispositivos hardware y software existentes además puede ser utilizado para controlar el comportamiento de una máquina, particularmente una computadora.

2.5.1. C++

C++ es un potente lenguaje de programación compilado y orientado a objetos, que permite programar compiladores, sistemas operativos, juegos, procesadores de texto, aplicaciones de base de datos, etc. Es una extensión del lenguaje de programación C.

Abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos.

Las principales características del C++ son las facilidades que proporciona para la programación orientada a objetos y para el uso de plantillas o programación genérica (*templates*). Permite trabajar

tanto a alto como a bajo nivel, sin embargo es a su vez uno de los que menos automatismos trae (obliga a hacerlo casi todo manualmente al igual que C) lo que "dificulta" mucho su aprendizaje. **(23)**

Características del lenguaje de programación C++:

- ✓ Compilado: más eficiente.
- ✓ Memoria automática gestionada por el RTS.
- ✓ Memoria dinámica gestionada por el programador.
- ✓ Métodos virtuales y no virtuales.
- ✓ Permite herencia múltiple.
- ✓ Punteros a void.

Además posee una serie de propiedades difíciles de encontrar en otros lenguajes de alto nivel:

- ✓ Posibilidad de redefinir los operadores (sobrecarga de operadores).
- ✓ Identificación de tipos en tiempo de ejecución (*RTTI*)

Paradigma:	Multiparadigma: orientado a objetos, imperativo, programación genérica.
Tipo de dato:	Fuerte, estático.
Implementaciones:	GNU Compiler Collection, Microsoft Visual C++, Borland C++ Builder, Dev-C++, C-Free
Dialectos:	ISO C++, ANSI C++ 1998, ANSI C++ 2003.
Influido por:	C, Simula.
Ha influido:	Ada, C#, Java, PHP, D, Perl

2.5.2. Java

Un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems.

Posee: Encapsulación, herencia y polimorfismo, fuertemente tipado, gestión automática de la memoria (*recogida de basura*), soporte para concurrencia (multihilo), gestión de excepciones, constructores independientes de la arquitectura del procesador.

Con la programación en Java, se pueden realizar distintos aplicativos, como son applets, que son aplicaciones especiales, que se ejecutan dentro de un navegador al ser cargada una pagina HTML en un servidor WEB, Por lo general los applets son programas pequeños y de propósitos específicos. **(24)**

Otra de las utilidades de la programación en Java es el desarrollo de aplicaciones, que son programas que se ejecutan en forma independiente, es decir con la programación Java, se pueden realizar aplicaciones como un procesador de palabras, una hoja que sirva para cálculos, una aplicación grafica, etc. En resumen cualquier tipo de aplicación se puede realizar con ella. Java permite la modularidad por lo que se pueden hacer rutinas individuales que sean usadas por más de una aplicación, por ejemplo se puede tomar una rutina de impresión que puede servir para el procesador de palabras, como para la hoja de calculo.

La programación en Java, permite el desarrollo de aplicaciones bajo el esquema de Cliente Servidor, como de aplicaciones distribuidas, lo que lo hace capaz de conectar dos o más computadoras u ordenadores, ejecutando tareas simultáneamente, y de esta forma logra distribuir el trabajo a realizar.

Desventajas del lenguaje de programación Java: **(24)**

- ✓ Eficiencia: Java es hasta 30 veces más lento que C++ a causa del tiempo invertido en:
  - o Recogida de basura.
  - o Sincronización de threads.
  - o Otras actividades (carga de clases, comprobación de límites, gestión de excepciones)
- ✓ Todos los métodos son virtuales.
- ✓ No tiene herencia múltiple.

2.5.3. Python

Python es un lenguaje de programación script, interpretado, interactivo y orientado a objetos. Destaca en una sintaxis muy sencilla y limpia pero con gran potencia. Contiene módulos, clases, tipos de datos de alto nivel y escritura dinámica. Tiene interfaces para diversos sistemas y librerías. También puede utilizarse como un lenguaje de extensión para aplicaciones que necesitan una interfaz programable. Otra ventaja es su portabilidad, funcionando en sistemas Unix y derivados, Windows, Dos, Mac y otros. Python fue creado principalmente para realizar páginas HTML. **(25)**

La característica más sobresaliente de Python es su gran legibilidad, con el resultado de que los programas resultantes a veces son denominados pseudocódigo ejecutable. La sintaxis es mínima, y a la vez la librería estándar es amplia y diversa. Python permite programar en diversos estilos o paradigmas: permite la programación imperativa, tiene muchos elementos de la funcional pero, sobre todo, es un lenguaje moderno, orientado a objetos.

Desventajas:

- ✓ Lenguaje Interpretado. Esto trae consigo menor velocidad de procesamiento.
- ✓ Al ser un lenguaje de scripting e interpretado no es adecuado para la programación a bajo nivel (drivers y kernels). No tiene control sobre la memoria.

- ✓ No es adecuado para aplicaciones que requieren de una alta capacidad de cómputo (procesamiento de imágenes).

2.5.4. ¿Por qué C++?

Luego de haber realizado un estudio detallado y profundo de las características que poseen cada uno de estos lenguajes de programación se llega a la conclusión de que el lenguaje de programación con mejores opciones para el desarrollo de la biblioteca es C++, tiene características que lo realzan por encima de los demás lenguajes, además en el caso de Java las aplicaciones resultantes son muchos más lentas que las derivadas de C++ y tienen dependencia de la maquina virtual, por lo que su uso no es el más adecuado a los propósitos finales de la Biblioteca y en el caso de Python al ser un lenguaje interpretado las aplicaciones finales se harían más lentas.

2.6. IDE de Desarrollo

Integrated Development Environment (IDE) ó Entorno de Desarrollo Integrado.

Los IDE no son más que programas compuestos por herramientas que son necesarias al programador a la hora de implementar. Puede ocurrir que el programa se dedique únicamente a un solo lenguaje de programación o a varios.

Es un entorno de programación que ha sido empaquetado como un programa de aplicación, es decir, consiste en un editor de código, un compilador, un depurador y un constructor de interfaz gráfica GUI. Los IDEs pueden ser aplicaciones por si solas o pueden ser parte de aplicaciones existentes.

2.6.1. KDevelop

KDevelop es un libre entorno de desarrollo integrado (IDE). Utiliza la colección de compiladores de GNU, en lugar de aplicar su propio compilador.

KDevelop integra una gran cantidad de herramientas, scripts, y las plantillas en un interfaz de usuario común. Consta de varios modos de interfaces de usuario, una aplicación asistente para varios sistemas de gestión de proyectos, herramientas de edición, distintos navegadores, un depurador de interfaces, varios plugin y un conjunto de otros diagnósticos, documentación y optimización de herramientas de ayuda. **(26)**

KDevelop es un entorno de desarrollo que facilita la creación y el desarrollo de aplicaciones estándar de GNU. La actual versión incluye: **(27)**

- ✓ Los usuarios pueden añadir, reemplazar y eliminar funcionalidad sin cambiar el código fuente básico.

- ✓ Soporta perfiles que determinan los complementos que deben ser cargados. Archivo asistente para la gestión de archivos y la creación de nuevas plantillas de los archivos.
- ✓ Configuración de los perfiles que especifican diferentes compiladores, los parámetros del compilador, crear directorios, arquitectura, etc.
- ✓ Asistente que informa problemas a medida que escribes una clase del navegador que muestra una jerarquía de clases y otros símbolos, y un analizador de idiomas.
- ✓ Soporta 15 diferentes lenguajes de programación.
- ✓ Un editor que destaca la sintaxis.
- ✓ Funciona con cualquier editor que ejecuta KDE KTextEditor.
- ✓ Genera aplicaciones completas de muestra.
- ✓ Plantillas para cada idioma y apoya sistema de construcción.

KDevelop es un entorno de desarrollo integrado para sistemas Linux y otros sistemas Unix, publicado bajo licencia GPL.

Su última versión se encuentra actualmente bajo desarrollo y funciona con distintos lenguajes de programación como C, C++, Java, Ada, SQL, Python, Perl y Pascal, así como guiones para el intérprete de comandos Bash. **(28)**

2.6.2. Eclipse

Eclipse es principalmente una plataforma de programación, usada para crear entornos integrados de desarrollo (IDEs) que puedan ser utilizados para la construcción de aplicaciones web, aplicaciones Java de todo tipo, programas C++, y Enterprise JavaBeans (EJBs).

La Plataforma Eclipse es una herramienta para todo, y para nada en particular. A pesar de que la funcionalidad de Eclipse es mucha, gran parte de la funcionalidad es muy genérica. Permite que nuevos componentes puedan utilizar nuevos tipos de contenido, para realizar nuevas tareas con contenidos existentes. La Plataforma Eclipse permite descubrir, e invocar funcionalidad implementada en componentes llamados plugins. Un fabricante proporciona una herramienta independiente como un plugin que permite llevar a cabo una determinada actividad. Cuando la Plataforma se inicializa, se le mostrarán al usuario además del entorno de Eclipse todos los plugins que estén instalados en el entorno. La calidad de la experiencia del usuario depende de cómo se integren los diferentes plugins con la Plataforma y como aquellos puedan comunicarse entre sí. **(29) (30)**

El principal objetivo de la Plataforma Eclipse es proporcionar mecanismos, reglas que puedan ser, seguidas por los fabricantes para integrar de manera transparente sus herramientas. Mediante APIs, interfaces, clases y métodos, se exponen estos mecanismos. La Plataforma también nos posibilita la construcción de nuevas herramientas que extenderán la funcionalidad de la Plataforma.

La plataforma Eclipse es mejor para la programación Java que para las demás aun cuando los demás lenguajes también sirven.

Si bien Eclipse es multiplataforma, los plugins no tienen por qué serlo. Existen plugins que sólo corren en una plataforma, o que aún no han sido desarrollados para más de una. Al ser una herramienta open – source, se desarrollan plugins que no tienen todas las funcionalidades que tienen en otras herramientas comerciales.

La desventaja al utilizar la ayuda de Eclipse es la pérdida de flexibilidad brindada por la estructura jerárquica que poseen los archivos toc.xml, con lo cual no es posible recorrer la ayuda de la herramienta en forma aleatoria sino que solo se puede contar con los enlaces que ella posee.

2.6.3. Code::Blocks

Code::Blocks es una herramienta de entorno de desarrollo integrado para el desarrollo de programas en lenguaje C++ programado en GCC y basado en la plataforma de interfaces gráficas WxWidgets, lo que le permite que corra libremente en diversos sistemas operativos (actualmente corre en Windows, Linux, Unix).

Cuenta con soporte multicompilador. Debido a que es solo la interfaz del entorno de desarrollo, puede enlazarse a varios compiladores, GCC (MingW / GNU GCC), MSV C++, Digital Mars, Borland C++, Open Watcom. Todos estos compiladores pueden ser detectados automáticamente si ya están instalados al iniciar Code::Blocks. **(31)**

Code::Blocks trae integradas plantillas para generar varias clases de programas, ya sea la clásica aplicación de consola, bibliotecas estáticas o dinámicas, o proyectos completos enlazados con populares bibliotecas como OpenGL y SDL; sin embargo, Code::Blocks integra sólo las plantillas, las bibliotecas deben instalarse por separado.

Code::Blocks tiene muchas facilidades en cuanto a interfaz, entre otras se pueden citar, sintaxis personalizable y extensible, interfaz con pestañas, completamiento de código. Además permite debuggear y es extensible a través de plugins.

2.6.4. ¿Por qué Code::Blocks?

Después de estudiar detalladamente los IDE de Desarrollo expuestos en epígrafes anteriores se llega a la conclusión de que el IDE más óptimo que daría paso a la solución del objetivo general será Code::Blocks. Primeramente KDevelop tiene solamente distribuciones para Linux por lo que la exportación de bibliotecas a plataformas Windows habría que desarrollarlo con otro IDE, Eclipse se acerca más a nuestra solución pero en cuestiones de velocidad Code::Blocks lo supera como entorno

de desarrollo, y lo más importante en nuestro caso es que, Code::Blocks está bajo licencia GPL, el software cubierto bajo esta licencia es software libre.

2.7. Conclusiones Parciales

El presente capítulo recogió el resultado de todo el estudio y la investigación relacionada a las tendencias y tecnologías actuales a desarrollar para la BMN, se describieron las características fundamentales de las metodologías más utilizadas actualmente en el desarrollo de aplicaciones y se analizaron sus ventajas y desventajas, se llegó a la conclusión de que la propuesta consiste en desarrollar la Biblioteca de Métodos Numéricos usando la metodología RUP; UML como lenguaje de modelado; Visual Paradigm como herramienta case, C++ como lenguaje de programación; y como IDE de desarrollo Code::Blocks por ser una herramienta de software libre. Teniendo en cuenta las estructuras de la Universidad de las Ciencias Informática en cuanto las tecnologías que se utilizan, se puede afirmar que la selección realizada es la más óptima para proporcionar una solución eficiente al problema planteado.

CAPÍTULO 3: Presentación de la solución propuesta

3.1. Introducción

En este capítulo se hace una descripción de la solución propuesta a través del modelo de dominio representando los principales conceptos asociados a nuestro trabajo. Se detallan los requisitos funcionales y no funcionales, casos de usos que se generan a partir de los requisitos funcionales y una explicación de cada uno de ellos a través de las descripciones textuales.

3.2. Modelo de Dominio

Un modelo de dominio captura los tipos más importantes de objetos en el contexto del sistema. Los objetos del dominio representan las “cosas” que existen o los eventos que suceden en el entorno en el que trabaja el sistema.

Las clases del dominio aparecen en tres formas típicas:

- ✓ Objetos del negocio que representan cosas que se manipulan en el negocio.
- ✓ Objetos del mundo real y conceptos de los que el sistema debe hacer un seguimiento.
- ✓ Sucesos que ocurrirán o han ocurrido.

El modelo de dominio se describe mediante diagramas UML (especialmente mediante diagrama de clases). Estos diagramas muestran las clases del dominio y cómo se relacionan unas con otras mediante asociaciones. **(15)**

El modelo de dominio ayuda además a los usuarios, clientes, desarrolladores y otros interesados a utilizar un vocabulario común.

3.2.1. ¿Cuándo se aplica un modelo de dominio?

El Modelo de Dominio se aplica cuando:

- ✓ Los flujos de información son difusos (múltiples orígenes, sólo eventos, sucesos).
- ✓ Imposibilidad de determinar subsistemas (exceso de interconexiones).
- ✓ Solapamiento de responsabilidades.
- ✓ Múltiples responsabilidades.
- ✓ Difícil establecimiento de reglas de funcionamiento.

3.2.2. Diagrama de clases del Modelo de Dominio

Luego de un estudio de todo lo anteriormente planteado y en base a esto se llega a la conclusión de que debido a no tener una estructura definida de los procesos del negocio (fronteras bien establecidas, donde se logren ver claramente quiénes son las personas que lo inician, quiénes son los beneficiados, pero además quiénes son las personas que desarrollan las actividades en cada uno de estos procesos), se plantea un modelo de dominio. Se realizará a través de un diagrama de clases de UML e identificando los conceptos representados en el diagrama en un glosario de términos; logrando que todo el interesado adquiera un entendimiento mínimo del contexto en que se emplaza la biblioteca.

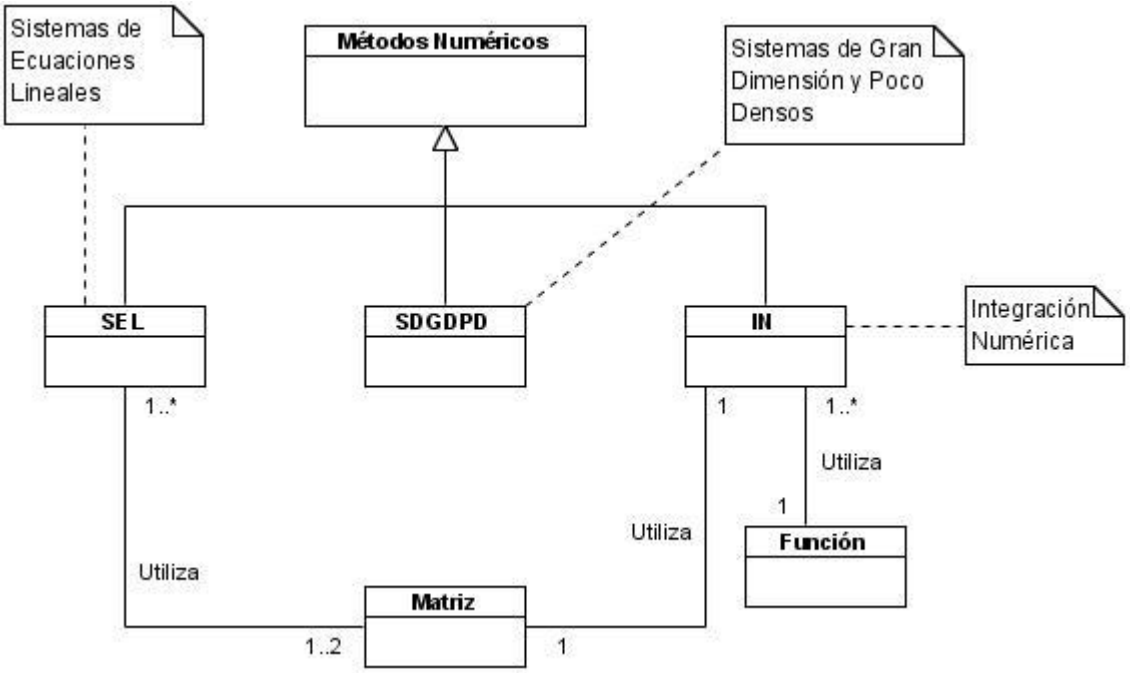


Figura 8: Diagrama de Clases del Dominio

3.2.3. Conceptos principales del Modelo de Dominio

Métodos Numéricos: algoritmos diseñados por la Matemática Numérica para resolver problemas matemáticos.

Sistemas de Ecuaciones Lineales (SEL): concepto que abarca a los métodos numéricos de sistemas de ecuaciones lineales.

Matriz: tabla o arreglo rectangular de números que dispone de estos en forma de filas y columnas.

Sistemas de Gran Dimensión y Poco Densos (SGDPD): concepto que abarca a los métodos numéricos de sistemas de gran dimensión y poco densos.

Integración Numérica (IN): concepto que abarca a los métodos numéricos de integrales numéricas.

Función: concepto que abarca a las funciones matemáticas.

3.3. Requerimientos

A través de los años se ha podido constatar que los requerimientos o requisitos son la pieza fundamental en un proyecto de desarrollo de software, ya que marcan el punto de partida para actividades como la planeación, básicamente en lo que se refiere a las estimaciones de tiempos y costos, así como la definición de recursos necesarios y la elaboración de cronogramas que será uno de los principales mecanismos de control con los que se contará durante la etapa de desarrollo. Además la especificación de requerimientos es la base que permite verificar si se alcanzaron o no los objetivos establecidos en el proyecto ya que estos son un reflejo detallado de las necesidades de los clientes o usuarios del sistema y es contra lo que se va a estar verificando si se están cumpliendo las metas trazadas. Es muy frecuente escuchar entre los conocedores del desarrollo de software (programas de computadoras), que un gran número de los proyectos de software fracasan por no realizar una adecuada definición, especificación, y administración de los requerimientos. Dentro de esa mala administración se pueden encontrar factores como la falta de participación del usuario, requerimientos incompletos y el mal manejo del cambio a los requerimientos.

La Ingeniería de Requerimientos (IR) cumple un papel primordial en el proceso de producción de software, ya que se enfoca un área fundamental: la definición de lo que se desea producir. Su principal tarea consiste en la generación de especificaciones correctas que describan con claridad, sin ambigüedades, en forma consistente y compacta, las necesidades de los usuarios o clientes; de esta manera, se pretende minimizar los problemas relacionados por la mala gestión de los requerimientos en el desarrollo de sistemas.

Los requerimientos pueden dividirse en requerimientos funcionales y requerimientos no funcionales.

- ✓ *Los requerimientos funcionales:* definen las funciones que el sistema será capaz de realizar. Describen las transformaciones que el sistema realiza sobre las entradas para producir salidas.
- ✓ *Los requerimientos no funcionales:* tienen que ver con características que de una u otra forma puedan limitar el sistema, como por ejemplo, el rendimiento (en tiempo y espacio), interfaces de usuario, fiabilidad (robustez del sistema, disponibilidad de equipo), mantenimiento, seguridad, portabilidad, estándares, etc.

3.3.1. Requerimientos Funcionales

Tabla 1: Requisitos Funcionales

Referencia	Requisito Funcional
R 1	Hallar solución de un Sistema de Ecuaciones Lineales.
R 1.1	Hallar solución por el método de Eliminación Gaussiana Pivote Parcial.
R 1.2	Hallar solución por el método de Eliminación Gaussiana Pivote Total.
R 1.3	Hallar solución por el método de Gauss – Jordan.
R 1.4	Hallar solución por el método de Jacobi.
R 1.5	Hallar solución por el método de Gauss – Seidel.
R 2	Hallar solución de un Sistema de Gran Dimensión y Poco Denso.
R 2.1	Hallar solución por el método de Thomas para matrices tridiagonales.
R 2.2	Hallar solución por el método de Gauss para matrices tridiagonales.
R 3	Calcular la integral numérica de una función
R 3.1	Calcular la integral por el método de la Regla Rectangular
R 3.2	Calcular la integral por el método de la Regla de los Trapecios
R 3.3	Calcular la integral por el método de Simpson
R 3.4	Calcular la integral por el método de Romberg

3.3.2. Requerimientos No Funcionales

Requerimiento de soporte

- ✓ Para garantizar el soporte de la biblioteca, la misma deberá tener un documento. Dicho documento deberá abordar cada uno de los métodos numéricos que se implementarán, parámetros que se necesitan para trabajar con el método, ventajas y desventajas del mismo, etc.

Requerimiento de portabilidad

- ✓ La biblioteca deberá poder ser utilizada en Sistemas Operativos Windows (2000, XP, Vista), Linux (Suse, Ubuntu, Debian).

Requerimiento legal

- ✓ La biblioteca se deberá regir por la licencia GNU/GPL.

3.4. Descripción del Sistema. Modelos de Casos de Uso del Sistema

Luego de haber realizado una pequeña descripción del sistema, se da paso a la descripción del modelo de casos de uso del sistema haciendo uso de las ventajas que brinda el lenguaje de modelado UML, se formulan las funcionalidades del sistema y representación mediante un diagrama, para ello es de vital importancia definir los actores y los casos de uso que representarán las responsabilidades del mismo.

3.4.1. Determinación y justificación de los actores del sistema

Un actor es un rol que cumple un usuario, puede intercambiar información o puede ser un recipiente pasivo de información y representa a un ser humano, a un software o a una máquina que interactúa con el sistema. En la biblioteca se ve reflejado los dos casos, la explicación se describe a continuación. **(15)**

Tabla 2: Actor del Sistema y su funcionalidad.

Actores	***Justificación***
Usuario	Representa a la persona (programador) o software que haga uso de la biblioteca.

3.4.2. Casos de Uso del Sistema

Un caso de uso constituye una técnica utilizada para describir el comportamiento del sistema, a través de un documento narrativo que define la secuencia de acciones que obtienen resultados de valor para un actor que utiliza un sistema para completar un proceso, sin importar los detalles de la implementación.

Tabla 3: Prioridad de los Casos de Uso

Caso de Uso del Sistema	***Prioridad***
Hallar solución SEL.	Crítico
Hallar solución SGDPD.	Crítico
Calcular IN de una función.	Crítico

3.4.3. Diagrama de Casos de Uso del Sistema

Los diagramas de casos de uso sirven para especificar la comunicación y el comportamiento de un sistema mediante su interacción con los usuarios y/u otros sistemas. O lo que es igual, un diagrama que muestra la relación entre los actores y los casos de uso en un sistema. Los diagramas de casos de

uso se utilizan para ilustrar los requerimientos del sistema al mostrar como reacciona una respuesta a eventos que se producen en el mismo.

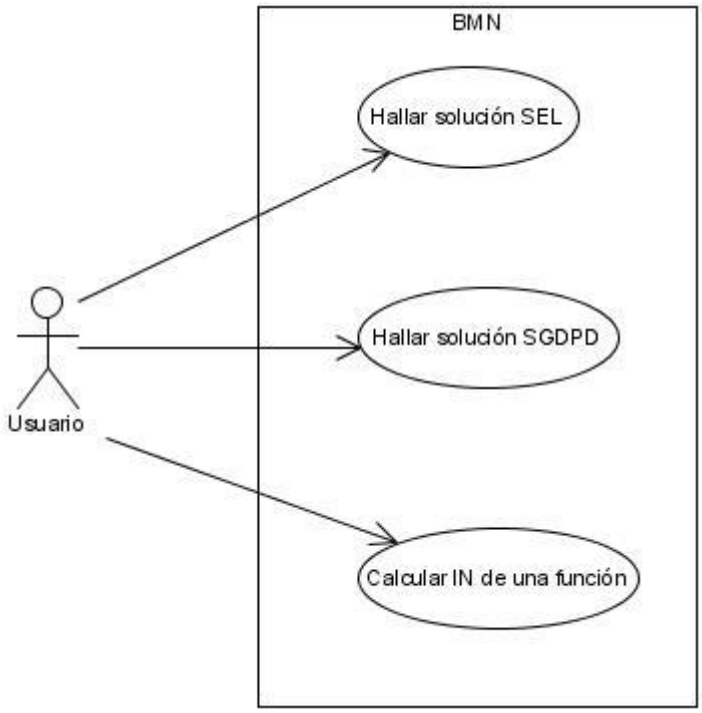


Figura 9: Diagrama de Casos de Uso del Sistema

3.4.4. Expansión de los Casos de Uso

Tabla 4: Descripción Textual Caso de Uso Calcular Solución SEL.

Descripción del Caso de Uso	
Nombre del Caso de Uso	Hallar solución de SEL.
Actor(es)	Usuario.
Propósito	Hallar las soluciones de un Sistema de Ecuaciones Lineales.
Descripción	El caso de uso se inicia cuando el usuario solicita hallar la solución de un Sistema de Ecuaciones Lineales, pasando los datos necesarios según el método escogido. El sistema realiza las operaciones necesarias y devuelve las soluciones.
Referencias	R.F 1 R.F 1.1 R.F 1.2 R.F 1.3 R.F 1.4 R.F 1.5
Pre condiciones	
Pos condiciones	Se hallan las soluciones del sistema.
Flujo de Eventos	

	Actor		**Respuesta del Sistema**
1	El usuario solicita hallar la solución de un Sistema de Ecuaciones Lineales.		
Sección "Métodos Directos"			
2	El usuario escoge un método directo pasando como dato la matriz ampliada del sistema.		
		3	El sistema realiza las operaciones de escalonar la matriz utilizando una estrategia de pivote y realizar el proceso inverso determinando las soluciones que satisfacen el sistema.
		4	El sistema devuelve las soluciones.
Sección "Métodos Iterativos"			
2	El usuario escoge un método iterativo pasando como datos la matriz de los coeficientes, la matriz de términos independientes, la convergencia, la tolerancia y la aproximación inicial.		
		3	El sistema realiza operaciones de hallar soluciones iterativas al sistema hasta que el error calculado en esta iteración sea menor que la tolerancia.
		4	El sistema devuelve estas soluciones.
Flujo Alterno			
		3.1	El sistema detecta datos no válidos.
		3.2	El sistema lanza un mensaje de error.

Tabla 5: Descripción Textual Caso de Uso Calcular Solución SGDPD

Descripción del Caso de Uso	
Nombre del Caso de Uso	Hallar solución de SGDPD.
Actor(es)	Usuario.
Propósito	Hallar las soluciones de un Sistema de Gran Dimensión y Poco Denso.
Descripción	El caso de uso se inicia cuando el usuario solicita hallar la solución de un Sistema de Gran Dimensión y Poco Denso, pasando los datos necesarios según el método escogido. El sistema halla y devuelve las soluciones.
Referencias	R.F 2 R.F 2.1 R.F 2.2
Pre condiciones	

Pos condiciones	Se hallan las soluciones del sistema.		
Flujo de Eventos			
	Actor		**Respuesta del Sistema**
1	El usuario solicita hallar la solución de un Sistema de Gran Dimensión y Poco Denso pasando como datos la diagonal principal y las dos adyacentes.		
		2	El sistema halla las soluciones mediante transformaciones y cálculos con los valores de la diagonal principal y las dos adyacentes.
		3	El sistema devuelve las soluciones.
Flujo Alterno			
		2.1	El sistema detecta errores en los datos.
		2.2	El sistema lanza una excepción.

Tabla 6: Descripción Textual Casos de Uso Calcular Integral Numérica

Descripción del Caso de Uso			
Nombre del Caso de Uso	Calcular I N de una función.		
Actor(es)	Usuario.		
Propósito	Calcular la Integral Numérica de una función.		
Descripción	El caso de uso se inicia cuando el usuario solicita hallar la integral numérica de una función, pasando los datos necesarios según el método escogido. El sistema halla y devuelve el valor de la integral.		
Referencias	R.F 3 R.F 3.1 R.F 3.2 R.F 3.3 R.F 3.4		
Pre condiciones			
Pos condiciones	Queda calculada la integral numérica de la función.		
Flujo de Eventos			
	Actor		**Respuesta del Sistema**
1	El usuario solicita calcular la Integral Numérica.		
Sección "Métodos Rectángulos, Trapecios y Simpson"			
2	El usuario escoge uno de los métodos siguientes: Rectángulo, Trapecios o Simpson.		

3	Pasa como datos la función a la cual se quiere hallar la integral, el limite inferior, el limite superior y la cantidad de sub intervalos con los que desea trabajar		
		4	El sistema divide el intervalo en la cantidad especificada y calcula el valor de la integral con los datos pasados.
		5	El sistema devuelve el valor de la integral.
Sección "Método Romberg"			
2	El usuario escoge trabajar con el método de Romberg.		
3	Pasa como datos la función a la cual se quiere hallar la integral, el límite inferior, el límite superior, la cantidad de sub intervalos y la tolerancia.		
		4	El sistema divide el intervalo en la cantidad de sub intervalos especificados y va calculando el valor de la integral mediante un proceso iterativo hasta que el error sea menor que la tolerancia.
		5	El sistema devuelve el valor de la integral.
Flujo Alterno			
		4.1	El sistema detecta datos no válidos.
		4.2	El sistema lanza una excepción.

3.5. Conclusiones Parciales

En este capítulo se dio paso a desarrollar la propuesta de solución planteada, obteniéndose a partir de un análisis las funcionalidades que debe poseer el sistema, las cuales se representaron mediante un Diagrama de Casos de Uso describiéndose, paso a paso todas las acciones de los actores del sistema con los casos de uso con los que interactúan. Se identificaron los actores. Teniendo en cuenta todas estas características se puede comenzar a construir el sistema poniendo en práctica el cumplimiento de los requisitos tanto funcionales como no funcionales planteados en este capítulo.

CAPÍTULO 4: Construcción de la solución propuesta.

4.1. Introducción

En este capítulo se pone en marcha la construcción de la solución propuesta en los capítulos anteriores. Se describe la arquitectura del sistema, los modelos de diseño e implementación además de las pruebas realizadas. Se definen los modelos de componentes tanto de código fuente como de código ejecutable. Se describen además las pruebas realizadas al producto para verificar que todo lo antes planteado se cumpliera.

En la elaboración de esta biblioteca se realizó un diseño lo más claro posible para su posterior programación. Además se tuvo muy en cuenta la escalabilidad puesto que esta es una primera versión que estará sujeta a cambios para mejorar su eficiencia y posibilitar mayor funcionalidad al usuario final.

4.2. Patrones

A grandes rasgos se puede decir que un patrón es un modelo a seguir para realizar una actividad en específico. Estos surgen con la experiencia de los seres humanos de tratar de lograr ciertos objetivos. Por lo tanto capturan las experiencias existentes y probadas para promover buenas prácticas.

Los patrones:

- ✓ Son una abstracción de “problema – solución”.
- ✓ Se ocupan de problemas recurrentes.
- ✓ Identifican y especifican abstracciones de niveles más altos que componentes o clases individuales.
- ✓ Proporcionan vocabulario y entendimiento común.

4.2.1. Patrones de Diseño

Los patrones de diseño son la base para la búsqueda de soluciones a problemas comunes en el desarrollo de software y otros ámbitos referentes al diseño de interacción o interfaces.

Un patrón de diseño es una solución a un problema de diseño. Para que una solución sea considerada un patrón debe poseer ciertas características. Una de ellas es que debe haber comprobado su efectividad resolviendo problemas similares en ocasiones anteriores. Otra es que debe ser reusable, lo que significa que es aplicable a diferentes problemas de diseño en distintas circunstancias.

Para un mejor diseño de la aplicación se tuvieron en cuenta varios patrones de diseño, una vez aplicados estos patrones los diseños se hicieron más flexibles, modulares y reutilizables.

Para la asignación de responsabilidades se tuvieron en cuenta algunos de los Patrones GRASP. Según Booch y Rambaugh la responsabilidad es un "contrato u obligación de un tipo de clase". Los patrones de asignación de responsabilidades usados para el desarrollo de la aplicación fueron el patrón Experto, el cual plantea que se debe asignar la responsabilidad al experto en información que en este caso sería la clase que cuenta con la información necesaria para cumplir la responsabilidad. También se hizo uso del patrón Creador, el cual plantea la necesidad de asignarle a una clase la responsabilidad de crear una instancia de otra clase siempre y cuando agregue los objetos de la clase, los contenga, registre las instancias de estos objetos y los utilice específicamente. Se utiliza además el patrón de Alta Cohesión que expresa que se debe asignar una responsabilidad de modo que la cohesión siga siendo alta, la cohesión no es más que la medida de cuán relacionadas y enfocadas están las responsabilidades de una clase. **(19)**

Otros patrones de diseño utilizados fueron los GoF o patrones creacionales. Se hace uso del patrón Fábrica Abstracta el cual proporciona una interfaz para la creación de familias relacionadas con los objetos o dependientes sin especificar sus clases concretas. También se utiliza el patrón Fachada el cual proporciona una interfaz unificada para un conjunto de interfaces o subsistemas, por lo tanto se puede afirmar que este patrón define una interfaz de alto nivel que hace más fácil el uso de los subsistemas. **(19)**

4.2.2. Patrones de Arquitectura

4.2.2.1. Arquitectura en Capas

La arquitectura en capas definida según Garlan y Shaw es una organización jerárquica tal que cada capa proporciona servicios a la capa inmediata superior y se sirve de las prestaciones que le brinda la inmediata inferior. **(32)**

Las ventajas del estilo en capas son muchas, primero que nada, el estilo soporta un diseño basado en niveles de abstracción crecientes, lo cual a su vez permite a los programadores la participación de un problema complejo en una secuencia de pasos incrementales. En segundo lugar admite optimizaciones y refinamientos. Proporciona además amplia reutilización, lo que se convierte en uno de los fuertes de esta arquitectura.

4.2.2.2. Arquitectura Orientada a Objetos

Arquitectura cuyos componentes representas objetos, o más bien instancias de los tipos de datos abstractos. David Garlan y Mary Shaw plantean que los objetos representan una clase de componentes que llaman managers, debido que son los responsables de preservar la integridad de su

propia representación. Un rasgo importante de este aspecto es que la representación interna de un objeto no es accesible desde otros objetos. **(32)**

Las características fundamentales de esta arquitectura son:
Los componentes del estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son asimismo las unidades de modelado, diseño e implementación, y los objetos y sus interacciones son el centro de las incumbencias en el diseño de la arquitectura y en la estructura de la aplicación.

Las interfaces están separadas de las implementaciones.

Los objetos interactúan a través de invocaciones de funciones y procedimientos.

4.2.2.3. Arquitectura Final

Para el desarrollo de la arquitectura de la biblioteca se utilizan los dos patrones antes mencionado. En el caso de la Arquitectura en Capas se aplica estructurando el trabajo en dos capas, una de ellas representa la interfaz de comunicación y la otra contiene toda la lógica del negocio. En el caso de la Arquitectura Orientada a Objetos sirve de guía para entender mejor la programación orientada a objetos siendo aplicada en la biblioteca según todos sus conceptos.

4.3. Diagramas de clase del diseño

El modelo de diseño es un modelo de objetos que describe la realización física de los casos de uso centrándose en como los requisitos funcionales y no funcionales, junto con otras restricciones relacionadas con el entorno de implementación, tienen impacto en el sistema a considerar.

Sirve de abstracción de la implementación y es utilizada como entrada fundamental de las actividades de implementación.

En el modelo del diseño, los casos de uso son realizados por las clases del diseño y sus objetos. Esto se representa por colaboraciones en el modelo de diseño y denota la realización de casos de uso del diseño.

Un diagrama de clases muestra un conjunto de clases, interfaces y colaboraciones, así como sus relaciones. Los diagramas de clases se utilizan para modelar la vista de diseño estática de un sistema Principalmente, esto incluye modelar el vocabulario del sistema, modelar las colaboraciones o modelar esquemas. Los diagramas de clases también son la base para los diagramas de componentes. Los diagramas de clases son importantes no sólo para visualizar, especificar y documentar modelos estructurales, sino también para construir sistemas ejecutables, aplicando ingeniería directa e inversa.

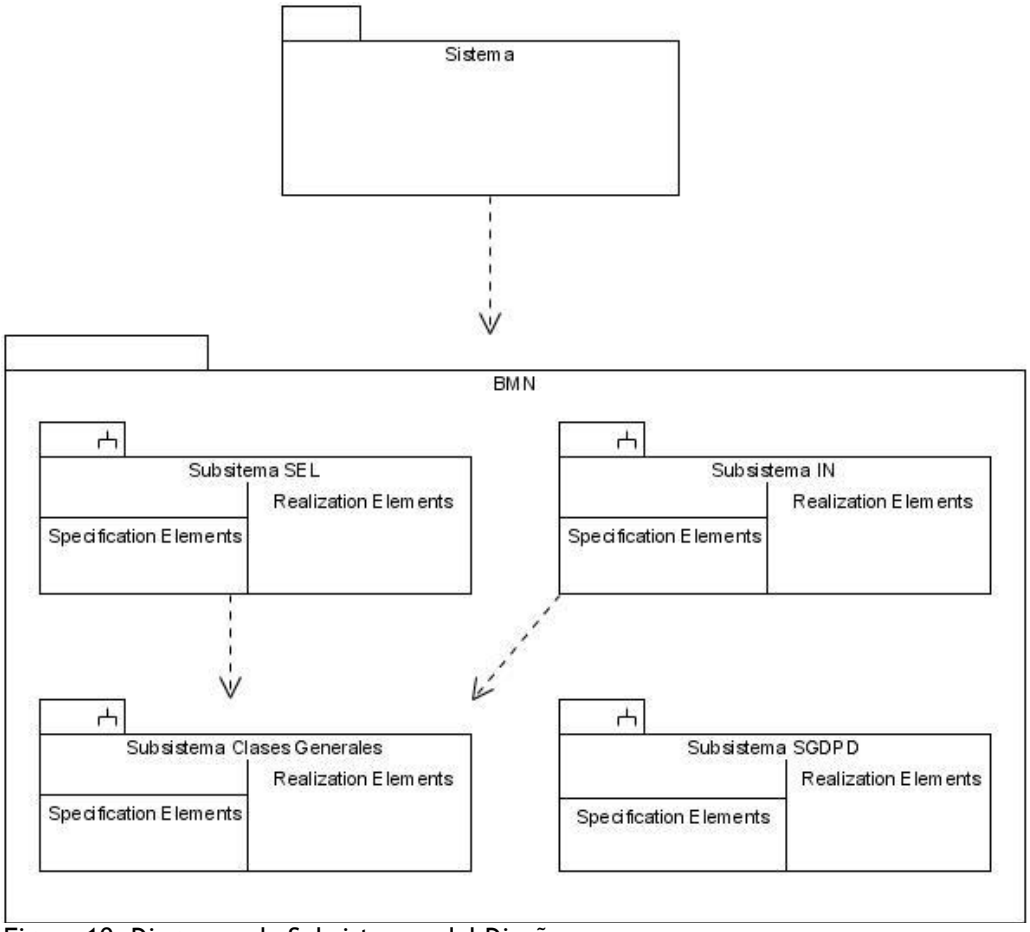


Figura 10: Diagrama de Subsistemas del Diseño

4.3.1. Diagrama de Clases Subsistema de Clases Generales

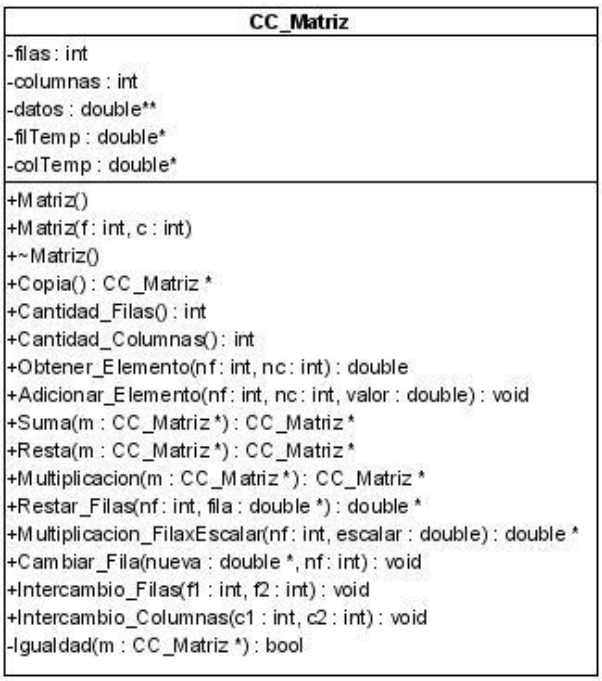


Figura 11: Diagrama de Clases Subsistema de Clases Generales

4.3.2. Diagrama de Clases Subsistema de SEL

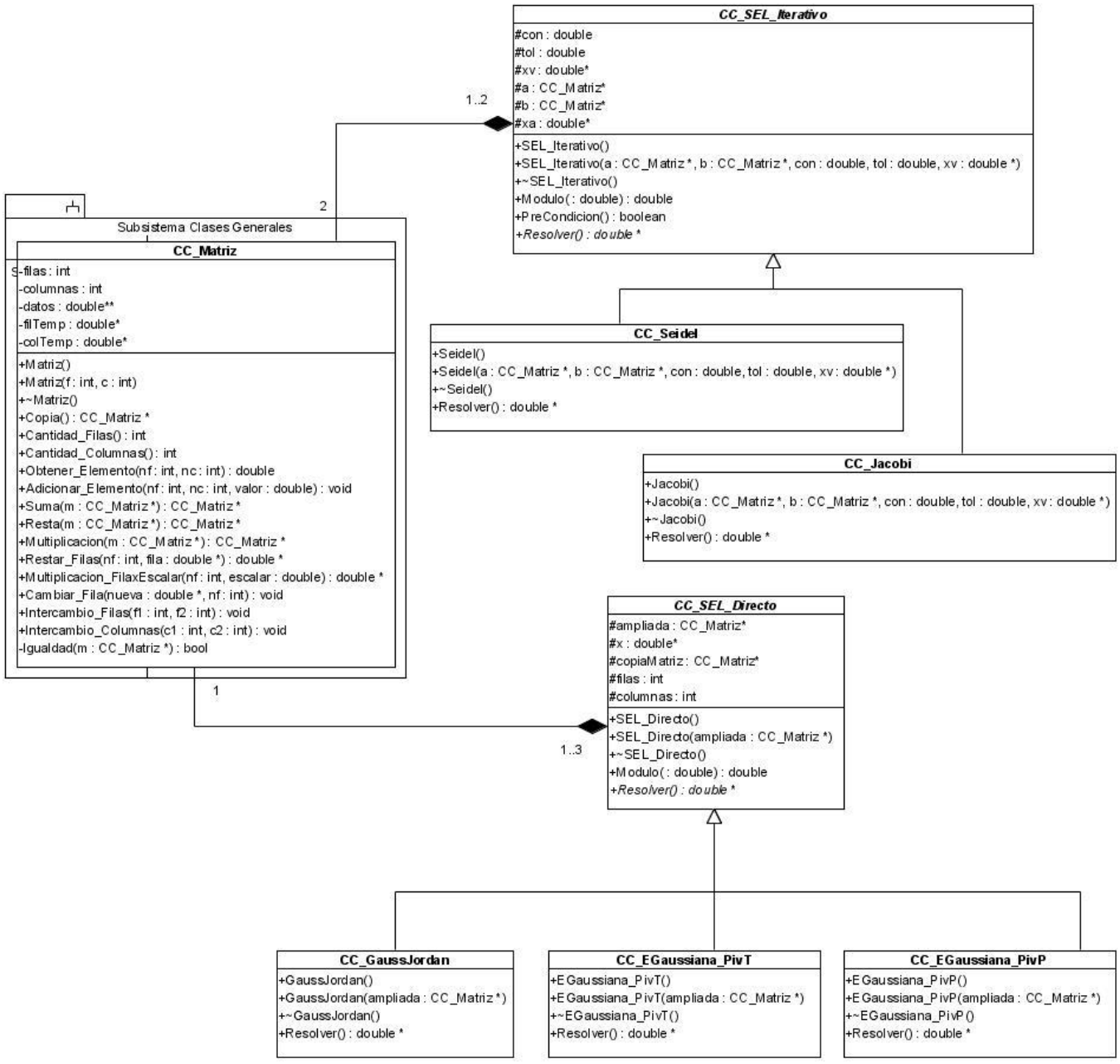


Figura 12: Diagrama de Clases Subsistema de SEL

4.3.3. Diagrama de Clases Subsistema de SGDPD

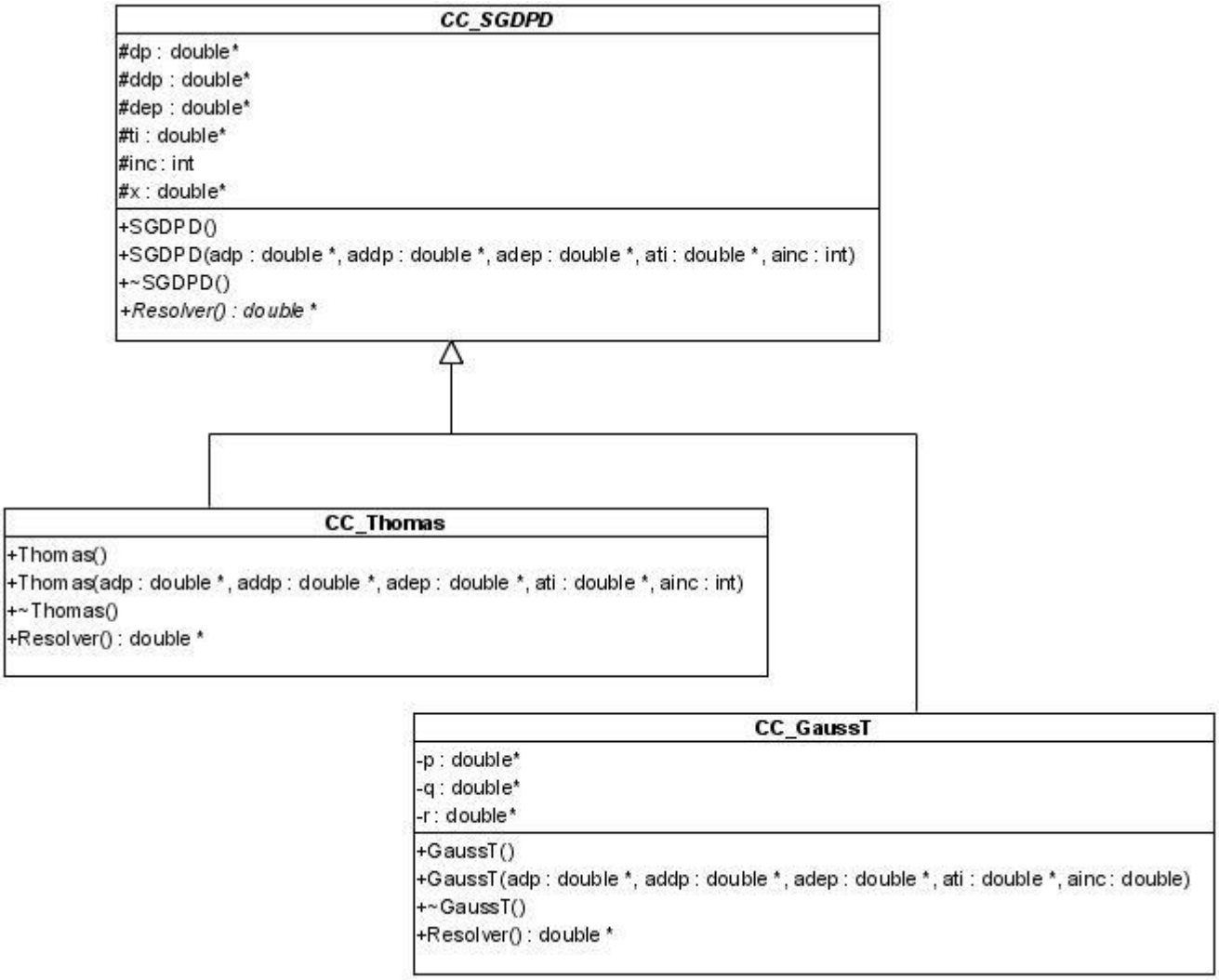


Figura 13: Diagrama de Clases Subsistema de SGDPD

4.3.4. Diagrama de Clases Subsistema de IN

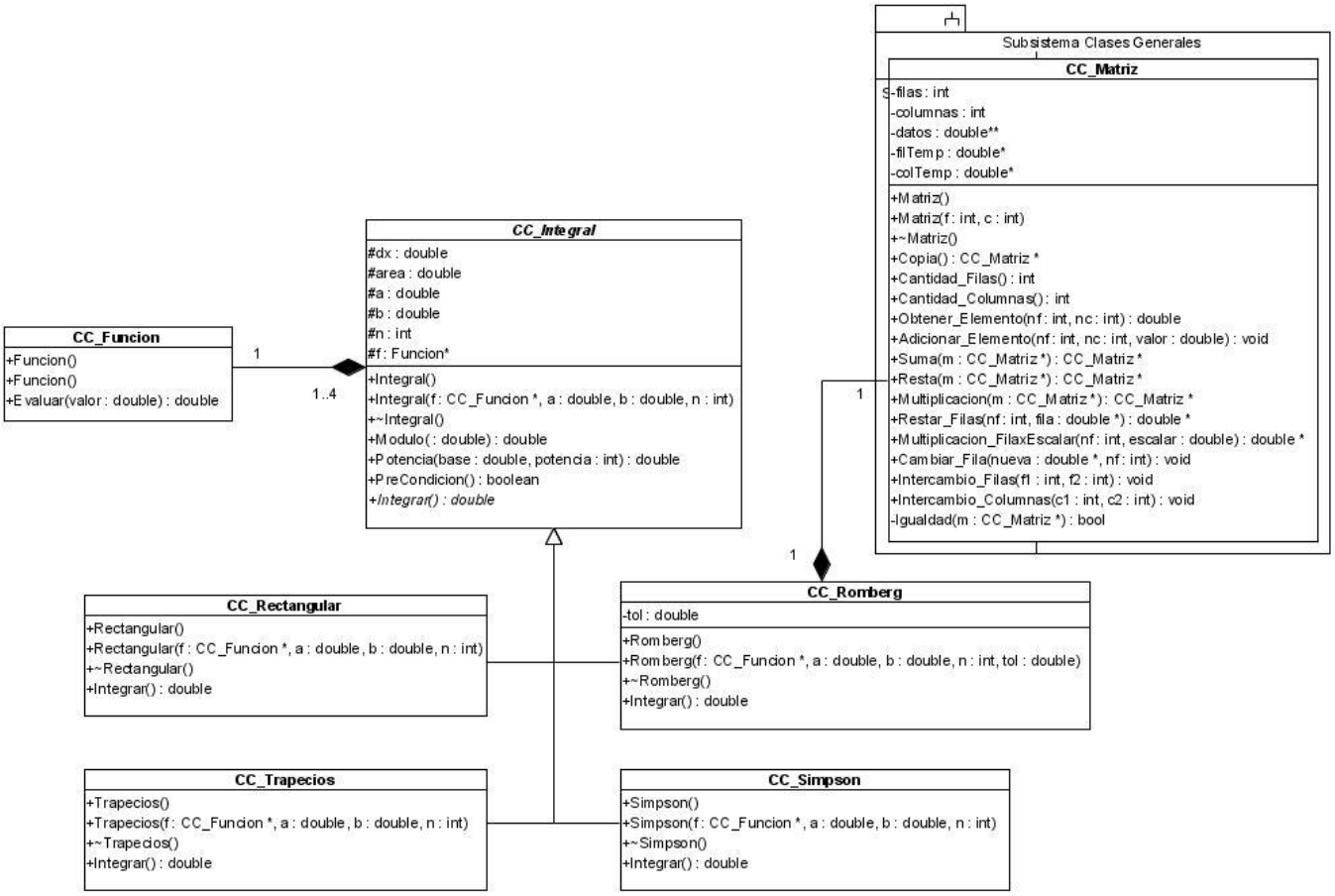


Figura 14: Diagrama de Clases Subsistema de IN

4.4. Diagramas de Interacción

Los diagramas de interacción muestran una interacción y valga la redundancia, que consiste de un conjunto de objetos y sus relaciones, incluyendo los mensajes que puedan ser realizados entre ellos. Son importantes para modelar los aspectos dinámicos de un sistema y para construir sistemas ejecutables a través de ingeniería hacia adelante e ingeniería inversa.

Comúnmente contienen:

- Objetos
- Enlaces
- Mensajes

Pueden servir para visualizar, especificar, construir y documentar los aspectos dinámicos de una sociedad particular de objetos, o pueden ser usados para modelar un flujo particular de control de un caso de uso.

Los diagramas de interacción están conformados por los diagramas de secuencia y los diagramas de colaboración.

4.4.1 Diagramas de Secuencia

Los diagramas de secuencia **(Anexo 2)** destacan la ordenación temporal de los mensajes. Un diagrama de secuencia se forma colocando en primer lugar los objetos que participan en la interacción en la parte superior del diagrama, a lo largo del eje X. Normalmente, se coloca a la izquierda el objeto que inicia la interacción, y los objetos subordinados a la derecha. A continuación, se colocan los mensajes que estos objetos envían y reciben a lo largo del eje Y, en orden de sucesión en el tiempo, desde arriba hasta abajo. Esto ofrece al lector una señal visual clara del flujo de control a lo largo del tiempo.

Los diagramas de secuencia tienen dos características que los distinguen de los diagramas de colaboración. En primer lugar, está la línea de vida. La línea de vida de un objeto es la línea discontinua vertical que representa la existencia de un objeto a lo largo de un período de tiempo. La mayoría de los objetos que aparecen en un diagrama de interacción existirán mientras dure la interacción, así que los objetos se colocan en la parte superior del diagrama, con sus líneas de vida dibujadas desde arriba hasta abajo. Pueden crearse objetos durante la interacción. Sus líneas de vida comienzan con la recepción del mensaje estereotipado como create. Los objetos pueden destruirse durante la interacción. Sus líneas de vida acaban con la recepción del mensaje estereotipado como destroy (además se muestra la señal visual de una gran x que marca el final de sus vidas).

En segundo lugar, está el foco de control. El foco de control es un rectángulo delgado y estrecho que representa el período de tiempo durante el cual un objeto ejecuta una acción, bien sea directamente o a través de un procedimiento subordinado. La parte superior del rectángulo se alinea con el comienzo de la acción; la inferior se alinea con su terminación (y puede marcarse con un mensaje de retorno).

4.4.2 Diagramas de Colaboración

Los diagramas de colaboración **(Anexo 3)** destacan la organización de los objetos que participan en una interacción. Un diagrama de colaboración se construye colocando en primer lugar los objetos que participan en la colaboración como nodos del grafo. A continuación se representan los enlaces que conectan esos objetos como arcos del grafo. Por último, estos enlaces se adornan con los mensajes que envían y reciben los objetos.

Los diagramas de colaboración tienen dos características que los distinguen de los diagramas de secuencia.

En primer lugar, el camino. Para indicar cómo se enlaza un objeto a otro, se puede asociar un estereotipo de camino al extremo más lejano de un enlace (como «local», que indica que el objeto designado es local al emisor). Normalmente, sólo se necesita representar explícitamente el camino del enlace para los caminos local, parameter, global y self (pero no association).

En segundo lugar, está el número de secuencia. Para indicar la ordenación temporal de un mensaje, se precede de un número (comenzando con el mensaje número 1), que se incrementa secuencialmente por cada nuevo mensaje en el flujo de control (2, 3, etc.).

4.5. Modelo de Implementación

En el modelo de implementación se empieza con los resultados obtenidos en el diseño y se crean los componentes físicos de la aplicación que se traducen en ficheros de código fuente .cpp y .h debido a su implementación en el lenguaje C++.

4.5.1. Diagramas de Componentes

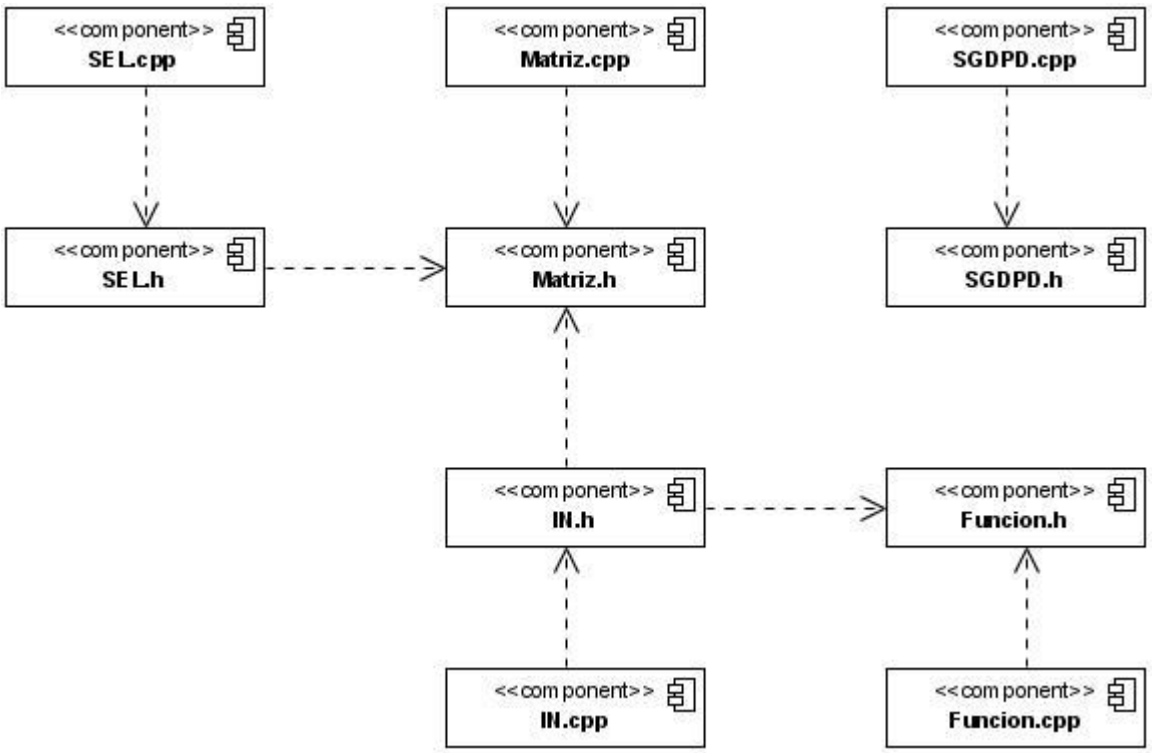


Figura 15: Diagrama de Componentes de Código Fuente

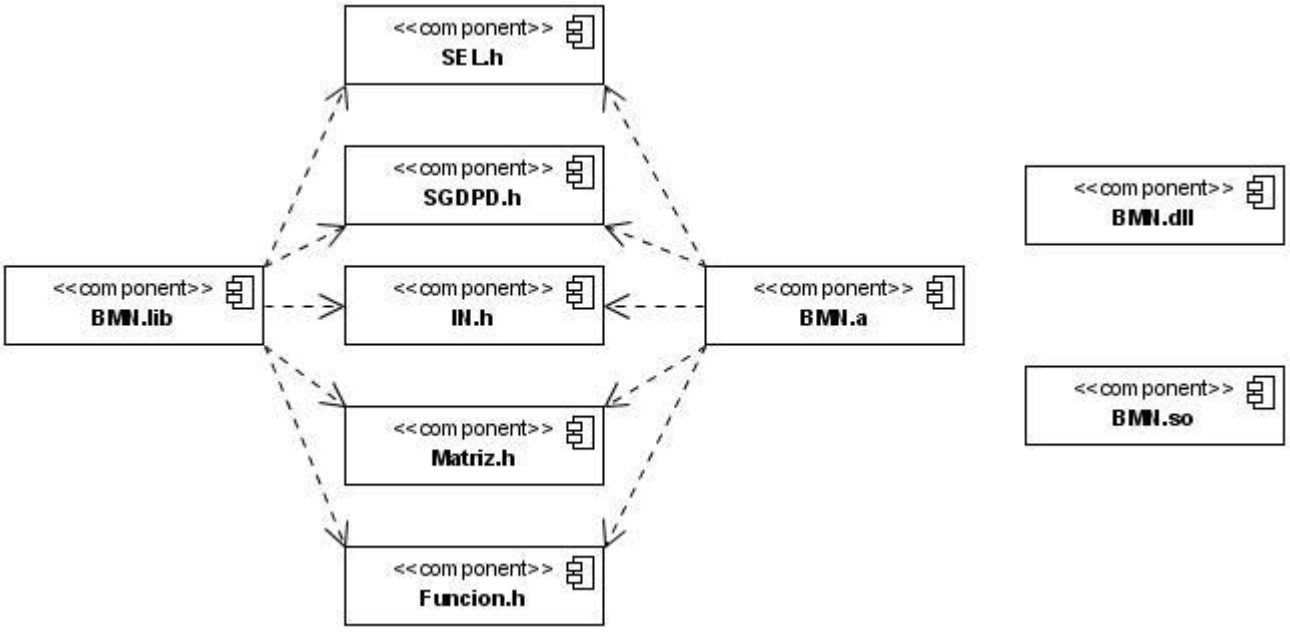


Figura 16: Diagrama de Componentes de Código Ejecutable

4.6. Prueba del Sistema Propuesto

Tabla 7: Caso de Prueba Hallar solución de un SEL

Caso de Uso	Hallar solución de SEL.
Caso de Prueba	Hallar solución de un Sistema de Ecuaciones Lineales.
Entrada (Eliminación Gaussiana Pivote Parcial)	Matriz Ampliada del Sistema: $\begin{bmatrix} 15 & -5 & 0 & 20 \\ -5 & 15 & -5 & 0 \\ 0 & -5 & 20 & 0 \end{bmatrix}$
Salida	Solución del Sistema $x_1 = 1.51724$ $x_2 = 0.551724$ $x_3 = 0.137931$
Entrada (Eliminación Gaussiana Pivote Total)	Matriz Ampliada del Sistema: $\begin{bmatrix} 15 & -5 & 0 & 20 \\ -5 & 15 & -5 & 0 \\ 0 & -5 & 20 & 0 \end{bmatrix}$
Salida	Solución del Sistema $x_1 = 1.51724$ $x_2 = 0.551724$ $x_3 = 0.137931$

Entrada (Gauss – Jordan)	Matriz Ampliada del Sistema: $\begin{bmatrix} 15 & -5 & 0 & 20 \\ -5 & 15 & -5 & 0 \\ 0 & -5 & 20 & 0 \end{bmatrix}$
Salida	Solución del Sistema $x_1 = 1.51724$ $x_2 = 0.551724$ $x_3 = 0.137931$
Entrada (Jacobi)	Matriz $\begin{bmatrix} 10 & -1 & 2 \\ 1 & -5 & 1 \\ 2 & -1 & 8 \end{bmatrix}$ Matriz de términos independientes. $\begin{bmatrix} 6 \\ -10 \\ 8 \end{bmatrix}$ $convergencia = 0.4$ $tolerancia = 0.021$ $Aproximación\ inicial$: $[0 \quad 0 \quad 0]$
Salida	$x_1 = 0.99734$ $x_2 = 2.0027$ $x_3 = -1.00195$
Entrada (Seidel)	Matriz $\begin{bmatrix} 10 & -1 & 2 \\ 1 & -5 & 1 \\ 2 & -1 & 8 \end{bmatrix}$ Matriz de términos independientes. $\begin{bmatrix} 6 \\ -10 \\ 8 \end{bmatrix}$ $convergencia = 0.4$ $tolerancia = 0.021$ $Aproximación\ inicial$: $[0 \quad 0 \quad 0]$
Salida	$x_1 = 1.00101$ $x_2 = 2.00127$ $x_3 = -1.00009$

Tabla 8: Caso de prueba Hallar solución de un SGDPD

Caso de Uso	Hallar solución de SGDPD.
Caso de Prueba	Hallar solución de un Sistema de Gran Dimensión y Poco Denso.
Entrada (Matrices Tridiagonales por el método de Gauss)	Diagonal Principal $[2, 5, 8, 4]$ Diagonal Encima Principal $[1, 2, 3]$ Diagonal Debajo Principal $[2, 2, 1]$ Términos Independientes $[5, 18, 24, 17]$ Incógnitas 4
Salida	$x_1 = 3,\ x_2 = 2,\ x_3 = 1,\ x_4 = 4$
Entrada (Matrices Tridiagonales por el método de Thomas)	Diagonal Principal $[2, 5, 8, 4]$ Diagonal Encima Principal $[1, 2, 3]$ Diagonal Debajo Principal $[2, 2, 1]$ Términos Independientes $[5, 18, 24, 17]$ Incógnitas 4
Salida	$x_1 = 3,\ x_2 = 2,\ x_3 = 1,\ x_4 = 4$

Tabla 9: Caso de Prueba Calcular la IN de una función.

Caso de Uso	Calcular IN de una función.
Caso de Prueba	Calcular la integral numérica de una función.
Entrada (Regla Rectangular)	Límite inferior : 1 Límite superior: 3 Número de particiones: 300 Función: $\frac{e^x}{x}$
Salida	8.03818
Entrada (Regla de los Trapecios)	Límite inferior : 1 Límite superior: 3 Número de particiones: 300 Función: $\frac{e^x}{x}$
Salida	8.03821

Entrada (Regla de Simpson)	Límite inferior : 1 Límite superior: 3 Número de particiones: 300 Función: $\frac{e^x}{x}$
Salida	8.00844
Entrada (Método de Romberg)	Límite inferior : 1 Límite superior: 3 Número de particiones: 300 Función: $\frac{e^x}{x}$ Tolerancia: 0.00008
Salida	8.03819

4.7. Conclusiones Parciales

En el presente capítulo se mostró el diseño detallado de la aplicación completa, así como la arquitectura de la misma basada en patrones de arquitectura y diseño. Además se presenta como queda el sistema expresado en componentes de implementación y las pruebas realizadas al mismo las que demostraron que la aplicación cumple con las funcionalidades previstas.

CAPÍTULO 5: Estudio de Factibilidad.

5.1. Introducción

En el siguiente capitulo se realiza un calculo de los costos asociados a nuestro trabajo, se describen los beneficios tangibles e intangibles, la planificación, la gestión de proyectos, el tipo de estimación realizada en el trabajo por punto de casos de uso.

5.2. Planificación

Para llevar a cabo la realización de cualquier proyecto se hace necesario la planificación continua de las tareas y las actividades, para luego organizarlas en orden de prioridad. Además se necesita diseñar planes estratégicos, se definen y establecen los objetivos y se escoge el medio más apropiado para el logro de los mismos antes de emprender la acción. La planificación muestra la serie de consecuencias, causas y efectos que acarrea la puesta en marcha del desarrollo del proyecto.

Se encontraron dos grandes fases en las que la planificación cobra el máximo protagonismo:

- ✓ La primera es necesaria para estudiar y establecer la viabilidad de un proyecto, ya sea interno o externo a la organización. Hay que hacer los correspondientes estudios técnicos, de mercado, financieros, de rentabilidad... así como una estimación de los recursos necesarios y los costes generados. Todo ello constituye el elemento fundamental en el que se apoya el cliente (que puede ser la propia organización en el caso de proyectos internos) para decidir sobre la realización o no del proyecto.
- ✓ La segunda fase importante de planificación tiene lugar una vez se ha decidido ejecutar el proyecto. Ahora es el momento de realizar una planificación detallada punto por punto. Uno de los errores más importantes y graves en gestión de proyectos es querer arrancar con excesiva premura la obra, sin haber prestado la atención debida a una serie de tareas previas de preparación, organización y planificación que son imprescindibles para garantizar la calidad de la gestión y el éxito posterior.

Es por ello que la planificación constituye un paso imprescindible para alcanzar la máxima calidad en el desarrollo de un proyecto.

5.2.1. Objetivos de la Planificación

Un principio básico en la gestión de proyectos, así como en toda actividad de gestión, es que los objetivos estén definidos a priori y con un grado suficiente de claridad y precisión. Hay proyectos

donde la definición de objetivos se hace realmente difícil, pero esa dificultad no significa que no deba hacerse, puesto que cuanto más inmaterial es o más arriesgado sea un proyecto más necesario será contar con un marco de referencia, aunque sus contornos sean menos nítidos que en otras ocasiones. El objetivo fundamental de la planificación es, *la realización del procesamiento de la información que conduzca a estimaciones razonables*, para ello se necesita de:

- ✓ Experiencia y sentido común alcanzado en proyectos anteriores.
- ✓ Diferentes técnicas de estimación para estimar los costos, esfuerzos y tiempo necesarios para obtener el software.

Existen 3 variantes para la estimación:

1. Análisis de Puntos de Casos de Uso.
2. Análisis de Puntos de Función y COCOMO II.
3. Las estimaciones por líneas de código (LDC) .

Se alega que no es factible realizar la estimación para la BMN con las variantes de Análisis de puntos de función y COCOMO II a pesar de ser técnicas eficientes porque no se presentan entradas externas, salidas externas, peticiones, ficheros internos, entre otras, que son aspectos relacionados a la parte de puntos de función.
Es por esto que se considera que los más eficaz sería utilizar para este trabajo la variante de Análisis de Puntos de Casos de Uso el cuál es un método de estimación del tiempo de desarrollo mediante la asignación de pesos lo que garantiza la posibilidad de predecir el tamaño del sistema a partir de sus características y requisitos.

5.3. Estimación de los Costos

Cálculo de Puntos de Casos de Uso sin ajustar:

$$\mathrm{UUCP} = \mathrm{UAW} + \mathrm{UUCW}$$

- UUCP: Puntos de Casos de Uso sin ajustar
- UAW: Factor de Peso de los Actores sin ajustar
- UUCW: Factor de Peso de los Casos de Uso sin ajustar

Factor de Peso de los Actores sin ajustar (UAW)

Tabla 10: Factor de peso de los actores sin ajustar

Tipo Actor	Descripción	Factor	#Actores	Resultados
Simple	Sistema con sistema a través de interfaz de programación.	1	1	$1 \cdot 1 = 1$
Promedio	Sistema con sistema mediante protocolo de interfaz basada en texto.	2	0	$2 \cdot 0 = 0$
Complejo	Persona que interactúa con el sistema mediante interfaz gráfica.	3	0	$3 \cdot 0 = 0$

Total: 1

$\text{UAW} = \sum(\text{factor} \cdot \#\text{actores}) = 1.$

Factor de Peso de los Casos de Uso sin ajustar (UUCW)

Tabla 11: Factor de peso de los casos de uso sin ajustar

Tipo Actor	Descripción	Factor	#Casos de Uso	Resultados
Simple	$1-3$ Transacciones	5	3	$5 \cdot 3 = 15$
Promedio	$4-7$ Transacciones	10	0	$10 \cdot 0 = 0$
Complejo	$+8$ Transacciones	15	0	$15 \cdot 0 = 0$

Total: 15

$\text{UUCW} = \sum(\text{factor} \cdot \#\text{Casos de Uso}) = 15$

Puntos de Casos de Uso sin ajustar

$\text{UUCP} = \text{UAW} + \text{UUCW}$

$\text{UUCP} = 1 + 15 = 16$

$\text{UUCP} = 16$

Cálculo de Puntos de Casos de Uso ajustados

$\text{UCP} = \text{UUCP} \cdot \text{TCF} \cdot \text{EF}$

- UCP: Puntos de Casos de Uso ajustados.
- UUCP: Puntos de Casos de Uso sin ajustar.
- TCF: Factor de complejidad técnica.
- EF: Factor de ambiente

Factor de Complejidad Técnica (TCF)

Tabla 12: Factor de Complejidad Técnica

Factor	Descripción	Peso	Valor Asignado	Total	Comentario
T1	Sistema distribuido.	2	0	0	El sistema es centralizado.
T2	Objetivo de performance o tiempo de respuesta.	1	5	5	El tiempo de respuesta tiene que ser lo más corto posible.
T3	Eficiencia del usuario final.	1	1	1	Escasas restricciones de eficiencia.
T4	Funcionamiento Interno complejo.	1	5	5	Existencia de cálculos complejos.
T5	El código debe ser reutilizable.	1	5	5	Se requiere que el código sea reutilizable.
T6	Facilidad de instalación.	0,5	5	2,5	No requiere de instalación.
T7	Facilidad de uso.	0,5	4	2,0	Debe ser de fácil uso para los usuarios.
T8	Portabilidad.	2	5	10	El sistema debe ser portable
T9	Facilidad de cambio.	1	4	4	El sistema deberá ser reutilizable y fácil de modificar.
T10	Concurrencia.	1	0	0	No existe concurrencia.
T11	Incluye objetivos especiales de seguridad.	1	0	0	No existen objetivos especiales de seguridad.
T12	Provee acceso directo a terceras partes.	1	0	0	No existe acceso a terceras partes.
T13	Se requieren facilidades especiales de entrenamiento de usuarios.	1	0	0	El sistema posee facilidades de uso.
			Total:	34.5	

$$TCF = 0{,}6 + 0{,}01 \cdot \sum(\text{Peso i} \cdot \text{Valor i})$$

$$TCF = 0{,}6 + 0{,}01 \cdot (34{,}5)$$

$$TCF = 0{,}945 \approx 0{,}95.$$

Cálculo del Factor de ambiente (EF)

Tabla 13: Factor de Ambiente

Factor	Descripción	Peso	Valor Asignado	Total	Comentario
E1	Familiaridad con el modelo de proyecto utilizado.	1,5	4	6	Existe familiaridad con el modelo del proyecto ya que no es algo nuevo.
E2	Experiencia en la aplicación.	0,5	3	1,5	Se posee alguna experiencia ya que se viene trabajando en algunos métodos.
E3	Experiencia en orientación a objetos.	1	3	3	Hace 5 años que se trabaja con esta filosofía.
E4	Capacidad del analista líder.	0,5	3	1,5	Se podría decir que es la primera vez que lleva a la práctica sus conocimientos de IS.
E5	Motivación.	1	5	5	Se posee bastante motivación.
E6	Estabilidad de requerimientos.	2	4	8	Se mantienen bastante inalterables.
E7	Personal Part–Time.	−1	3	−3	Se trabajan algunas horas al día.
E8	Dificultad del lenguaje de programación.	−1	2	−2	Lenguaje que no es por lo general difícil de utilizar ya que se ha trabajado bastante con él.
			Total:	20	

$$EF = 1{,}4 - 0.03 \cdot (\text{Peso i} \cdot \text{Valor i})$$

$$EF = 1{,}4 - 0.03 \cdot (20)$$

$$EF = 0{,}8.$$

Estimación del Esfuerzo

$$UCP = UUCP \cdot TCF \cdot EF$$

$$UCP = 16 \cdot 0{,}95 \cdot 0{,}8$$

$$UCP = 12{,}16.$$

Llevar de Puntos de Casos de Uso a Esfuerzo (E)

$E = UCP \cdot CF$

CF: Factor de Conversión

Se contabilizan cuantos factores de los que afectan al Factor de ambiente están por debajo del valor medio (3), para los factores E1 a E6.

Se contabilizan cuántos factores de los que afectan al Factor de ambiente están por encima del valor medio (3), para los factores E7 y E8.

Si el total es 2 o menos, se utiliza el factor de conversión 20 horas-hombre/Punto de Casos de Uso, es decir, un Punto de Caso de Uso toma 20 horas-hombre.

Si el total es 3 o 4, se utiliza el factor de conversión 28 horas-hombre/Punto de Casos de Uso, es decir, un Punto de Caso de Uso toma 28 horas-hombre.

Si el total es mayor o igual que 5, se recomienda efectuar cambios en el proyecto, ya que se considera que el riesgo de fracaso del mismo es demasiado alto.

Se contabilizaron cuantos factores de los que afectan al Factor de ambiente están por debajo del valor medio (3), para los factores E1 a E6 y el total es 0, por lo que se utiliza el factor de conversión 20 horas-hombre/Punto de Casos de Uso, es decir, un Punto de Caso de Uso toma 20 horas-hombre.

$CF = 20$

$E = UCP \cdot CF$

$E = 12{,}16 \cdot 20$

$E = 243{,}2\ \text{Horas/Hombre.}$

Tabla 14: Distribución del esfuerzo por actividades

Actividades	Porcentaje (%)	Horas – Hombre.
Análisis	10	60,8
Diseño	20	121,6
Programación	40	243,2
Pruebas	15	91,2
Sobrecarga	15	91,2
Total:	100	608

Como el valor del esfuerzo calculado representa el esfuerzo del FT implementación, por comparación y regla de tres salen el resto de los esfuerzo y la suma de ellos es el esfuerzo total (E_T).

Suponiendo que una persona labora a un ritmo de 8 horas por día, como promedio un mes posee 30 días y en un mes existen 24 días laborables y 6 días no laborables, por lo que la cantidad de horas que puede trabajar una persona en un mes implica 192 horas.

Si $E_T = 608\ horas/hombre$ para un mes de trabajo que son 192 horas, el proyecto necesitaría un $E_T = 3{,}16\ mes/hombre$.

En el caso de la BMN que se propone, el equipo de desarrollo de software es de 2 personas, entonces el problema analizado puede terminarse en aproximadamente 2 meses ($1{,}58\ meses$).

Costo del Proyecto

$\mathrm{C(total)} = \mathrm{E(total)x2Th(media)}$

C: Costo

Th: Tarifa Horaria $= \$225.00$

$\mathrm{Th\ (media)}$: Th/horas total

$$\mathrm{Th(media)} = \frac{225}{192} = 1.17$$

$$\mathrm{C(total)} = 608 * 2(1.17)$$

$$\mathrm{C(total)} = 608 * 2.34$$

$$\mathrm{C(total)} = 1422.72$$

5.4. Beneficios

5.4.1. Beneficios tangibles

Con el desarrollo de la BMN se obtienen beneficios tangibles ya que se obtiene una solución que apoyará el desarrollo de aplicaciones que necesiten de métodos numéricos relacionados con Sistemas de Ecuaciones Lineales, Sistemas de Gran Dimensión y Poco Densos e Integración Numérica. Todo el tiempo que se debería emplear en estudiar e implementar el método se revierte en tiempo que se dedica al desarrollo de la aplicación.

Trae además consigo beneficios económicos ya que no haría falta ninguna herramienta adicional para desarrollar una aplicación que requiera de estos métodos, ni de una gran cantidad de profesionales especializados en matemática.

5.4.2. Beneficios Intangibles

Aumentaría la rapidez y terminación de los productos que dependan de la Biblioteca.

Aumenta el interés por parte de la comunidad de programadores, una vez palpadas las ventajas de utilizar la biblioteca, de mejorar considerablemente la misma y agregarles más prestaciones haciendo su uso cada vez mayor.

Apoya la idea de la informatización de la sociedad cubana ya que consiste en un producto nacional.

5.5. Análisis de costos y beneficios

Analizando los datos calculados anteriormente se llega a la conclusión de que la aplicación no requiere de grandes gastos ni recursos (solo dos PC), no requiere además de mucho tiempo ya que en un plazo de dos meses a un costo de 1422.72 pesos la aplicación estaría terminada por lo que comparando esto con los beneficios que aporta para el desarrollo de futuras aplicaciones se vuelve una solución factible.

5.6. Conclusiones Parciales

En este capítulo se trata el tema de la factibilidad y para ello se ha hecho un estudio detallado de los costos, recursos humanos y tiempo de desarrollo además de los beneficios que aportaría el producto. Se dan argumentos para hacer cierta la afirmación de que es “factible la realización la BMN”.

Conclusiones Generales

En este punto se consideran cumplidos los objetivos trazados al tener desarrollada la Biblioteca de Métodos Numéricos. La misma cumple todos los requisitos planteados para su desarrollo por lo que se convierte en una herramienta cuyas funcionalidades básicas muestran los resultados esperados.

Se creó además una documentación relacionada con la Biblioteca la cual ayudará a los usuarios finales en su uso, en la misma se da una explicación de cada método numérico implementado, además de ventajas y desventajas del mismo, los parámetros de entrada y salida y un pseudocódigo del método en cuestión.

Este trabajo representa un aporte importante ya que cualquier usuario que desarrolle una aplicación en la cual se requiera de métodos numéricos referentes a Sistemas de Ecuaciones Lineales, Sistemas de Gran Dimensión y Poco Densos e Integración Numérica, puede hacer uso de la Biblioteca lo cual representa tiempo ganado el que debería emplear para estudiarlo e implementarlo, además de que evita la dependencia de software privativos que realicen estas operaciones.

Con los beneficios presentados y datos palpables en cuanto a costo, cantidad de hombres, beneficios tangibles e intangibles, se determinó que el desarrollo de la aplicación es realmente factible y que constituye un primer paso en futuras automatizaciones de procesos relacionados con las aplicaciones informáticas.

Se cuenta entonces con una herramienta flexible, segura, que por su diseño y estar desarrollada en Software Libre permite su modificación para adaptarlo a las necesidades del usuario.

Recomendaciones

Se recomienda:

- ✓ Estudiar e implementar nuevos métodos numéricos con el fin de ampliar las funcionalidades de la Biblioteca.
- ✓ Hacer uso de la biblioteca en aplicaciones reales con el fin de aprovechar sus prestaciones.

Referencias Bibliográficas

1. Vadillo, Fernando. ***Matemática Numérica.*** **s.l. : 22, 2003. 1131-7787.**

2. Trefethen, L. N. **OXFORD UNIVERSITY COMPUTING LABORATORY. [En línea] [Citado el: 27 de noviembre de 2007.] http://web.comlab.ox.ac.uk/oucl/work/nick.trefethen/.**

3. Sanz-Serna, J. M. ***10 Lecciones de Cálculo Numérico.*** **Valladolid : Universidad de Valladolid, 1998.**

4. Chapra, Steven C. y Canale, Raymond P. ***Métodos numéricos para ingenieros: con aplicaciones en computadoras personales.*** **México : McGraw Hill, 1985.**

5. ESTUDIO DE SISTEMAS DE ECUACIONES LINEALES (s.e.l.). [En línea] [Citado el: 8 de diciembre de 2007.] http://personal5.iddeo.es/ztt/Tem/T8_Estudio_SEL.htm.

6. Alvarez, Manuel, Guerra, Alfredo y Lau, Rogelio. ***Matemática Numérica.***

7. CFD Online. [En línea] [Citado el: 21 de febrero de 2008.] http://www.cfd-online.com/Wiki/Tridiagonal_matrix_algorithm_-_TDMA_(Thomas_algorithm).

8. García González, Ana. **[En línea] [Citado el: 17 de diciembre de 2007.] http://www2.eco.uva.es/anagar/derive.pdf.**

9. [En línea] [Citado el: 11 de febrero de 2008.] http://www.granbazar.com.mx/cesta/index.php?act=viewProd&productId=196.

10. Universidad Pública de Navarra. **Servicio Informático: servicio de software específico. [En línea] [Citado el: 16 de diciembre de 2007.] http://www.unavarra.es/si/sisoftes.htm.**

11. Facultad de Informática, Dpto. Sistemas Informáticos y Computación. ***Benchmarks y Bibliotecas Numéricas Optimizadas.*** **Valencia : s.n., 2004.**

12. Cuenca, Javier y Giménez, Domingo. ***Computación Matricial y Paralela.*** **Murcia : s.n.**

13. SLEPc. [En línea] [Citado el: 16 de enero de 2008.] http://www.grycap.upv.es/slepc/description.htm.

14. Joshi, Mahesh. **PSPASES Home Page. [En línea] [Citado el: 2 de diciembre de 2007.] http://www-users.cs.umn.edu/%7Emjoshi/pspases/.**

15. Jacobson, Ivar, Booch, Grady y Rumbaugh, James. ***El Proceso Unificado de Desarrollo de Software.*** **Madrid : Addison Wesley, 2000. 84-7829-036-2.**

16. Beck, Kent. ***Extreme Programming Explained.*** **Madrid : Addison Wesley, 2000. 201-61641-6.**

17. Mendoza Sánchez, María A. **[En línea] 2004. [Citado el: 25 de enero de 2008.] http://www.informatizate.net/articulos/pdfs/metodologias_de_desarrollo_de_software_07062004.pdf.**

18. Centro de Informática Ingeniería. [En línea] 2007. [Citado el: 6 de marzo de 2008.] http://www.ciinsrl.com/index.php?modules=productos.

19. Larman, Craig. ***UML y Patrones Introducción al análisis y diseño orientado a objetos.*** **México : PRENTICE HALL, 1999. 970-17-0261-1.**

20. Perissé, Marcelo Claudio. ***PROYECTO INFORMATICO: Una metodología simplificada.*** **Buenos Aires : s.n., 2001. 987-43-2947-5.**

21. Visual Paradigm. [En línea] 2004. [Citado el: 20 de diciembre de 2007.] http://www.visual-paradigm.com/product/vpuml/.

22. Soluciones y propuestas RATIONAL. [En línea] 2006. [Citado el: 14 de abril de 2008.] http://www.rational.com.ar/brochures/rose.pdf.

23. Stroustrup, Bjarne. ***El lenguaje de programación C++.*** **Madrid : Addison Wesley, 1998. 84-7829-019-2.**

24. Méndez Pozo, Gonzalo. **Programación Orientada a Objetos Java. [En línea] 2006-2007. [Citado el: 28 de febrero de 2008.] http://www.fdi.ucm.es/profesor/gmendez/Tema1-Intro.pdf.**

25. Python Programming Language. [En línea] 1990-2008. [Citado el: 16 de febrero de 2008.] http://www.python.org/.

26. KDevelop - LinuxLinks.com. [En línea] abril de 2008. [Citado el: 3 de mayo de 2008.] http://www.linuxlinks.com/article/20070801153806402/KDevelop.html.

27. KDevelop - Free Software Directory. [En línea] 2008. [Citado el: 15 de marzo de 2008.] http://directory.fsf.org/project/KDevelop/.

28. KDevelop - An Integrated Development Environment. [En línea] 2008. [Citado el: 20 de enero de 2008.] http://www.kdevelop.org/.

29. Eclipse Platform Technical Overview. [En línea] 2003. [Citado el: 24 de febrero de 2008.] http://www.eclipse.org/whitepapers/eclipse-overview.pdf.

30. Vanrell, Juan Angel y Vaucheret, Claudio. **Implementando ayuda dinámica en la plataforma Eclipse. [En línea] [Citado el: 24 de febrero de 2008.] http://ficcte.unimoron.edu.ar/wicc/Trabajos/III%20-%20isbd/596-paperhelpecl.pdf.**

31. Code::Blocks Features. [En línea] [Citado el: 10 de marzo de 2008.] http://www.codeblocks.org/features.

32. David Garlan y Mary Shaw**. "An introduction to software architecture". CMU Software Engineering Institute Technical Report, CMU/SEI-94-TR-21, ESC-TR-94-21,**

Bibliografía Consultada

- Rogan C., José y Muñoz G., Víctor. *PROGRAMACION Y METODOS NUMERICOS.* Chile : s.n. 562-978-7276.
- Llin, V y Pozniak, E. *Fundamentos del Analisis Matemáticos.* Moscú : Mir Moscú, 1982. 5-03-002061-6.
- Sáez Martínez, José, García Molina, Jesús y J., Pedro. *Una Arquitectura para una Herramienta de Patrones de Diseño.* Murcia : s.n.
- Mayo Abad, Orestes. *Técnicas Básicas de Optimización.* La Habana : Preliminar, 1998.
- Hypotrech Ltd. *HYSYS.* Cánada : s.n., 2001.
- Quarteroni, Alfio, Sacco, Riccardo y Saleri, Fausto. *Numerical Mathematics.* New York : Springer-Verlag, 2000. 0-387-98959-5.
- Universidad de las Ciencias Informáticas. Entorno Virtual de Aprendizaje. *Ingenieria del Software I y II.* [En línea] UCI. teleformacion.uci.cu.
- Biblioteca. [En línea] UCI. biblioteca.uci.cu.
- J. Higham, Nicholas. *Accuracy and Stability of Numerical Algorithms.* Manchester : SIAM, 1996. 1098765432.
- Conte, S.D. y de Boor, Carl. *ELEMENTARY NUMERICAL ANALYSIS An Algorithmic Approach.* 3. 1980. 0-07-012447-7.
- Jacobson, Ivar, Booch, Grady y Rumbaugh, James. *El Proceso Unificado de Desarrollo de Software.* Madrid : Addison Wesley, 2000. 84-7829-036-2.
- Larman, Craig. *UML Y PATRONES Introducción al análisis y diseño orientado a objetos.* Mexico : PRENTICE HALL, 1999. 970-17-0261-1.
- Alvarez, Manuel, Guerra, Alfredo y Lau, Rogelio. *Matemática Numérica.* 2. s.l. : Felix Varela.

Anexos

Anexo 1

Tabla 15: Descripción de CC_Matriz

Nombre: CC_Matriz	
Atributo	**Tipo de Dato**
filas	Int
columnas	Int
datos	double **
filTemp	double *
colTemp	double *
Para cada funcionalidad	
Nombre	Matriz * Copia ( )
Descripción	Método para hacer una copia de la matriz.
Nombre	int Cantidad_Filas ( )
Descripción	Método que devuelve la cantidad de filas de la matriz.
Nombre	int Cantidad_Columnas()
Descripción	Método que devuelve la cantidad de columnas de la matriz.
Nombre	double Obtener_Elemento (int, int)
Descripción	Método que devuelve el elemento en una posición.
Nombre	void Adicionar_Elemento (int, int, double)
Descripción	Método que asigna un nuevo elemento a la matriz.
Nombre	Matriz * Suma (Matriz *)
Descripción	Método que devuelve la suma de la matriz original con la introducida por parámetros.
Nombre	Matriz * Resta (Matriz *)
Descripción	Método que devuelve la resta de la matriz original con la introducida por parámetros.
Nombre	Matriz * Multiplicacion (Matriz *)
Descripción	Método que devuelve la multiplicación de la matriz original con la introducida por parámetros.
Nombre	double * Restar_Filas (int, double *)
Descripción	Método que devuelve la resta de una fila de la matriz con otra introducida por parámetros.
Nombre	double * Multiplicacion_FilaxEscalar (int, double)

Descripción	Método que devuelve la multiplicación de una fila de la matriz por un escalar.
Nombre	void Cambiar_Fila (double *, int)
Descripción	Método que se encarga de cambiar una fila de la matriz por la introducida por parámetro.
Nombre	void Intercambio_Filas (int, int)
Descripción	Método que se encarga de intercambiar las filas de una matriz.
Nombre	void Intercambio_Columnas (int, int)
Descripción	Método que se encarga de intercambiar las columnas de una matriz.
Nombre	bool Igualdad (matriz *)
Descripción	Método que se encarga de verificar si la matriz que se entra por parámetros es igual a la original en cuanto a tamaño.

Tabla 16: Descripción de CC_SEL_Directo

Nombre: CC_SEL_Directo	
Atributo	**Tipo**
ampliada	Matriz *
x	double *
copiaMatriz	Matriz *
filas	int
columnas	int
Para cada responsabilidad:	
Nombre	double Modulo (double)
Descripción	Método que devuelve el módulo de un número.
Nombre	virtual double * Resolver( )
Descripción	Método que halla las soluciones del sistema.

Tabla 17: Descripción de CC_E_Gaussiana_PivT

Nombre: CC_E_Gaussiana_PivT	
Para cada responsabilidad:	
Nombre	double * Resolver( )
Descripción	Método que halla las soluciones del sistema.

Tabla 18: Descripción de CC_E_Gaussiana_PivP

Nombre: E_Gaussiana_PivP	
Para cada responsabilidad:	
Nombre	double * Resolver( )
Descripción	Método que halla las soluciones del sistema.

Tabla 19: Descripción de CC_E_GaussJordan

Nombre: CC_E_GaussJordan	
Para cada responsabilidad:	
Nombre	double * Resolver( )
Descripción	Método que halla las soluciones del sistema.

Tabla 20: Descripción de CC_SEL_Iterativo

Nombre: CC_SEL_Iterativo	
Atributo	**Tipo**
con	double
tol	double
xv	double *
a	Matriz *
b	Matriz *
xa	double *
Para cada responsabilidad:	
Nombre	double Modulo (double)
Descripción	Método que devuelve el módulo de un número.
Nombre	bool PreCondicion()
Descripción	Método que verifica que los datos sean correctos para que el sistema converja.
Nombre	virtual double * Resolver( )
Descripción	Método que halla las soluciones del sistema.

Tabla 21: Descripción de CC_Jacobi

Nombre: CC_Jacobi	
Para cada responsabilidad:	
Nombre	double * Resolver( )
Descripción	Método que halla las soluciones del sistema.

Tabla 22: Descripción de CC_Seidel

Nombre: CC_Seidel	
Para cada responsabilidad:	
Nombre	double * Resolver( )
Descripción	Método que halla las soluciones del sistema.

Tabla 23: Descripción de CC_SGDPD

Nombre: CC_SGDPD	
Atributo	**Tipo**
dp	double *
ddp	double *
dep	double *
ti	double *
inc	int
x	double *
Para cada responsabilidad:	
Nombre	virtual double * Resolver( )
Descripción	Método que halla las soluciones del sistema.

Tabla 24: Descripción de CC_Thomas

Nombre: CC_Thomas	
Para cada responsabilidad:	
Nombre	double * Resolver( )
Descripción	Método que halla las soluciones del sistema.

Tabla 25: Descripción de CC_GaussT

Nombre: CC_GaussT	
Para cada responsabilidad:	
Nombre	double * Resolver( )
Descripción	Método que halla las soluciones del sistema.

Tabla 26: Descripción de CC_Funcion

Nombre: CC_Funcion	
Para cada responsabilidad:	
Nombre	double Evaluar (double)
Descripción	Método para evaluar la función.

Tabla 27: Descripción de CC_Integral

Nombre: CC_Integral	
Atributo	**Tipo**
dx	double
area	double
a	double
b	double
n	int
f	Funcion *
Para cada responsabilidad:	
Nombre	double Modulo (double)
Descripción	Método que devuelve el módulo de un número.
Nombre	double Potencia(double, int)
Descripción	Método para calcular la potencia de un número.
Nombre	bool PreCondicion()
Descripción	Método que verifica que los datos sean correctos para la integración.
Nombre	virtual double Integrar( )
Descripción	Método que calcula y devuelve la integral.

Tabla 28: Descripción de CC_Rectangular

Nombre: CC_Rectangular	
Para cada responsabilidad:	
Nombre	double Integrar( )
Descripción	Método que calcula y devuelve la integral.

Tabla 29: Descripción de CC_Trapecios

Nombre: CC_Trapecios	
Para cada responsabilidad:	
Nombre	double Integrar( )
Descripción	Método que calcula y devuelve la integral.

Tabla 30: Descripción de CC_Simpson

Nombre: CC_Simpson	
Para cada responsabilidad:	
Nombre	double Integrar( )
Descripción	Método que calcula y devuelve la integral.

Tabla 31: Descripción de CC_Romberg

Nombre: CC_Romberg	
Atributo	**Tipo**
tol	double
Para cada responsabilidad:	
Nombre	double Integrar( )
Descripción	Método que calcula y devuelve la integral.

Anexo 2

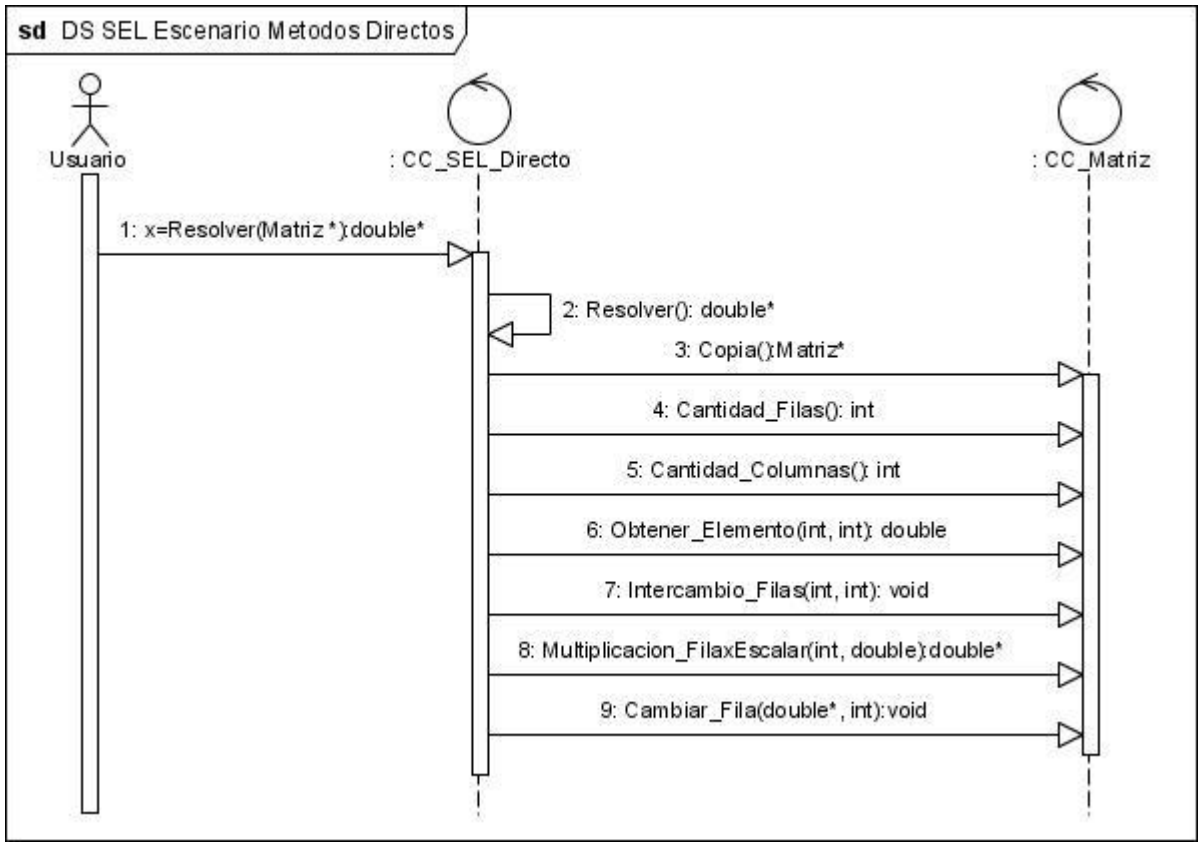


Figura 17: Diagrama de Secuencia SEL Escenario Métodos Directos

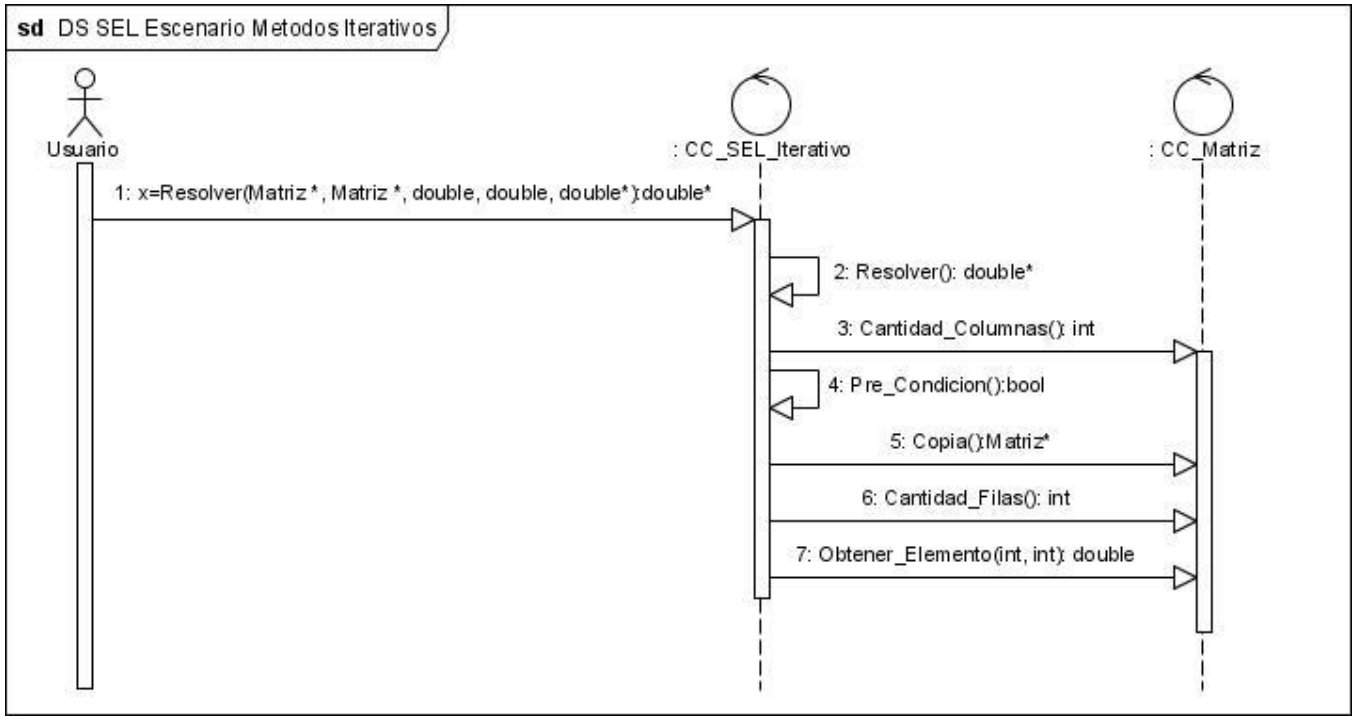


Figura 18: Diagrama de Secuencia SEL Escenario Métodos Iterativos

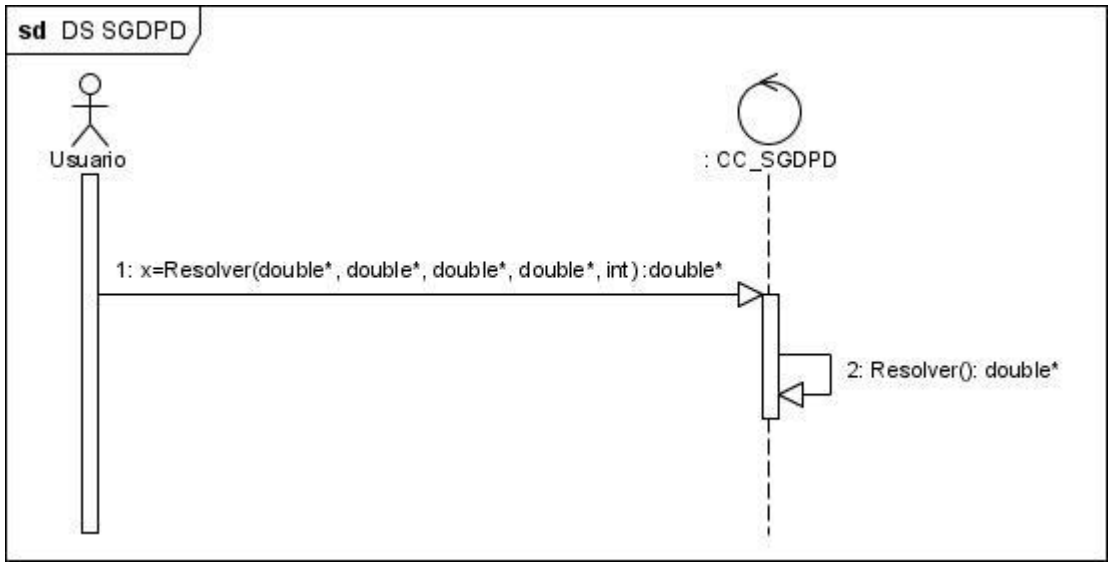


Figura 19: Diagrama de Secuencia SGDPD

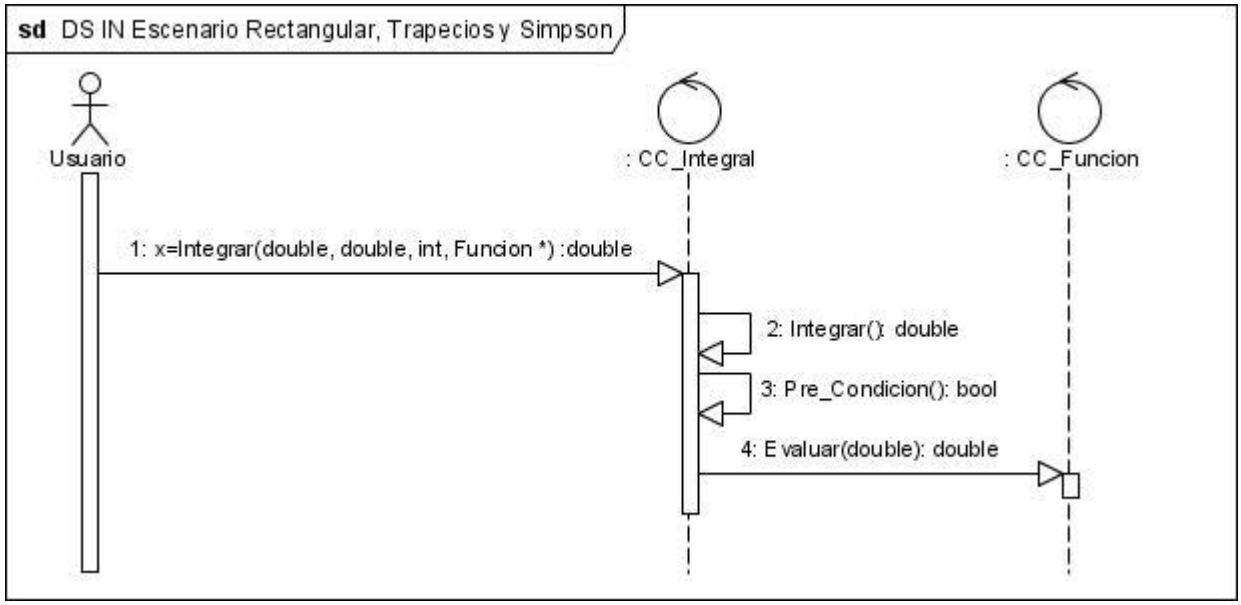


Figura 20: Diagrama de Secuencia IN Escenario Rectangular, Trapecios y Simpson

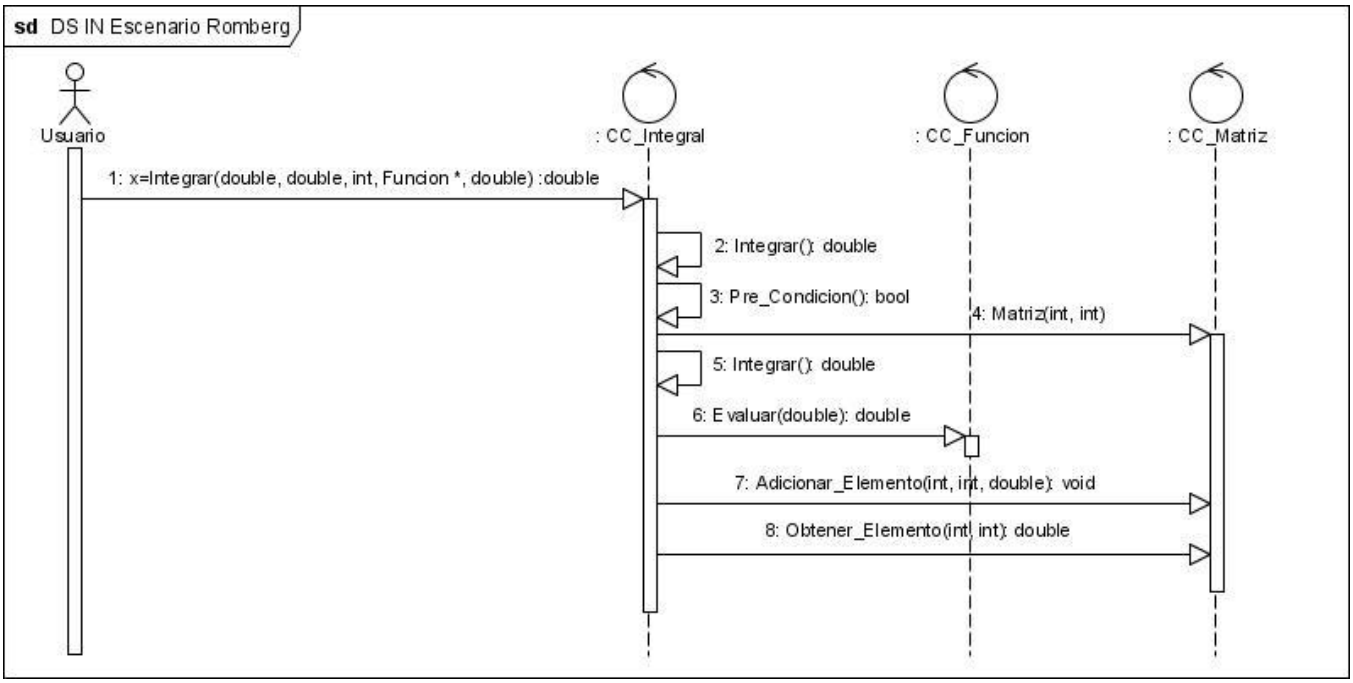


Figura 21: Diagrama de Secuencia IN Escenario Romberg

Anexo 3

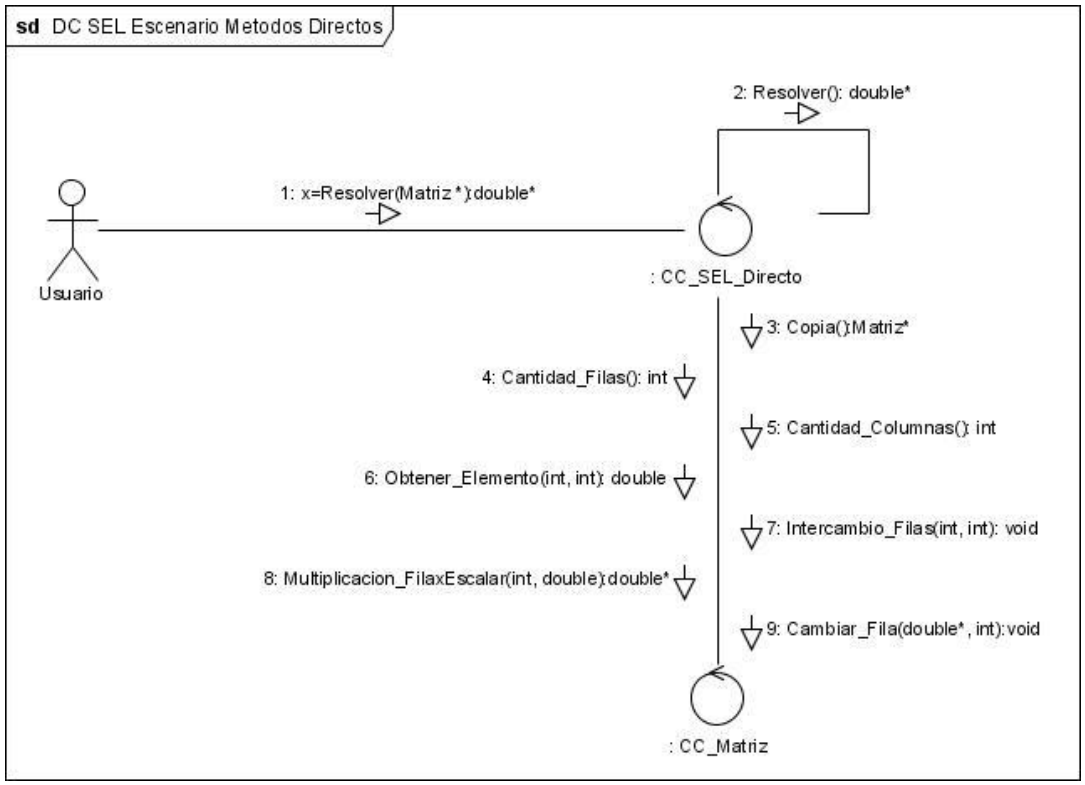


Figura 22: Diagrama de Colaboración SEL Escenario Métodos Directos

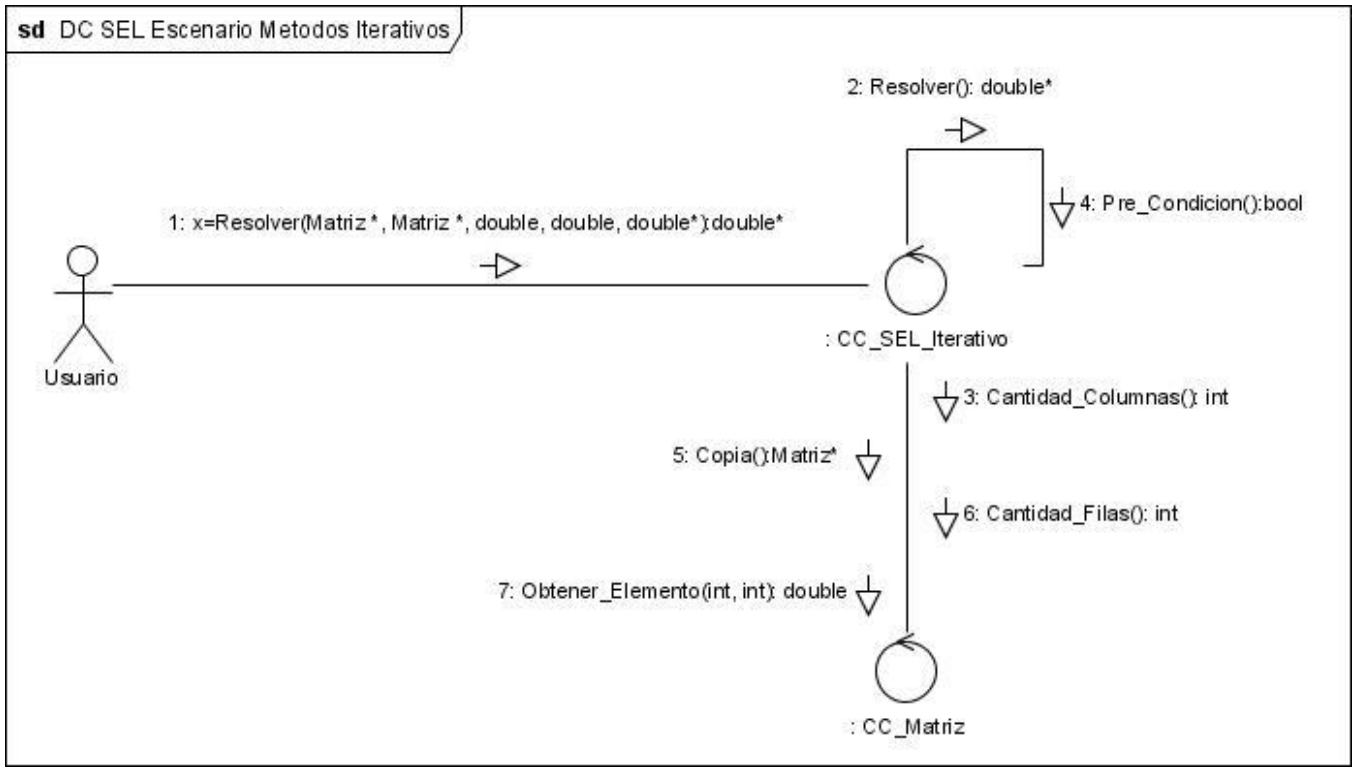


Figura 23: Diagrama de Colaboración SEL Escenario Métodos Iterativos

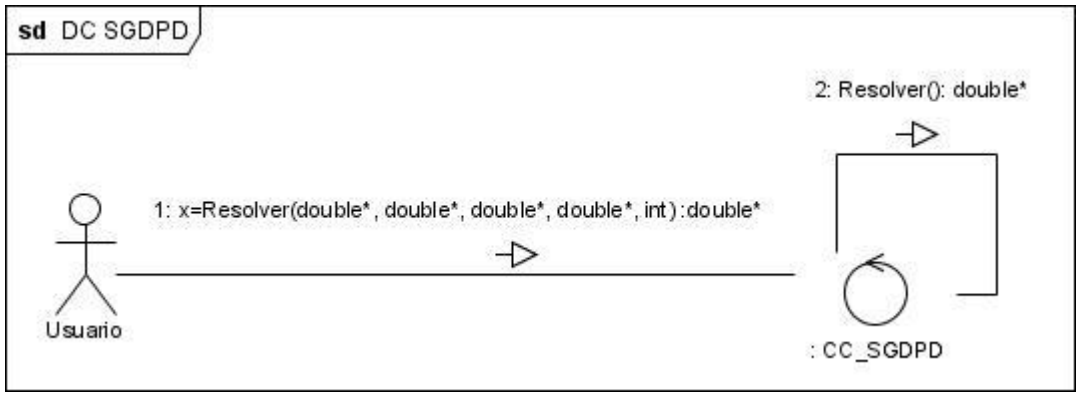


Figura 24: Diagrama de Colaboración SGDPD

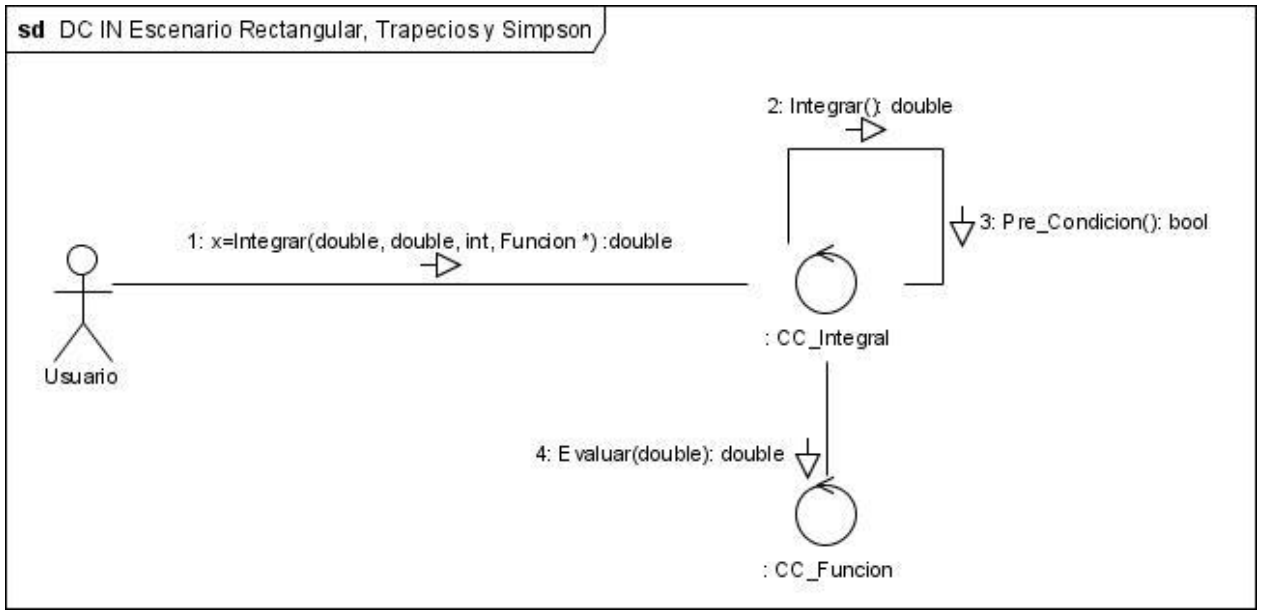


Figura 25: Diagrama de Colaboración IN Escenario Rectangular, Trapecios y Simpson

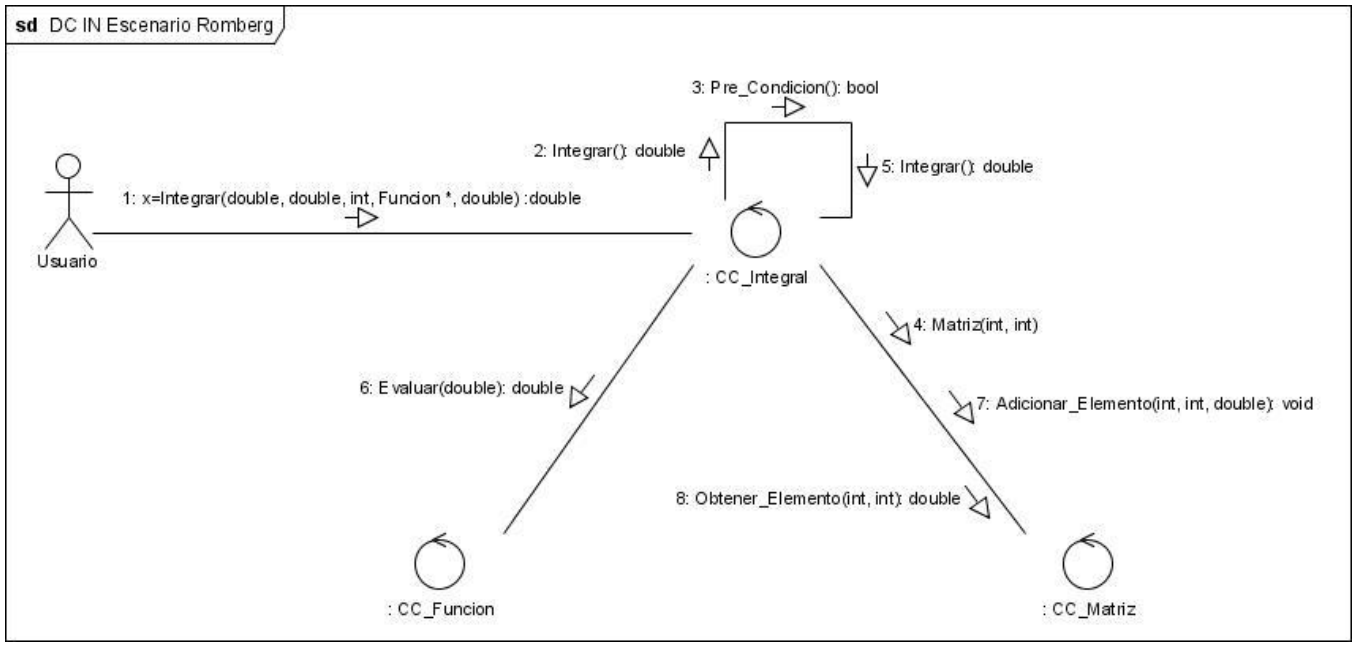


Figura 26: Diagrama de Colaboración IN Escenario Romberg

Glosario

Aquí se incluye una explicación de determinados términos y siglas utilizados en el texto para facilitar su comprensión. Generalmente se incluyen los términos que tienen menor difusión en nuestro campo profesional, los de otro campo profesional, o aquellos términos conocidos, pero que se usan con un significado diferente en el texto. Además, pueden ser incluidas las siglas utilizadas en el documento para facilitar su lectura.

Actuario: Persona versada en los cálculos matemáticos y en los conocimientos estadísticos, jurídicos y financieros concernientes a los seguros y a su régimen, la cual asesora a las entidades aseguradoras y sirve como perito en las operaciones de estas.

API: *Application Programming Interface - Interfaz de Programación de Aplicaciones* es el conjunto de funciones y procedimientos (o métodos si se refiere a programación orientada a objetos) que ofrece cierta biblioteca para ser utilizado por otro software como una capa de abstracción.

Applets: Componente de una aplicación que se ejecuta en el contexto de otro programa, por ejemplo un navegador web. El applet debe ejecutarse en un contenedor, que lo proporciona un programa anfitrión, mediante un plugin, o en aplicaciones como teléfonos móviles que soportan el modelo de programación por applets.

CBSE: (Component-based Software Engineering - Componente basado en Ingeniería de Software) tiene que ver con el desarrollo de software de sistemas intensivos de piezas reutilizables (componentes), el desarrollo de partes reutilizables, y el mantenimiento del sistema y la mejora de los medios de los componentes de reemplazo y personalización.

Ciencias Actuariales: Es la ciencia que aplica procedimientos matemáticos a problemas relacionados con actividades de seguros públicos o privados, de sociedades financieras y de cualquier entidad pública que requiere un estudio matemático actuarial.

Clúster: Grupo de múltiples ordenadores unidos mediante una red de alta velocidad, de tal forma que el conjunto es visto como un único ordenador, más potente que los comunes de escritorio.

Feedback: Es un proceso por el cual cierta proporción de la señal de salida de un sistema se pasa (alimentado de nuevo) a la entrada. Esta se usa a menudo para controlar el comportamiento dinámico del sistema. Ejemplos de información se puede encontrar en la mayoría de los sistemas complejos, tales como la ingeniería, arquitectura, economía, la termodinámica y la biología. Un ejemplo de un complejo sistema de votos es el sistema de dirección de un automóvil. Mientras conduce, una persona que recibe las señales del medio ambiente, tales como signos y peligros. El cerebro del conductor procesa la información y envía señales al automóvil a través del volante y los pedales. El automóvil responde al cambiar de dirección o velocidad en consecuencia.

Frameworks: Estructura de soporte definida en la cual otro proyecto de software puede ser organizado y desarrollado. Típicamente, un framework puede incluir soporte de programas, bibliotecas y un lenguaje interpretado entre otros software para ayudar a desarrollar y unir los diferentes componentes de un proyecto.

GUI: Graphical User Interface- Interfaz Gráfica de Usuario es el artefacto tecnológico de un sistema interactivo que posibilita, a través del uso y la representación del lenguaje visual, una interacción amigable con un sistema informático. Utiliza un conjunto de imágenes y objetos gráficos para representar la información y acciones disponibles en la interfaz.

Hyperlink: Es una referencia o elemento de navegación en un documento a otra sección del mismo documento o a otro documento.

IGP: Interior Gateway Protocol – Protocolo de Pasarela Interno hace referencia a los protocolos usados dentro de un sistema autónomo.

Iteración: En *matemática* se refiere al proceso de iteración de una función o a las técnicas que se usan en métodos iterativos para la resolución de problemas numéricos y en *programación* es la repetición de una serie de instrucciones en un programa de computadora. Puede usarse tanto como un término genérico (como sinónimo de repetición) así como para describir una forma específica de repetición con un estado mutable.

Kernel: Núcleo. Parte fundamental de un programa, por lo general de un sistema operativo, que reside en memoria todo el tiempo y que provee los servicios básicos. Es la parte del sistema operativo que está más cerca de la máquina y puede activar el hardware directamente o unirse a otra capa de software que maneja el hardware.

Latex: Lenguaje de marcado para documentos, y un sistema de preparación de documentos, formado por un gran conjunto de macros de TeX.

Layers: Un layer o capa es un soporte que contiene cierta información, ya sea gráfica, lógica o de cualquier otra índole y que es, en sí mismo, un objeto integral e independiente.

Licencia GNU/GPL: Está diseñada para asegurar la libertad de distribuir copias de Software Libre (y cobrar por ese servicio), asegurar que recibirá el código fuente del programa o bien podrá conseguirlo si quiere, asegurar que puede modificar el programa o modificar algunas de sus piezas para un nuevo programa y para garantizar que puede hacer todas estas cosas.

Mathlink: MathLink es una librería de funciones que implementa un protocolo para enviar y recibir.

MFC: (Microsoft Foundation Class Library) es una biblioteca que envuelve porciones de la API de Windows en clases C + +, incluyendo la función que les permite utilizar un marco de aplicación por defecto.

Middleware: El Middleware es un software de conectividad que ofrece un conjunto de servicios que hacen posible el funcionamiento de aplicaciones distribuidas sobre plataformas heterogéneas. Funciona como una capa de abstracción de software distribuida, que se sitúa entre las capas de aplicaciones y las capas inferiores (sistema operativo y red). El Middleware nos abstrae de la complejidad y heterogeneidad de las redes de comunicaciones subyacentes, así como de los sistemas operativos y lenguajes de programación, proporcionando una API para la fácil programación y manejo de aplicaciones distribuidas.

MIMD: (Multiple Instruction stream, Multiple Data stream) es una técnica empleada para lograr el paralelismo.

MPI: ("Message Passing Interface", Interfaz de Paso de Mensajes) es un estándar que define la sintaxis y la semántica de las funciones contenidas en una librería de paso de mensajes diseñada para ser usada en programas que exploten la existencia de múltiples procesadores.

Open Source: El término Open Source nace cuando se pretende obtener aplicaciones comerciales para el software libre. Puesto a que no pueden obligar a pagar por el código, ofrecen asesoramiento y

mantenimiento de sistemas. A la hora de llamar la atención de los inversores, el nombre de software libre (free software) no resultaba atractivo, por lo que se buscó un término alternativo que permitiera ofrecer a los inversores conceptos que no tuvieran nada que ver con “free”. La realidad no enseña que el concepto “free” significa en inglés tanto “libre” como “gratis”. Con el fin de eliminar toda referencia a “gratis”, se creo el concepto de Código Abierto (Open Source). Así otorgaron cobertura legal a aquellos programas que optaran por una licencia Open Source frente a una de software libre.

Objective-C: Es un lenguaje de programación orientado a objetos creado como un súper-conjunto de C pero que implementase un modelo de objetos parecido al de Smalltalk.

Organigrama: Es un cuadro sintético que indica los aspectos importantes de una estructura de organización, incluyendo las principales funciones y sus relaciones, los canales de supervisión y la autoridad relativa de cada empleado encargado de su función respectiva.

Plugin: Es un módulo de hardware o software que añade una característica o un servicio específico a un sistema más grande. Un programa puede tener uno o más conectores. Son muy utilizados en los programas navegadores para ampliar sus funcionalidades.

PVM: (*Parallel Virtual Machine*) es un paquete de software que permite a una colección heterogénea de computadoras, con sistema operativo *UNIX*, que estén conectadas a través de una red, ser usadas como una sola máquina paralela.

Rutina driver: Pertenece a la serie de rutinas de generación de números y permutaciones aleatorias. La rutina drive se reduce a devolver un conjunto de M permutaciones de tamaño N, utilizando para ello las rutinas siguientes: Rutina Aleator y Rutina gen_perm.

Smalltalk: Es un sistema informático que permite realizar tareas de computación mediante la interacción con un entorno de objetos virtuales. Metafóricamente, se puede considerar que un Smalltalk es un mundo virtual donde viven objetos que se comunican mediante el envío de mensajes.

Spline: En el subcampo matemático del análisis numérico, un *spline* es una curva definida a trozos mediante polinomios. El término "spline" hace referencia a una amplia clase de funciones que son utilizadas en aplicaciones que requieren la interpolación de datos, y/o un alisado en la interpolación.

Testing: Pruebas que se le realizan a los software que permiten encontrar las fallas antes de la puesta en producción. Esto garantiza la calidad de un software.

Thread: Un Thread (que de una forma un poco 'basta' se puede traducir como hilo) es la unidad básica de ejecución de OS/2. Cualquier programa que se ejecute consta de, al menos, un thread. Se puede considerar como la agrupación de un trozo de programa junto con el conjunto de registros del procesador que utiliza y una pila de máquina.

Toolbox: (Caja de herramientas), una colección especializada de archivos .m diseñada para trabajar en algunos problemas específicos. Algunos de ellos pueden ser Toolbox de identificación de sistemas, Toolbox de ecuaciones derivadas parciales, Toolbox de matemática simbólica, etc.