UNIVERSIDAD DE LAS CIENCIAS INFORMÁTICAS

Facultad 9

TÍTULO: Desarrollo de una biblioteca de métodos numéricos (BMN), referente al cálculo de raíces de funciones, interpolación y ajuste de curvas y resolución de ecuaciones diferenciales.

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

AUTORES: Romanuel Ramón Antunez

Lidisy Hernández Montero

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,3 de julio, 2008.

"Año 50 de la Revolución"

DEDICATORIA

Dedico este trabajo a mis padres y a mi hermana, para que conozcan un poquito sobre el mundo de las matemáticas y la informática.

Lidisy

Dedico este trabajo a mis padres por toda la confianza depositada en mí y a mi hermano menor, lo que te propongas lo lograrás.

Romanuel

AGRADECIMIENTOS

A mis padres por estar siempre a mi lado, por quererme tanto, porque sin el esfuerzo de ellos este sueño nunca se habría hecho realidad.

A mi hermana, mi mejor amiga, por ser mi sostén en los momentos más difíciles de mi vida.

A mi amado esposo, por hacerme feliz, por confiar en mí, por ser mi brazo derecho, mi vida.

A mi familia por el amor y apoyo que siempre me han dado.

A Lago y Alexey por ayudarnos tanto.

A Fidel y la Revolución, por haber creado esta universidad de excelencia.

Lidisy

A mi mamá por todo su apoyo y confianza, ya tienes un ingeniero.

A mi papá y a Bernardo (mi segundo padre) por hacer de mí el hombre que soy.

A mi hermana Elizabeth por todo su apoyo durante estos 5 años de carrera.

A mi hermanito (Lolo) por ser mi fuente de inspiración.

A mi esposa preciosa por todo su apoyo, comprensión y amor hacia mí.

A mi abuelo por ser mi motor impulsor.

A toda mi familia por su ayuda siempre.

A mis socios de Santiago por tantos años de alegría y guerrilla juntos (desde el pre) en especial a Adrián Carlos.

A mi tutor quien me incursionó por primera vez en el tema de numérica, y por todo su apoyo para la culminación de este trabajo.

A mi cotutor y a Yancy por su ayuda incondicional.

A todos los que de una forma u otra ayudaron a culminar este trabajo.

Romanuel

DECLARACIÓN DE AUTORÍA

Declaramos que somos los únicos autores de este trabajo y autorizamos al Polo Conceptualización PDVSA de 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 3 días del mes de julio del año 2008.

Romanuel Ramón Antunez	Lidisy Hernández Montero	José Ángel Lago Graverán

OPINION DEL TUTOR

El tutor del presente Trabajo de Diploma considera que durante su ejecución los estudiantes mostraron las cualidades que a continuación se detallan:

Presentaron un alto grado de independencia al asumir las tareas individuales asignadas. La dedicación y seriedad con que asumieron la investigación trajeron como resultado que su trabajo se convirtiera en referencia para otros estudiantes.

Alcanzaron un profundo dominio del campo donde se realizó la investigación.

En la documentación generada se destaca la originalidad y creatividad expuesta por los autores.

La herramienta obtenida como resultado es robusta y los resultados arrojados a raíz de las pruebas hechas fueron satisfactorios.

En el trabajo realizado se puede apreciar una alta calidad científico-técnica. Los resultados obtenidos presentan un elevado nivel de rigurosidad y formalismo, lo cual corrobora que el objetivo propuesto al inicio de la investigación fue alcanzado.

Por todo lo anteriormente expresado se considera que ambos estudiantes están aptos para ejercer como Ingeniero en Ciencias Informáticas, y se propone:

Calificación máxima de 5 puntos a la diplomante Lidisy Hernández Montero.
Calificación máxima de 5 puntos al diplomante Romanuel Ramón Antunez.

José Ángel Lago Graverán

_____ _____

Firma Fecha

OPINION DEL OPONENTE

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

Título: Desarrollo de una biblioteca de métodos numéricos (BMN), referentes al cálculo de raíces de funciones, interpolación y ajuste de curvas y resolución de ecuaciones diferenciales.

Autores: Romanuel Ramón Antunez
Lidisy Hernández Montero

El documento presentado está correctamente estructurado, cuenta con una introducción, 5 capítulos, conclusiones, recomendaciones y anexos; en los capítulos se exponen la Fundamentación Teórica, las soluciones técnicas y tendencias, la presentación de la solución propuesta, la solución propuesta como tal y el análisis de factibilidad. El documento cuenta con una buena redacción, ortografía y está en correspondencia con un trabajo de ciclo completo, investigación, diseño e implementación.

Con el desarrollo de este trabajo se puede apreciar calidad científico-técnica del mismo, un elevado nivel de rigurosidad y formalismo que certifica el cumplimiento del objetivo propuesto en la investigación. Aporta un aceptable resumen teórico que por si solo puede servir de breve documento introductorio a los temas de Matemática Numérica contenidos. El análisis de las soluciones existentes constituye un catálogo primario para profundizar en las soluciones técnicas existentes y/o seleccionar un asistente adecuado para el trabajo matemático. La solución propuesta aporta una biblioteca de clases libre, ajustada a las necesidades nacionales y del momento histórico, y con un manual de trabajo y uso oportuno para sus usuarios, carencia esta último de muchas otras soluciones libres. Además de ajustarse a las necesidades del proyecto productivo y sus aplicaciones que necesiten cálculo numérico.

La bibliografía consultada es abundante y actualidad. Las respuestas a las preguntas formuladas fueron adecuadas y constataron el dominio del trabajo de tesis y de los temas afines por parte de los diplomantes.

Este trabajo puede resultar el comienzo de una solución potente para este tipo de aplicaciones.

Teniendo en cuenta lo antes expresado, la exposición realizada por los diplomantes, la calidad de las respuestas y el cumplimiento de los objetivos propuestos, se propone al Tribunal la calificación máxima de 5 puntos a ambos diplomantes.

Oponente: M.Sc. Walter Carballosa Torres

Fecha: 3 de Julio del 2008

“Con números se puede demostrar cualquier cosa"

Isócrates

"Las matemáticas poseen no sólo la verdad, sino cierta belleza suprema. Una belleza fría y austera, como la de una escultura"

Einstein

RESUMEN

El siguiente trabajo pone en manos de los usuarios finales una alternativa para el desarrollo de aplicaciones que requieran siempre el uso de métodos numéricos. El trabajo aborda el desarrollo de una biblioteca de clases libre que proporciona facilidades a los usuarios a la hora de implementar aplicaciones como simuladores, software de música o diseño, entre otras que requieran algoritmos para el cálculo de raíces de funciones no lineales, hallar la interpolación de un valor, ajustar funciones y solucionar sistemas de ecuaciones diferenciales ordinarias(EDO). Nacionalmente no se han hecho muchas investigaciones relevantes respecto al tema y este trabajo puede resultar el comienzo de una solución potente para este tipo de aplicaciones.

Para alcanzar la meta trazada se estudian otras soluciones existentes y se trabaja la investigación de algoritmos eficientes para resolver los problemas numéricos antes expuestos; además de realizarse un estudio de las tendencias y tecnologías actuales que pueden ser útiles para la construcción de la solución propuesta.

PALABRAS CLAVES

ajuste, biblioteca de clases, ecuación diferencial, interpolación, métodos numéricos, raíz.

ÍNDICE

FIGURAS

TABLAS

INTRODUCCIÓN

La ciencia describe los fenómenos reales mediante modelos matemáticos que permiten un conocimiento más profundo del fenómeno, así como de su futuro desarrollo. Las herramientas más utilizadas en los problemas asentados en estos modelos, son los sugeridos por la matemática aplicada, pero no siempre se puede hacer uso de métodos analíticos clásicos por múltiples razones:

- No se adecúan al modelo concreto.
- Su aplicación resulta excesivamente compleja.
- La solución formal es tan complicada que hace imposible cualquier interpretación posterior.
- Simplemente no existen métodos analíticos capaces de proporcionar soluciones al problema.

Siendo útiles en este caso los métodos numéricos, que son técnicas mediante las cuales es posible obtener la solución de un problema, que mediante una labor de cálculo intensa, llevan a soluciones aproximadas siempre numéricas, manejando adecuadamente las fuentes de errores. El importante esfuerzo de cálculo que implica la mayoría de estos métodos hace que su uso esté intrínsecamente ligado al empleo de computadoras.

En la actualidad existen muchos software tales como: software de música, software de diseño, simuladores, etc. que requieren del uso de métodos numéricos. Existiendo varias herramientas o software, que permiten a los programadores desarrollar o trabajar con algunos de estos métodos.

Desafortunadamente estas herramientas no resuelven el problema a los programadores de economizar tiempo y evitar posibles errores en la implementación de estos métodos numéricos, pues la mayoría son herramientas privativas por lo que no posibilitan el código para adaptarlo a las necesidades particulares y/o las libres no están totalmente documentadas.

Siendo entonces el problema científico la poca disponibilidad de bibliotecas libres que agrupen la mayoría de los métodos numéricos, para ser usadas en software de diseño, sonido y simuladores.

Por consiguiente el objeto de estudio es la matemática numérica y el campo de acción los métodos numéricos referentes a raíces de funciones, interpolación y ajuste de curvas y la resolución de ecuaciones diferenciales.

Con este trabajo se pretende desarrollar una biblioteca de métodos numéricos libre que incluya rutinas que abarquen las distintas ramas de la matemática numérica, para ser usada en software que requieran soluciones de este tipo.

La idea a defender que se plantea es que, con el desarrollo de este trabajo aumentaría la disponibilidad de bibliotecas libres y la documentación de uso de métodos numéricos, que agilicen la implementación de software de diseño, sonido y simuladores.

Para dar cumplimiento al objetivo trazado, se plantearon las siguientes tareas de investigación:

1. Estudiar estado del arte.
2. Escoger herramientas para el desarrollo de la BMN.
3. Realizar análisis y diseño de la BMN.
4. Implementar métodos para el cálculo de raíces de funciones.
5. Implementar métodos de interpolación.
6. Implementar métodos para ajuste de curvas.
7. Implementar métodos para la resolución de ecuaciones diferenciales.
8. Realizar pruebas a la biblioteca.
9. Elaborar documentación de la biblioteca.

Para lograr un mejor entendimiento y obtener una mayor comprensión de lo que está sucediendo, se pondrán en práctica los siguientes métodos científicos:

1. Métodos Teóricos
   - 1.1. Lógicos
     - 1.1.1. Modelación: para crear un modelo (abstracción) que explique el resultado final del trabajo.
     - 1.1.2. Análisis y Síntesis: que permita el estudio de la bibliografía utilizada y posibilite que se realice una recapitulación de la misma.
   - 1.2. Históricos: Para conocer todo lo referente a los antecedentes que existen respecto a bibliotecas de este tipo y programas o software que trabajen con los diferentes métodos numéricos.
2. Métodos Particulares
   - 2.1. Entrevistas: al tutor, cotutor, consultor y otros especialistas de la matemática numérica, y matemática aplicada para un mayor conocimiento sobre el uso de estos métodos numéricos.
   - 2.2. Otros: revisión de libros, sitios web, y otros documentos especializados en matemática para una mejor comprensión de los métodos y de sus usos.

Este trabajo fue estructurado de la siguiente forma:

Capítulo 1 Fundamentación Teórica: En este capítulo se analizan los conceptos asociados al dominio del problema, además de ser explicados cada uno de los métodos a implementar en la BMN y un análisis del estado del arte sobre bibliotecas y herramientas para el trabajo con métodos numéricos.

Capítulo 2 Tendencias y tecnologías actuales a desarrollar: En este capítulo se tocan las tecnologías a utilizar para el desarrollo de la biblioteca. IDE Code::Blocks como entorno de desarrollo, lenguaje de programación C++, Lenguaje Unificado de Modelado (UML) para la modelación de la solución propuesta, como herramienta CASE Visual Paradigm y el Proceso Unificado de Desarrollo de Software (RUP) como metodología 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 de los actores y casos de uso del sistema así como el diseño de estos.

Capítulo 4 Construcción de la solución propuesta: Se construye de forma completa la solución propuesta a la biblioteca. Además, se le va dando la forma final a los diagramas necesarios a utilizar a la hora de realizar el análisis y diseño del trabajo para completar la modelación del sistema.

Capítulo 5 Estudio de Factibilidad: Descripción de la planificación y gestión del proyecto por puntos de casos de uso como método de planificación.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA.

1.1. INTRODUCCIÓN

En este capítulo se abordarán diferentes conceptos asociados al dominio del problema, los que resultan necesarios para una mejor comprensión del tema a desarrollar. Se trata de forma independiente cada método a implementar en la BMN para un mejor entendimiento del funcionamiento del mismo, así como un estudio de las tendencias actuales para el trabajo de métodos numéricos.

1.2. CONCEPTOS ASOCIADOS A LA MATEMÁTICA NUMÉRICA

Biblioteca: (1)

En computación, una biblioteca es un conjunto de procedimientos y funciones (subprogramas) agrupadas en un archivo con el fin de ser aprovechadas por otros programas. Al proceso de hacer accesibles estos subprogramas al programa principal se le llama enlace (*link*).

Existen dos tipos de bibliotecas:

- las estáticas, o de enlace estático.
- y las compartidas, o de enlace dinámico.

En Windows, archivos de bibliotecas dinámicas poseen extensión .DLL (*Dynamic Link Library*), mientras que las estáticas generalmente terminan en .LIB. En Unix y Linux, las bibliotecas dinámicas tienen extensión .so (*Shared Object*) y las estáticas .a (*archive*).

Habitualmente se emplea el término librería para referirse a una biblioteca, por la similitud con el original inglés *library*. Ambos términos, biblioteca y librería son correctos según las definiciones de la Real Academia Española, aunque los puristas consideran como correcta biblioteca.

Matemática numérica: (2)

La matemática numérica es una rama de las matemáticas cuyos límites no son del todo precisos. De una forma rigurosa, se puede definir como la disciplina ocupada de describir, analizar y crear

algoritmos numéricos que nos permitan resolver problemas matemáticos, en los que estén involucradas cantidades numéricas, con una precisión determinada.

En el contexto del cálculo numérico, un algoritmo es un procedimiento que nos puede llevar a una solución aproximada de un problema mediante un número finito de pasos que pueden ejecutarse de manera lógica. En algunos casos, se les da el nombre de métodos constructivos a estos algoritmos numéricos.

La matemática numérica cobra especial importancia con la llegada de las computadoras. Estas últimas son útiles para cálculos matemáticos extremadamente complejos, pero en última instancia operan con números binarios y operaciones matemáticas simples.

Desde este punto de vista, el análisis numérico proporcionará todo el andamiaje necesario para llevar a cabo todos aquellos procedimientos matemáticos susceptibles de expresarse algorítmicamente, basándose en algoritmos que permitan su simulación o cálculo en procesos más sencillos empleando números.

Métodos numéricos: (3)

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.

Función: (4) (5)

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

$$f: X \to Y$$

que cumple con las siguientes dos condiciones:

1. Condición de existencia: Todos los elementos de X están relacionados con los elementos de Y, es decir:

   $\forall x \in X, \exists\, y \in Y\ (x, y) \in f$

2. Condición de unicidad: Cada elemento de X está relacionado con un único elemento de Y, es decir, si $(x, y_1) \in f \wedge (x, y_2) \in f \Longrightarrow y_1 = y_2$

   Y se de denota $f(x) = y$

Dominio: (4) (5)

Es el conjunto de existencia de la misma, es decir, los elementos para los cuales la función está definida. Es el conjunto de todos los objetos que puede transformar, se denota Dom_f o bien D_f y está definido por:

$$D_f = \{x \in X : \exists y \in Y,\ f(x) = y\}$$

Raíz de función: (6)

En matemática, se conoce como raíz (o cero) de una función $f(x)$ a todo elemento x perteneciente al dominio de dicha función tal que se cumpla:

$$f(x) = 0$$

Polinomio: (7)

En matemáticas un polinomio es una expresión que se construye por una o más variables, usando solamente las operaciones de adición, sustracción, multiplicación y exponentes enteros numéricos positivos. Debe mencionarse en particular que la división por una expresión que contiene una variable no es un polinomio sino una función racional.

Debido a su estructura simple, los polinomios son muy sencillos de evaluar, y son usados extensivamente en análisis numérico para interpolación polinomial o para integrar numéricamente funciones más complejas.

Spline: (8)

En el subcampo matemático del análisis numérico, un spline es una curva definida a trozos mediante polinomios.

En los problemas de interpolación, se utiliza a menudo la interpolación mediante splines porque da lugar a resultados similares requiriendo solamente el uso de polinomios de bajo grado a la vez que se evitan las oscilaciones, que en la mayoría de las aplicaciones resultan indeseables, que aparecen al interpolar mediante polinomios de grado elevado.

Para el ajuste de curvas, los splines se utilizan para aproximar formas complicadas. La simplicidad de la representación y la facilidad de cómputo de los splines los hacen populares para la representación de curvas en informática, particularmente en el terreno de los gráficos por ordenador.

Interpolación: (9)

El método de interpolación es un método científico-lógico que consiste en suponer que el curso de los acontecimientos continuará en el futuro, convirtiéndose en las reglas que utilizamos para llegar a una nueva conclusión. Es decir, sabemos a ciencia cierta que existen unos axiomas y éstos son extrapolables a la nueva situación. Toma como base la inducción y la analogía.

Utilizado para buscar la solución a un problema (lógica) o de enseñar la misma (pedagogía), lo que lo convierte en una herramienta muy utilizada en el marco profesional y de enseñanza.

En el subcampo matemático del análisis numérico, se denomina interpolación a la construcción de nuevos puntos partiendo del conocimiento de un conjunto discreto de puntos.

Ajuste de curvas: (10)

Consiste en encontrar una curva que se aproxime lo mejor posible a una serie de puntos y que posiblemente cumpla una serie de restricciones adicionales.

Ecuaciones diferenciales: (11)

Una ecuación diferencial es una ecuación en la que intervienen derivadas de una o más funciones.

Dependiendo del número de variables independientes respecto a las que se derivan, las ecuaciones diferenciales se dividen en:

- Ecuaciones diferenciales ordinarias: aquellas que contienen derivadas respecto a una sola variable independiente.
- Ecuaciones en derivadas parciales: aquellas que contienen derivadas respecto a dos o más variables.

Resolución de ecuaciones diferenciales: (12)

Es un tipo de problema matemático que consiste en buscar una función que cumpla una determinada ecuación diferencial. Se puede llevar a cabo mediante un método específico para la ecuación diferencial en cuestión o mediante una transformada.

1.3. OBJETO DE ESTUDIO

La matemática numérica tiene como propósito el desarrollo de métodos para la solución de los más diversos problemas matemáticos mediante una cantidad finita de operaciones numéricas. Uno de estos problemas más frecuentes es del tipo

$$f(x) = 0$$

Donde $f(x)$ es una función no lineal genérica y se busca (si es que existen) los valores de x en un intervalo I que satisfacen la ecuación. El intervalo I donde se buscan las soluciones puede ser cerrado o abierto y de amplitud infinita.

La importancia de los métodos numéricos encargada de resolver este problema viene dada por la frecuencia de estos problemas en cualquier área de la ciencia y la técnica y la limitación de los métodos analíticos para dar solución a este problema.

Otro de los problemas es la aproximación de funciones en el que se halla una expresión analítica $g(x)$ que sirva para aproximar otra función $f(x)$ para x en algún intervalo [a, b].

Problemas de este tipo se dan con mucha frecuencia en la simulación pues por lo general no se conoce $f(x)$ sino valores aislados. En ocasiones el algoritmo algebraico para hallar $f(x)$, aunque se conoce, es tan complejo que se prefiere hallar una $g(x)$ de una clase más simple, aún sabiendo que se incurre en un error.

Una ecuación diferencial como hemos dicho antes es aquella donde aparecen derivadas y, si todas las derivadas son respecto a una variable independiente entonces se llaman ordinarias.

$$\frac{dy}{dx} = 2y$$

El comportamiento de muchos procesos físico-químicos en particular los que experimentan cambios en dependencia del tiempo se modelan mediante sistemas de ecuaciones diferenciales ordinarias (EDO). De ahí la importancia de los métodos de resolución de EDO. Si muchas de las EDO importantes se pueden resolver por métodos analíticos bien conocidos, hay otro gran número de EDO físicamente significativa que no es posible resolver por estos métodos, recurriendo entonces a métodos numéricos.

1.3.1. MÉTODOS PARA EL CÁLCULO DE RAÍCES DE FUNCIONES.

1.3.1.1. MÉTODO DE BISECCIÓN

Es el método más elemental y antiguo para determinar las raíces de una ecuación, de los llamados *de convergencia asegurada,* lo cual quiere decir que si se cumplen las condiciones de trabajo, el método converge hacia la raíz.

Este método consiste en aproximar la raíz de la ecuación como el punto medio del intervalo [*a, b*], minimizándose el error absoluto máximo:

$$x_k = \frac{a_k + b_k}{2}$$

Evaluando la función en este punto se decide si la raíz se encuentra en la mitad izquierda del intervalo o en la mitad derecha. De esta manera, una de las dos mitades queda descartada y la amplitud del nuevo intervalo de búsqueda es exactamente un medio de la anterior. A medida que este proceso se repite, el intervalo de búsqueda va disminuyendo en amplitud. (12)

Siendo a_k y b_k los límites del intervalo en cada iteración k ($0 \leq k \leq n$, con n igual al número de iteraciones totales), donde está contenida la raíz.

La elección del nuevo intervalo se hace de acuerdo con los siguientes criterios: (13)

$$Si\ \ f(a_k)f(x_k) < 0\ , a_{k+1} = a_k\ y\ b_{k+1} = x_k$$

$$Si\ \ f(x_k)f(b_k) < 0\ , a_{k+1} = x_k\ y\ b_{k+1} = b_k$$

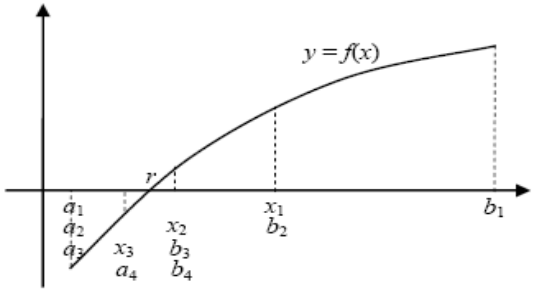

Fig. 1 Método de Bisección.

1.3.1.2. MÉTODO REGULA – FALSI

El nombre de este método proviene de una frase latina que significa regla inclinada y geométricamente consiste en tomar como aproximación de la raíz en el intervalo $[a_n, b_n]$ el punto de intersección con el eje x de un segmento que une los extremos del arco de la gráfica en ese intervalo. Por esta razón, también se le conoce como método de las cuerdas.

Puede considerarse como una codificación del método de bisección para mejorar la velocidad de convergencia. Para lograr una buena velocidad de convergencia, la gráfica de la función $f(x)$ debe presentar poca curvatura, en el caso extremo en que la gráfica sea lineal la convergencia se produce en una sola iteración.

Este, como en el método de la bisección, parte de dos puntos que rodean a la raíz. La siguiente aproximación, se calcula como la intersección con el eje x de la recta que une ambos puntos como en el método de la secante que se verá más adelante y la asignación del nuevo intervalo de búsqueda se realiza como en el método de la bisección.

Este método consiste en trazar una secante que une los puntos de la función evaluados en los extremos del intervalo donde se encuentra la raíz. Lo mismo que en el método de la bisección, el intervalo donde se encuentra la raíz se determina con la diferencia de signos en la función para los valores extremos del intervalo.

El cero de esta secante se toma como aproximación de la raíz de la función. Se usa esta aproximación para reemplazar uno de los dos extremos del último intervalo, de tal manera que se conserve el cambio de signo en el valor de la función en los extremos del nuevo intervalo de aproximación y se continúa la búsqueda en un nuevo subintervalo de acuerdo con el siguiente procedimiento:

$$x_k = \frac{f(b_k)a_k - f(a_k)b_k}{f(b_k) - f(a_k)}$$

$$x_k = b_k + f(b_k)\frac{(b_k - a_k)}{[f(a_k) - f(b_k)]}$$

Siendo a_k y b_k los límites del intervalo en cada iteración k ($0 \leq k \leq n$, con n igual al número de iteraciones totales) donde está contenida la raíz. (12) (13)

La elección del nuevo intervalo se hace de acuerdo con los siguientes criterios:

$$\text{Si } f(a_k)f(x_k) < 0, \quad a_{k+1} = a_k \ y \ b_{k+1} = x_k$$

$$\text{Si } f(x_k)f(b_k) < 0, \quad a_{k+1} = x_k \ y \ b_{k+1} = b_k$$

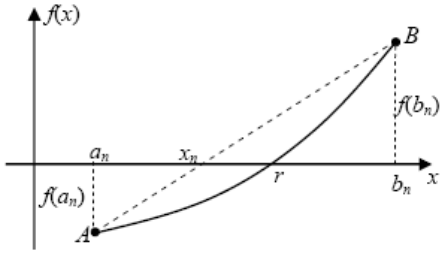

Fig. 2 Método de Regula-Falsi.

1.3.1.3. MÉTODO REGULA-FALSI MODIFICADO

Como su nombre lo indica, este método es una modificación del método de la falsa posición, y consiste en disminuir la pendiente de la secante. Esto se logra tomando como punto fijo el punto de la función definido por la última aproximación de la raíz y reduciendo a la mitad el valor de la función en el otro extremo del intervalo y con esto determinar el segundo punto de la secante. Se continúa con el proceso de la misma manera como en el método anterior. La aproximación de la raíz en cada etapa se hace con la fórmula: (13)

$$x_k = \frac{Ga_k - Fb_k}{G - F}$$

Siendo a_k y b_k los límites del intervalo en cada iteración k ($0 \leq k \leq n$, con n igual al número de iteraciones totales), donde está contenida la raíz y $G = f(b_k)$ y $F = f(a_k)$.

La elección del nuevo intervalo se hace de acuerdo con los siguientes criterios:

$$\text{Si } f(a_k)f(x_k) < 0\ , \qquad a_{k+1} = a_k$$
$$b_{k+1} = x_k$$
$$G = f(x_k)$$
$$F = F/2$$

$$\text{Si } f(b_k)f(x_k) < 0\ , \qquad a_{k+1} = x_k$$
$$b_{k+1} = b_k$$
$$F = f(x_k)$$

$$G = G/2$$

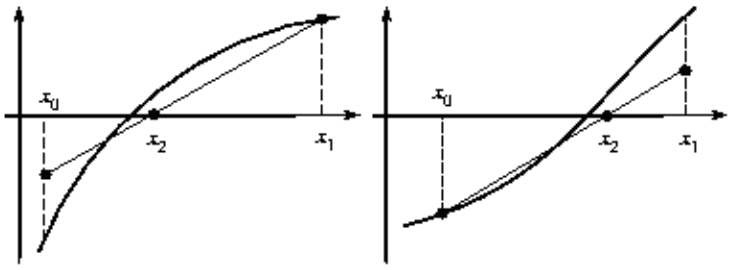

Fig. 3 Método de Regula-Falsi modificado.

1.3.1.4. MÉTODOS DE NEWTON-RAPHSON

En análisis numérico, el método de Newton-Raphson es un algoritmo eficiente para encontrar aproximaciones de los ceros o raíces de una función real. También puede ser usado para encontrar el máximo o mínimo de una función, encontrando los ceros de su primera derivada.

En este método se comienza con un valor razonablemente cercano al cero (denominado punto de arranque), se reemplaza la función por la recta tangente en ese valor, se iguala a cero y se despeja (fácilmente, por ser una ecuación lineal). Este cero será, generalmente, una aproximación mejor a la raíz de la función. Luego, se aplican tantas iteraciones como se deseen.

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$$

donde k es un número tal que $0 \leq k \leq n$, con n igual al número de iteraciones totales.

El orden de convergencia de este método es, por lo menos, cuadrático. (12) (13)

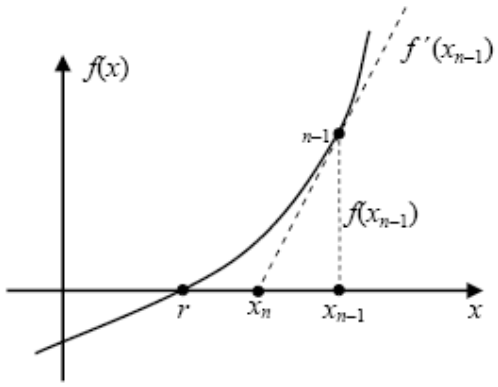

Fig. 4 Método de Newton-Raphson.

Para que el método converja las dos primeras derivadas de $\boldsymbol{f(x)}$ deben ser continuas y no nulas en el intervalo donde se encuentre la raíz, además $\boldsymbol{f(x) * f''(x) > 0}$.

1.3.1.5. APROXIMACIONES SUCESIVAS

Dada la ecuación $f(x) = 0$, el método de las aproximaciones sucesivas reemplaza esta ecuación por una equivalente, $x = g(x)$ definida en la forma $g(x) = f(x) + x$. Para encontrar la solución, se parte de un valor inicial x_0 y se calcula una nueva aproximación $x_1 = g(x_0)$. Reemplazando el nuevo valor obtenido y se repite entonces el proceso. Esto da lugar a una sucesión de valores $\{x_0, x_1, \dots, x_n\}$, que si converge, tendrá como límite la solución del problema. Sin embargo, el método puede divergir fácilmente; sólo podrá converger si la derivada $g'(x)$ es menor en valor absoluto que la unidad. (14) (13)

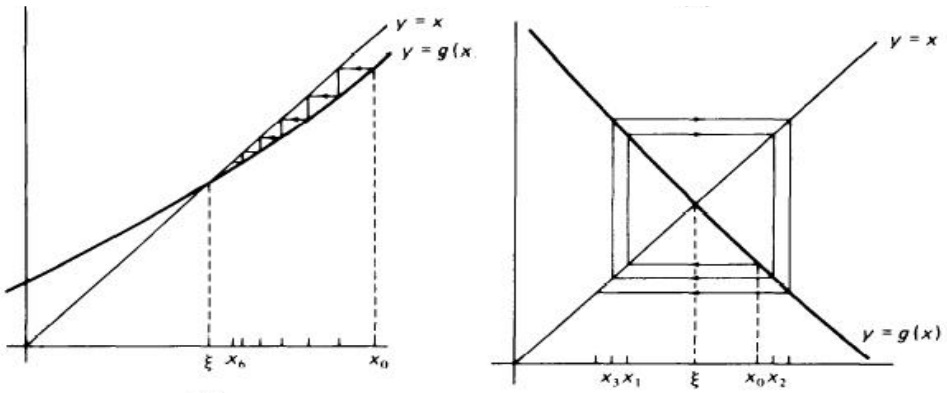

Fig. 5 Método de aproximaciones sucesivas.

1.3.1.6. MÉTODO DE LA SECANTE

El método de la secante es un método para encontrar los ceros de una función de forma iterativa. Es una modificación del método de Newton-Raphson dirigida a eliminar la necesidad de utilizar la función derivada. Para ello, se sustituye la pendiente de la recta tangente por la pendiente de una recta secante a la gráfica de $f(x)$. El método de las secantes requiere de dos aproximaciones iniciales de la raíz r ya que una secante se determina por dos puntos de la curva:

$$x_{k+1} = \frac{x_{k-1} f(x_k) - x_k f(x_{k-1})}{f(x_k) - f(x_{k-1})}$$

donde k es un número tal que $-1 \leq k \leq n$, con n igual al número de iteraciones totales.

Para el intervalo inicial de aproximación se hace la siguiente transformación: $x_{-1} = a_0$ y $x_0 = b_0$.

La convergencia de este método en un punto cercano a la solución, es el número áureo, por lo que se trata de una convergencia superlineal. En caso de que la aproximación inicial sea demasiado lejana, este método no asegura la convergencia y tiene un comportamiento similar al de Newton-Raphson. (12) (13)

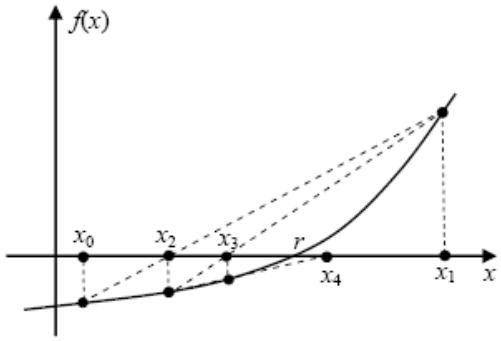

Fig. 6 Método de la secante.

1.3.1.7. Método de Steffensen

El método de Steffensen presenta una convergencia rápida y no requiere, como en el caso del método de la secante, la evaluación de derivada alguna. Presenta además, la ventaja adicional de que el proceso de iteración sólo necesita un punto inicial. Este método calcula el siguiente punto de iteración a partir de la expresión: (15)

$$x_{n+1} = x_n - \frac{[f(x_n)]^2}{f\big(x_n + f(x_n)\big) - f(x_n)}$$

1.3.1.8. Método de Wegstein

Este método, igual que el método de la falsa posición y el de la secante utiliza una recta secante a una curva para obtener aproximaciones de la raíz. Como en el método de las sustituciones sucesivas, requiere de la transformación de la función $f(x) = 0$ en la fórmula $x = F(x)$.

La aproximación de la raíz se hace con la fórmula algorítmica:

$$x_{k+1} = \frac{x_{k-1}F(x_k) - x_kF(x_{k-1})}{x_{k-1} - x_k + F(x_k) - F(x_{k-1})}$$

donde k es un número tal que $0 \leq k \leq n$, con n igual al número de iteraciones totales.

Para comenzar el método se necesitan dos aproximaciones iniciales. Una de ellas puede ser un punto arbitrario x_0, y el otro se puede generar utilizando la técnica de aproximaciones sucesivas. (14) (13)

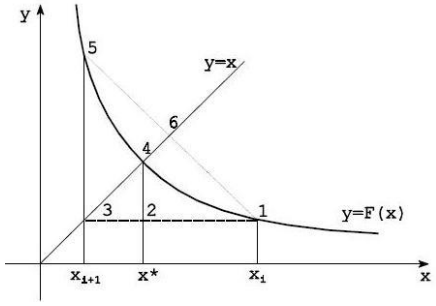

Fig. 7 Procedimiento de Wegstein.

1.3.2. MÉTODOS DE INTERPOLACIÓN

1.3.2.1. MÉTODO DE POLINOMIOS DE LAGRANGE

En análisis numérico, el polinomio de Lagrange, llamado así en honor a Joseph-Louis de Lagrange, es el polinomio que interpola un conjunto de puntos dado en la forma de Lagrange. Fue descubierto por Edward Waring en 1779 y redescubierto más tarde por Leonhard Euler en 1783.

Dado que existe un único polinomio interpolador para un determinado conjunto de puntos, resulta algo confuso llamar a este polinomio el polinomio interpolador de Lagrange. Un nombre más conciso es interpolación polinomial en la forma de Lagrange.

El método de Lagrange brinda un algoritmo eficiente para hallar el polinomio interpolador pero, además, permite encontrar el valor interpolado para una x específica sin necesidad de hallar la expresión analítica del polinomio interpolador.

Un polinomio de interpolación de Lagrange, p se define en la forma: (12)

$$p(x) = y_0 L_0(x) + y_1 L_1(x) + \cdots + y_n L_n(x) = \sum_{k=0}^{n} y_k L_k(x)$$

$$L_i(x) = \frac{(x - x_0)(x - x_1) \cdots (x - x_{i-1})(x - x_{i+1}) \cdots (x - x_n)}{(x_i - x_0)(x_i - x_1) \cdots (x_i - x_{i-1})(x_i - x_{i+1}) \cdots (x_i - x_{n1})} \qquad i = 1,2, \ldots, n$$

1.3.2.2. DIFERENCIAS DIVIDIDAS DE NEWTON

La idea fundamental del método de Newton es realizar la interpolación en un punto de forma sucesiva: partiendo de dos nodos ir agregando los demás, uno por uno, en el orden que se desee, de tal manera que en cada paso solo se requiera agregar un nuevo término a los cálculos precedentes. El método permite, sin realizar ninguna operación adicional, ir obteniendo en cada paso del proceso una estimación del error de interpolación, de manera que el proceso iterativo se pueda detener si se alcanza un error suficientemente pequeño. (12)

donde:

$$f[x_0, x_1] = \frac{y_1 - y_0}{x_1 - x_0}$$

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0}$$

$$\vdots$$

$$f[x_0, x_1, \cdots, x_k] = \frac{f[x_1, x_2, \cdots, x_k] - f[x_0, x_1, \cdots, x_{k-1}]}{x_k - x_0} \qquad k = 1,2, \dots, n$$

quedando:

$$p_n(x) = f(x_0) + f[x_0, x_1](x - x_0) + \cdots + f[x_0, x_1, \cdots, x_n](x - x_0)(x - x_1) \cdots (x - x_{n-1})$$

1.3.2.3. INTERPOLACIÓN POR SPLINE

Una función spline es una función polinomial por tramos que es continua y posee derivadas continuas hasta un cierto orden. En nuestro caso nos centraremos en los splines cúbicos.

$$s(x) = \begin{cases} a_1x^3 + b_1x^2 + c_1x + d_1 \;\; si \;\; x_0 < x < x_1 \\ \vdots \\ a_nx^3 + b_nx^2 + c_nx + d_n \;\; si \;\; x_{n-1} < x < x_n \end{cases}$$

Cada uno de los n polinomios posee 4 coeficientes por lo que el spline tendrá $4n$ condiciones que son:

- Condición de interpolación $s(x_i) = y_i \qquad i = 1,2,3, \dots, n$
- Condición continuidad $s(x) = en\ continua\ en\ x_i \qquad i = 1,2,3, \dots, n-1$
- Condición de suavidad $s'(x) = en\ continua\ en\ x_i \qquad i = 1,2,3, \dots, n-1$

$$s''(x) = en\ continua\ en\ x_i \qquad i = 1,2,3,\dots,n-1$$

Estas serian $4n-2$ condiciones por lo que nos quedan 2 por imponer. Estas suelen ponerse de diversas formas para lograr distintos propósitos.

Para determinar las formulas de los splines utilizaremos:

$$M_i = s''(x_i) \quad i = 1,2,3,\dots,n$$

$$h_i = x_{i+1} - x_i \quad i = 1,2,3,\dots,n-1$$

Y resolveremos un problema de tipo $HM = Y$, donde H es una matriz tridiagonal formada con ayuda de las longitudes de los tramos en que están definidos los n polinomios, Y matriz columna formada con ayuda de los valores de las imágenes de los nodos y M la matriz columna de incógnitas del sistema de ecuaciones lineales. (12)

1.3.2.3.1. SPLINE CÚBICO NATURAL

Este es el spline interpolador más empleado. Se obtiene añadiendo las dos condiciones:

$$s''(x_0)=M_0=0 \quad \text{y} \quad s''(x_n)=M_n=0$$

Con estas condiciones el spline que resulta es la función que hace mínima la integral:

$$\int_{x_0}^{x_n} [g''(x)]^2 dx$$

Lo que significa geométricamente que minimiza la curvatura de la función interpoladora.

Con estas condiciones el sistema de ecuaciones para el spline natural queda: (12)

$$\frac{h_0+h_1}{3}M_1 + \frac{h_1}{6}M_2 = \frac{y_2-y_1}{h_1} - \frac{y_1-y_0}{h_0}$$

$$\frac{h_1}{6}M_1 + \frac{h_1+h_2}{3}M_2 + \frac{h_2}{6}M_3 = \frac{y_3-y_2}{h_2} - \frac{y_2-y_1}{h_1}$$

$$\vdots$$

$$\frac{h_{n-2}}{6}M_{n-2} + \frac{h_{n-2}+h_{n-1}}{3}M_{n-1} = \frac{y_n-y_{n-1}}{h_{n-1}} - \frac{y_{n-1}-y_{n-2}}{h_{n-2}}$$

1.3.2.3.2. SPLINE CÚBICO ANCLADO

En el spline interpolador anclado las dos condiciones adicionales que se toman son: $s'(x_0+)=m_0$ y $s'(x_n-)=m_n$, es decir, se fijan las pendientes en los extremos del spline a valores deseados m_0 y m_n, lo cual es muy importante en muchas aplicaciones.

Ahora el sistema queda: (12)

$$\frac{h_0}{3}M_0+\frac{h_0}{6}M_1=\frac{y_1-y_0}{h_0}-m_0$$

$$\frac{h_0}{6}M_0+\frac{h_0+h_1}{3}M_1+\frac{h_1}{6}M_2=\frac{y_2-y_1}{h_1}-\frac{y_1-y_0}{h_0}$$

$$\vdots$$

$$\frac{h_{n-2}}{6}M_{n-2}+\frac{h_{n-2}+h_{n-1}}{3}M_{n-1}+\frac{h_{n-1}}{6}M_n=\frac{y_n-y_{n-1}}{h_{n-1}}-\frac{y_{n-1}-y_{n-2}}{h_{n-2}}$$

$$\frac{h_{n-1}}{6}M_{n-1}+\frac{h_{n-1}}{3}M_n=m_n-\frac{y_n-y_{n-1}}{h_{n-1}}$$

1.3.2.3.3. SPLINE CÚBICO PERIÓDICO

Si al spline interpolador se le imponen las condiciones $y_0=y_n$, $s'(x_0+)=s'(x_n-)$ y $s''(x_0+)=s''(x_n-)$ se logra una función cuya gráfica es tal que, si ella se repite una y otra vez, en los puntos de unión la función es continua y son continuas las derivadas de orden 1 y 2.

El sistema a resolver: (12)

$$\frac{h_0}{6}M_0+\frac{h_0+h_1}{3}M_1+\frac{h_1}{6}M_2=\frac{y_2-y_1}{h_1}-\frac{y_1-y_0}{h_0}$$

$$\vdots$$

$$\frac{h_{n-2}}{6}M_{n-2}+\frac{h_{n-2}+h_{n-1}}{3}M_{n-1}+\frac{h_{n-1}}{6}M_n=\frac{y_n-y_{n-1}}{h_{n-1}}-\frac{y_{n-1}-y_{n-2}}{h_{n-2}}$$

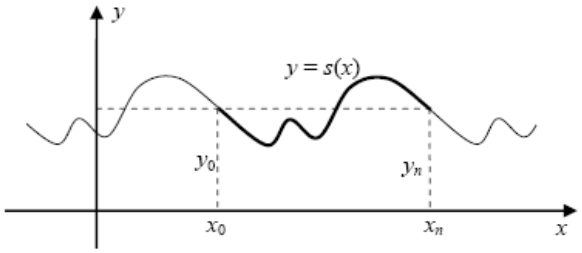

Fig. 8 Spline cúbico periódico.

1.3.3. MÉTODOS DE AJUSTE DE CURVAS

Sea $A=\{x_1,x_2,\dots,x_m\}$ un conjunto de valores de una variable x entre los cuales pueda existir cualquier orden e incluso, pueda existir repeticiones. Sea f una función definida en A, que toma valores $f_j=f(x_j)$ con $j=1,2,\dots m$, los cuales usualmente se desconocen. Sea y_j el valor observado de f_j.

Sea G una familia de funciones de aproximaciones, el problema de ajuste de curvas consiste, en encontrar aquella función g de la familia G que se aproxime lo mejor posible a los datos (x_j,y_j) con $j=1,2,\dots m$. En términos mas formales, se trata de encontrar la función $g\,\epsilon\,G$ que haga mínima la desviación cuadrática:

$$D=\sum_{j=1}^{m}[g(x_j)-y_j]^2$$

1.3.3.1. AJUSTE POR MÍNIMOS CUADRADOS

El método se apoya en la minimización de la suma del cuadrado del error dado por la función a la que se está aproximando la función original. El error está expresado como:

$$Q=\sum_{i=1}^{N}(y_i-\widehat{Y}_i)^2$$

donde:

y_i: valores de la función original.

$\widehat{Y}_i$: valores obtenidos con la función propuesta como aproximación.

Los parámetros o coeficientes que aparecen en la fórmula $\widehat{Y}_i$ se ajustan o determinan minimizando la función error Q. Esta minimización es posible realizarla analíticamente para algunas formas sencillas de la función $\widehat{Y}_i$.

Cuando la función $\hat{Y}$es de forma polinomial:

$$\hat{\hat{Y}} = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Para el caso en que $n = 3$, las ecuaciones que se producen son:

$$\sum a_0 + a_1 \sum x_i + a_2 \sum {x_i}^2 + a_3 \sum {x_i}^3 = \sum y_i$$

$$a_0 \sum x_i + a_1 \sum {x_i}^2 + a_2 \sum {x_i}^3 + a_3 \sum {x_i}^4 = \sum y_i x_i$$

$$a_0 \sum {x_i}^2 + a_1 \sum {x_i}^3 + a_2 \sum {x_i}^4 + a_3 \sum {x_i}^5 = \sum y_i {x_i}^2$$

$$a_0 \sum {x_i}^3 + a_1 \sum {x_i}^4 + a_2 \sum {x_i}^5 + a_3 \sum {x_i}^6 = \sum y_i {x_i}^3$$

Este sistema de ecuaciones lineales simultáneas en a_i se resuelve por cualquier método, entre ellos, los incluidos en este módulo, en la sección de solución de ecuaciones.

En el caso en que $\hat{Y}$ tenga una forma como:

$$\hat{Y} = \frac{a}{b + k}$$

Las ecuaciones que resultan para evaluar los parámetros a y b son no lineales

$$\frac{\partial Q}{\partial a} = 2 \sum \left[\left(y_i - \frac{a}{b + x_i} \right) \left(-\frac{1}{b + x_i} \right) \right] = 0$$

$$\frac{\partial Q}{\partial b} = 2 \sum \left[\left(y_i - \frac{a}{b + x_i} \right) \left(\frac{a}{(b + x_i)^2} \right) \right] = 0$$

de donde:

$$\sum\left[\left(y_i - \frac{a}{b + x_i}\right)\left(\frac{1}{b + x_i}\right)\right] = 0$$

$$\sum\left[\left(y_i - \frac{a}{b + x_i}\right)\left(\frac{1}{b + {x_i}^2}\right)\right] = 0$$

Estas dos ecuaciones pueden resolverse utilizando el método de Newton-Raphson para sistemas de ecuaciones lineales. (16)

1.3.4. MÉTODOS DE SOLUCIÓN DE EDO

1.3.4.1. EXPANSIÓN EN SERIES DE TAYLOR

En matemáticas, la serie de Taylor de una función f infinitamente derivable (real o compleja) definida en un intervalo abierto $(a - r, a + r)$ se define con la siguiente suma:

$$\sum_{n=0}^{\infty} \frac{f^{(n)}(a)}{n!}(x - a)^n$$

Aquí, $n!$ es el factorial n y $f^{(n)}(a)$ indica la n-ésima derivada de f en el punto a.

Si esta serie converge para todo x perteneciente al intervalo $(a - r, a + r)$ y la suma es igual a $f(x)$, entonces la función $f(x)$ se llama analítica. Para comprobar si la serie converge a $f(x)$, se suele utilizar una estimación del resto del teorema de Taylor. Una función es analítica si y solo si se puede representar con una serie de potencias; los coeficientes de esa serie son necesariamente los determinados en la fórmula de la serie de Taylor.

Si $a = 0$, a la serie se le llama serie de Maclaurin. (12) (14)

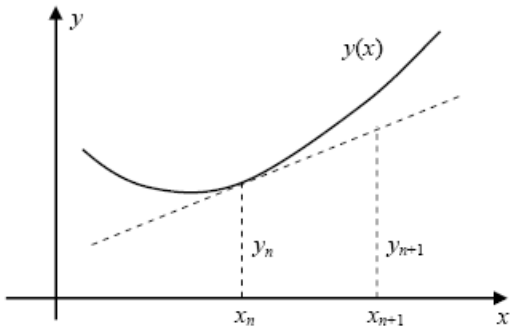

Fig. 9 Series de Taylor.

1.3.4.2. MÉTODO DE EULER

El método de Euler es el más elemental de los métodos de paso simple para resolver ecuaciones diferenciales ordinarias de la forma:

$$\frac{dy}{dx} = f(x, y)$$

con:

$$y(x_0) = y_0$$

Como se conoce el primer elemento de la solución, se puede determinar el campo de direcciones en ese punto. Para hallar el segundo punto de la solución aproximada se sigue una trayectoria rectilínea en esa dirección hasta alcanzar la abscisa.

La solución del método de Euler converge hacia la solución de la ecuación diferencial, cuando esta es inestable, el método de Euler también tiene que serlo.

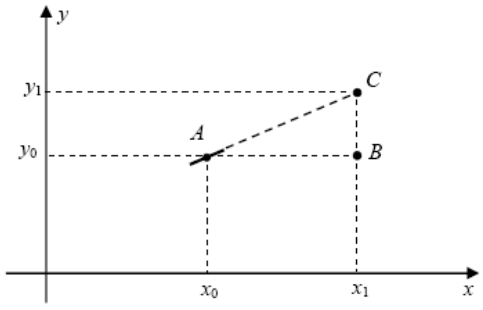

Fig. 10 Método de Euler.

Expresado lo anterior matemáticamente quedaría:

$$y_{n+1} = y_n + (x_n - x_{n-1}) f(x_{n-1}, y_{n-1})$$

Y si llamamos a $h = (x_n - x_{n-1})$ quedaría:

$$y_{n+1} = y_n + hf(x_{n-1}, y_{n-1})$$ (12) (14)

1.3.4.3. MÉTODOS DE RUNGE-KUTTA

Runge-Kutta no es solo un método sino una importante familia de métodos iterativos tanto implícitos como explícitos para aproximar las soluciones de ecuaciones diferenciales ordinarias. Este conjunto de métodos fue desarrollado alrededor del año 1900 por los matemáticos Carl David Tolmé Runge y Martin Wilhelm Kutta.

Se trata de un método por etapas que resuelve el problema de hallar las derivadas parciales de $f(x, y)$ del método de Taylor, evaluando la función $f(x, y)$ en varios puntos. Las formulas mas comúnmente usadas de esta familia de métodos son las de segundo y cuarto orden y tienen la forma: (12) (14)

$$RK2: \begin{cases} K_1 = hf(x_n, y_n) \\ K_2 = hf(x_n + h, y_n + K_1) \\ y_{n+1} = y_n + \dfrac{K_1 + K_2}{2} \end{cases}$$

$$RK4: \begin{cases} K_1 = hf(x_n, y_n) \\ K_2 = hf\left(x_n + \dfrac{h}{2}, y_n + \dfrac{K_1}{2}\right) \\ K_3 = hf\left(x_n + \dfrac{h}{2}, y_n + \dfrac{K_2}{2}\right) \\ hf(x_n + h, y_n + K_3) \\ y_{n+1} = y_n + \dfrac{K_1 + 2K_2 + 2K_3 + K_4}{6} \end{cases}$$

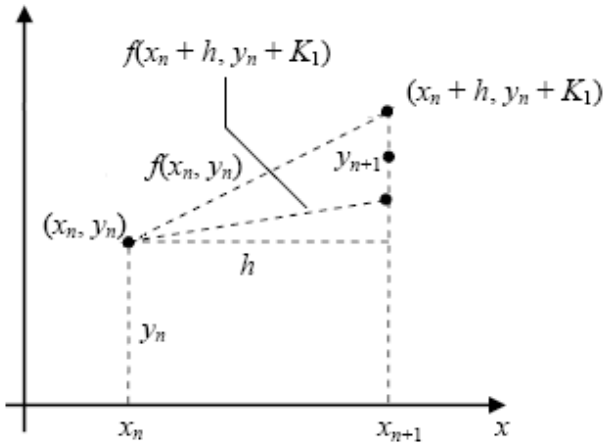

Fig. 11 Métodos de Runge-Kutta.

1.3.4.4. MÉTODOS DE ADAMS-BASHFORTH

Los métodos de Adams-Bashforth forman una familia de métodos de paso múltiple típica. Siendo uno de los más usados el de orden cuatro con paso cuádruple:

$$y_{n+1} = y_n + \frac{h}{24}(55f_n - 59f_{n-1} + 37f_{n-2} - 9f_{n-3})$$

Estos utilizan un polinomio interpolador en el intervalo $[x_{n-p}, x_n]$ para aproximar la función $f(x, y(x))$ en el intervalo $[x_n, x_{n+1}]$, siendo esta la causa principal de la menor exactitud de estos métodos. (12) (14)

1.3.4.5. MÉTODOS DE ADAMS-MOULTON

Los métodos de Adams-Moulton resuelven el problema de exactitud de los métodos de Adams-Bashforth tomando los nodos de interpolación en $[x_{n-p}, x_{n+1}]$, lo cual aumenta considerablemente la exactitud, pero introduce nuevas dificultades. En este caso se utiliza el método de cuarto orden con paso triple: (12) (14)

$$y_{n+1} = y_n + \frac{h}{24}(9f_{n+1} + 19f_n - 5f_{n-1} + f_{n-2})$$

1.3.4.6. MÉTODOS PREDICTORES-CORRECTORES

Los métodos predictores-correctores están constituidos por un método implícito de buena exactitud (ecuación correctora) y un método explícito (ecuación predictora) que aporta una aproximación inicial adecuada para el proceso iterativo del método implícito. Los algoritmos de Adams-Bashforth y Adams-Moulton suelen emplearse para formar pares predictor-corrector. Para ello se seleccionan métodos del mismo orden, de modo que la aproximación inicial sea suficientemente buena para que la ecuación correctora no haya que aplicarla más de una o dos veces. (12) (14)

1.4. ANÁLISIS DE SOLUCIONES EXISTENTES

1.4.1. MATLAB

MATLAB es un lenguaje de computación técnica de alto nivel y un entorno interactivo para desarrollo de algoritmos, visualización de datos, análisis de datos y cálculo numérico. Con MATLAB, se podrán resolver problemas de cálculo técnico más rápidamente que con lenguajes de programación tradicionales, tales como C, C++ y FORTRAN.

Se puede usar MATLAB en una amplia gama de aplicaciones que incluyen procesamiento de señales e imágenes, comunicaciones, diseño de sistemas de control, sistemas de prueba y medición, modelado y análisis financiero y biología computacional. Los conjuntos de herramientas complementarios (colecciones de funciones de MATLAB para propósitos especiales, que están disponibles por separado) amplían el entorno de MATLAB permitiendo resolver problemas especiales en estas áreas de aplicación.

Además, MATLAB contiene una serie de funciones para documentar y compartir su trabajo. Se puede integrar el código de MATLAB con otros lenguajes y aplicaciones, y distribuir los algoritmos y aplicaciones desarrolladas usando MATLAB.

Características principales:

- Lenguaje de alto nivel para cálculo técnico.
- Entorno de desarrollo para la gestión de código, archivos y datos.

- Herramientas interactivas para exploración, diseño y resolución de problemas iterativos.
- Funciones matemáticas para álgebra lineal, estadística, análisis de Fourier, filtraje, optimización e integración numérica.
- Funciones gráficas bidimensionales y tridimensionales para visualización de datos.
- Herramientas para crear interfaces gráficas de usuario personalizadas.
- Funciones para integrar los algoritmos basados en MATLAB con aplicaciones y lenguajes externos, tales como C/C++, FORTRAN, Java, COM y Microsoft Excel.

Plataformas:

Unix, Windows y Apple Mac OS X. (17)

1.4.2. DERIVE

Asistente matemático para la resolución de problemas donde se encuentren involucrados elementos de álgebra, ecuaciones, trigonometría, vectores y matrices. Con él se simplifica la resolución de problemas numéricos y simbólicos, y los resultados pueden representarse como gráficos 2D o superficies 3D.

Características principales:

- Gráficos 2D: explícitos, implícitos y paramétricos; coordenadas rectangulares y polares; funciones de variable compleja; especificación de colores; etiquetaje de ejes y anotaciones sobre los gráficos.
- Gráficos 3D: mallado para funciones de dos variables; selección del punto de vista; cambio de escala; rotación de gráficos en tiempo real.
- Álgebra: desarrollo y factorización de polinomios; simplificación de expresiones algebraicas; resolución de numérica y simbólica; resolución de sistemas lineales de ecuaciones.
- Aritmética: aritmética exacta y aritmética aproximada de precisión configurable; factorización de enteros; conversión de unidades métricas e inglesas; constantes físicas fundamentales.
- Cálculo: cálculo simbólico de límites finitos e infinitos; primera y n-ésima derivadas parciales; integrales definidas e indefinidas; integración numérica; sumas y productos finitos e infinitos; derivación implícita y paramétrica; desarrollos de Taylor y series de Fourier; longitud de arco, áreas y volúmenes; transformadas de Laplace; resolución exacta de EDOs de primer y segundo orden; resolución numérica por Runge-Kutta de sistemas de EDOs.

- Funciones: exponencial, trigonométricas, hiperbólicas, definidas a trozos.
- Programación: definición de funciones; construcciones de control If-then-else; operadores relacionales y booleanos; funciones recursivas e iterativas.
- Vectores, matrices y conjuntos: elementos numéricos y simbólicos; notación estándar de subíndice; producto escalar, matricial y externo; traspuesta, determinante, inversa y traza; valores y vectores propios; algebra no conmutativa; cálculo vectorial diferencial e integral; álgebra tensorial.

Plataformas soportadas y requerimientos:

- Windows® 98, Me, 2000 o XP.
- Los requerimientos mínimos de RAM y procesador son los mismos que los del sistema operativo.
- Unidad de CD ROM.
- 10 MB de espacio en el disco duro. (18)

1.4.3. MATHEMATICA

Mathematica 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.

Características principales:

- Realización de cálculos y simulaciones de cualquier nivel de complejidad mediante el uso de la amplia librería 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.

Plataformas:

Windows NT/2000/XP/Vista, Mac OS X, Linux x86/Itanium, Solaris UltraSPARC/x86, HP-UX, IBM AIX, y sistemas operativos compatibles. (19)

1.4.4. MAPLE

Maple es una potente herramienta, tecnológicamente avanzada, que incorpora algoritmos simbólicos propios reconocidos en todo el mundo. Así mismo Maple incorpora desde su versión 6 los prestigiosos resolvedores numéricos proporcionados por su socio *Numerical Algorithms Group* (NAG).

Características principales:

Maple incorpora más de 3000 funciones para cálculo simbólico y numérico entre las que se incluyen funciones para:

- Álgebra: aritmética simbólica con números reales y complejos o polinomios, factorización, expansión, combinación y simplificación de expresiones algebraicas y polinomios, secuencias y series.
- Cálculo: Derivadas, integrales y límites, rutinas de visualización para diferenciación e integración.
- Ecuaciones diferenciales: Resolución numérica y exacta de ecuaciones y sistemas de ecuaciones diferenciales ordinarias (EDO) y problemas de valor inicial, resolución numérica de problemas de valores de contorno, resolución exacta de ecuaciones y sistemas de ecuaciones en derivadas parciales (EDP), análisis estructural y reducción de orden de EDOs y EDPs.
- Álgebra Lineal: Más de 100 funciones para construir, resolver y programar en álgebra lineal, construcción de matrices de Hankel, Hilbert, identidad, Toeplitz, Vandermonde, Bezout y la matriz Silvester de dos polinomios.
- Cálculo Vectorial: Derivadas direccionales, gradientes, matrices Hessiana, Laplacianas, rotacionales y divergencias de un campo vectorial, matrices Jacobianas y Wronskian, productos escalares, vectoriales y externos de vectores y operadores diferenciales.
- Otras funciones: funciones para álgebras abstractas, álgebra de operadores lineales, curvas algebraicas, funciones y estructuras combinatorias, variables complejas, ajuste de curvas, álgebra diferencial, matemática financiera, series de potencia, teoría de grafos, programación lineal, lógica, estadística, etc.
- Programación: Maple da acceso al mismo lenguaje de programación, herramientas y rutinas básicas con las que ha sido desarrollado. Tiene un lenguaje de programación avanzado

que incluye programación funcional y procedural, sobrecarga de operadores, manipulación de excepciones, herramientas de depuración, etc.

- Visualización: Incluye un amplio conjunto de herramientas de visualización con gráficos típicos predefinidos, gráficos 2D y 3D, animaciones 2D y 3D, una amplia variedad de tipos de coordenadas, gráficos implícitos 2D y 3D, gráficos vectoriales, contornos, gráficos complejos, gráficos de EDOs y EDPs, rotación en tiempo real, objetos geométricos predefinidas, iluminación.
- Interfaz de usuario: Maple utiliza hojas de cálculo, tiene amplias capacidades de edición y procesado de textos, gestor de hiperenlaces, menús contextuales, paletas, exportación a HTML, LaTeX y RTF
- Conectividad: Maple está adherido a los estándares internacionales para comunicación de datos soportando un amplio número de formatos.

Plataformas soportadas:
Windows, Macintosh, Linux y UNIX. (20)

1.4.5. BIBLIOTECAS NUMÉRICAS NAG

Esta BIBLIOTECA proporciona más de 1.280 de las renovadas funciones Fortran de NAG, que abarcan un amplio rango de aplicaciones matemáticas y estadísticas. Los algoritmos Fortran de NAG se integran fácilmente en aplicaciones escritas en un extenso número de lenguajes.

Características principales:

- 95 nuevas rutinas para atender la evolución de los requerimientos de la industria.
- NAG proporciona una librería Fortran que es 100% *thread safe*.
- Más de 1.110 rutinas de usuario compiladas y probadas con opciones *thread safe*.
- Ecuaciones en derivadas parciales: resolvedores *Black-Scholes*.
- Generación de mallas.
- Optimización.
- Problemas de valores propios generalizados: rutinas LAPACK.
- Algebra lineal dispersa: un precondicionador Jacobi.
- Generadores de números cuasi-aleatorios.
- Análisis de series temporales GARCH: rutinas de estimación y previsión.
- Aproximaciones de funciones especiales.
- Ceros de polinomios. (21)

1.4.6. YORICK

Yorick es un lenguaje de programación interpretado para cálculos o simulaciones científicas, incluye un paquete de gráficos interactivos, y un paquete de archivos binario capaz de traducir desde y hacia los formatos numéricos en bruto de todos los ordenadores modernos. Yorick está escrito en ANSI C, y funciona en la mayoría de sistemas operativos (* nix sistemas, Windows, MacOS).

Yorick tiene una sintaxis compacta, similar al C, pero con los operadores de matriz. Es fácilmente ampliable a través de la vinculación dinámica de las bibliotecas C, permite la eficiente manipulación arbitraria de tamaño de los vectores, y ofrece una amplia gama de capacidades gráficas. (22)

1.4.7. YACAS

Yacas es un programa de álgebra computacional (SAC) de uso simple, de código abierto, y de propósito general. Su nombre es un acrónimo de *Yet Another Computer Algebra System*, su equivalente en español sería similar a: "Otro Programa Más de Algebra Computacional".

Diseñado en su propio lenguaje de programación, en los que nuevos algoritmos pueden ser fácilmente implementados. Este lenguaje se encuentra cercanamente emparentado con LISP WH89, e incluye la transformación de expresiones (re-escritura de términos) como una característica básica de este lenguaje.

Maneja entradas y salidas en archivos de texto ASCII o en OpenMath, tanto de forma interactiva, tanto individual como por lotes de archivos. (23)

1.4.8. YURIX

Posee un intérprete de lenguaje de programación de alto nivel que nos da la posibilidad no solo de crear funciones y más funciones como Matlab, sino nos sumerge en el mundo de la orientación a objetos, que desde luego es de mucha ayuda sobre todo en el desarrollo de simulaciones y además facilita la integración con las plataformas .NET como Mono desarrollada por Novell, todo esto desde un código fuente.

Yurix al igual que Matlab y Octave tiene un lenguaje diseñado para el fácil acceso a paquetes de algoritmos desarrollados por la Universidad de California, Berkeley y otras más, y publicados bajo licencia BSD, el paquete consiste en cientos de funciones de álgebra lineal desarrollados en el lenguaje Fortran, además de funciones propias de Yurix. El lenguaje que utiliza el software es similar al de Matlab u Octave, lo cual hace que la documentación sea reutilizada para Yurix.

Para el cálculo numérico usa variables tipo carácter, escalares, matriciales, complejas o tipo celdas (que es una matriz que puede contener otras matrices, cadenas de caracteres y los demás tipos), para el fácil manejo de base de datos, iguales al de Matlab u Octave, pero en Yurix adicionales a esas tiene un nuevo tipo de variable llamada Object, con el objetivo de facilitar el desarrollo de simulaciones.

En Yurix se puede desarrollar aplicaciones con Interfaz de Usuario Gráfica (GUI), que mediante ventanitas y la orientación a objetos, se optimizará el proceso de producción de software.

Posee funciones para el manejo de expresiones algebraicas matriciales y simbólicas, gráfica de funciones en 2D.

Está provisto de una GUI en un completo Entorno de Desarrollo Integrado, que posee herramientas como el editor, el espacio de trabajo, nuestra capeta de trabajo, historial de la línea de comandos, ayuda interactiva al escribir funciones en línea de comandos, etc. (24)

1.4.9. GNU OCTAVE

Octave o GNU Octave es un programa libre para realizar cálculos numéricos. Como indica su nombre es parte de proyecto GNU. MATLAB es considerado su equivalente comercial. Entre varias características que comparten se puede destacar que ambos ofrecen un intérprete permitiendo ejecutar órdenes en modo interactivo. Nótese que Octave no es un sistema de álgebra computacional como podría ser GNU Maxima, sino que usa un lenguaje que está orientado al análisis numérico.

El proyecto fue creado alrededor del año 1988 pero con una finalidad diferente: ser utilizado en un curso de diseño de reactores químicos. Posteriormente en el año 1992, se decide extenderlo y comienza su desarrollo a cargo de John W. Eaton. La primera versión alpha fue lanzada el 4 de enero de 1993. Un año más tarde, el 17 de febrero, 1994 aparece la versión 1.0.

El nombre surge del nombre de un profesor de unos de los autores conocido por sus buenas aproximaciones por medio de cálculos mentales a problemas numéricos.

Detalles técnicos:

- Octave está escrito en C++ usando la biblioteca STL.
- Tiene un intérprete de su propio lenguaje (de sintaxis similar a Matlab), y permite una ejecución interactiva o por lotes.
- Puede extenderse el lenguaje con funciones y procedimientos por medios de módulos dinámicos.
- Utiliza otros programas GNU para ofrecer al usuario crear gráficos para luego imprimirlos o guardarlos (Grace).
- Dentro del lenguaje también se comporta como una consola de órdenes (shell). Esto permite listar contenidos de directorios, por ejemplo.
- Además de correr en plataformas Unix también lo hace en Windows.
- Puede cargar archivos con funciones de Matlab de extensión *.m*.

Lenguaje octave:

- La sintaxis es similar a la utilizada en MATLAB.
- Es un lenguaje interpretado.
- No permite pasar argumentos por referencia. Siempre se pasan por valor.
- No permite punteros.
- Se pueden generar scripts.
- Soporta gran parte de las funciones de la librería estándar de C.
- Puede extenderse para ofrecer compatibilidad a las llamadas al sistema UNIX.
- El lenguaje está pensado para trabajar con matrices y provee mucha funcionalidad para trabajar con éstas.
- No es un lenguaje de programación orientado a objetos. Por lo tanto, no tiene clases ni objetos.
- Soporta estructuras similares a los *structs* de C.

Al ser su licencia GNU (*General Public License*), puede ser copiado y utilizado libremente. (25)

1.5. CONCLUSIONES PARCIALES

Se ha podido conocer la base teórica que fundamenta esta investigación, se trataron conceptos fundamentales relacionados con esta, se acercó al lector al conocimiento de los métodos numéricos y su uso en la actualidad. Se evidencia cómo, a pesar de existir distintas herramientas para el uso y trabajo con métodos numéricos no resuelven el problema inicialmente planteado siendo el desarrollo de la BMN beneficioso para el desarrollo de futuras aplicaciones.

Ya conocidos de forma independiente los métodos numéricos a implementar en la BMN se puede realizar una investigación para analizar las herramientas más factibles que posibiliten el desarrollo de la misma.

CAPÍTULO 2: TENDENCIAS Y TECNOLOGÍAS ACTUALES A DESARROLLAR.

2.1. INTRODUCCIÓN

En este capítulo realizaremos un análisis de las tecnologías y tendencias mundiales actuales que pudieran ser útiles en el desarrollo de nuestra biblioteca. Tocaremos las metodologías de desarrollo posibles a emplear, así como lenguajes de programación, entornos de desarrollo, etc. Estableciendo comparaciones entre ellos para poder ir decantando y seleccionar los que mejores se ajusten a nuestros intereses.

2.2. EL LENGUAJE UNIFICADO DE MODELADO COMO SOPORTE A LA PROGRAMACIÓN ORIENTADA A OBJETOS

La ingeniería es la profesión en la que el conocimiento de las matemáticas y ciencias naturales, obtenido mediante estudio, experiencia y práctica, se aplica con juicio para desarrollar formas de utilizar, económicamente, los materiales y las fuerzas de la naturaleza para beneficio de la humanidad. (26)

Por otro lado el modelado, o modelización, es una técnica cognitiva que consiste en crear una representación ideal de un objeto real mediante un conjunto de simplificaciones y abstracciones, cuya validez se pretende constatar. La validación del modelo se lleva a cabo comparando las implicaciones predichas por el mismo con observaciones. (27)

Lenguaje Unificado de Modelado (UML, por sus siglas en inglés, *Unified Modeling Language*) es el lenguaje de modelado de sistemas de software más conocido y utilizado en la actualidad. Es un lenguaje gráfico para visualizar, especificar, construir y documentar un sistema de software. UML ofrece un estándar para describir un "plano" del sistema (modelo), incluyendo aspectos conceptuales tales como procesos de negocios y funciones del sistema, y aspectos concretos como expresiones de lenguajes de programación, esquemas de bases de datos y componentes de software reutilizables.

Es importante resaltar que UML es un "lenguaje" para especificar y no para describir métodos o procesos. Se utiliza para definir un sistema de software, para detallar los artefactos en el sistema y para documentar y construir. En otras palabras, es el lenguaje en el que está descrito el modelo. Se puede aplicar en una gran variedad de formas para dar soporte a una metodología de desarrollo de software, pero no especifica en sí mismo qué metodología o proceso usar. (1)

Tiene una gran cantidad de propiedades que han sido las que, realmente, han contribuido a hacer de UML el estándar de la industria en la actualidad.

Algunas de las propiedades de UML como lenguaje de modelado son: (28)

- Es un lenguaje distribuido y adecuado a las necesidades de conectividades actuales y futuras. Ampliamente utilizado por la industria del software.
- Remplaza a decenas de notaciones empleadas por otros lenguajes.
- Modela estructuras complejas.
- Las estructuras más importantes que soporta tienen su fundamento en la tecnología orientada a objeto, tales como objetos, clases, componentes y nodos.
- Comportamiento del sistema: casos de usos, diagramas de secuencia, de colaboración, que sirve para evaluar el estado de las máquinas.

Se utilizará como notación el Lenguaje Unificado de Modelado para lograr un mayor entendimiento ya que se logra modelar y describir secuencialmente por pasos todos los procesos que se llevan a cabo según la problemática planteada. Sirve porque es un lenguaje gráfico para visualizar, especificar, construir y documentar los artefactos de un sistema de software.

2.3. METODOLOGÍAS DE DESARROLLO DE SOFTWARE

El desarrollo de software no es una tarea fácil. Prueba de ello es que existen numerosas propuestas metodológicas que inciden en distintas dimensiones del proceso de desarrollo. Debido a la diversidad de propuestas nos enfrentamos a un problema en el momento de decidir qué metodología de desarrollo utilizar.

El objetivo de una metodología de desarrollo es subir la calidad del software (en todas las fases por las que pasa) a través de una mayor transparencia y control sobre el proceso. Es labor de la

metodología de desarrollo hacer que esas medidas, para aumentar la calidad, sean reproducibles en cada desarrollo.

En los últimos años se han desarrollado dos corrientes en cuanto a metodologías de desarrollo de software se refiere, las llamadas metodologías ágiles y las pesadas. A continuación analizamos tres de las más usadas actualmente, dentro de las dos corrientes antes mencionadas.

2.3.1. Proceso Unificado de Desarrollo de Software (RUP)

RUP es uno de los procesos más generales de los existentes actualmente, ya que en realidad está pensado para adaptarse a cualquier proyecto, y no tan solo de software.

El proceso unificado de desarrollo, es el resultado de la evolución e integración de diferentes metodologías de desarrollo de software. Permite sacar el máximo provecho de los conceptos asociados a la orientación a objetos y al modelado visual. Cuenta con las mejoras prácticas del modelo de desarrollo de un software en particular. (29) (28)

1. Desarrollo de software de forma iterativa.
2. Manejo de requerimientos.
3. Utiliza arquitectura basada en componentes.
4. Modela el software de forma visual, usando UML.
5. Verifica la calidad del software.
6. Controla los cambios.
7. Dirige las tareas de cada desarrollador por separado y del equipo como un todo.
8. Especifica los artefactos que deben desarrollarse en cada fase de desarrollo del software.

Un proyecto realizado siguiendo RUP se divide en cuatro fases:

1. Inicio.
2. Elaboración.
3. Construcción.
4. Transición.

En cada fase se ejecutarán una o varias iteraciones (de tamaño variable según el proyecto), y dentro de cada una de ellas seguirá un modelo de cascada o *waterfall* para los flujos de trabajo que requieren las nuevas actividades anteriormente citadas.

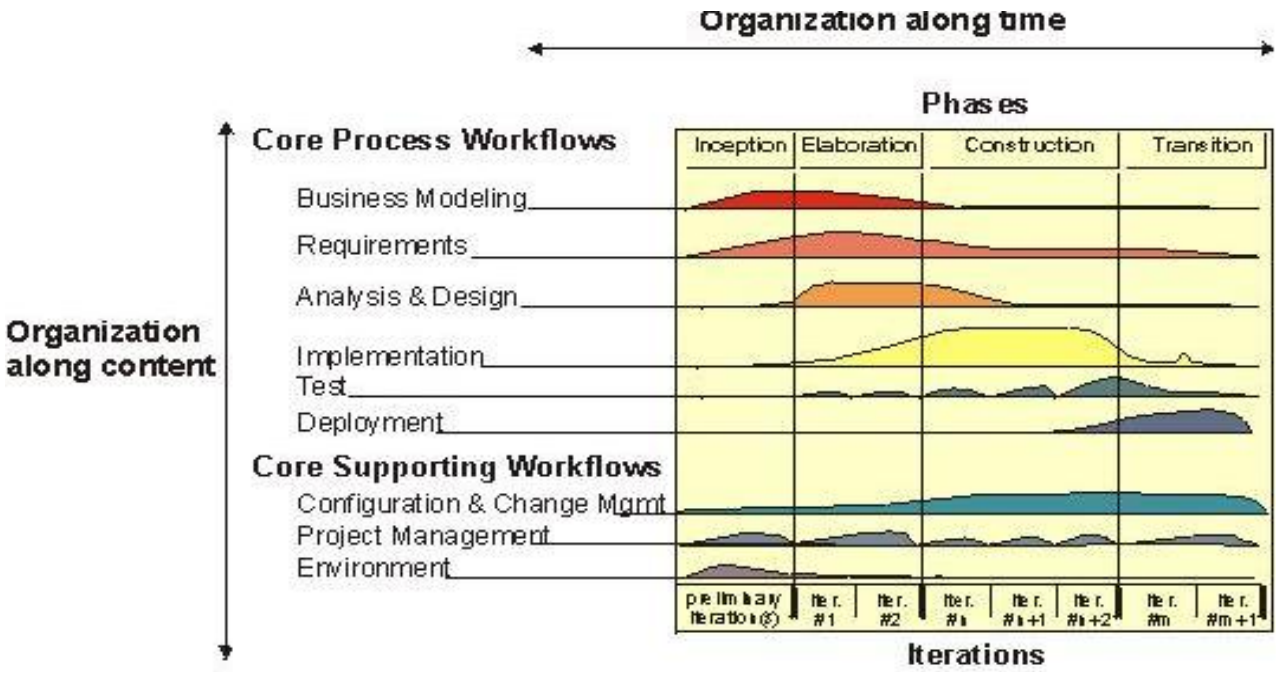

Fig. 12 Vista general de RUP

RUP define nueve actividades a realizar en cada fase del proyecto:

1. Modelado del negocio.
2. Análisis de requisitos.
3. Análisis y diseño.
4. Implementación.
5. Test.
6. Distribución.
7. Gestión de configuración y cambios.
8. Gestión del proyecto.
9. Gestión del entorno.

y el flujo de trabajo (*workflow*) entre ellas en base a los llamados diagramas de actividad.

El proceso define una serie de roles que se distribuyen entre los miembros del proyecto y que definen las tareas de cada uno y el resultado (*artefactos* en la jerga de RUP) que se espera de ellos.

Fig. 13 Flujos de trabajos de RUP

RUP se basa en casos de uso para describir lo que se espera del software y está muy orientado a la arquitectura del sistema, documentándose lo mejor posible, basándose en UML (*Unified Modeling Language*) como herramienta principal. (30)

2.3.2. PROGRAMACIÓN EXTREMA (XP)

Es una metodología ágil, intenta reducir la complejidad del software por medio de un trabajo orientado al objeto, basado en las relaciones interpersonales y la velocidad de reacción. Intenta minimizar el riesgo de fallo del proceso por medio de la disposición permanente de un representante competente del cliente a disposición del equipo de desarrollo. Este representante debe estar en condiciones de contestar rápida y correctamente a cualquier pregunta del equipo de desarrollo de forma que no se retrase la toma de decisiones.

XP define UserStories como base del software a desarrollar. Estas historias las escribe el cliente y describe escenarios sobre el funcionamiento del software, que no solo se limitan a las Interfaces Gráficas de Usuarios (GUI, siglas en inglés) sino también puede describir el modelo y el dominio. A partir de los UserStories y de la arquitectura perseguida se crea un plan de releases entre el equipo de desarrollo y el cliente. Para cada release se discutirán los objetivos de la misma con el representante del cliente y se definirán las iteraciones (de pocas semanas de duración) necesarias para cumplir con los objetivos del release. El resultado de cada iteración es un programa que se transmite al cliente para que lo juzgue.

En base a su opinión se definen las siguientes iteraciones del proyecto y si el cliente no está contento se adaptará el plan de releases e iteraciones hasta que el cliente dé su aprobación y el

software esté a su gusto. UserStories y casos de pruebas son la base sobre la que se asienta el trabajo del desarrollador. Se sigue un diseño evolutivo con la siguiente premisa: conseguir la funcionalidad deseada de la forma más sencilla posible. Este diseño evolutivo hace que no se le dé apenas importancia al análisis como fase independiente, puesto que se trabaja exclusivamente en función de las necesidades del momento.

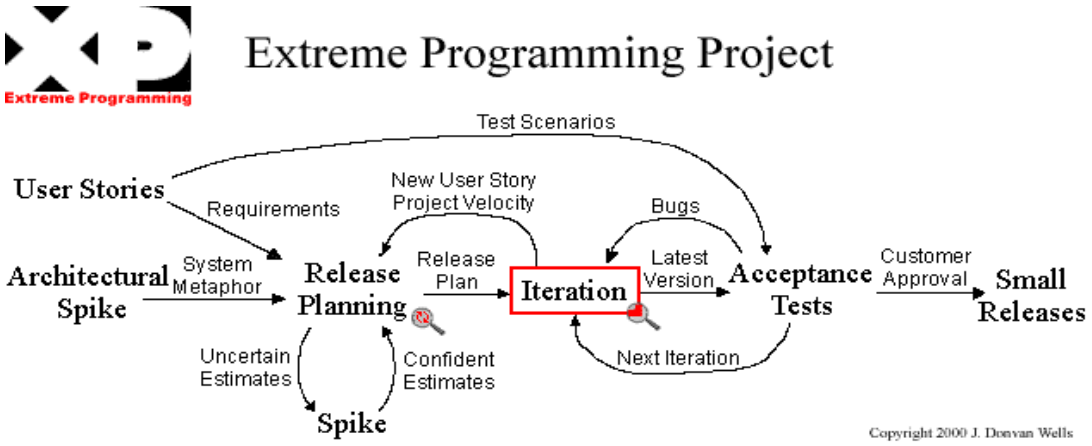

Fig. 14 Vista general de XP

La codificación del software en XP se produce siempre en parejas (dos programadores, un ordenador), por lo que se espera que la calidad del mismo suba en el mismo momento de escribirlo. Al contrario que muchos otros métodos, el código pertenece al equipo en completo, no a un programador o pareja, de forma que cada programador puede cambiar cualquier parte del código en cualquier momento si así lo necesita, dejándose en todo caso las mejoras orientadas al rendimiento para el final.

En XP se programará solo la funcionalidad que es requerida para el release actual. Es decir, una gran flexibilidad y capacidad de configuración solo será implementada cuando sea necesaria para cumplir los requerimientos del release. Se sigue un diseño evolutivo con la siguiente premisa: conseguir la funcionalidad deseada de la forma más sencilla posible. (30)

2.3.3. Desarrollo Guiado por Funcionalidad (FDD)

FDD es un proceso diseñado por Peter Coad, Erich Lefebvre y Jeff De Luca y se podría considerar a medio camino entre RUP y XP, aunque al seguir siendo un proceso ligero es más similar a este último.

Está pensado para proyectos con tiempo de desarrollo relativamente cortos (menos de un año). Se basa en un proceso iterativo con iteraciones cortas (~2 semanas). Las iteraciones se deciden en base a *features* (de ahí el nombre del proceso) o funcionalidades, que son pequeñas partes del software con significado para el cliente.

Un proyecto que sigue FDD se divide en 5 fases:

1. Desarrollo de un modelo general.
2. Construcción de la lista de funcionalidades.
3. Plan de releases en base a las funcionalidades a implementar.
4. Diseñar en base a las funcionalidades.
5. Implementar en base a las funcionalidades.

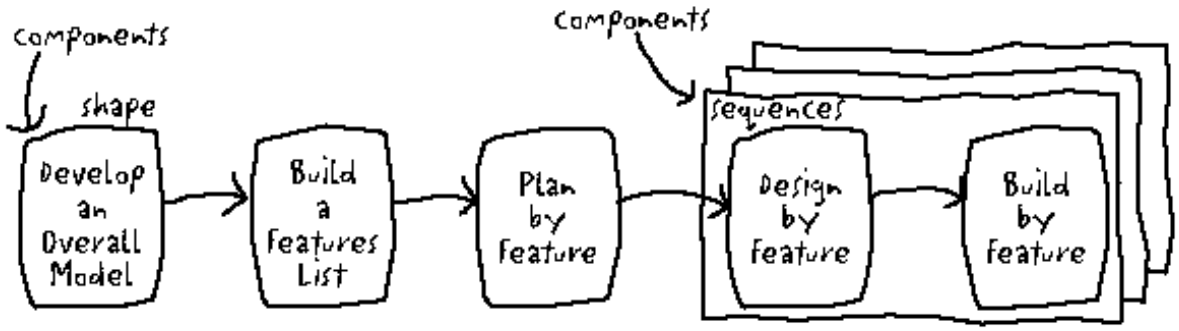

Fig. 15 Vista general de FDD.

Las primeras tres fases ocupan gran parte del tiempo en las primeras iteraciones, siendo las dos últimas las que absorben la mayor parte del tiempo según va avanzando el proyecto, limitándose las primeras a un proceso de refinamiento.

El trabajo (tanto de modelado como de desarrollo) se realiza en grupo, aunque siempre habrá un responsable último (arquitecto jefe o jefe de programadores en función de la fase en que nos encontremos), con mayor experiencia, que tendrá la última palabra en caso de no llegar a un acuerdo.

Las funcionalidades a implementar en un release se dividen entre los distintos subgrupos del equipo, y se procede a implementarlas. Las clases escritas tienen propietario (es decir, solo quién las crea puede cambiarlas), es por ello que en el equipo que implementa una funcionalidad dada deberán estar todos los dueños de las clases implicadas, pudiendo encontrarse un programador en varios grupos, implementando distintas funcionalidades.

FDD también define métricas para seguir el proceso de desarrollo de la aplicación, útiles para el cliente y la dirección de la empresa, y que pueden ayudar, además de para conocer el estado actual del desarrollo, a realizar mejores estimaciones en proyectos futuros. (30)

2.3.4. SELECCIÓN DE LA METODOLOGÍA A UTILIZAR

Después de hacer una estudio de las características de estas tres metodologías decidimos emplear RUP por ser una de las más completas y abarcadoras, además de estar muy bien organizada y documentada, aunque esta última se puede considerar como una de sus debilidades (pues a veces tiende a ser mucha documentación). Si comparamos con respecto a XP esta presenta una documentación muy pobre y UML juega un papel prácticamente nulo y en cuanto a FDD aunque su documentación es aceptable necesita rigurosamente de personal especializado en el tema, además de pequeñas debilidades potenciales para una buena obtención de requisitos del sistema.

2.4. HERRAMIENTAS CASE

Las Herramientas CASE (*Computer Aided Software Engineering*,) 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.

Estas tienen como objetivos:

- Mejorar la productividad en el desarrollo y mantenimiento del software.
- Aumentar la calidad del software.
- Mejorar el tiempo y coste de desarrollo y mantenimiento de los sistemas informáticos.
- Mejorar la planificación de un proyecto
- Aumentar la biblioteca de conocimiento informático de una empresa ayudando a la búsqueda de soluciones para los requisitos.
- Automatizar, desarrollo del software, documentación, generación de código, pruebas de errores y gestión del proyecto.
- Ayuda a la reutilización del software, portabilidad y estandarización de la documentación

- Gestión global en todas las fases de desarrollo de software con una misma herramienta.
- Facilitar el uso de las distintas metodologías propias de la ingeniería del software.

2.4.1. RATIONAL ROSE

Es la herramienta CASE que comercializan los desarrolladores de UML (Booch, Rumbaugh y Jacobson) y que soporta de forma completa la especificación del UML. Esta herramienta propone la utilización de cuatro tipos de modelos para realizar un diseño del sistema, utilizando una vista estática, otra dinámica de los modelos del sistema, una lógica y otra física; que permite crear y refinar estas vistas creando de esta forma un modelo completo que representa el dominio del problema y el sistema de software. (31)

Se puede generar código 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.

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.

2.4.2. VISUAL PARADIGM

Es una herramienta CASE que le facilita a los ingenieros de software diseñar, integrar y modelar visualmente los distintos diagramas que se generan a lo largo del desarrollo del software. Presenta un generador de código que soporta más de diez lenguajes y proporciona la ingeniería inversa: (32)

Características:

1. Es profesional, da la posibilidad de crear un gran conjunto de artefactos de las distintas fases del desarrollo del software, entre los que se encuentran los siguientes:

   Diagramas de Casos de Uso.

   Diagramas de Clases.

   Diagramas de Secuencia.

Diagramas de Comunicación.

Diagramas de Estado.

Diagramas de Componentes.

Diagramas de Despliegue.

Diagramas de Objetos.

Diagramas de Interacción.

Diagramas de Entidad Relación.

Diagramas ORM.

Diagramas de Procesos del Negocio.

Diagramas de EJB.

Diagramas de visión general.

2. Es amigable.

3. Contiene facilidades para redactar Especificaciones de Casos de Uso del Sistema.

4. Sincronización entre Diagramas de Entidad Relación y Diagramas de Clases.

5. Generación de Código / Ingeniería Inversa: entre los lenguajes conocidos por Visual Paradigm para la ingeniería inversa, se encuentran los siguientes:

Java Source.

Java Classes.

C++ Source.

Librerías dinámicas (DLL) y/o ficheros ejecutables de .NET.

CORBA IDL Source.

Ada 9x Source.

XML.

XML Shema.

JDBC.

Hibernate.

PHP 5.0 Source.

6. Generación de documentos.

7. Interoperabilidad con otras Aplicaciones.

8. Integración con distintos Ambientes de Desarrollo Integrados (IDE), entre ellos están:

Visual Studio.

Eclipse.

NetBeans / Suntm One.

ItellIJ IDEA.

WebLogic workshoptm.

Websphere.

JBuilder.

JDeveloper.

9. Generación de Código ORM.
10. Disponibilidad en múltiples plataformas.

2.4.3. Selección de la herramienta CASE a utilizar

Después de analizar las características de las herramientas antes expuestas hemos decidido que la herramienta a utilizar para un mejor desarrollo ingenieril de este trabajo es Visual Paradigm. Aunque ambas herramientas, no cabe duda son muy competentes, en el caso de la seleccionada encontramos versiones multiplataforma, además de ser mucho más intuitiva y proporcionar muchas facilidades para el trabajo. Así podremos obtener una mayor calidad de la documentación del software y exportar la misma para distribuciones Linux.

2.5. Software Libre y su utilización

Las tecnologías libres llegaron bastante rápido a Cuba, si se tiene en cuenta que su uso comenzó desde principios de la década de los 90, cuando el proyecto GNU no llegaba aún a los diez años de creado.

Muchos organismos vienen desarrollando en sus instituciones un trabajo constante y meritorio: Salud Pública, la Aduana General de la República, Educación Superior, el Ministerio de Ciencia, Tecnología y Medio Ambiente o el Ministerio de Informática y las Comunicaciones (MIC), entre otros tantos en los que las tecnologías libres muestran un crecimiento sostenido.

Hoy se estiman en más de 2 000 los usuarios de distribuciones GNU/Linux, cifra contrastante con unos 200 que se estimaban en 1995.

El gobierno cubano también ha hecho manifiesto su interés en este movimiento, considerándolo esencial para el desarrollo del país. Al respecto, en el año 2002 fue lanzada una estrategia guiada por el MIC para favorecer la inserción en el país de las tecnologías libres, convicción que reafirmó dos años después con el acuerdo 684/04.

Existen ejemplos como los de la Universidad de las Ciencias Informáticas (UCI), que acertadamente ha dedicado un por ciento de su matrícula al desarrollo de las tecnologías libres, y hoy es pilar en el desarrollo de este campo en el país, con un trabajo ascendente que irradia hacia sus filiales en provincias.

El proceso de «migrar» hacia las tecnologías libres no es trivial —aunque tampoco difícil—. Lo óptimo es que se realice cuando sea eficaz y dentro de un proceso armónico y seguro, de acuerdo con el alcance y desarrollo de cada profesional, empresa, ministerio y país.

La introducción del software libre en Cuba es aún más compleja, pues no pueden olvidarse las limitaciones para importar tecnologías y software. Aunque paradójicamente, por esa misma razón constituye una línea estratégica en el desarrollo de la informatización en el país.

Tampoco se puede desconocer que los niveles de informatización de la sociedad cubana actual se han alcanzado, en buena medida, con el uso sistemático de tecnologías propietarias, Microsoft más que ninguna otra, y por ende uno de los lados débiles es la falta de preparación en tecnologías libres y en el uso de distribuciones GNU/Linux.

Ya existe en la Isla un grupo multidisciplinario, encabezado por la Oficina Nacional de Informatización, encargado de analizar las diferentes experiencias de uso de software libre, además de directivas para su implementación paulatina.

De esta coexistencia obtendremos el escenario para desarrollar un proceso paulatino de transferencia, instrumentado por pasos bien documentados, y que nos aporte seguridad y buenas prácticas. (33)

Ventajas del software libre:

- Bajo o nulo costo.
- Libertad de uso y redistribución.
- Independencia tecnológica.
- Fomento de la libre competencia al basarse en servicios y no licencias.
- Soporte y compatibilidad a largo plazo.
- Formatos estándar.
- Sistemas sin puertas traseras y más seguros.
- Corrección más rápida y eficiente de fallos.
- Métodos simples y unificados de gestión de software.

- Sistema en expansión.

El software libre ya no es una promesa, es una realidad y se utiliza en sistemas de producción por algunas de las empresas tecnológicas más importantes como IBM, SUN Microsystems, Google, Hewlett-Packard, etc. Paradójicamente, incluso Microsoft, que posee sus propias herramientas, emplea GNU Linux en muchos de sus servidores. Podemos augurar sin lugar a dudas un futuro crecimiento de su empleo y una consolidación bien merecida. (34)

2.5.1. DISTRIBUCIÓN DE LINUX

Linux es solo un kernel, es decir, el núcleo del sistema operativo. Para que este núcleo sea de utilidad como un sistema operativo, el kernel se utiliza con otros programas que permiten poder interactuar con la máquina. Si se quiere ver de esta forma, esto es el equivalente de lo que se distribuye con "el disco de Windows". Pero en Linux se va un paso más allá: no solo se distribuye un sistema mínimo, con el cual usualmente "no se puede hacer nada", sino que adicionalmente se "empacan" otros programas de utilidades, aplicaciones, juegos, etc., para conformar lo que se conoce como una distribución de Linux.

Así existe una cantidad bastante grande de distribuciones de Linux. Las diferencias entre una distribución y otra son muy variadas pero derivan principalmente de los objetivos buscados y la forma en la que se implementan estos objetivos: facilidad de instalación, optimizadas para usuarios caseros, diseñadas para funcionar como firewalls, etc. y los paquetes (aplicaciones) incluidas.

- **Debian GNU/Linux:** Cuenta con más de 9000 paquetes, cada uno para 11 arquitecturas distintas, y es desarrollada por casi un millar de programadores en todo el mundo, los cuales se mantienen en contacto por medio de Internet. Su objetivo es desarrollar un sistema operativo que se distinga por su excelencia técnica, y que esté basado en software libre. El proyecto Debian, dentro del cual se desarrolla esta distribución, fue donde se originaron los lineamientos de Software Libre de Debian, los cuales son la base de la definición de Código Abierto (OSD). Debido a que Debian no es una compañía, el soporte técnico (pagado) es provisto por terceros.

- **Ubuntu:** Basada en la distribución Debian GNU/Linux. Concentra su objetivo en la facilidad y libertad de uso, los lanzamientos regulares y la facilidad en la instalación. Ubuntu es patrocinado por Canonical Ltd., una empresa privada fundada y financiada por el empresario sudafricano Mark Shuttleworth. Disponible oficialmente para 2 arquitecturas: Intel x86, AMD64.

Portada extraoficialmente a 5 arquitecturas más: PowerPC, SPARC (versión *"alternate"*), IA-64, Playstation 3 y HP PA-RISC. Al igual que casi cualquier distribución basada en Linux, Ubuntu es capaz de actualizar a la vez todas las aplicaciones instaladas en la máquina a través de repositorios, a diferencia de otros sistemas operativos comerciales, donde esto no es posible.

- **Mandrake Linux:** Es la distribución basada en RedHat que más éxito ha tenido. Su desarrollo se originó a partir de opiniones divergentes respecto a qué hacer con KDE (en términos de su integración en RedHat), optimizaciones específicas para procesadores Pentium, el proceso de instalación y otros temas.

- **RedHat Linux:** Es una distribución comercial, y fue en gran parte la responsable de introducir Linux en el mundo corporativo. Se orienta más que todo hacia la producción de una distribución que sea fácil de instalar y administrar, para lo cual han fundado el desarrollo de varios proyectos orientados a este fin. Además brinda soporte técnico (pagado) en forma directa.
- **Slackware Linux:** Es una de las distribuciones más antiguas y respetadas. En un inicio fue desarrollada completamente por una persona, Patrick Volkerding, y actualmente esto es básicamente aún así. La opinión generalizada es que Slackware es algo difícil de instalar y algo difícil de utilizar, pero eso es con certeza una exageración: Slackware puede ser tan difícil como uno guste, es posible que el nombre esté relacionado con la acepción corriente de slack (eliminar tensión), pero es más probable que sea una referencia a la teología de la Iglesia del SubGenio (Church of the SubGenius), que dice que slack es la materia prima de la que se origina la felicidad humana. (35)

En nuestro caso la distribución sobre la que se trabajará será Debian, debido a la gran cantidad de paquetes compilados estable que posee facilitándonos múltiples facilidades y además de ser muy cómodo el trabajo y presentar buena interfaz gráfica.

2.6. Lenguajes de Programación Orientado a Objeto

La Programación Orientada a Objetos (POO u OOP según sus siglas en inglés) es un paradigma de programación que usa objetos y sus interacciones para diseñar aplicaciones y programas de computadora. Está basado en varias técnicas, incluyendo herencia, modularidad, polimorfismo, y

encapsulamiento. Su uso se popularizó a principios de la década de 1990. Actualmente son muchos los lenguajes de programación que soportan la orientación a objetos. (1)

Hoy en día la tecnología orientada a objetos ya no se aplica solamente a los lenguajes de programación, además se viene aplicando en el análisis y diseño con mucho éxito, al igual que en las bases de datos. Es que para hacer una buena programación orientada a objetos hay que desarrollar todo el sistema aplicando esta tecnología, de ahí la importancia del análisis y el diseño orientado a objetos.

La programación orientada a objetos es una de las formas más populares de programar y viene teniendo gran acogida en el desarrollo de proyectos de software desde los últimos años. Esta acogida se debe a sus grandes capacidades y ventajas frente a las antiguas formas de programar.

Ventajas de un lenguaje orientado a objetos:

- Fomenta la reutilización y extensión del código.
- Permite crear sistemas más complejos.
- Relacionar el sistema al mundo real.
- Facilita la creación de programas visuales.
- Construcción de prototipos.
- Agiliza el desarrollo de software.
- Facilita el trabajo en equipo.
- Facilita el mantenimiento del software. (36)

2.6.1. C++

El C++ (pronunciado *"ce más más"* o *"ce plus plus"*) es un lenguaje de programación, diseñado a mediados de los años 1980, por Bjarne Stroustrup, como extensión del lenguaje de programación C.

Se puede decir que C++ es un lenguaje que abarca tres paradigmas de la programación: la programación estructurada, la programación genérica y la programación orientada a objetos.

Actualmente existe un estándar, denominado ISO C++, al que se han adherido la mayoría de los fabricantes de compiladores más modernos. Existen también algunos intérpretes como ROOT (enlace externo).

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*).

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*).

El nombre C++ fue propuesto por Rick Masciatti en el año 1983, cuando el lenguaje fue utilizado por primera vez fuera de un laboratorio científico. Antes se había usado el nombre "C con clases". En C++, "c++" significa "incremento de C" y se refiere a que C++ es una extensión de C. (37) (38)

2.6.2. C#

C# (pronunciado *"si sharp"* o C sostenido) es un lenguaje de programación orientado a objetos desarrollado y estandarizado por Microsoft como parte de su plataforma .NET, que después fue aprobado como un estándar por la ECMA e ISO.

Su sintaxis básica deriva de C/C++ y utiliza el modelo de objetos de la plataforma.NET el cual es similar al de Java aunque incluye mejoras derivadas de otros lenguajes (más notablemente de Delphi y Java). C# fue diseñado para combinar el control de lenguajes de bajo nivel como C y la velocidad de programación de lenguajes de alto nivel como Visual Basic.

C#, como parte de la plataforma.NET, está normalizado por ECMA desde diciembre de 2001 (ECMA-334 "Especificación del Lenguaje C#"). El 7 de noviembre de 2005 acabó la beta y salió la versión 2.0 del lenguaje que incluye mejoras tales como tipos genéricos, métodos anónimos, iteradores, tipos parciales y tipos anulables. Ya existe la versión 3.0 de C# destacando los tipos implícitos y el LINQ (Language Integrated Query).

Aunque C# forma parte de la plataforma.NET, esta es una interfaz de programación de aplicaciones; mientras que C# es un lenguaje de programación independiente diseñado para generar programas sobre dicha plataforma.

En la actualidad existen los siguientes compiladores para el lenguaje C#:

- Microsoft.NET framework SDK incluye un compilador de C#, pero no un IDE.

- Microsoft Visual Studio .NET, IDE por excelencia de este lenguaje, versión 2002, 2003, 2005 Y próximamente la versión 2008 que actualmente se encuentra en fase RTM.
- #develop, es un IDE libre para C# bajo licencia LGPL, muy similar a Microsoft Visual C#.
- Mono, es una implementación GPL de todo el entorno .NET desarrollado por Novell. Como parte de esta implementación se incluye un compilador de C#.
- Delphi 2006, de Borland Software Corporation.
- DotGNU Portable.NET, de la Free Software Foundation. (39) (40)

2.6.3. JAVA

Java es un lenguaje de programación orientado a objetos desarrollado por Sun Microsystems a principios de los años 90. El lenguaje en sí mismo toma mucha de su sintaxis de C y C++, pero tiene un modelo de objetos más simple y elimina herramientas de bajo nivel, que suelen inducir a muchos errores, como la manipulación directa de punteros.

Las aplicaciones Java están típicamente compiladas en un *bytecode*, aunque la compilación en código máquina nativo también es posible. En el tiempo de ejecución, el *bytecode* es normalmente interpretado o compilado a código nativo para la ejecución, aunque la ejecución directa por hardware del *bytecode* por un procesador Java también es posible.

La implementación original y de referencia del compilador, la máquina virtual y las librerías de clases de Java fueron desarrolladas por Sun Microsystems en 1995. Desde entonces, Sun ha controlado las especificaciones, el desarrollo y evolución del lenguaje a través del Java Community Process.

Entre noviembre de 2006 y mayo de 2007, Sun Microsystems liberó la mayor parte de sus tecnologías Java bajo la licencia GNU/GPL, de acuerdo con las especificaciones del Java Community Process, de tal forma que prácticamente todo el Java de Sun es ahora software libre (aunque la biblioteca de clases de Sun que se requiere para ejecutar los programas Java todavía no es software libre). (41)

2.6.4. PYTHON

Python es un lenguaje de programación creado por Guido van Rossum en el año 1990.

Es comparado habitualmente con TCL, Perl, Scheme, Java y Ruby. En la actualidad Python se desarrolla como un proyecto de código abierto, administrado por la Python Software Foundation. La última versión estable del lenguaje es actualmente la 2.5.1 (18 de abril de 2007) (Se anunció la llegada de la versión 3.0 para el 2008).

Python es considerado como la "oposición leal" a Perl, lenguaje con el cual mantiene una rivalidad amistosa. Los usuarios de Python consideran a éste mucho más limpio y elegante para programar.

Python permite dividir el programa en módulos reutilizables desde otros programas Python. Viene con una gran colección de módulos estándar que se pueden utilizar como base de los programas (o como ejemplos para empezar a aprender Python). También hay módulos incluidos que proporcionan E/S de ficheros, llamadas al sistema, sockets y hasta interfaces a GUI (interfaz gráfica con el usuario) como Tk, GTK, Qt entre otros...

Python es un lenguaje interpretado, lo que ahorra un tiempo considerable en el desarrollo del programa, pues no es necesario compilar ni enlazar. El intérprete se puede utilizar de modo interactivo, lo que facilita experimentar con características del lenguaje, escribir programas desechables o probar funciones durante el desarrollo del programa. También es una calculadora muy útil.

El principal objetivo que persigue este lenguaje es la facilidad, tanto de lectura, como de diseño.

Python es un lenguaje de programación multiparadigma. Esto significa que más que forzar a los programadores a adoptar un estilo particular de programación, permite varios estilos: programación orientada a objetos, programación estructurada y programación funcional. Otros muchos paradigmas más están soportados mediante el uso de extensiones. Python usa tipo de dato dinámico y reference counting para el manejo de memoria. Una característica importante de Python es la resolución dinámica de nombres, lo que enlaza un método y un nombre de variable durante la ejecución del programa (también llamado ligadura dinámica de métodos).

Otro objetivo del diseño del lenguaje era la facilidad de extensión. Nuevos módulos se pueden escribir fácilmente en C o C++. Python puede utilizarse como un lenguaje de extensión para módulos y aplicaciones que necesitan de una interfaz programable. (42) (43)

Algunas de sus principales características distintivas son:

- Sintaxis muy clara y legible.
- Fuerte capacidad de introspección.

- Intuitiva orientación de objetos.
- Expresión natural del código de procedimiento.
- Completa modularidad, paquetes de apoyo jerárquico.
- Excepción basada en el manejo de errores.
- Nivel muy alto tipos de datos dinámicos.
- Extensas bibliotecas estándar y módulos para casi todas las tareas virtuales.
- Extensiones y módulos fácilmente escritos en C, C ++.
- Incrustado dentro de las aplicaciones como un interfaz de scripting.

2.6.5. FUNDAMENTACIÓN DEL LENGUAJE A UTILIZAR

Luego de estudiar las características de los lenguajes antes expuesto decidimos utilizar C++, pues es indudablemente un lenguaje muy potente y nos permite programar tanta a alto como a bajo nivel.

Aunque las aplicaciones en C# estén orientadas a ser económicas respecto a los requisitos de memoria y proceso, el lenguaje no fue hecho para competir directamente en velocidad o tamaño con C++ o lenguaje ensamblador.

Por otro lado el de hecho de que python sea interpretado hace al programa más lento y en cuanto a java la dependencia de la tecnología de la maquina virtual no es muy tentativa.

2.7. ENTORNOS DE DESARROLLO

Un entorno de desarrollo integrado o en inglés *Integrated Development Environment* (*IDE*) es un programa compuesto por un conjunto de herramientas para un programador. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien, poder utilizarse para varios.

Un IDE 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 sí solas o pueden ser parte de aplicaciones existentes. El lenguaje Visual Basic por ejemplo puede ser usado dentro de las aplicaciones de Microsoft Office, lo que hace posible escribir sentencias Visual Basic en forma de macros para Microsoft Word.

Los IDEs proveen un marco de trabajo amigable para la mayoría de los lenguajes de programación tales como C++, Java, C#, Delphi, Visual Basic, Object Pascal, Velneo, etc. En algunos lenguajes, un IDE puede funcionar como un sistema en tiempo de ejecución, en donde se permite utilizar el lenguaje de programación en forma interactiva, sin necesidad de trabajo orientado a archivos de texto, como es el caso de Smalltalk u Objective-C.

Es posible que un mismo IDE pueda funcionar con varios lenguajes de programación. Este es el caso de Eclipse, que mediante pluggins se le puede añadir soporte de lenguajes adicionales. (44)

2.7.1. KDevelop

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

A diferencia de muchas otras interfaces de desarrollo, KDevelop no cuenta con un compilador propio, por lo que depende de gcc para producir código binario.

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. (45)

KDevelop ofrece las siguientes características:

- KAppWizard, que genera aplicaciones de ejemplo completas.
- ClassGenerator, que crea nuevas clases y las integra en el proyecto actual.
- Administración de archivos para archivos fuente, archivos de encabezado, documentación del proyecto, etc.
- La creación de manuales de usuario escritos con SGML y la generación automática de salida en HTML con la look and feel de KDE.
- Documentación automática API, basada en HTML para las referencias cruzadas de las clases de su proyecto con las bibliotecas utilizadas.
- Soporte internacional para su aplicación, lo cual le permite agregar fácilmente sus nuevos lenguajes a un proyecto.
- Creación WYSIWYG de interfaces de usuario con un editor de cuadros de diálogo integrado.

- Administración de proyectos mediante CVS y la provisión de una interfaz fácil de usar para las funciones más necesarias.
- Depuración de programas por medio de KDbg.
- Edición de iconos por medio de KiconEdit.
- La inclusión de cualquier otro programa que necesite para el desarrollo, agregándolo al menú Herramientas de acuerdo con sus necesidades individuales.

KDevelop facilita el trabajo con todos los programas en un mismo lugar y ahorra tiempo al automatizar los procesos estándar de desarrollo, proporciona un acceso directo y transparente a toda la información necesaria para el control de un proyecto. Los mecanismos de explotación integrados están diseñados para soportar solicitudes de documentación que tengan los desarrolladores junto con su proyecto. (46) (47)

2.7.2. ANJUTA

Anjuta es un IDE para programar en C y C++ en sistemas GNU/Linux. Su principal objetivo es trabajar con GTK y en el escritorio GNOME, además es software libre, liberado bajo la licencia GPL.

Incluye un administrador de proyectos, asistentes, plantillas, depurador interactivo y un poderoso editor que verifica y resalta la sintaxis escrita.

Anjuta ejecuta casi todas las funciones usando plugins. Se puede seleccionar qué extensiones se quieren activar y el modo por defecto. Al igual que el diseño de interfaz de usuario, los complementos de selección también son persistentes para cada proyecto, lo cual facilita la labor en los diferentes niveles de complejidad del proyecto.

Todos los plugins en Anjuta son fácilmente sustituibles con diferentes plugins de la aplicación de las mismas características. Esto permite, por ejemplo, a tener varios editores para elegir. Esto se aplica a cualquier plugin.

Anjuta además, enumera todos los directorios y archivos en el proyecto.
Posee un potente gestor de proyectos. (48)

2.7.3. ECLIPSE

Eclipse es una plataforma de software de código abierto independiente de una plataforma para desarrollar lo que el proyecto llama "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Esta plataforma, típicamente ha sido usada para desarrollar IDE, como el llamado *Java Development Toolkit* (JDT) y el compilador (ECJ) que se entrega como parte de Eclipse (y que son usados también para desarrollar el mismo Eclipse). Sin embargo, también se puede usar para otros tipos de aplicaciones cliente, como BitTorrent Azureus.

Eclipse fue desarrollado originalmente por IBM como el sucesor de su familia de herramientas para VisualAge. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

La versión actual de Eclipse dispone de las siguientes características:

- Editor de texto.
- Resaltado de sintaxis.
- Compilación en tiempo real.
- Pruebas unitarias con JUnit.
- Control de versiones con CVS.
- Integración con Ant.
- Asistentes (*wizards*): para creación de proyectos, clases, test, etc.
- Refactorización

Asimismo, a través de "plugins" libremente disponibles es posible añadir:

- Control de versiones con Subversion, vía Subclipse.
- Integración con Hibernate, vía Hibernate Tools.

Proyectos Eclipse:

- JDT (Java Development Tools) - Herramientas de desarrollo Java.
- AJDT (AspectJ Development Tools Project) - Herramientas de desarrollo de proyecto aspecto.
- C/ C + + Desarrollo de herramientas (CDT).
- COBOL.
- PDT (PHP Development Tools)- Herramientas de desarrollo de PHP. (49)

2.7.4. CODE::BLOCKS

Code::Blocks es un IDE libre de C++ construido para satisfacer las más exigentes necesidades de sus usuarios. Está diseñado para ser muy extensible y totalmente configurable, con un consecuente aspecto y operación a través de plataformas.

Construido alrededor de un plugin, Code::Blocks puede extenderse con plugins. Cualquier tipo de funcionalidad se puede añadir mediante la instalación / codificación de un plugin. La compilación y depuración de la funcionalidad ya está disponible a través de los plugins.

El servidor es muy poderoso, elegido para ofrecer el contenido de manera eficiente y confiable, incluso fue diseñado para manejar grandes cargas.

Lo más destacado:

- Open Source. GPLv3, sin costes ocultos.
- Cross-plataforma. Corre en Linux, Mac, Windows (usa wxWidgets).
- Escrito en C + +.
- Extensible a través de plugins.

Compilador:

- Múltiples compiladores de apoyo:
  - GCC (MingW / GNU GCC).
  - MSV C++.
  - Digital Mars.
  - Borland C++ 5.5.
  - Open Watcom.
- Muy rápido.
- Soporte a la construcción paralela (la utilización de la CPU de núcleos extra).
- Proyectos multi-meta.
- Las importaciones de proyectos Dev-C++.

Debugger:

- Interfaces de GNU GDB
- También apoya MS Banco de Desarrollo del Caribe (no completamente ofrecido)

- Interrupción de apoyo:
  - Código de interrupción
  - Interrupción de datos (lectura, escritura y de lectura / escritura)
  - Breakpoint o condiciones (pausa sólo cuando la expresión es verdadera)
  - Breakpoint o hacer caso omiso de los cargos (romper sólo después de cierto número de votos)
- Pantalla función de los símbolos locales y argumentos
- Relojes definida por el usuario (ver apoyo a los tipos definidos por el usuario a través de scripting)
- Llame a pila
- Desmontaje
- Custom volcado de la memoria
- Cambia de hilos
- Ver los registros de la CPU

Interfaz:

- Sintaxis personalizable y extensible.
- Códigos plegables para C++ y archivos XML.
- Interfaz con pestañas.
- Completamiento de código.
- El archivo de la clase.
- Llave principal (swap) entre archivos .h y .c / .cpp
- Para la lista de tareas de gestión con los diferentes usuarios. (50)

2.7.5. FUNDAMENTACIÓN DEL ENTONRNO DE DESARROLLO UTILIZADO

Luego del análisis de los IDE antes expuestos, decidimos desarrollar la BMN en Code::Blocks para obtener mejores resultados. Debido a que aunque KDevelop y Anjuta son muy competentes solo los encontramos para distribuciones Linux, por lo que la exportación del proyecto para otras plataformas será algo engorrosa. Por otra parte con Eclipse y Code::Blocks resolveríamos esto, pero cuando comparamos en velocidad sin ninguna duda nos quedamos con este último.

2.8. CONCLUSIONES PARCIALES

Con el análisis realizado en este capítulo se ha profundizado en la selección de la metodología y herramientas a utilizar para el desarrollo de este trabajo. Quedando seleccionadas RUP como metodología de desarrollo por su adaptabilidad y documentación, UML como lenguaje de modelado para describir todos los procesos y Visual Paradigm como herramienta CASE para modelar la aplicación. C++ como lenguaje de programación por sus potencialidades y como IDE de desarrollo Code::Blocks. Logrando con esta selección dar paso a la descripción detallada de la BMN para obtener un resultado óptimo en este trabajo.

CAPÍTULO 3: PRESENTACIÓN DE LA SOLUCIÓN PROPUESTA.

3.1. INTRODUCCIÓN

En este capítulo se hará una descripción de la propuesta a defender, en vista a solucionar la situación actual en el campo de acción. Se muestran los requisitos, tanto los funcionales como los no funcionales, y los casos de uso generados a partir de dichos requisitos funcionales con sus respectivas especificaciones. Esta propuesta se elaboró con el fin de facilitar el trabajo con los métodos numéricos, y agilizar el desarrollo de aplicaciones informáticas que requieran el uso de estos.

3.2. MODELO CONCEPTUAL

Luego de un exhaustivo análisis y debido a no tener una estructura definida de los procesos de 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.

3.2.1. DIAGRAMA DE CLASES DEL MODELO DE DOMINIO

El Modelo de Dominio (o Modelo Conceptual) es una representación visual de los conceptos u objetos del mundo real significativos para un problema o área de interés. Representa clases conceptuales del dominio del problema, conceptos del mundo real, no de los componentes de software. Una clase conceptual puede ser una idea o un objeto físico (símbolo, definición y extensión). El modelo de dominio se representa en UML con un Diagrama de Clases en los que se muestra:

- conceptos u objetos del dominio del problema: clases conceptuales

- asociaciones entre las clases conceptuales
- atributos de la clase conceptuales

La identificación y la asignación de un nombre para estos objetos ayudan a desarrollar un glosario de términos que permitirán comunicarse mejor a todos los que están trabajando en el sistema. Más adelante, los objetos del dominio ayudan a identificar algunas de las clases que se analizan y diseñan en el sistema.

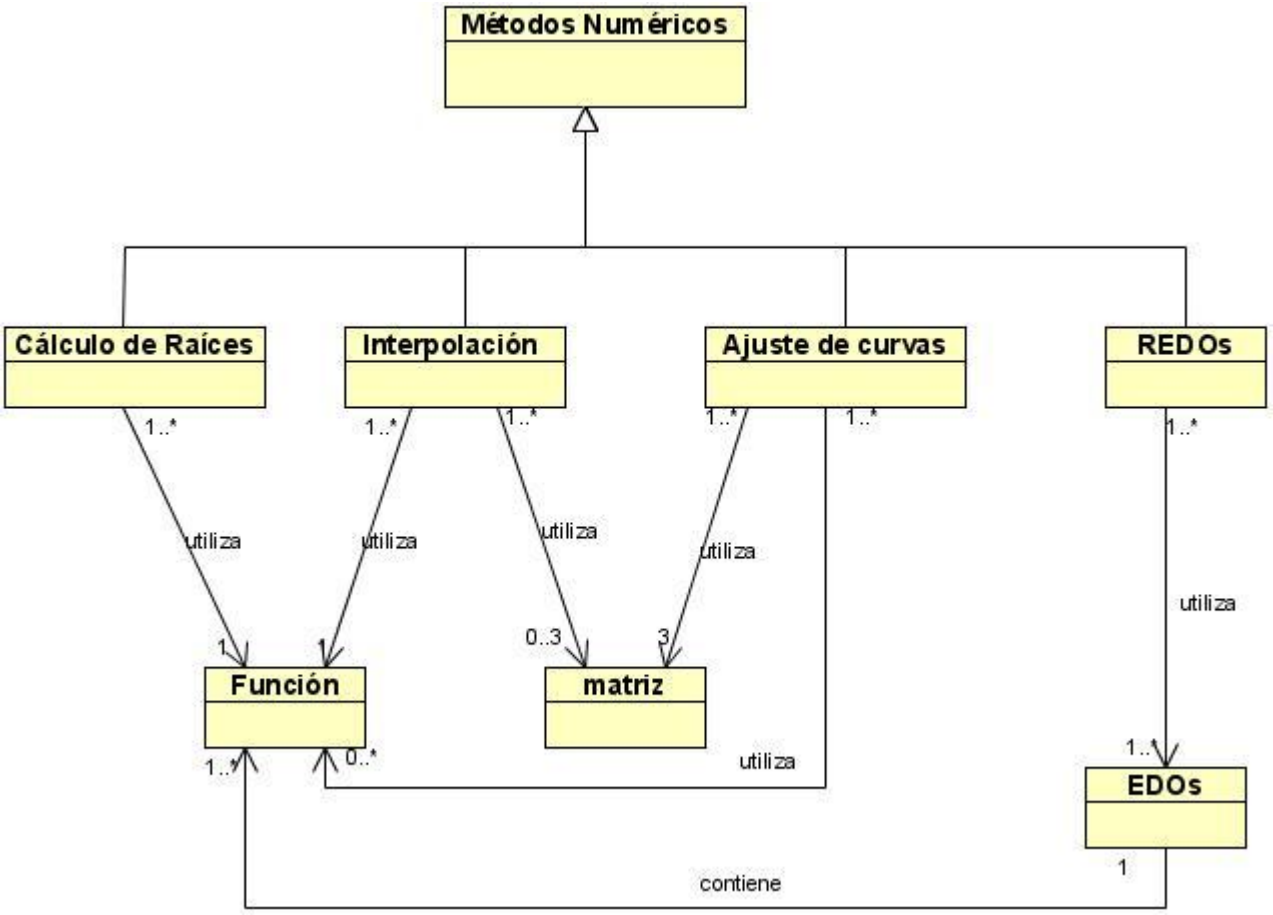

Fig. 16 Diagrama de clases del dominio

3.2.2. ANÁLISIS DE LOS CONCEPTOS DEL DOMINIO

Mediante un glosario de términos se quiere definir cada uno de los términos empleados en el dominio. Los cuales son:

- Se denomina **Métodos Numéricos** a todos los algoritmos diseñados por la matemática numérica para resolver problemas matemáticos.

- Se denomina **Cálculo de Raíces** a todos los métodos numéricos encargados de solucionar los problemas matemáticos de cálculo de raíces de funciones no lineales.
- Se denomina **Interpolación** a todos los métodos numéricos encargados de solucionar los problemas matemáticos de interpolación de funciones.
- Se denomina **Ajuste de Curvas** a todos los métodos numéricos encargados de solucionar los problemas matemáticos de cálculo de ajuste de curvas mediante un modelo lineal.
- Se denomina **REDO** a todos los métodos numéricos encargados de solucionar los problemas matemáticos de calcular la solución de sistemas de EDO.
- Se denomina **Función** a todas las funciones matemáticas, independiente del tipo de función que sea, polinomial, trigonométrica, logarítmica, etc.
- Se denomina **Matriz** a la disposición de elementos en filas y columnas.
- Se denomina **EDO** a las ecuaciones formadas por n funciones (con n >= 1) derivadas independientemente del orden de la derivada.

3.3. SOLUCIÓN PROPUESTA

Como solución propuesta se tiene la creación de una biblioteca de objetos de métodos numéricos (BMN), que agrupe algoritmos para el cálculo de raíces de funciones, interpolación, ajuste de curvas y resolución de ecuaciones diferenciales. La BMN es el resultado de la unión de un conjunto de bibliotecas, cada una de éstas encargada de agrupar los distintos algoritmos pertenecientes a los temas antes mencionados. Esta distribución se hace con el objetivo de ganar en simplicidad y portabilidad de la BMN.

3.4. DESCRIPCIÓN DE LA FUNCIONALIDAD DEL SISTEMA PROPUESTO

Una especificación de requisitos del software es una descripción completa del comportamiento del sistema a desarrollar. Incluye un conjunto de casos de uso que describen todas las interacciones que se prevén que los usuarios tendrán con el software. También contiene requisitos no funcionales (o suplementarios). Los requisitos no funcionales son los requisitos que imponen restricciones al diseño o funcionamiento del sistema (tal como requisitos de funcionamiento, estándares de calidad, o requisitos del diseño).

3.4.1. REQUISITOS FUNCIONALES DEL SISTEMA

Los requisitos funcionales describen las capacidades o condiciones del producto, o sea lo que se quiere que haga el sistema a construir.

Tabla 1. Requisitos funcionales.

Referencia	Requisito Funcional
R 1	**Calcular la raíz de una función no lineal.**
R 1.1	Calcular la raíz por métodos de intervalos.
R 1.1.1	Calcular la raíz por el método de Bisección.
R 1.1.2	Calcular la raíz por el método de Regula-Falsi.
R 1.1.3	Calcular la raíz por el método de Regula-Falsi Modificado.
R 1.2	Calcular la raíz por métodos Abiertos.
R 1.2.1	Calcular la raíz por el método de Newton-Raphson.
R 1.2.2	Calcular la raíz por el método de la Secante.
R 1.2.3	Calcular la raíz por el método de Aproximaciones Sucesivas.
R 1.2.4	Calcular la raíz por el método de Wegstein.
R 1.2.5	Calcular la raíz por el método de Steffensen.
R 2	**Hallar la interpolación de un valor mediante una función interpoladora.**
R 2.1	Hallar la interpolación de un valor mediante una función polinomial.
R 2.1.1	Hallar la interpolación mediante el método de polinomios de Lagrange.
R 2.1.2	Hallar la interpolación mediante el método de Diferencias divididas de Newton.
R 2.2	Hallar la interpolación de un valor mediante una función Spline Cúbico.
R 2.2.1	Hallar la interpolación mediante el método de Spline Cúbico Natural.
R 2.2.2	Hallar la interpolación mediante el método de Spline Cúbico Anclado.
R 2.2.3	Hallar la interpolación mediante el método de Spline Cúbico Periódico.
R 3	**Ajustar una función a un conjunto de puntos.**
R 3.1	Ajustar una función por el método de mínimos cuadrados.
R 4	**Solucionar ecuaciones diferenciales ordinarias (EDO).**
R 4.1	Solucionar EDO por métodos de paso simple.
R 4.1.1	Solucionar EDO por el método de Euler.
R 4.1.2	Solucionar EDO por el método de Runge-Kutta (2^{do} y 4^{to} orden).
R 4.2	Solucionar EDO por métodos de pasos múltiple.
R 4.2.1	Solucionar EDO por el método de Adams-Bashforth.
R 4.2.2	Solucionar EDO por el método de Adams-Bashforth-Moulton.

R 4.3	Solucionar sistemas de EDO. (Generalización de los métodos de Euler y RK4).

3.4.2. REQUISITOS NO FUNCIONALES DEL SISTEMA

Los requisitos no funcionales son aquellos que describen las cualidades, características y propiedades que el producto a construir debe tener. Estos se clasifican por categorías:

Requerimiento de usabilidad

La herramienta será utilizada por programadores, el producto debe estar concebido para que el usuario piense qué desea hacer, y no cómo hacerlo.

Requerimiento de Soporte

Para garantizar el soporte de esta herramienta se creará un manual de usuario para esclarecer las dudas sobre el trabajo con la misma que le puedan surgir al usuario. Además de un manual de uso y trabajo con los métodos numéricos pues el usuario no tiene que ser un especialista en este tema.

Requerimiento de portabilidad

La herramienta propuesta podrá ser utilizada en sistemas operativos como Windows, distribuciones Unix, etc.

Requerimiento de rendimiento

Se debe realizar una biblioteca de objeto eficiente, precisa y rápida, que permita dar respuesta en el menor tiempo posible.

Requerimiento legal

La biblioteca se regirá por la licencia GNU/GPL.

3.5. MODELO DE CASOS DE USO DEL SISTEMA

El modelo de casos de uso describe la funcionalidad propuesta del nuevo sistema. Un caso de uso representa una unidad discreta de interacción entre un usuario (humano o máquina) y el sistema. Un caso de uso es una unidad de trabajo significativo y se ejecuta en su totalidad o no se ejecuta nada, devolviendo algo de valor al usuario.

Tabla 2. Prioridad de casos de uso

Caso de uso del sistema	Prioridad
Calcular raíz de función.	Crítico
Hallar interpolación de un valor	Crítico
Hallar Ajuste de función	Crítico
Calcular solución de un sistema de EDO	Crítico

3.5.1. DETERMINACIÓN Y JUSTIFICACION DE LOS ACTORES DEL SISTEMA

Un actor no es parte del sistema, es un rol de un usuario, que 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 este caso con el sistema interactúa un único actor que se define a continuación:

Tabla 3. Descripción del actor del sistema.

Actor	Justificación
Usuario	Representa a otro sistema (ejemplo: simulador) o a un programador que utiliza la BMN.

3.5.2. 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. Una relación es una conexión entre los elementos del modelo, por ejemplo la especialización y la generalización son relaciones. 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.

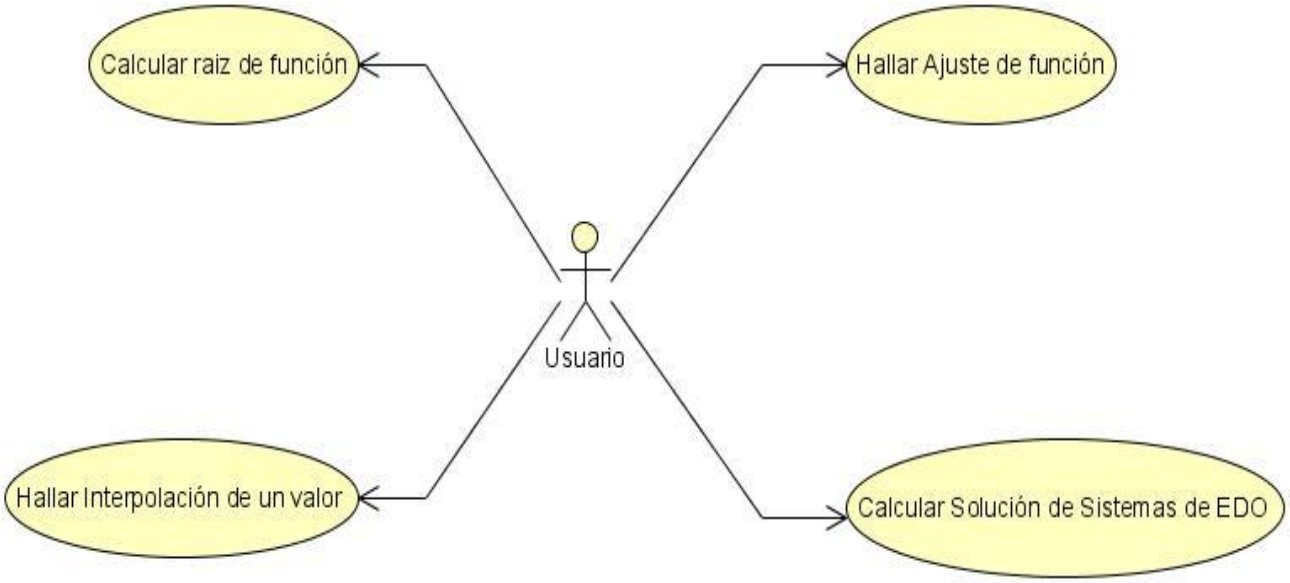

Fig. 17 Diagrama de casos de uso del sistema.

3.5.3. DESCRIPCIÓN TEXTUAL DE LOS CASOS DE USO DEL SISTEMA

Cada caso de uso tiene una descripción que especifica la funcionalidad que se incorporará al sistema propuesto. Un caso de uso puede 'incluir' la funcionalidad de otro caso de uso o puede 'extender' otro caso de uso con su propio comportamiento.

Tabla 4. Descripción del CUS Calcular raíz de función.

Descripción del CUS Calcular raíz de función.	
Nombre del Caso de Uso	Calcular raíz de función.
Actor(es)	Usuario.
Propósito	Calcular la raíz de una función no lineal.
Descripción	El caso de uso se inicia cuando el usuario solicita calcular la raíz de una función no lineal, pasando los datos necesarios según el método escogido. El sistema valida los datos y calcula la raíz con su error.
Referencias	RF 1 RF 1.1 RF 1.1.1 RF 1.1.2

<table>
<tr><td></td><td>RF 1.1.3

RF 1.2

RF 1.2.1

RF 1.2.2

RF 1.2.3

RF 1.2.4

RF 1.2.5</td></tr>
<tr><td>Precondiciones</td><td></td></tr>
<tr><td>Poscondiciones</td><td>Quedan calculados la raíz aproximada y su error.</td></tr>
<tr><td>Flujo de eventos</td><td>
<table>
<tr><th></th><th>Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1</td><td>El usuario solicita calcular la raíz de una función por métodos de Intervalo, pasando el intervalo, la función, el error, la cantidad de iteraciones y el id del método a emplear.</td><td></td></tr>
<tr><td>2</td><td></td><td>El sistema valida los datos.</td></tr>
<tr><td>3</td><td></td><td>[Los datos son correctos.] El sistema calcula y devuelve la raíz con su error y cantidad de iteraciones reales.</td></tr>
</table>
</td></tr>
<tr><td>Flujo Alterno</td><td>
<table>
<tr><th></th><th>Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1a.1</td><td>El usuario solicita calcular la raíz de una función por método Abierto, pasando el intervalo, la función, el error, la cantidad de iteraciones, el punto inicial para iterar y el id del método a emplear.</td><td></td></tr>
<tr><td>3a.1</td><td></td><td>[Los datos son incorrectos.] El sistema lanza una excepción.</td></tr>
</table>
</td></tr>
</table>

Tabla 5. Descripción del CUS Hallar interpolación de un valor.

Descripción del CUS Hallar interpolación de un valor.	
Nombre del Caso de Uso	Hallar interpolación.
Actor(es)	Usuario.
Propósito	Hallar la imagen de un valor x mediante una función interpoladora.
Descripción	El caso de uso se inicia cuando el usuario solicita hallar la interpolación de un valor, pasando los datos necesarios según el método de interpolación seleccionado. El sistema halla y devuelve el valor

<table>
<tr><td></td><td>interpolado.</td></tr>
<tr><td>Referencias</td><td>RF 2

RF 2.1

RF 2.1.1

RF 2.1.2

RF 2.2

RF 2.2.1

RF 2.2.2

RF 2.2.3</td></tr>
<tr><td>Precondiciones</td><td></td></tr>
<tr><td>Poscondiciones</td><td>Queda calculado el valor interpolado.</td></tr>
<tr><td>Flujo de eventos</td><td>
<table>
<tr><th></th><th>Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1</td><td>El usuario solicita hallar la interpolación de un valor mediante métodos de polinomio, pasando los nodos de interpolación, el valor a interpolar y el id del método a emplear.</td><td></td></tr>
<tr><td>2</td><td></td><td>El sistema calcula y devuelve el valor interpolado.</td></tr>
</table>
</td></tr>
<tr><td>Flujo Alterno</td><td>
<table>
<tr><th></th><th>Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1a.1</td><td>El usuario solicita hallar la interpolación de un valor mediante métodos de spline cúbicos, pasando los nodos de interpolación, el valor a interpolar y el id del método a emplear.</td><td></td></tr>
<tr><td>1a. 2</td><td></td><td>El sistema construye el sistema de ecuaciones pertinente.</td></tr>
</table>
</td></tr>
</table>

Tabla 6. Descripción del CUS Hallar Ajuste de función.

Descripción del CUS Hallar Ajuste de función.	
Nombre del Caso de Uso	Hallar Ajuste de función.
Actor(es)	Usuario.
Propósito	Calcular los coeficientes de la función que se ajusta a un conjunto de puntos.
Descripción	El caso de uso se inicia cuando el usuario solicita ajustar una función a un conjunto de puntos mediante un modelo lineal, brindando los datos

	necesarios. El sistema construye el sistema de ecuaciones y halla la lista de coeficientes de la función que se ajusta al modelo planteado.
Referencias	RF 3 RF 3.1
Precondiciones	
Poscondiciones	Quedan calculados los coeficientes de la función que se ajusta al modelo.

Flujo de eventos

	Actor	Respuesta del Sistema
1	El usuario solicita ajustar una función a un conjunto de puntos mediante un modelo lineal pasando la lista de puntos (valores x, y) y la lista de funciones.	
2		El sistema construye el sistema de ecuaciones pertinente.
3		El sistema calcula y devuelve los coeficientes de la función que se ajusta al modelo.

Tabla 7. Descripción del CUS Calcular solución de un sistema de EDO.

Descripción del CUS Calcular solución de un sistema de EDOs.	
Nombre del Caso de Uso	Calcular solución de un sistema de EDOs.
Actor(es)	Usuario.
Propósito	Calcular la solución de un sistema de n EDOs con n≥1.
Descripción	El caso de uso se inicia cuando el usuario solicita calcular la solución de un sistema de EDO, brindando los datos necesarios según el tipo de método a emplear. El sistema verifica los datos y calcula la solución del sistema de EDO en cuestión.
Referencias	RF 4 RF 4.1 RF 4.1.1 RF 4.1.2 RF 4.2 RF 4.2.1 RF 4.2.2 RF 4.3
Precondiciones	
Poscondiciones	Queda calculada la solución del sistema de EDO en cuestión.

<table>
<tr><td>Flujo de eventos</td><td>
<table>
<tr><th></th><th>Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1</td><td>El usuario solicita calcular la solución de un sistema de EDO por métodos de paso simple pasando x e y iniciales, x final, el paso, la lista de EDOs y el id del método a usar.</td><td></td></tr>
<tr><td>2</td><td></td><td>El sistema calcula y devuelve la solución del sistema de EDO.</td></tr>
</table>
</td></tr>
<tr><td>Flujo Alterno</td><td>
<table>
<tr><th></th><th>Actor</th><th>Respuesta del Sistema</th></tr>
<tr><td>1a.1</td><td>El usuario solicita calcular la solución de un sistema de EDO por métodos de paso múltiple pasando x e y iniciales, x final, el paso, la lista de EDOs y el id del método a usar.</td><td></td></tr>
<tr><td>1a. 2</td><td></td><td>El sistema halla los próximos dos valores de x e y respectivamente por Runge-Kutta 2 ó 4.</td></tr>
</table>
</td></tr>
</table>

3.6. CONCLUSIONES

En este capítulo se hizo un esbozo detallado de la propuesta de solución que brinda este trabajo para cumplir con los objetivos planteados. Se realizó un modelo de dominio para tener una mejor comprensión de las clases y términos a tratar, se listaron los requerimientos funcionales y no funcionales del sistema, además de determinar quién es al actor que interactuará con la BMN y de detallar los casos de usos que satisfacen los requerimientos del sistema anteriormente mencionados. Dando paso a la construcción del sistema propuesto, que facilitará el trabajo con los métodos numéricos para agilizar el desarrollo de las aplicaciones informáticas que requieran su uso.

CAPÍTULO 4: CONSTRUCCIÓN DE LA SOLUCIÓN PROPUESTA.

4.1. INTRODUCCIÓN

En el presente capítulo se hace una descripción de la construcción de la solución propuesta, se plantean los patrones utilizados en la elaboración de la aplicación, así como los modelos de diseño e implementación y pruebas realizadas. 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 esta sería una primera versión que deberá estar sujeta a cambios para mejorar su eficiencia y posibilitar mayor funcionalidad al usuario final.

4.2. PATRONES

Los desarrolladores con experiencia acumulan un repertorio tanto de principios generales como de soluciones basadas en aplicar ciertos estilos que les guían en la creación de software. Estos principios y estilos, si se codifican con un formato estructurado que describa el problema y la solución, y se les da un nombre, podrían llamarse “Patrones”.

De manera más simple un patrón es un par problema/solución con nombre que se puede aplicar en nuevos contextos, con consejos acerca de cómo aplicarlo en nuevas situaciones y discusiones sobre sus compromisos.

Un patrón es una descripción de un problema bien conocido que suele incluir:

- Descripción.
- Escenario de Uso.
- Solución concreta.
- Las consecuencias de utilizar este patrón.
- Ejemplos de implementación.
- Lista de patrones relacionados.

4.2.1. PATRONES DE ARQUITECTURA

Arquitectura en Capas

Los sistemas o arquitecturas en capas constituyen uno de los patrones que aparecen con mayor frecuencia mencionados, o, por el contrario, como una de las posibles encarnaciones de algún estilo más envolvente. Garlan y Shaw definen el estilo en capas como una organización jerárquica tal que cada capa proporciona servicios a la capa inmediatamente superior y se sirve de las prestaciones que le brinda la inmediatamente inferior. En algunos ejemplares, las capas internas están ocultas a todas las demás, menos para las capas externas adyacentes. En la práctica, las capas suelen ser entidades complejas, compuestas de varios paquetes o subsistemas.

Casos representativos de este estilo son muchos de los protocolos de comunicación en capas. En ellos cada capa proporciona un sustrato para la comunicación a algún nivel de abstracción, y los niveles más bajos suelen estar asociados con conexiones de hardware. El ejemplo más característico es el modelo OSI con los siete niveles: nivel físico, vínculo de datos, red, transporte, sesión, presentación y aplicación. El estilo también se encuentra en forma más o menos pura en arquitecturas de bases de datos y sistemas operativos.

Arquitecturas Orientadas a Objetos (AOO)

Los nombres alternativos para este patrón han sido Arquitecturas Basadas en Objetos, Abstracción de Datos y Organización Orientada a Objetos. Los componentes de este patrón son los objetos, o más bien instancias de los tipos de datos abstractos. En la caracterización clásica de David Garlan y Mary Shaw, los objetos representan una clase de componentes que ellos llaman managers, debido a que son 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.

En la semblanza de estos autores curiosamente no se establece como cuestión definitoria el principio de herencia. Ellos piensan que, a pesar de que la relación de herencia es un mecanismo organizador importante para definir los tipos de objeto en un sistema concreto, ella no posee una función arquitectónica directa. En particular, en dicha concepción la relación de herencia no puede concebirse como un conector, puesto que no define la interacción entre los componentes de un sistema. Además, en un escenario arquitectónico la herencia de propiedades no se restringe a los tipos de objeto, sino que puede incluir conectores e incluso estilos arquitectónicos enteros.

Resumiendo las características de las arquitecturas OO, se podría decir que:

Los componentes del estilo se basan en principios OO: encapsulamiento, herencia y polimorfismo. Son 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.

En cuanto a las restricciones, puede admitirse o no que una interfaz pueda ser implementada por múltiples clases.

En tantos componentes, los objetos interactúan a través de invocaciones de funciones y procedimientos.

Arquitectura de la BMN

Para conformar la arquitectura de la BMN se integran los patrones arquitectónicos antes expuestos, de manera tal que la BMN estaría dividida en dos capas, una capa interfaz de comunicación y otra para la lógica de la BMN, pero, como la segunda capa es mucho más compleja que la primera, se divide en subsistemas los cuales se organizan y comunican según la AOO.

4.2.2. PATRONES DE DISEÑO

Para lograr un mejor resultado en el diseño de la aplicación se tuvieron en cuenta algunos patrones para la asignación de responsabilidades (GRASP) tales como Experto y Creador. El primero plantea que siempre se debe asignar una responsabilidad al experto en información, o sea, la clase con toda la información necesaria para llevarla a cabo. El segundo expresa que la responsabilidad de crear una instancia de una determinada clase debe asignarse a otra clase, siempre que esta agregue, contenga, registre o utilice específicamente los objetos de aquella.

Además de los patrones GRASP se tuvo en cuenta otros patrones GoF(*Gang of Four*) como es el caso del patrón Fachada; este proporciona una interfaz unificada de alto nivel que, representando a todo un subsistema, facilite su uso. La fachada satisface a la mayoría de los clientes, sin ocultar las funciones de menor nivel a aquellos que necesiten acceder a ellas. Para seguir una buena técnica de diseño combinamos el patrón Fachada con el Fábrica Abstracta para así obtener una interfaz que permita crear objetos del subsistema de manera independiente a éste.

4.3. DIAGRAMA DE CLASES DEL DISEÑO

El modelo de diseño es un modelo de objetos que describe la realización de los CU, y sirve como una abstracción del modelo de implementación y el código fuente. Es usado como una entrada inicial en las actividades de implementación y prueba.

Es usado para concebir un documento del diseño del sistema de SW. Es abarcador, compuesto por artefactos que engloban todas las clases del diseño, subsistemas, paquetes, colaboraciones, y las relaciones entre ellos.

En el diseño se modela el sistema y se encuentra su forma (incluida la arquitectura) para que soporte todos los requisitos, incluyendo los no funcionales y las restricciones que se le suponen. Además impone una estructura del sistema que debemos esforzarnos por conservar lo más fielmente posible cuando demos forma al sistema.

Una clase de diseño es una construcción similar en la implementación del sistema:

- El lenguaje utilizado para especificar una clase del diseño es lo mismo que el lenguaje de programación.
- Las relaciones entre clases de diseño se traducen de manera directa al lenguaje:
  - generalización: herencia
  - asociaciones, agregaciones: atributos
- Los métodos de una clase del diseño tienen correspondencia directa con el correspondiente método en la implementación de las clases.
- Se pueden postergar algunos requisitos a implementación.
- Una clase de diseño puede proporcionar interfaces si tiene sentido hacerlo en el lenguaje de programación.

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.

Estos constituyen uno de los elementos más importantes dentro de un proyecto de software, ya que brindan una visión bastante completa de todo el sistema, mostrando todas las clases con sus métodos

y atributos, así como las relaciones entre estas. Un software se puede dividir por subsistemas, ese es el caso del sistema que se propone en este trabajo.

En el Anexo 1 se muestra una descripción de todas las clases del sistema.

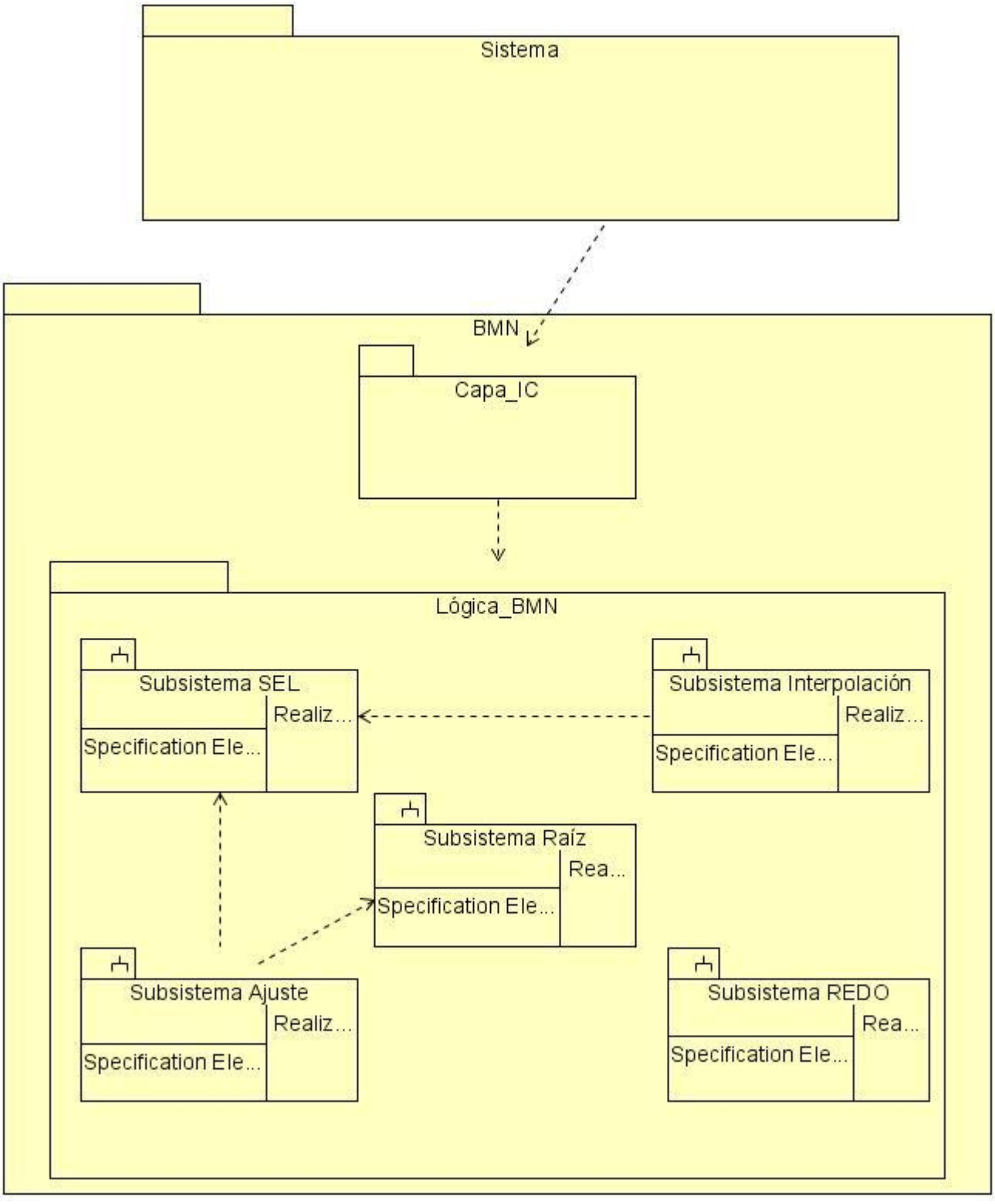

Fig. 18 Diagrama de Subsistemas de clases del Sistema.

4.3.1. SUBSISTEMA RAÍCES DE FUNCIONES

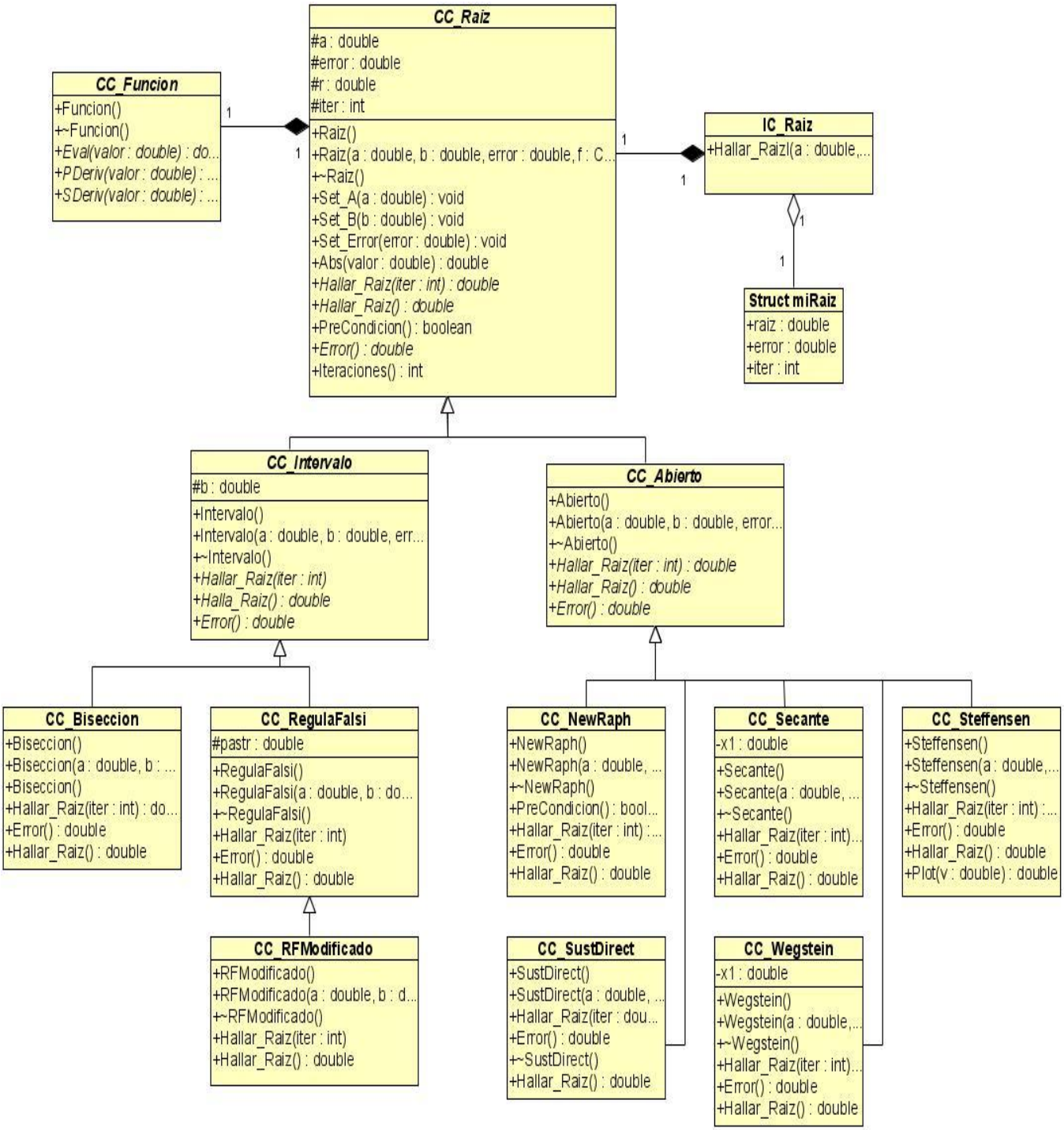

Fig. 19 Diagrama de clases subsistema Raíz.

4.3.2. SUBSISTEMA INTERPOLACIÓN

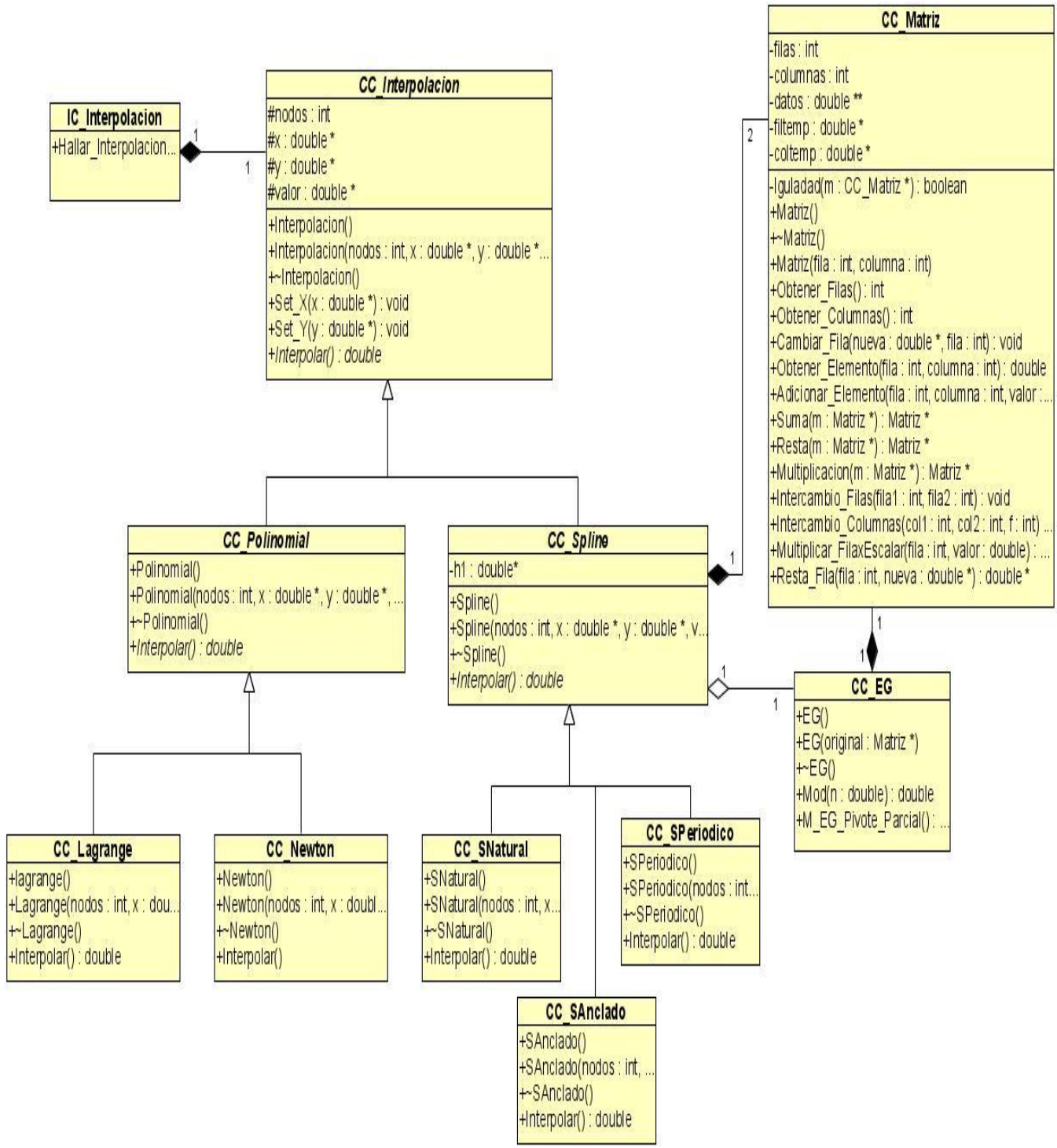

Fig. 20 Diagrama de clases subsistema interpolación.

4.3.3. SUBSISTEMA AJUSTE DE CURVAS

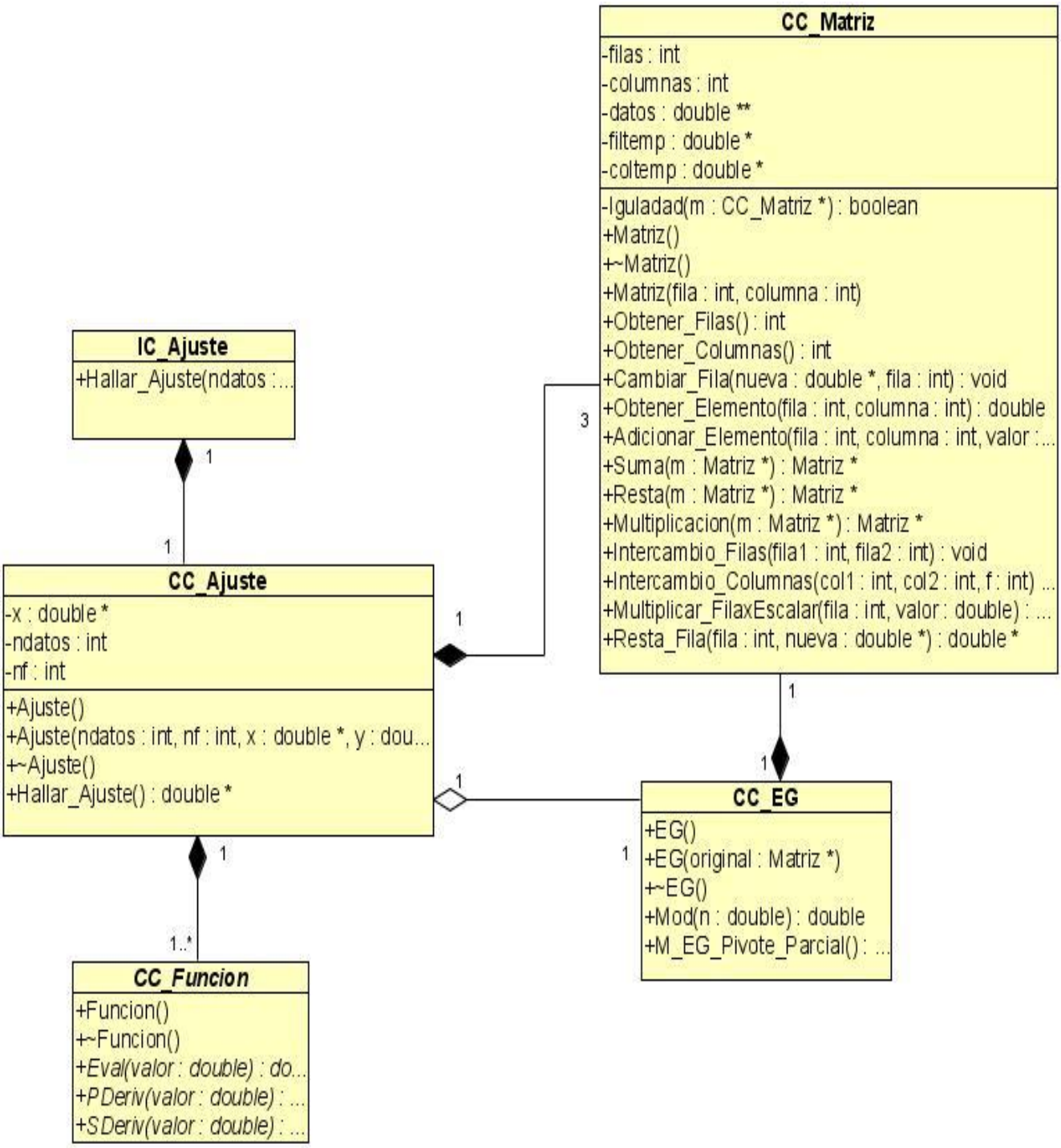

Fig. 21 Diagrama de clases subsistema ajuste.

4.3.4. SUBSISTEMA REDO

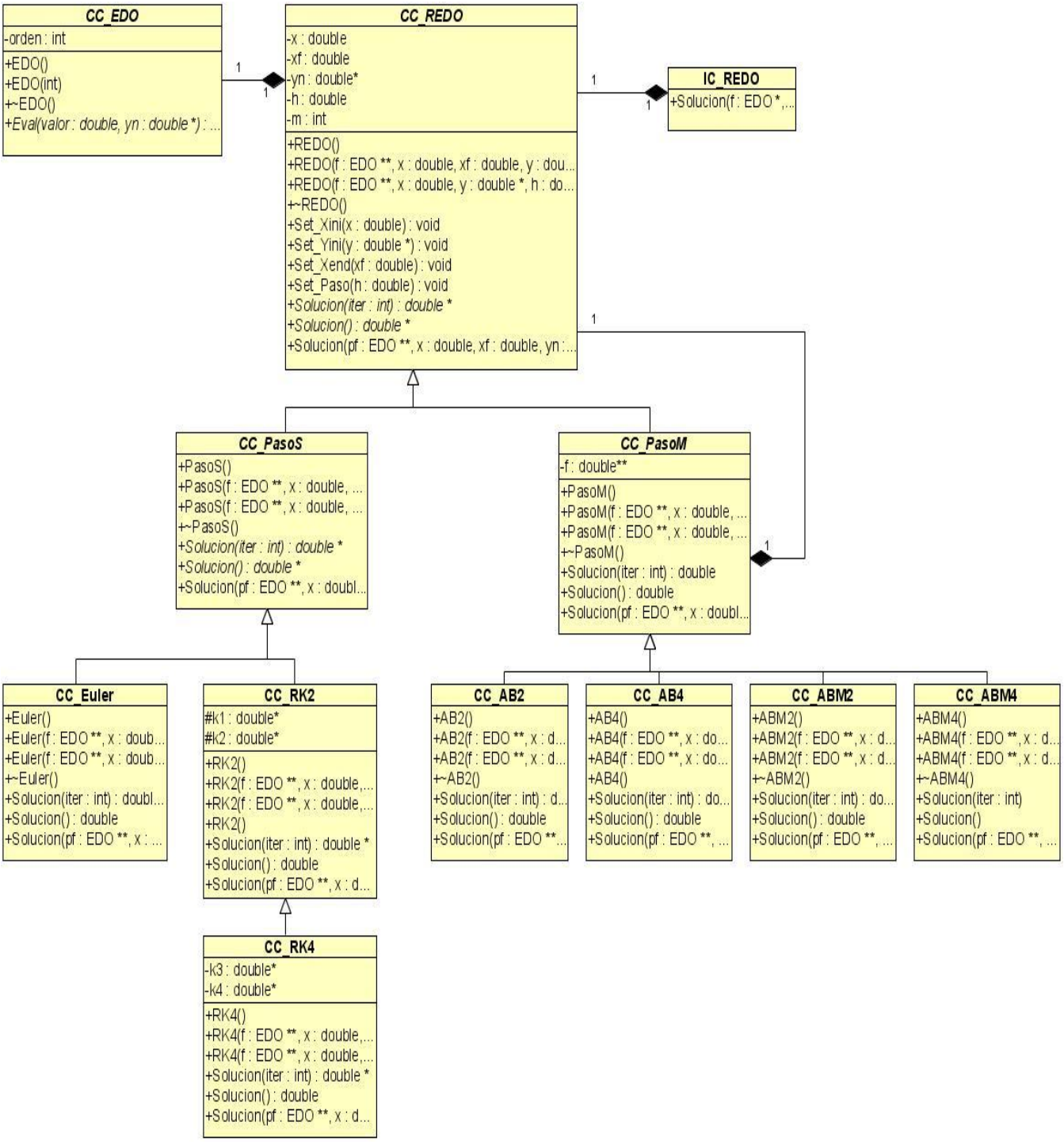

Fig. 22 Diagrama de clases subsistema REDO.

4.4. DIAGRAMAS DE INTERACCIÓN

Los diagramas de interacción están constituidos por dos tipos: los Diagramas de Colaboración y los Diagramas de secuencia. Ambos expresan información similar, pero en una forma diferente.

Los diagramas de secuencia forman parte del modelado dinámico del sistema y proporcionan una vista detallada de los casos de uso. Son diagramas que muestran la interacción organizada de objetos, mediante mensajes que se envían entre sí, en una secuencia de tiempo. Son útiles para observar la vida de los objetos en un sistema, identificar llamadas a realizar o posibles errores del modelado estático que imposibiliten el flujo de información.

Los Diagramas de Colaboración son una forma alternativa al diagrama de secuencia de mostrar un escenario. Este tipo de diagrama muestra las interacciones entre objetos organizadas entorno a los objetos y los enlaces entre ellos. Proporcionan la representación principal de un escenario, ya que las colaboraciones se organizan entorno a los enlaces de unos objetos con otros.

Los diagramas de colaboración resaltan la organización estructural de los objetos que envían y reciben mensajes. Un diagrama de colaboración muestra un conjunto de objetos, enlaces entre estos objetos y mensajes enviados y recibidos por estos objetos. Los objetos normalmente son instancias con nombre o anónimas de clases, pero también pueden representar instancias de otros elementos, como colaboraciones, componentes y nodos.

En el Anexo 2 se encuentran los diagramas de colaboración y secuencia correspondientes a los casos de uso de la BMN.

4.5. MODELO DE IMPLEMENTACIÓN

En el modelo de implementación se muestra el paso del diseño de clases a la creación de componentes físicos, que se traducen en ficheros .cpp debido a su implementación en C++ para el caso del diagrama de componentes de código fuente (fig. 23a) y para el diagrama de código compilado (fig. 23b) se traducen en los ficheros .lib y .a para las bibliotecas estáticas y .dll y .so para las dinámicas.

4.5.1. DIAGRAMAS DE COMPONENTES

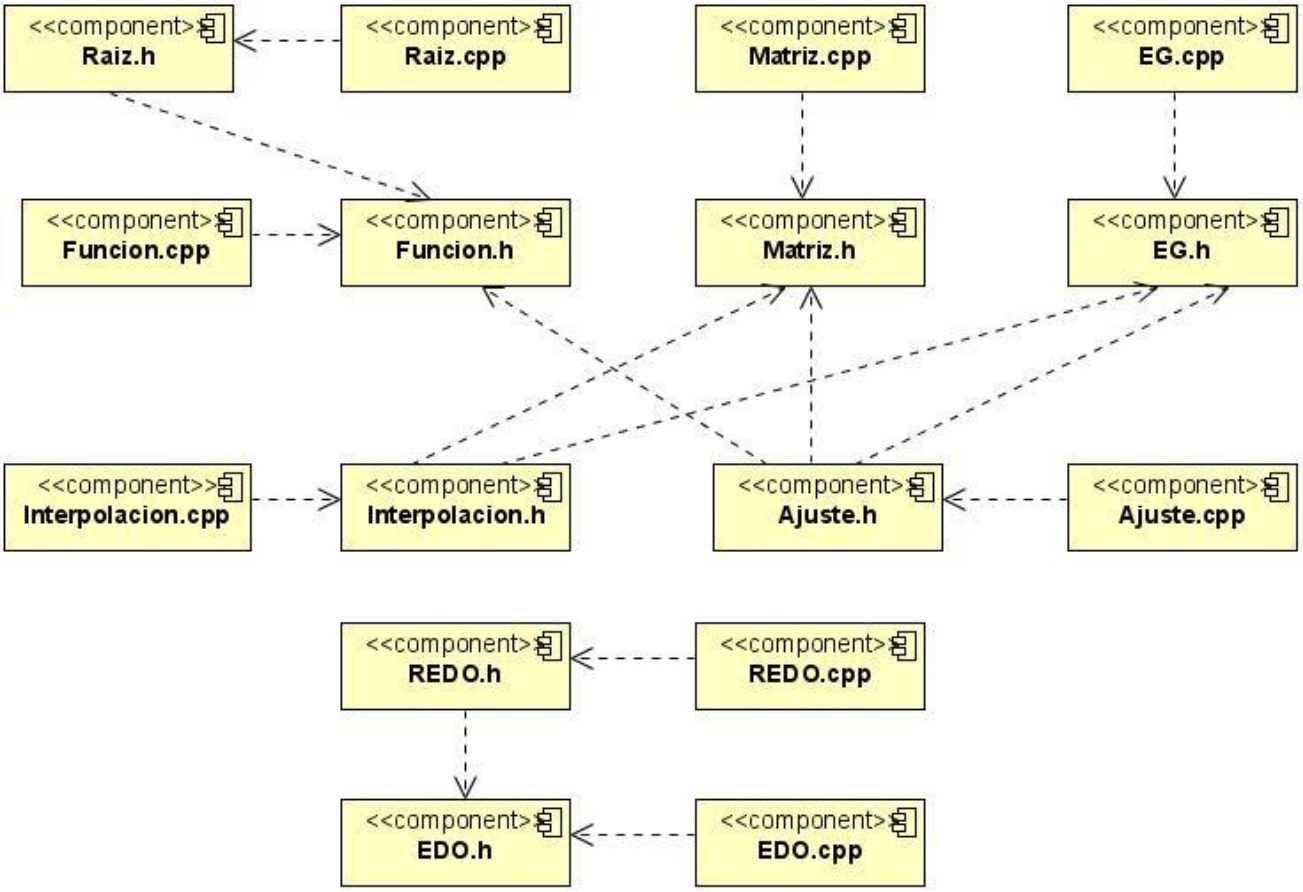

Fig. 23a Diagrama de Componentes código fuente.

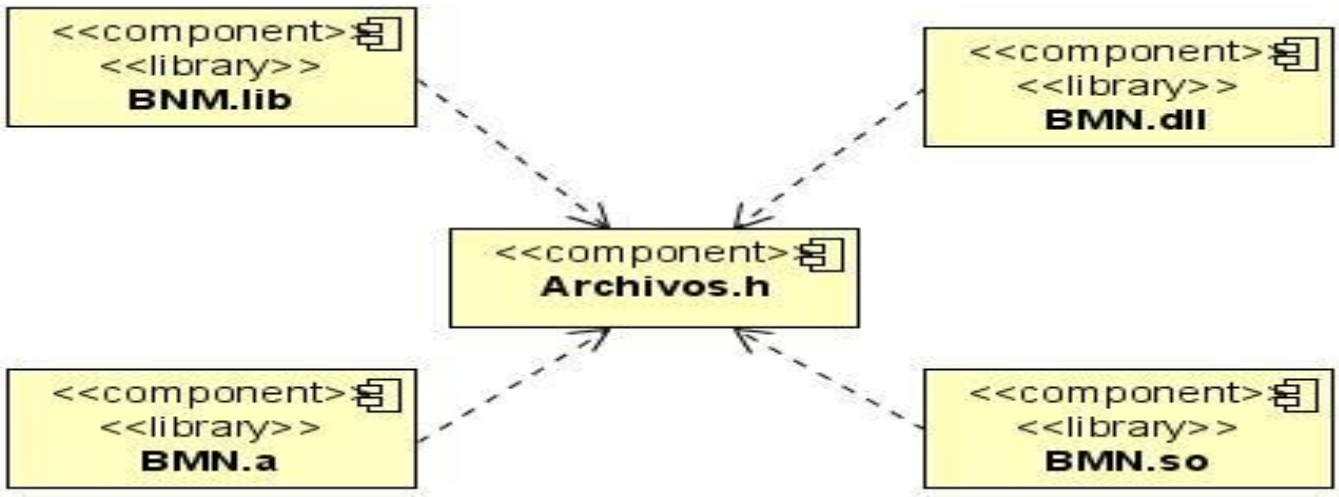

Fig. 23b Diagrama de Componentes código compilado.

4.6. PRUEBA DEL SISTEMA PROPUESTO

Tabla 8. Caso de prueba calcular la raíz de una función no lineal con valores válidos.

Caso de Uso	Calcular la raíz de una función no lineal.
Caso de Prueba	Calcular la raíz de una función no lineal con valores válidos.
Entrada (Bisección)	Intervalo [1, 7], error: 0.00005, función: $x^2 - 3x = 0$, cantidad de iteraciones: 100.
Salida	Raíz: 2.99997, error: 4.57764e-05, iteraciones: 16.
Entrada (Regula-Falsi)	Intervalo [1, 7], error: 0.00005, función: $x^2 - 3x = 0$, cantidad de iteraciones: 100.
Resultado	Raíz: 2.99995, error: 3.54319e-05, iteraciones: 20 .
Entrada (Regula-Falsi Modificado)	Intervalo [1, 7], error: 0.00005, función: $x^2 - 3x = 0$, cantidad de iteraciones: 100.
Resultado	Raíz: 3, error: 2.4144e-06, iteraciones: 7.
Entrada (Newton-Raphson)	Intervalo [1, 7], error: 0.00005, función: $x^2 - 3x = 0$, cantidad de iteraciones: 100.$x_0 = 4$.
Resultado	Raíz: 3.00008, error: 3.866442e-05, iteraciones: 22.
Entrada (Secante)	Intervalo [1, 7], error: 0.00005, función: $x^2 - 3x = 0$, cantidad de iteraciones: 100., $x_0 = 4$, $x_1 = 7$.
Resultado	Raíz: 3, error: 3.32742e-05, iteraciones: 5.
Entrada (Aproximaciones Sucesivas)	Intervalo [1, 7], error: 0.00005, función: $\sqrt{3x} = x$, cantidad de iteraciones: 100., $x_0 = 4$.
Resultado	Raíz: 3.00003, error: 2.63384e-05, iteraciones:14.
Entrada (Wegstein)	Intervalo [1, 7], error: 0.00005, función: $\sqrt{3x} = x$, cantidad de iteraciones: 100., $x_0 = 4$.
Resultado	Raíz: 3, error: 2.2599e-06, iteraciones:3.
Entrada (Steffensen)	Intervalo [1, 7], error: , función: $x^2 - 3x = 0$,cantidad de iteraciones: 100, $x_0 = 4$.
Resultado	Raíz: 3, error: 1.68814e-05, iteraciones: 5.

Tabla 9. Caso de prueba calcular la raíz de una función no lineal con valores no válidos.

Caso de Uso	Calcular la raíz de una función no lineal.
Caso de Prueba	Calcular la raíz de una función no lineal con valores no válidos.

Entrada (Bisección)	Intervalo [1, 2], error: 0.00005, función: $x^2 - 3x = 0$, cantidad de iteraciones: 100.
Resultado	"Error, intervalo no valido".
Entrada (Regula-Falsi)	Intervalo [1, 2], error: 0.00005, función: $x^2 - 3x = 0$, cantidad de iteraciones: 100.
Resultado	"Error, intervalo no valido".
Entrada (Regula-Falsi modificado)	Intervalo [1, 2], error: 0.00005, función: $x^2 - 3x = 0$, cantidad de iteraciones: 100.
Resultado	"Error, intervalo no valido".
Entrada (Newton-Raphson)	Intervalo [1, 2], error: 0.00005, función: $x^2 - 3x = 0$, cantidad de iteraciones: 100, $x_0 = 1$.
Resultado	"Error, intervalo o puntos de iteración no válidos".
Entrada (Secante)	Intervalo [1, 2], error: 0.00005, función: $x^2 - 3x = 0$, cantidad de iteraciones: 100, $x_0 = 1$, $x_1 = 2$.
Resultado	"Error, intervalo o puntos de iteración no validos".

Tabla 10. Caso de prueba calcular interpolación de un valor.

Caso de Uso	Hallar interpolación de un valor.
Caso de Prueba	Calcular interpolación de un valor.
Entrada (Polinomios de Lagrange)	Nodos de interpolación: $x_0 = 1,\ x_1 = 2,\ x_2 = 4$, imágenes: $y_0 = 2,\ y_1 = 3,\ y_2 = 1$, valor a interpolar: 2,35.
Resultado	Valor de p(2.35): 3,035.
Entrada (Newton)	Nodos de interpolación: $x_0 = 1,\ x_1 = 2,\ x_2 = 4$, imágenes: $y_0 = 2,\ y_1 = 3,\ y_2 = 1$, valor a interpolar: 2,35.
Resultado	Valor de p(2.35): 3,035.
Entrada (Spline cúbico natural)	Nodos de interpolación: $x_0 = 0, x_1 = 20, x_2 = 28, x_3 = 36, x_4 = 50, x_5 = 60, x_6 = 70, x_7 = 80, x_8 = 90$,imágenes: $y_0 = 20, y_1 = 10, y_2 = 9, y_3 = 10, y_4 = 20, y_5 = 22, y_6 = 20, y_7 = 12, y_8 = 12$, valor a interpolar: 2,35.
Resultado	Valor de p(2.35): 20.1978
Entrada (Spline cúbico periódico)	Nodos de interpolación: $x_0 = 1,\ x_1 = 2,\ x_2 = 3,\ x_3 = 4,\ x_4 = 5$, imágenes: $y_0 = 1,\ y_1 = 3,\ y_2 = 2, y_3 = 4$, $y_3 = 1$, valor a interpolar: 2,35.
Resultado	Valor de p(2.35): 2.76944.
Entrada (Spline cúbico anclado)	Nodos de interpolación: $x_0 = 2,\ x_1 = 3,\ x_2 = 5,\ x_3 = 6$, imágenes: $y_0 = 4,\ y_1 = 3,\ y_2 = 4, y_3 = 3, m_0 = 1, m_n = -1$, valor a interpolar: 2,35.
Resultado	Valor de p(2.35): 3.94916.

Tabla 11. Caso de prueba hallar ajuste de una función.

Caso de Uso	Hallar ajuste de una función.
Caso de Prueba	Hallar ajuste de una función.
Entrada (Ajuste por mínimos cuadrados)	Valores de x: $x_0 = 0.10,\ x_1 = 0.17,\ x_2 = 0.24,\ x_3 = 0.30,\ x_4 = 0.36,\ x_5 = 0.40,\ x_6 = 0.53,\ x_7 = 0.70,\ x_8 = 0.85,\ x_9 = 1.03$, Imágenes: $y_0 = 5.1, y_1 = 7.7, y_2 = 10.8, y_3 = 13.2, y_4 = 15.6, y_5 = 18.6, y_6 = 22.2, y_7 = 23.9, y_8 = 26.3, y_9 = 27.5$, Funciones: $f_1 = 1,\ f_1 = s,\ f_1 = s^2$,
Resultado	Coeficientes: $c_1 = -0.693878, c_2 = 56.5834, c_3 = -28.6967,$
Condiciones	

Tabla 12. Caso de Prueba calcular solución de un sistema de EDOs con n ecuaciones.

Caso de Uso	Calcular solución de un sistema de EDOs.
Caso de Prueba	Calcular solución de un sistema de EDOs con n ecuaciones.
Entrada (Euler)	$x_0 = 0$, $y_0 = 0.5$, $x_f = 4.0$, paso: 0.2 , lista de ecuaciones: $\frac{dx}{dy} = \frac{1}{2}y$.
Resultado	$y_f = 3.36375$
Entrada (Runge-Kutta 2)	$x_0 = 0$, $y_0 = 0.5$, $x_f = 4.0$, paso: 0.2 , lista de ecuaciones: $\frac{dx}{dy} = \frac{1}{2}y$.
Resultado	$y_f = 3.68312$
Entrada (Runge-Kutta 4)	$x_0 = 0$, $y_0 = 0.5$, $x_f = 4.0$, paso: 0.2 , lista de ecuaciones: $\frac{dx}{dy} = \frac{1}{2}y$.
Resultado	$y_f = 3.81359$
Entrada (Adams-Bashforth 2)	$x_0 = 0$, $y_0 = 0.5$, $x_f = 4.0$, paso: 0.2 , lista de ecuaciones: $\frac{dx}{dy} = \frac{1}{2}y$.
Resultado	$y_f = 3.66676$
Entrada (Predictores-Correctores 2)	$x_0 = 0$, $y_0 = 0.5$, $x_f = 4.0$, paso: 0.2 , lista de ecuaciones: $\frac{dx}{dy} = \frac{1}{2}y$.
Resultado	$y_f = 3.69822$

4.7. CONCLUSIONES PARCIALES

Para esta primera versión de la BMN, se tienen concebidos detalladamente el diseño de la aplicación completa, así como la arquitectura de la misma gracias al estudio de los patrones de arquitectura y diseño; además se presenta cómo queda el sistema expresado en componentes de implementación y las pruebas realizadas al mismo.

CAPÍTULO 5: ESTUDIO DE FACTIBILIDAD.

5.1. INTRODUCCIÓN

Estudiar la factibilidad de un proyecto se realiza para hacer una estimación del tamaño del producto y el esfuerzo que se requiere para que salga adelante, para estimar cuánto tiempo puede durar, el presupuesto que requiere para su desarrollo y lo que se necesita en términos de recursos humanos y materiales. En este trabajo de diploma se hace uso de la variante de estimación, Análisis de Puntos de Casos de Uso para obtener algunos de los datos mencionados anteriormente.

5.2. ESTIMACIÓN BASADA EN PUNTOS DE CASOS DE USOS

Se trata de un método de estimación del tiempo de desarrollo de un proyecto mediante la asignación de "pesos" a un cierto número de factores que lo afectan, para finalmente, contabilizar el tiempo total estimado para el proyecto a partir de esos factores. Con este método se obtiene la estimación del esfuerzo en horas-hombre, teniendo en cuenta solamente la funcionalidad especificada en cada caso de uso.

Para la aplicación de este método se desarrollan una serie de pasos, tales como:

1. Cálculo de Puntos de Casos de Uso sin ajustar.
2. Cálculo de Puntos de Casos de Uso ajustados.
3. Estimación de esfuerzo a través de los puntos de casos de uso. Cálculo del esfuerzo del flujo de trabajo Implementación.
4. Cálculo del Esfuerzo Total de todo el proyecto.
5. Costo del proyecto.

1. Cálculo de Puntos de Casos de Uso sin ajustar.

Para obtener los puntos de casos de usos desajustados se hace uso de la fórmula:
UUCP = UAW +UUCW

Donde:

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.

Paso 1.1: Cálculo del Factor de Peso de los Actores sin ajustar (UAW).

Se calcula teniendo en cuenta la cantidad de actores con que cuenta el sistema y el nivel de complejidad de ellos.

Tabla 13. Factor de peso de los actores sin ajustar.

Tipo de Actor	Descripción	Factores de peso	Actores	Total
Simple	Otro sistema que interactúa con el sistema a desarrollar mediante una interfaz de programación. (API, *Aplication Programming Interface*)	1	1	1
Medio	Otro sistema que interactúa con el sistema a desarrollar mediante un protocolo o una interfaz basada e texto.	2	0	0
Complejo	Una persona que interactúa con el sistema mediante una interfaz gráfica.	3	0	0

Se calcula mediante la siguiente ecuación:

UAW = Sumatoria (Factor * Actores)

UAW = 1

Paso 1.2: Cálculo del Factor de Peso de los Casos de Uso sin ajustar (UUCW).

Se calcula teniendo en cuenta la cantidad de casos de uso con que cuenta el sistema y el nivel de complejidad de ellos. O sea hay que tener en cuenta cuantos casos de usos cumplen con la categoría (simple, medio y complejo), para esto se necesita saber de cuantas transacciones cuenta cada caso de uso. Las transacciones se obtienen de la descripción de los casos de uso.

Tabla 14. Factor de peso de los casos de uso sin ajustar.

Tipo de CU	Descripción	Factores de peso	Cantidad de CU	Total
Simple	El casos de uso tiene de 1 a 3 transacciones	5	4	20
Medio	El caso de uso tiene de 4 a 7 transacciones	10	0	0
Complejo	El caso de uso tiene más de 8 transacciones	15	0	0

Se calcula mediante la siguiente ecuación:

UUCW = Sumatoria (Factor * CantCU)

UUCW = 20

Finalmente se tiene que:

UUCP = UAW +UUCW

UUCP =1 + 20

UUCP= 21

2. Cálculo de Puntos de Casos de Uso ajustados.

Para obtener los puntos de casos de usos ajustados se hace uso de la fórmula:

UCP=UUCP * TCF * EF

Donde:

UCP: Puntos de Casos de Uso ajustados.

UUCP: Puntos de Casos de Uso sin ajustar.

TCF: Factor de complejidad técnica.

EF: Factor de ambiente.

Paso 2.1: Cálculo del Factor de complejidad técnica (TCF).

$TCF = 0.6 + 0.01 * \Sigma (Peso_i * Valor_i)$ (Donde Valor es un número del 0 al 5)

Significado de los valores:

0: No presente o sin influencia.

1: Influencia incidental o presencia incidental.

2: Influencia moderada o presencia moderada.

3: Influencia media o presencia media.

4: Influencia significativa o presencia significativa.

5: Fuerte influencia o fuerte presencia.

Tabla 15. Factor de complejidad técnica.

Factor	Descripción	Peso	Valor	Comentario	Σ (Peso$_i$ * Valor$_i$)
T1	Sistema distribuido	2	0	El Sistema es centralizado	0
T2	Tiempo de respuesta	1	5	Se requiere que la velocidad de respuesta sea la mínima posible.	5
T3	Eficiencia del usuario final	1	1	Escasas restricciones de eficiencia.	1
T4	Procesamiento interno complejo	1	5	Si presenta cálculos Complejos.	5
T5	El código debe ser reutilizable	1	5	Se requiere que el código sea reutilizable.	5
T6	Facilidad de instalación	0.5	5	No se requiere de instalación.	2.5
T7	Facilidad de uso	0.5	4	Que sea de fácil uso para los usuarios	2
T8	Portabilidad	2	4	Se requiere que el sistema sea portable	8
T9	Facilidad de cambio	1	4	Se requiere adaptabilidad y que sea fácil de modificar	4
T10	Concurrencia	1	0	No hay concurrencia	0
T11	Incluye objetivos especiales de seguridad	1	0	No incluye objetivos de seguridad	0
T12	Provee acceso directo a terceras partes	1	0	El sistema no provee acceso a terceras partes	0
T13	Se requieren facilidades especiales de entrenamiento para los usuarios.	1	0	Sistema fácil de usar.	0
Total					**32.5**

TCF = 0.6 + 0.01 * 32.5

TCF = 0.925

Paso 2.2: Cálculo del Factor de ambiente (EF).

La preparación, habilidades que tienen las personas involucradas en el desarrollo del sistema también se tienen en cuenta en las estimaciones de tiempo. A cada uno de los factores que se muestran en la tabla se le asignan un valor de 0 a 5.

Tabla 16. Factor de ambiente

Factor	Descripción	Peso	Valor	Σ (Peso$_i$ * Valor$_i$)
E1	Familiaridad con el modelo de proyecto utilizado	1,5	3	4,5
E2	Experiencia en la aplicación	0,5	3	1,5
E3	Experiencia en la orientación a objetos.	1	3	3
E4	Capacidad del analista líder.	0,5	3	1,5
E5	Motivación.	1	5	5
E6	Estabilidad de requerimientos	2	4	8
E7	Personal Part–Time	-1	2	-2
E8	Dificultad del lenguaje de programación	-1	2	-2
			Sumatoria	19,5

Se calcula mediante la siguiente ecuación:

EF = 1.4 - 0.03 * Σ(Pesoi * Valor asignadoi)

EF = 1.4 - 0.03 * 19.5

EF = 0.815

Finalmente se tiene que:

UCP = UUCP * TCF * EF

UCP = 21 *0.925* 0.815

UCP = 15.831375

3. Estimación del esfuerzo a través de los puntos de casos de uso. Cálculo del esfuerzo del flujo de trabajo Implementación.

Para obtener el esfuerzo estimado en horas-hombre se hace uso de la fórmula:

E = UCP * CF

Donde:

E: Esfuerzo (horas-hombre).

UCP: Puntos de Casos de Uso ajustados.

CF: Factor de conversión.

Paso 3.1: Cálculo del Factor de conversión (CF).

CF = 20 horas-hombre (si $Total_{EF} \leq 2$)

CF = 28 horas-hombre (si $Total_{EF} = 3$ ó $Total_{EF} = 4$)

CF = abandonar o cambiar proyecto (si $Total_{EF} \geq 5$)

Como EF = 0.815 entonces CF = 20 horas-hombre.

Finalmente se tiene que:

E = UCP * CF

E = 15.831375* 20

E = 316.6275 horas-hombre

4. Cálculo del Esfuerzo Total de todo el proyecto.

Tabla 17. Distribución del esfuerzo entre las diferentes actividades.

Actividad	Porcentaje%	Horas-Hombre
Análisis	10	79.164375
Diseño	20	158.32875
Implementación	40	316.6275
Prueba	15	118.7465625
Sobrecargas	15	118.7465625
Total	100	791.64375

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

Suponiendo que una persona trabaje 8 horas por día, y un mes tiene como promedio 30 días y que los domingos sean días no laborables, en 1 mes se trabajan 26 días y la cantidad de horas que puede trabajar una persona en 1 mes equivale 208 horas.

Si ET = 791.64375 horas-hombre y por cada 208 horas se tiene 1 mes, entonces daría un ET = 3.8059 mes-hombre.

En el caso del desarrollo de la BMN el equipo de desarrollo de software es de 2 personas y todas realizan el mismo esfuerzo, entonces el problema analizado puede terminarse en aproximadamente 2 meses (1.9030 meses).

5. Costo del producto.

Para obtener el costo del producto se hace uso de la fórmula:
Costo = CHM x ET

Donde:
ET: Esfuerzo total (mes-hombre)
CHM: Costo hombre-mes

Paso 5.1: Cálculo del Costo hombre-mes.

Donde:
SBM: Salario básico mensual
Salario básico de una persona: SBM = $225.
CH: Cantidad de Hombre
CH = 2
Se calcula mediante está fórmula:
CHM = CH x SBM
CHM = 2 x $225
CHM = $ 450

Finalmente se tiene que:
Costo = CHM x ET
Costo = $ 450 x 3.8059
Costo = $ 1712.655 CUP.
Costo = $ 68,5062 CUC.

5.3. BENEFICIOS TANGIBLES E INTANGIBLES

Con el desarrollo de la BMN se obtienen beneficios tangibles e intangibles, los tangibles se refieren al producto que se utiliza como medio de apoyo en el desarrollo de aplicaciones que requieren el uso de métodos numéricos, dando a los programadores facilidades para el trabajo con estos, los intangibles representan el avance en cuanto a tiempo de desarrollo que tendrán las aplicaciones futuras.

5.4. ANÁLISIS DE COSTOS Y BENEFICIOS

Para desarrollar este sistema no se requiere de grandes gastos de recursos, ni tampoco de mucho tiempo (un costo de 1712.655 pesos y 2 meses), por lo que se vuelve factible su desarrollo en vista a los beneficios que reporta para el desarrollo de futuras aplicaciones tales como simuladores, software de sonido o diseño, entre otras.

5.5. CONCLUSIONES PARCIALES

A lo largo de este capítulo se ha detallado claramente los costos a incurrir, los recursos humanos implicados, el tiempo de desarrollo y los beneficios que aporta la terminación del producto en cuestión por lo que se llega a la conclusión que es factible la realización de esta aplicación principalmente por los aportes económicos y sociales que genera.

El estudio de factibilidad constituye una de las bases para evaluar el trabajo de investigación y es una valiosa herramienta que permite establecer con seguridad el alcance, el enfoque y los diferentes aspectos que deben considerarse al efectuar el análisis y la evaluación económica de los proyectos de software.

CONCLUSIONES GENERALES

Al realizar un estudio detallado de los conceptos fundamentales, algoritmos y las herramientas similares a este trabajo, se dio cumplimiento a los objetivos planteados con la conclusión de la BMN. De manera que esta cumpliera los requisitos funcionales establecidos para lograr las metas trazadas con la realización de un paquete de clases que muestra su funcionalidad con resultados satisfactorios.

Al ser el diseño del paquete de clases adaptable a los posibles cambios, posibilita que se puedan agregar nuevas funcionalidades en versiones posteriores sin hacer grandes modificaciones. Esto aumentaría la utilidad de la biblioteca con vistas a confeccionar un producto cada vez más completo y eficaz.

En esta primera versión de la BMN el aporte principal ha sido el desarrollo de una biblioteca de clases y un manual de trabajo y uso de los métodos numéricos que brinda al desarrollador de aplicaciones informáticas, tales como simuladores, software de sonido y diseño, etc., una serie de facilidades para implementar el módulo de métodos numéricos, específicamente, el cálculo de raíces, la interpolación de valores, el ajuste de funciones y la resolución de EDOs respondiendo a las necesidades del usuario en menor tiempo. Además de mejorar, fácil y rápidamente, nuestras aplicaciones que necesiten cálculo numérico, así como su robustez. Los programadores pueden concentrarse mejor en los aspectos especializados de su aplicación eliminando la dependencia de especialistas en el tema de métodos numéricos.

RECOMENDACIONES

- Continuar el estudio de nuevos métodos numéricos.
- Agregar otros métodos numéricos para aumentar el alcance de la biblioteca.
- Utilizar la documentación generada sobre métodos numéricos como material de consulta.
- Poner en explotación la biblioteca en un sistema real.

REFERENCIAS BIBLIOGRÁFICAS

BIBLIOGRAFÍA CITADA

1. **UCI.** wikiprod. *wikiprod.* [En línea] UCI, enero de 2008. http://wiki.prod.uci.cu.

2. **Vadillo, Fernando.** *Matemática Numérica.* 2002.

3. **Efren García Guerra, Juan Manuel.** [En línea] enero de 2008. http://www.itnuevolaredo.edu.mx/maestros/sis_com/ing_garciaguerra/docs/fan_UNIDAD%20I%2006.doc.

4. **Wales, Jimmy y Sanger, Larry.** Wikipedia. [En línea] enero de 2008. http://es.wikipedia.org/wiki/Funci%C3%B3n_matem%C3%A1tica.

5. **msn.** msn encarta. [En línea] enero de 2008. http://es.encarta.msn.com/encyclopedia_761575032/Funci%C3%B3n_(matem%C3%A1ticas).html.

6. **Wales, Jimmy y Sanger, Larry.** Wikipedia. [En línea] enero de 2008. http://es.wikipedia.org/wiki/Ra%C3%ADz_de_una_funci%C3%B3n.

7. **Sanger, Larry y Wales, Jimmy.** Wikipedia. [En línea] enero de 2008. http://es.wikipedia.org/wiki/Polinomio.

8. **Wales, Jimmy y Sanger, Larry.** Wikipedia. [En línea] enero de 2008. http://es.wikipedia.org/wiki/Spline.

9. **Sanger, Larry y Wales, Jimmy.** Wikipedia. [En línea] enero de 2008. http://es.wikipedia.org/wiki/Interpolaci%C3%B3n.

10. **Wales, Jimmy y Sanger, Larry.** Wikipedia. [En línea] enero de 2008. http://es.wikipedia.org/wiki/Ajuste_de_curvas.

11. **Sanger, Larry y Wales, Jimmy.** Wikipedia. [En línea] enero de 2008. http://es.wikipedia.org/wiki/Ecuaciones_diferenciales.

12. **Alvarez, Manuel, Guerra, Alfedro y Lau, Rogelio.** *Matemática Numérica.* 2. s.l. : Felix Varela.

13. aprende en linea. [En línea] enero de 2008. http://aprendeenlinea.udea.edu.co/lms/moodle/file.php?file=/107/ceros_reales.html.

14. **J.Scenna, Nicolás y autores, Colectivo de.** *Modelado, Simulación y optimización de procesos químicos.* 1999. 950-42-0022-2.

15. tauro. [En línea] diciembre de 2007. http://tauro.unex.es/vaguiti/METODOS_MATEMATICOS/metodos/442.html.

16. aprende en linea. [En línea] febrero de 2008. http://aprendeenlinea.udea.edu.co/lms/moodle/file.php?file=/107/Aprox_de_funciones.html.

17. Soft Perú. [En línea] enero de 2008. http://soft-peru.blogspot.com/.

18. AddLink. [En línea] enero de 2008. http://www.addlink.es/productos.asp?pid=76.

19. Addlink. [En línea] enero de 2008. http://www.addlink.es/productos.asp?pid=1.

20. Addlink. [En línea] enero de 2008. http://www.addlink.es/productos.asp?pid=41.

21. Addlink. [En línea] febrero de 2008. http://www.addlink.es/productos.asp?pid=185.

22. yorick. [En línea] febrero de 2008. http://yorick.sourceforge.net.

23. Yacas. [En línea] diciembre de 2007. http://yacas.sourceforge.net/yacas.html.

24. Yurix-project. [En línea] enero de 2008. http://yurix-project.sourceforge.net/doc/Proyecto%20Yurix%20v%200.9.9_update.pdf.

25. Octave. [En línea] enero de 2008. http://www.gnu.org/software/octave/about.html.

26. **UCI.** Wikiprod. [En línea] febrero de 2008. http://wiki.prod.uci.cu/index.php/Ingenier%C3%ADa.

27. **Wales, Jimmy y Sanger, Larry.** Wikipedia. [En línea] enero de 2008. http://es.wikipedia.org/wiki/Modelado.

28. **RUMBAUHG, J. y Jacobson, I.** *El Lenguaje Unificado de Modelado. Manual de referencia.* Madrid : Pearson Educación, 2000.

29. **RESSMAN, R. S.** Ingienería del software. Un enfoque práctico. [En línea] 2002. http://bibliodoc.uci.cu/pdf/reg02689.pdf.

30. **Molpeceres, Alberto.** *Procesos de desarrollo: RUP, XP y FDD.* 2002.

31. **LARMAN, C.** UML y Patrones. Introducción al análisis y diseño orientado a objetos . [En línea] 1999. http://bibliodoc.uci.cu/pdf/reg00061.pdf.

32. **PARADIGM, VISUAL.** Visual Paradigm for UML. [En línea] 2007. http://www.visual-paradigm.com/product/vpuml/.

33. **Grau, David.** Software libre II: Una estrategia decisiva de desarrollo. [En línea] marzo de 2008. http://www.juventudrebelde.cu/.

34. abadia digital. [En línea] enero de 2008. http://www.abadiadigital.com/noticia2010.html.

35. Que es una distribucion de linux? [En línea] febrero de 2008. http://bulma.net/body.phtml?nIdNoticia=1508.

36. **Java** [En línea] enero de 2008. http://java.ciberaula.com/articulo/tecnologia_orientada_objetos/.

37. **Systems, Zator.** Programación C++. [En línea] enero de 2008. http://www.zator.com/Cpp/E1.htm.

38. **UCI.** Wikiprod. [En línea] enero de 2008. http://wiki.prod.uci.cu/index.php/C.

39. —. Wikiprod. [En línea] enero de 2008. http://wiki.prod.uci.cu/index.php/C_Sharp.

40. C# Online. [En línea] enero de 2008. http://es.csharp-online.net/Main%28%29.

41. **UCI.** WikiProd. [En línea] enero de 2008. http://wiki.prod.uci.cu/index.php/Java.

42. —. Wikiprod. [En línea] enero de 2008. http://wiki.prod.uci.cu/index.php/Python.

43. mi lugar. [En línea] enero de 2008. http://users.servicios.retecal.es/tjavier/python/Un_poco_de_Python-2.html.

44. **UCI.** Wikiprod. [En línea] enero de 2008. http://wiki.prod.uci.cu/index.php/IDE.

45. —. Wikiprod. [En línea] febrero de 2008. http://wiki.prod.uci.cu/index.php/KDevelop.

46. [En línea] febrero de 2008. http://books.google.com.cu/books?id=jFKoq1psLk0C&pg=PA937&lpg=PA937&dq=kdevelop+caracteristicas&source=web&ots=6IubE6hfZb&sig=RBUYbbPf0gqqqTYoPOe4U3BR2vM&hl=es#PPA937,M1.

47. Kdevelop. [En línea] febrero de 2008. http://www.kdevelop.org/index.html?filename=main.html&set_lang=es.

48. Anjuta. [En línea] febrero de 2008. http://anjuta.sourceforge.net/features.

49. Eclipse. [En línea] febrero de 2008. http://www.eclipse.org/home/categories/languages.php.

50. Code Blocks. [En línea] febrero de 2008. http://www.codeblocks.org/features.

BIBLIOGRAFÍA CONSULTADA

1. Bulma. [En línea] http://bulma.net/body.phtml?nIdNoticia=723.

2. **Conte, S. D. y de Boor, Carl.** *Elementary numeric analysis an algorithmic approach.* New York : McGraw-Hill Book Company, 1980. 0-07-012447-7.

3. Eclipse [En línea] http://www.eclipse.org/whitepapers/eclipse-overview.pdf.

4. Ejemplo de desarrollo software. *Ejemplo de desarrollo software.* [En línea] http://www.dsic.upv.es/asignaturas/facultad/lsi/ejemplorup/.

5. Ejemplos java y C/linux. *Ejemplos java y C/linux.* [En línea] http://www.chuidiang.com/ood/metodologia/metodologia.php.

6. **Figueroa, Pablo.** Metodología de desarrollo de software Orientado a Objetos. *Metodología de desarrollo de software Orientado a Objetos.* [En línea] http://www.cs.ualberta.ca/~pfiguero/soo/metod/.

7. **Higham, Nicholas J.** *Accuracy and Stability of Numerical Algorithms.* Filadelphia : SIAM. O-8987 l-355-2.

8. **Infante del Rio, Juan Antonio y Rey Cabezas, Jose Maria.** *Metodos Numericos, teorias, problemas y practicas con matlab.* s.l. : piramide.

9. **Kravanja, Peter y Van Barel, Marc.** *Computing the Zeros of Analytic Functions .* Berlín : Springer. 0075-8434 .

10. **Landeta, Juan Manuel Izar.** *Elementos de metodos numericos para ingenieros.*

11. **Lara, Gabriel, Reyes, Luis y Vega, Sergio.** Desarrollo Basado en la Reutilización. *Desarrollo Basado en la Reutilización.* [En línea] 2006. http://www.uvmsf.cl/~fleon/wp-content/uploads/2006/12/reutilizacion.pdf.

12. **Letelier, Patricio y Penadés, Mª Carmen.** Métodologías ágiles para el desarrollo de software:eXtreme Programming (XP). s.l. : Universidad Politécnica de Valencia.

13. **L.F.Shampine, Allen, R.C. y S.Pruess.** *Fundamentals of numerical computing.* New York : JOHN WILEY & SONS, INC. 0-471-16363-5.

14. Metodos Numericos. *Metodos Numericos.* [En línea] http://aprendeenlinea.udea.edu.co/lms/moodle/file.php?file=/107/ceros_reales.html.

15. **Nash, J. C.** *Compact Numerical Methods for Computers, linear algebra and function minimisation.* 2. New York : s.n., 1990. 0-85274-318-1.

16. **Nash, Mary M. y Nash, Jhon C.** *Scientific Computing with PCs.* Ottawa : s.n., 1993. 0 88769 008 4.

17. **Neta, Beny.** *NumericalL solution of partial differential equations.* california : s.n., 2003.

18. *Numerical* Recipes *in C.* s.l. : Cambridge University Press. 0-521-43108-5.

19. PROGRAMACION EN C++. [En línea] http://ctc.aspira.org/PDF%20files/cplus.pdf.

20. **Quarteroni, Alfio, Sacco, Riccardo y Saleri, Fausto.** *Numerical Mathematics.* New York : Springer-Verlag New York, Inc., 2000. 0-387-98959-5.

21. **Saad, Y.** *Numerical Methods for large eigenvalue problems.* [DJVU] s.l. : Manchester University Press Series in Algorithms and Architectures for Advanced Scientific Computing.

22. **Sanchez, María A. Mendoza.** *Metodologias de desarrollo de software.* [pdf] s.l. : http://www.informatizate.net , 2004.

23. Softonic. [En línea] http://anjuta.softonic.com/linux.

24. **Stoer, J. y Bulirsch, R.** *Introduction to Numerical Analysis.* s.l. : Springer-Verlag . 0-387-97878-X .

25. TAURO.UNEX.ES. *TAURO.UNEX.ES.* [En línea] http://tauro.unex.es/vaguiti/METODOS_MATEMATICOS/metodos/.

26. **Thompson, William J.** *Computing for Scientist and engineers .* New York : JOHN WILEY & SONS, INC, 1992. O-471-54718-2.

27. yahoo. [En línea] http://ar.groups.yahoo.com/group/desarrollocsharp/.

ANEXOS

Anexo1

Tabla 18. Descripción de CC_Funcion.

Nombre: Funcion	
Para cada responsabilidad:	
Nombre:	virtual double Eval(double)
Descripción:	Método para evaluar una función.
Nombre:	virtual double PDeriv(double)
Descripción:	Método para hallar la primera derivada de una función.
Nombre:	virtual double SDeriv(double)
Descripción:	Método para hallar la segunda derivada de una función.

Tabla 19. Descripción de CC_Biseccion

Nombre: Biseccion	
Para cada responsabilidad:	
Nombre:	double Hallar_raiz(int)
Descripción:	Método para hallar la raíz de una función mediante bisección.
Nombre:	double Hallar_raiz()
Descripción:	Redefine el método: double Hallar_raiz(int).
Nombre:	double Error()
Descripción:	Método que devuelve el error hallado.

Tabla 20. Descripción de CC_RegulaFalsi

Nombre: RegulaFalse	
Atributo	**Tipo**
pastr	double
Para cada responsabilidad:	
Nombre:	double Error()
Descripción:	Método que devuelve el error hallado.
Nombre:	virtual double Hallar_raiz(int);
Descripción:	Método para hallar la raíz de una función mediante Regula-Falsi.
Nombre:	virtual double Hallar_raiz()
Descripción:	Redefine el método: double Hallar_raiz(int).

Tabla 21. Descripción de RF_Modificado

Nombre: RFModificado	
Para cada responsabilidad:	
Nombre:	double Hallar_raiz(int)
Descripción:	Método para hallar la raíz de una función mediante Regula-Falsi modificado.
Nombre:	double Hallar_raiz()
Descripción:	Redefine el método: double Hallar_raiz(int).

Tabla 22. Descripción de CC_NewRaph

Nombre: NewRaph	
Para cada responsabilidad:	
Nombre:	bool Pre_Condicion()
Descripción:	Comprueba que los datos cumplen las precondiciones para hallar una raíz.
Nombre:	double Hallar_raiz(int)
Descripción:	Método para hallar la raíz de una función mediante Newton-Raphson.
Nombre:	double Hallar_raiz()
Descripción:	Redefine el método: double Hallar_raiz(int).
Nombre:	double Error()
Descripción:	Método que devuelve el error hallado.

Tabla 23. Descripción de CC_Secante

Nombre: Secante	
Atributo	**Tipo**
x1	double
Para cada responsabilidad:	
Nombre:	double Hallar_raiz(int)
Descripción:	Método para hallar la raíz de una función mediante secante.
Nombre:	double Hallar_raiz()
Descripción:	Redefine el método: double Hallar_raiz(int).
Nombre:	double Error()
Descripción:	Método que devuelve el error hallado.

Tabla 24. Descripción de CC_SustDirect

Nombre: SustDirect	
Para cada responsabilidad:	
Nombre:	double Hallar_raiz(int)
Descripción:	Método para hallar la raíz de una función mediante aproximaciones sucesivas.
Nombre:	double Hallar_raiz()
Descripción:	Redefine el método: double Hallar_raiz(int).
Nombre:	double Error()
Descripción:	Método que devuelve el error hallado.

Tabla 25. Descripción de CC_Wegstein

Nombre: Wegstein	
Atributo	**Tipo**
x1	double
Para cada responsabilidad:	
Nombre:	double Hallar_raiz(int)
Descripción:	Método para hallar la raíz de una función mediante Wegstein.
Nombre:	double Hallar_raiz()
Descripción:	Redefine el método: double Hallar_raiz(int).
Nombre:	double Error()
Descripción:	Método que devuelve el error hallado.

Tabla 26. Descripción de CC Steffensen

Nombre: Steffensen	
Para cada responsabilidad:	
Nombre:	double Hallar_raiz(int)
Descripción:	Método para hallar la raíz de una función mediante Steffensen.
Nombre:	double Hallar_raiz()
Descripción:	Redefine el método: double Hallar_raiz(int).
Nombre:	double Error()
Descripción:	Método que devuelve el error hallado.
Nombre:	double Plot(double)
Descripción:	Método para elevar al cuadrado un valor.

Tabla 27. Descripción de CC_Lagrange

Nombre: Lagrange	
Para cada responsabilidad:	
Nombre:	double Interpolar()
Descripción:	Método para hallar la interpolación de un valor mediante Lagrange.

Tabla 28. Descripción de CC_Newton

Nombre: Newton	
Para cada responsabilidad:	
Nombre:	double Interpolar()
Descripción:	Método para hallar la interpolación de un valor mediante Newton.

Tabla 29. Descripción de CC_SNatural

Nombre: SNatural	
Para cada responsabilidad:	
Nombre:	double Interpolar()
Descripción:	Método para hallar la interpolación de un valor mediante spline natural.

Tabla 30. Descripción de CC_SPeriodico

Nombre: SPeriodico	
Para cada responsabilidad:	
Nombre:	double Interpolar()
Descripción:	Método para hallar la interpolación de un valor mediante spline periódico.

Tabla 31. Descripción de CC_SAnclado

Nombre: SAnclado	
Atributo	**Tipo**
m0	double
mn	double
Para cada responsabilidad:	
Nombre:	double Interpolar()
Descripción:	Método para hallar la interpolación de un valor mediante spline anclado.

Tabla 32. Descripción de CC_Ajuste

Nombre: Ajuste	
Atributo	**Tipo**
x	double*
f	Funcion**
ndatos	int
nf	int
G	Matriz*
Gt	Matriz*
Y	Matriz*
g	EG*
Para cada responsabilidad:	
Nombre:	double*Hallar_Ajuste()
Descripción:	Método para hallar los coeficientes de la función que se ajusta a los

Tabla 33. Descripción de CC_EDO

Nombre: EDO	
Atributo	**Tipo**
orden	int
Para cada responsabilidad:	
Nombre:	int Get_Orden()
Descripción:	Método que devuelve el orden de la ecuación diferencial.
Nombre:	virtual double Eval(double,double*)
Descripción:	Método para evaluar una ecuación diferencial.

Tabla 34. Descripción de CC_Euler

Nombre: Euler	
Para cada responsabilidad:	
Nombre:	double* Solucion()
Descripción:	Método para hallar la solución de la ecuación diferencial en cuestión por
Nombre:	double* Solucion(int)
Descripción:	Sobrecarga del método anterior pero fijando un numero de iteraciones.
Nombre:	double* Solucion(EDO**, double, double, double*, double, int)
Descripción:	Sobrecarga del método double* Solución() antes descrito pasando todos los valores necesarios por parámetros.

Tabla 35. Descripción de CC_RK2

Nombre: RK2	
Atributo	**Tipo**
k1	double*
k2	double*
Para cada responsabilidad:	
Nombre:	Virtual double* Solucion()
Descripción:	Método para hallar la solución de la ecuación diferencial en cuestión por el método de Runge-Kutta 2.
Nombre:	virtual double* Solucion(int)

Descripción:	Sobrecarga del método anterior pero fijando un numero de iteraciones.
Nombre:	virtual double* Solucion(EDO**, double, double, double*, double, int)
Descripción:	Sobrecarga del método virtual double* Solución() antes descrito pasando todos los valores necesarios por parámetros.

Tabla 36. Descripción de CC_RK4

Nombre: RK4	
Atributo	**Tipo**
k3	double*
k4	double*
Para cada responsabilidad:	
Nombre:	double* Solucion()
Descripción:	Método para hallar la solución de la ecuación diferencial en cuestión por el método de Runge-Kutta 4.
Nombre:	double* Solucion(int)
Descripción:	Sobrecarga del método anterior pero fijando un numero de iteraciones.
Nombre:	double* Solucion(EDO**, double, double, double*, double, int)
Descripción:	Sobrecarga del método double* Solución() antes descrito pasando todos los valores necesarios por parámetros.

Tabla 37. Descripción de CC_AB2

Nombre: AB2	
Para cada responsabilidad:	
Nombre:	double* Solucion()
Descripción:	Método para hallar la solución de la ecuación diferencial en cuestión por el método de Adams-Bashforth 2.
Nombre:	double* Solucion(int)
Descripción:	Sobrecarga del método anterior pero fijando un numero de iteraciones.
Nombre:	double* Solucion(EDO**, double, double, double*, double, int)
Descripción:	Sobrecarga del método double* Solución() antes descrito pasando todos los valores necesarios por parámetros.

Tabla 38. Descripción de CC_AB4

Nombre: AB4	
Para cada responsabilidad:	
Nombre:	double* Solucion()
Descripción:	Método para hallar la solución de la ecuación diferencial en cuestión por el método de Adams-Bashforth 4.
Nombre:	double* Solucion(int)
Descripción:	Sobrecarga del método anterior pero fijando un numero de iteraciones.
Nombre:	double* Solucion(EDO**, double, double, double*, double, int)
Descripción:	Sobrecarga del método double* Solución() antes descrito pasando todos los valores necesarios por parámetros.

Tabla 39. Descripción de CC_ABM2

Nombre: ABM2	
Para cada responsabilidad:	
Nombre:	double* Solucion()
Descripción:	Método para hallar la solución de la ecuación diferencial en cuestión por el método de Adams-Bashforth-Moulton 2.
Nombre:	double* Solucion(int)
Descripción:	Sobrecarga del método anterior pero fijando un numero de iteraciones.
Nombre:	double* Solucion(EDO**, double, double, double*, double, int)
Descripción:	Sobrecarga del método double* Solución() antes descrito pasando todos los valores necesarios por parámetros.

Tabla 40. Descripción de CC_ABM4

Nombre: ABM4	
Para cada responsabilidad:	
Nombre:	double* Solucion()
Descripción:	Método para hallar la solución de la ecuación diferencial en cuestión por el método de Adams-Bashforth-Moulton 4.
Nombre:	double* Solucion(int)
Descripción:	Sobrecarga del método anterior pero fijando un numero de iteraciones.
Nombre:	double* Solucion(EDO**, double, double, double*, double, int)
Descripción:	Sobrecarga del método double* Solución() antes descrito pasando todos los valores necesarios por parámetros.

Anexo 2

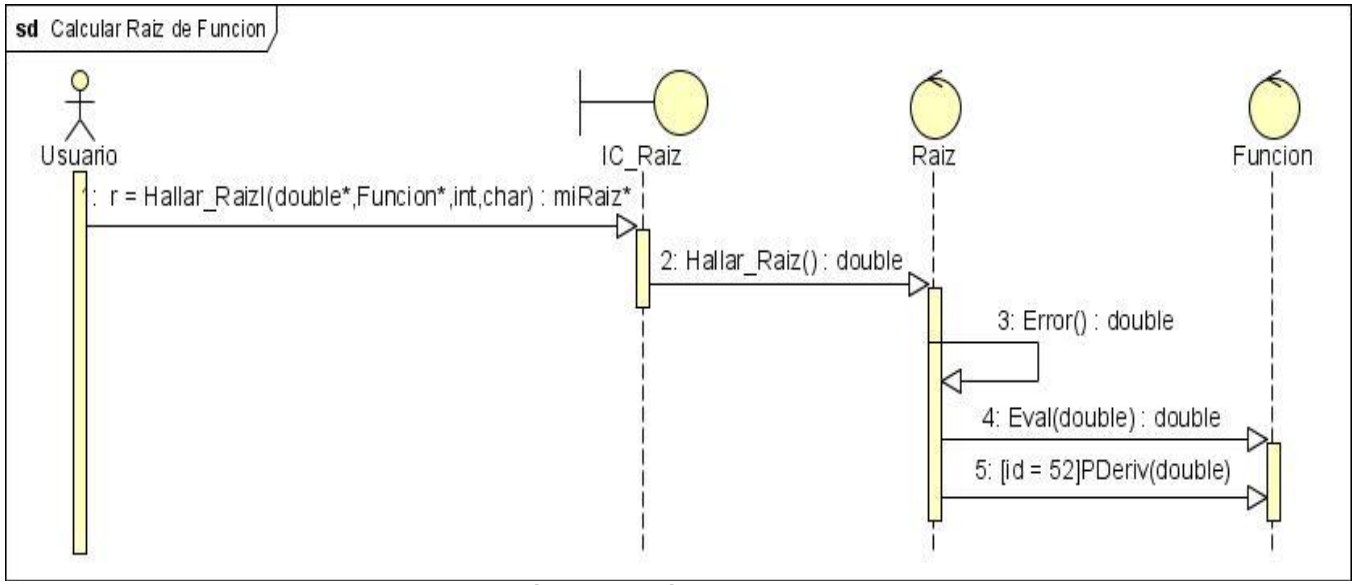

Fig. 24 Diagrama de secuencia calcular raíz de función.

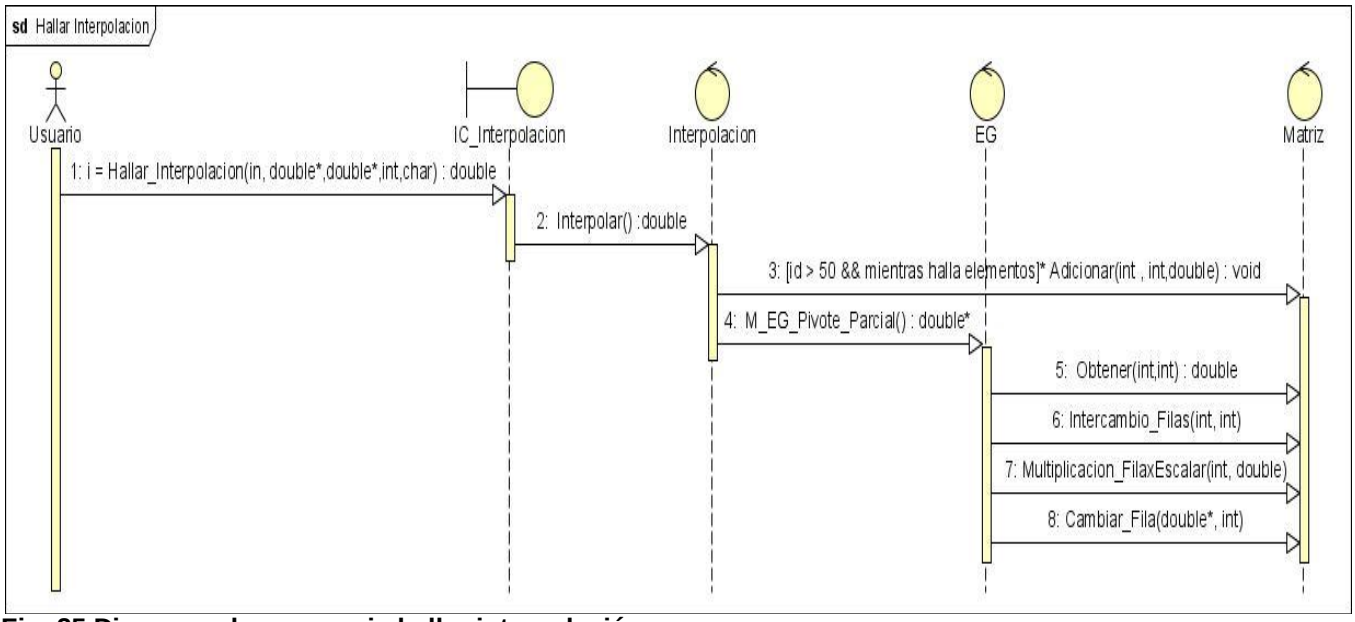

Fig. 25 Diagrama de secuencia hallar interpolación.

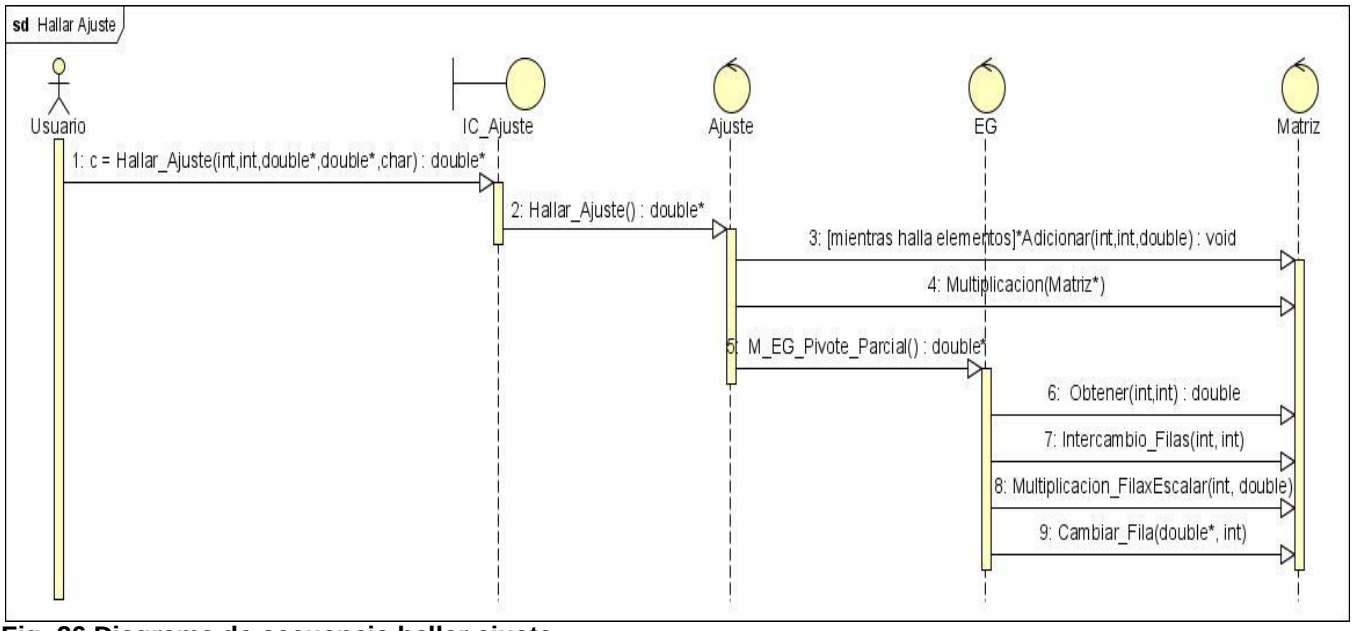

Fig. 26 Diagrama de secuencia hallar ajuste.

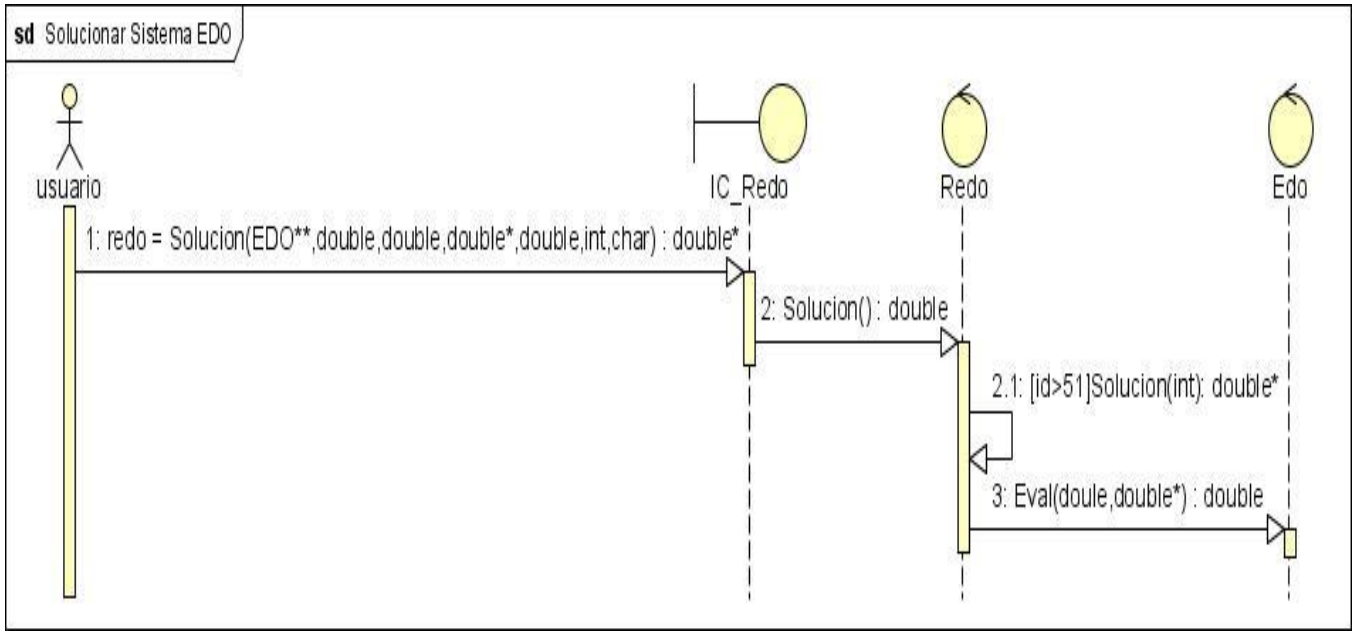

Fig. 27 Diagrama de secuencia solucionar sistema de EDO.