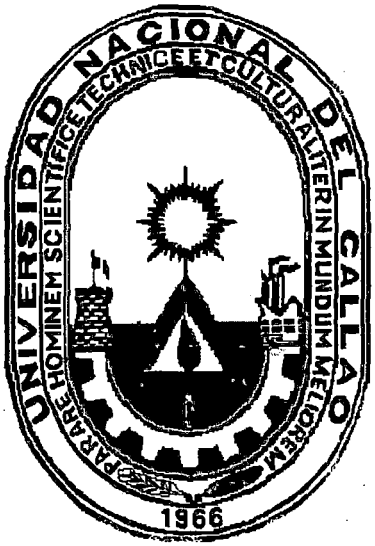
101

IF

ABR 2014

UNIVERSIDAD NACIONAL DEL CALLAO

VICERRECTORADO DE INVESTIGACIÓN



FACULTAD DE CIENCIAS NATURALES Y MATEMATICA

INSTITUTO DE INVESTIGACIÓN

INFORME FINAL DEL PROYECTO DE INVESTIGACIÓN

ALGORITMOS DE MÉTODOS NUMÉRICOS CODIFICADOS EN LENGUAJE DE PROGRAMACION C++ PARA LA SOLUCION DE PROBLEMAS DE ANÁLISIS NUMÉRICO

AUTOR : Lic. ELMER ALBERTO LEÓN ZÁRATE

PERIODO DE EJECUCIÓN: 01 de Diciembre del 2011 al 30 de Noviembre del 2013

Resolución **RR No.1275-2011-R** y Resolución **RCG No 032-2011-CG-FCNM**

FEBRÈRO DEL 2014

CALLAO – PERÚ
2014

I INDICE

II RESUMEN

La presente investigación tuvo como propósito la elaboración de un sistema computacional en el lenguaje de programación C++ que permita codificar los algoritmos para la solución de problemas de análisis numéricos.

Este trabajo de investigación tiene por título: **"ALGORITMOS DE MÉTODOS NUMÉRICOS CODIFICADOS EN LENGUAJE DE PROGRAMACION C++ PARA LA SOLUCION DE PROBLEMAS DE ANÁLISIS NUMÉRICO"** y ha sido preparado para apoyar la formación de los estudiantes de ciencias e ingeniería, quienes podrán aplicar este sistema cuando tengan que resolver problemas de análisis numérico.

En el capítulo del marco teórico se hace una revisión de los temas del análisis numérico y el lenguaje de programación C++.

En el capítulo de resultados obtenemos la codificación de algoritmos de métodos numéricos en el lenguaje de programación C++ para la solución de problemas de análisis numérico de los siguientes temas:

- Ecuaciones no lineales
- Sistemas de ecuaciones lineales y no lineales
- Aproximación funcional e Interpolación
- Integrales
- Derivadas
- Ecuaciones diferenciales ordinarias y parciales

III INTRODUCCIÓN

El Lenguaje de Programación C++ es Científico y Orientado a Objetos. También se debe tener en cuenta que no se presentan aplicaciones en el área del análisis numérico, por lo tanto la bibliografía acerca de este tema es muy escasa.

El Análisis numérico es el desarrollo y estudio de procedimientos para resolver problemas con ayuda de una computadora. Las técnicas mediante las cuales es posible formular problemas de tal manera que se puedan resolver usando operaciones aritméticas. Combinan dos de las herramientas más importantes de la ingeniería: matemáticas y computadoras.

La ventaja fundamental del análisis numérico es que puede obtenerse una respuesta numérica, aun cuando un problema no tenga solución analítica.

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. Existen algoritmos de métodos numéricos aplicados a las diferentes temas del análisis numérico; las mismas que se pretenden implementar en este trabajo de investigación utilizando el Lenguaje de programación C++.

El objetivo principal de este trabajo es codificar algoritmos de métodos numéricos en el lenguaje de programación C++ para la solución de problemas de análisis numérico.

Los métodos numéricos son técnicas mediante las cuales es posible formular problemas matemáticos de tal forma que puedan resolverse usando operaciones aritméticas.

El análisis numérico trata de diseñar métodos para "aproximar" de una manera eficiente las soluciones de problemas expresados matemáticamente.

El objetivo principal del análisis numérico es encontrar soluciones "aproximadas" a problemas complejos utilizando sólo las operaciones más simples de la aritmética. Se requiere de una secuencia de operaciones algebraicas y lógicas que producen la aproximación a la solución del problema matemático.

Los métodos numéricos pueden ser aplicados para resolver procedimientos matemáticos en:

- Ecuaciones no lineales
- Sistemas de ecuaciones lineales y no lineales
- Aproximación funcional e Interpolación
- Integrales
- Derivadas
- Ecuaciones diferenciales ordinarias y parciales

Los métodos numéricos se aplican en áreas tales como: Ciencias, Ingeniería Industrial, Ingeniería Química, Ingeniería Civil, Ingeniería Mecánica y la Ingeniería eléctrica.

Sin embargo, la posibilidad de utilizar un medio computacional provee un diseño didáctico que contribuye a un ambiente experimental y dinámico, al pasar de los medios tradicionales de pizarrón, tiza, plumón, lápiz y papel a presentar ejemplos y problemas simulados con los que es posible experimentar y explorar sus propiedades; por lo que es una posibilidad de acercamiento a la enseñanza y aprendizaje del análisis numérico. Un software tutorial que promueva la enseñanza y el aprendizaje de conceptos importantes del análisis numérico presenta algunas ventajas para los dos agentes del proceso, profesor y alumno.

Por ello, en este trabajo se propone la creación de un entorno computacional para la enseñanza y aprendizaje, con una componente didáctica integrada, que presente un problema en cuya resolución se muestre la necesidad de los conceptos propios del análisis numérico.

Con ello, se espera promover una mayor comprensión de algunos conceptos del análisis numérico al usar la computadora como un elemento auxiliar cognitivo que incorpora actividades didácticas. Es importante hacer notar que el objetivo de este estudio se centra en crear un sistema computacional que sirva como una herramienta de apoyo en el proceso de enseñanza-aprendizaje del análisis numérico para el alumno bajo la dirección de un profesor. En este sentido, la descripción de las herramientas computacionales a programar en Lenguaje de Programación C++ para cubrir los principios didácticos es la componente principal en este trabajo. En particular, las rutinas de cálculo numérico y algoritmos son aportaciones

determinantes en el manejo de una aritmética continua real en una aritmética discreta, que es la que maneja la computadora. Así como la programación de objetos computacionales que incorporan una aritmética de racionales y rutinas mediante las que se implementan las ideas de corte didáctico.

Un Trabajo de investigación como el propuesto contribuye a que el alumno resuelva problemas por sí mismo, señale sus fallas y corrija sus errores sometiendo sus respuestas de forma repetida; le proporciona ayuda para resolver los problemas en el momento que lo necesite independientemente de la presencia física del profesor pero como si éste lo estuviera acompañando. Además, se adecua a cada estudiante de acuerdo a su nivel de profundidad en el tratamiento de los conceptos y a la necesidad de ayudas; no se imponen soluciones ni métodos. Para el profesor resulta de gran ayuda en la etapa de ejercitación, descargándole de la labor rutinaria y agotadora de evaluar las tareas asignadas y señalar los errores de manera individual, permitiéndole entonces profundizar en el desarrollo conceptual del tema más que en la parte operativa que le requiere tanto tiempo. Así, se contribuye a modificar el papel de los agentes involucrados en el proceso de enseñanza-aprendizaje, alumno y profesor, en roles de auto aprendizaje y tutor respectivamente.

IV MARCO TEÓRICO

4.1 ANALISIS NUMERICO

El análisis numérico tiene que ver con el desarrollo y evaluación de métodos para calcular los resultados numéricos requeridos a partir de datos numéricos (Scheid Francis, 1972).

Los análisis numéricos han sido desarrollados con el objeto de resolver problemas matemáticos cuya solución es difícil o en ocasiones imposible de resolver por medio de los análisis tradicionales.

Las soluciones que ofrecen los análisis numéricos son aproximaciones de los valores reales y, por tanto, se tendrá un cierto grado de error que será conveniente determinar.

Son herramientas muy poderosas utilizadas para la solución de problemas. Tienen la capacidad de manejar sistemas de ecuaciones grandes, no linealidades y geometrías complicadas las cuales son bastante comunes en la práctica de la ingeniería y que, en ocasiones, son imposibles de resolverse de una manera analítica. Por lo tanto los análisis numéricos amplían la habilidad de quien los estudia para la solución de problemas.

La gran mayoría de los análisis numéricos son procesos cíclicos o iterativos, en los cuales se repite una serie de pasos y estos se basan en las denominadas ecuaciones o fórmulas de recurrencia, las cuales relacionan dos o más elementos consecutivos de una sucesión de números, funciones, matrices, etc.

4.1.1.-ECUACIONES NO LINEALES

A.- ALGORITMO DEL METODO DEL PUNTO FIJO

El método de punto fijo es parte de una serie de métodos los cuales son llamados métodos abiertos, se basan en fórmulas que requieren de un solo valor x o de un par de ellos pero que no necesariamente encierran a la raíz. Como tales, algunas veces divergen o se alejan de la raíz a medida que se crece el número de iteraciones. Sin embargo, Cuando los métodos abiertos

convergen, en general lo hacen mucho más rápido que los métodos que usan intervalos. Se empieza el análisis de los métodos abiertos con una versión simple que es útil para ilustrar su forma general y también para demostrar el concepto de la convergencia.

En este método se despeja la ecuación f(x)=0 de tal forma que X quede al lado izquierdo de la ecuación:

$$X = g(x).....(1)$$

Esta transformación se puede llevar a cabo por medio de operaciones algebraicas o simplemente agregando x a cada lado de la ecuación original.

La utilidad de la ecuación (1) es que proporciona una fórmula para predecir un valor de x en función de x. De esta forma, dada una aproximación inicial a la raíz, x_i, la ecuación (1) puede ser utilizada para obtener una nueva aproximación x_{i+1}, expresada por la formula iterativa:

$$X_{i+1} = g(x_i).....(2)$$

Para encontrar una raíz real de la ecuación g(x)=x proporcionar la función G(X) y los:

DATOS : Valor inicial X_0, criterio de convergencia EPS y número máximo de iteraciones MAXIT.

RESULTADOS: La raíz aproximada X o un mensaje de falla.

PASO 1: Hacer I =1

PASO 2: Mientras I < MAXIT, realizar los pasos 3 a 6.

PASO 3: Hacer $X=G(X_0)$ (calcular (X_i))

PASO 4: Si $ABS(X-X_0) <= EPS$ entonces IMPRIMIR X y TERMINAR.
De otro modo CONTINUAR.

PASO 5: Hacer I = I + 1.

PASO 6: Hacer X_0 = (actualiza X_0).

PASO 7: IMPRIMIR mensaje de falla: "EL METODO NO CONVERGE A UNA RAIZ" y TERMINAR.

B.- ALGORITMO DEL METODO DE NEWTON-RAPHSON

El método de Newton-Raphson asume que la función f(x) es derivable sobre un intervalo cerrado [a,b]. Entonces f(x) tiene una pendiente definida y una única línea tangente en cada punto en [a,b]. La tangente en (x0,f(x0)) es una aproximación a la curva de f(x) cerca del punto (x0,f(x0)). En consecuencia, el cero de la línea tangente es una aproximación del cero de f(x).

Formula de recurrencia:

$$x_n = x_{n+1} - \frac{f(x)}{f'(x)}$$

el método de newton de segundo orden:

$$\frac{1}{\Delta x_n} = \frac{1}{2}\left[\frac{f''(x_n)}{f'(x_n)}\right] - \left[\frac{f'(x_n)}{f(x_n)}\right]$$

El método de Newton-Raphson es uno de los métodos numéricos para resolver un problema de búsqueda de raíces f(x)=0 más poderosos y conocidos.

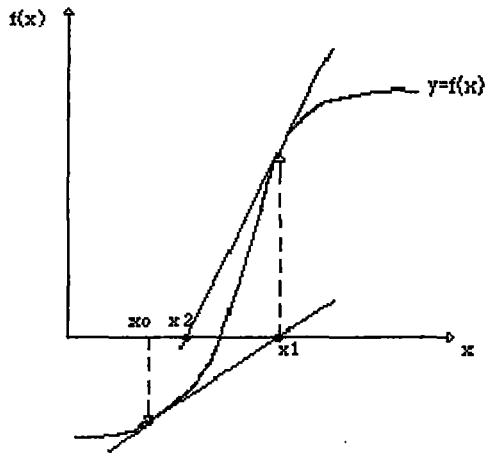


Figura 4.1.1. Aproximaciones con tangentes sucesivas.

Esta figura muestra como se obtienen las aproximaciones usando tangentes sucesivas. Comenzando con la aproximación inicial x_0, la aproximación x_1 es la

intersección con el eje x de la línea tangente a la gráfica de f en $(x_o,f(x_o))$. La aproximación x_2 es la intersección con el eje de las x de la línea tangente a la gráfica de f en $(x_1,f(x_1))$ y así sucesivamente.

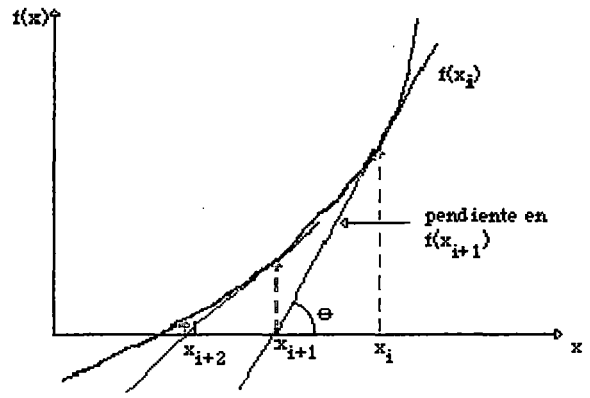


Figura 4.1.2. Aproximaciones conociendo los valores x_i´s.

$m=\tan \Theta = f'(x)$ pendiente de la recta que pasa por $(x_i,f(x_i))$.

$$m=\tan \Theta = \text{Cateto opuesto / Cateto adyacente} = \frac{f(x_i)}{x_i - x_{i+1}} - f'(x_i)$$

Lo que en realidad se desea saber es cuánto vale x_{i+1} para tomarlo en cuenta para la siguiente iteración, y así seguiría sucesivamente, hasta obtener la raíz.

$$\frac{f(x_i)}{x_i - x_{i+1}} = f'(x_i)$$

$$\frac{x_i - x_{i+1}}{f(x_i)} = \frac{1}{f'(x_i)}$$

$$-x_{i+1} = -x_i + \frac{f(x_i)}{f'(x_i)}$$

$$x_{i+1} - x_i \quad \frac{f(x_i)}{f'(x_i)}$$

Para encontrar una raíz real de la ecuación f(x)=0 proporcionar la función F(X) y su derivada DF(X) y los:

DATOS : Valor inicial X_0, criterio de convergencia EPS, criterio de exactitud EPS1 y número máximo de iteraciones MAXIT.

RESULTADOS: La raíz aproximada X o un mensaje de falla.

PASO 1: Hacer I =1

PASO 2: Mientras I < MAXIT, realizar los pasos 3 a 7.

PASO 3: Hacer $X=(X_0- F(X_0))/DF(X_0)$ (calcular (X_i))

PASO 4: Si $ABS(X-X_0) < EPS$ entonces IMPRIMIR X y TERMINAR.
De otro modo CONTINUAR.

PASO 5: Si $ABS(F(X)) < EPS1$ entonces IMPRIMIR X y TERMINAR.
De otro modo CONTINUAR.

PASO 6: Hacer I = I + 1.

PASO 7: Hacer $X_0 = X$.

PASO 8: IMPRIMIR mensaje de falla: "EL METODO NO CONVERGE A UNA RAIZ" y TERMINAR.

C.- ALGORITMO DEL METODO DE LA SECANTE

El método de la secante parte de dos puntos y estima la tangente (es decir, la pendiente de la recta) por una aproximación de acuerdo con la expresión:

$$f'(x_0) = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

El método de la secante nos proporciona el siguiente punto de iteración:

$$x_2 = x_0 - \frac{x_1 - x_0}{f(x_1) - f(x_0)} f(x_0)$$

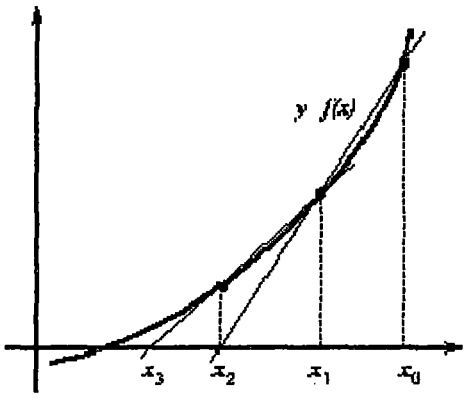


Figura 4.1.3. Representación geométrica del método de la secante.

Para encontrar una raíz real de la ecuación f(x)=0, dada f(x) analíticamente, proporcionar la función F(X) y los:

DATOS : Valores iníciales X_0, X_1; criterio de convergencia EPS, criterio de exactitud EPS1 y número máximo de iteraciones MAXIT.

RESULTADOS: La raíz aproximada X o un mensaje de falla.

PASO 1: Hacer I =1

PASO 2: Mientras I < MAXIT, realizar los pasos 3 a 8.

PASO 3: Hacer $X=(X_0- (X_1-X_0)*F(X_0))/(F(X_1)-F(X_0))$

PASO 4: Si $ABS(X-X_1) < EPS$ entonces IMPRIMIR X y TERMINAR.
De otro modo CONTINUAR.

PASO 5: Si $ABS(F(X)) < EPS1$ entonces IMPRIMIR X y TERMINAR.
De otro modo CONTINUAR.

PASO 6: Hacer $X_0 = X_1$.

PASO 7: Hacer $X_1 = X$.

PASO 8: Hacer I = I+1.

PASO 9: IMPRIMIR mensaje de falla: "EL METODO NO CONVERGE A UNA RAIZ" y TERMINAR.

D.- ALGORITMO DEL METODO DE POSICION FALSA

Se busca una solución de la ecuación f(x) = 0, una raíz de f(x), se parte de un intervalo inicial $[a_0,b_0]$ con $f(a_0)$ y $f(b_0)$ de signos opuestos, lo que garantiza que en su interior hay al menos una raíz). El algoritmo va obteniendo sucesivamente en cada paso un intervalo más pequeño [ak, bk] que sigue incluyendo una raíz de la función f.

A partir de un intervalo [ak, bk] se calcula un punto interior ck:

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

Dicho punto es la intersección de la recta que pasa por (a,f(ak)) y (b,f(bk)) con el eje de abscisas.

Se evalúa entonces f(ck). Si es suficientemente pequeño, ck es la raíz buscada. Si no, el próximo intervalo [ak+1, bk+1] será:

- [ak, ck] si f(ak) y f(ck) tienen signos opuestos;
- [ck, bk] en caso contrario.

Para encontrar una raíz real de la ecuación f(x)=0, dada f(x) analíticamente, proporcionar la función F(X) y los:

DATOS : Valores iníciales XI y XD que forman un intervalo en donde se halla una raíz X (F(XI)*F(XD)<0), criterio de convergencia EPS, criterio de exactitud EPS1 y número máximo de iteraciones MAXIT.

RESULTADOS: La raíz aproximada X o un mensaje de falla.

PASO 1: Hacer I =1; FI = F(XI); FD=F(XD).

PASO 2: Mientras I < MAXIT, realizar los pasos 3 a 8.

PASO 3: Hacer XM=(XI*FD – XD*FI)/(FD-FI);
FM=F(XM).

PASO 4: Si ABS(FM) < EPS1 entonces IMPRIMIR XM y TERMINAR.

PASO 5: Si ABS(XD-XI) < EPS Hacer
XM=(XD+XI)/2; IMPRIMIR "LA RAIZ BUSCADA ES:". IMPRIMIR XM y TERMINAR.

PASO 6: Si FD*FM>0, hacer XD=XM (actualiza XD).

PASO 7: Si FD*FM<0, hacer XI=XM (actualiza XI).

PASO 8: Hacer I = I+1.

PASO 9: IMPRIMIR mensaje de falla: "EL METODO NO CONVERGE A UNA RAIZ" y TERMINAR.

4.1.2.- SISTEMAS DE ECUACIONES LINEALES

A.- ALGORITMO DEL METODO DE ELIMINACION DE GAUSS

El proceso de eliminación de *Gauss* consiste en realizar transformaciones elementales en el sistema inicial (intercambio de filas, intercambio de columnas, multiplicación de filas o columnas por constantes,

operaciones con filas o columnas, . . .), destinadas a transformarlo en un sistema triangular superior, que resolveremos por remonte. Además, la matriz de partida tiene el mismo determinante que la matriz de llegada, cuyo determinante es el producto de los coeficientes diagonales de la matriz.

Gracias a un teorema, podemos afirmar que, si la matriz A es estrictamente diagonal dominante, es decir, si

$$|a_{ii}| > \sum_{\substack{j=1 \\ j \neq i}}^{N} |a_{ij}| \quad , \qquad \forall i = 1,\ldots,N$$

entonces, el método de *Gauss* se puede llevar a cabo.

El método de *Gauss normal,* presenta dos problemas, el primero proviene de encontrar en alguna de las sucesivas etapas, algún coeficiente diagonal igual a cero y el segundo es debido a los errores de redondeo que se pueden producir en este método.

El primer método que se presenta usualmente en álgebra, para la solución de ecuaciones algebraicas lineales simultáneas, es aquel en el que se eliminan las incógnitas mediante la combinación de las ecuaciones. Este método se conoce como *Método de Eliminación*. Se denomina *eliminación Gaussiana* si en el proceso de eliminación se utiliza el esquema particular atribuido a *Gauss*.

Utilizando el método de Gauss, un conjunto de ***n*** ecuaciones con ***n*** incógnitas se reduce a un *sistema triangular equivalente* (un sistema equivalente es un sistema que tiene iguales valores de la solución), que a su vez se resuelve fácilmente por "sustitución inversa"; un procedimiento simple que se ilustrará con la presentación siguiente.

El esquema de Gauss empieza reduciendo un conjunto de ecuaciones simultáneas, a un *sistema triangular equivalente* como:

$$
\begin{aligned}
a_{11}X_1 + a_{12}X_2 + a_{13}X_3 + a_{14}X_4 + \ldots + a_{1n}X_n &= C_1 \\
a^1_{22}X_2 + a^1_{23}X_3 + a^1_{24}X_4 + \ldots + a^1_{2n}X_n &= C^1_2 \\
a^2_{33}X_3 + a^2_{34}X_4 + \ldots + a^2_{3n}X_n &= C^2_3 \\
&\ldots \\
a^{n-2}_{(n-1)(n-1)}X_{n-1} - a^{n-2}_{(n-1)n}X_n &= C^{n-2}_{n-1} \\
a^{n-1}_{nn}X_n &= C^{n-1}_n
\end{aligned}
\qquad (1)
$$

en el cual los superíndices indican los nuevos coeficientes que se forman en el proceso de reducción. La reducción real se logra de la siguiente manera:

1. La primera ecuación se divide entre el coeficiente de *X1* en esa ecuación para obtener:

$$
X_1 + \frac{a_{12}}{a_{11}}X_2 + \frac{a_{13}}{a_{11}}X_3 + \ldots + \frac{a_{1n}}{a_{11}}X_n = \frac{C_1}{a_{11}} \qquad (2)
$$

2. La ecuación (2) se multiplica entonces por el coeficiente de *X1* de la segunda ecuación (2) y la ecuación que resulta se resta de la misma, eliminando así *X1*. La ecuación (2) se multiplica entonces por el coeficiente de *X1* de la tercera ecuación (2), y la ecuación resultante se resta de la misma para eliminar *X1* de esa ecuación. En forma similar, X1 se elimina de todas las ecuaciones del conjunto excepto la primera, de manera que el conjunto adopta la forma:

$$
\begin{aligned}
a_{11}X_1 + a_{12}X_2 + a_{13}X_3 + a_{14}X_4 + \ldots + a_{1n}X_n &= C_1 \\
a^1_{22}X_2 + a^1_{23}X_3 + a^1_{24}X_4 + \ldots + a^1_{2n}X_n &= C^1_2 \\
a^1_{32}X_2 + a^1_{33}X_3 + a^1_{34}X_4 + \ldots + a^1_{3n}X_n &= C^1_3 \\
&\ldots \\
a^1_{n2}X_2 + a^1_{n3}X_3 + a^1_{n4}X_4 + \ldots + a^1_{nn}X_n &= C^1_n
\end{aligned}
\qquad (3)
$$

3. La ecuación utilizada para eliminar las incógnitas en las ecuaciones que la siguen se denomina *Ecuación Pivote*. En la ecuación pivote, el coeficiente de la incógnita que se va a eliminar de las ecuaciones que la siguen se denomina el *Coeficiente Pivote* (a_{11} en los pasos previos).

4. Siguiendo los pasos anteriores, la segunda ecuación (3) se convierte en la *ecuación pivote*, y los pasos de la parte 1 se repiten para eliminar X_2 de todas las ecuaciones que siguen a esta ecuación pivote.

   Esta reducción nos conduce a:

$$\begin{aligned} a_{11}X_1 + a_{12}X_2 + a_{13}X_3 + a_{14}X_4 + \ldots + a_{1n}X_n &= C_1 \\ a^1_{22}X_2 + a^1_{23}X_3 + a^1_{24}X_4 + \ldots + a^1_{2n}X_n &= C^1_2 \\ a^2_{33}X_3 + a^2_{34}X_4 + \ldots + a^2_{3n}X_n &= C^2_3 \\ \ldots & \\ a^2_{n3}X_3 + a^2_{n4}X_4 + \ldots + a^2_{nn}X_n &= C^2_n \end{aligned} \quad (4)$$

5. A continuación se utiliza la tercer ecuación (4) como ecuación pivote, y se usa el procedimiento descrito para eliminar X_3 de todas las ecuaciones que siguen a la tercer ecuación (4). Este procedimiento, utilizando diferentes ecuaciones pivote, se continúa hasta que el conjunto original de ecuaciones ha sido reducido a un conjunto triangular tal como se muestra en la ecuación (1).

6. Una vez obtenido el conjunto triangular de ecuaciones, la última ecuación de este conjunto equivalente suministra directamente el valor de X_n (ver ecuación 1). Este valor se sustituye entonces en la antepenúltima ecuación del conjunto triangular para obtener un valor de X_{n-1}, que a su vez se utiliza junto con el valor de X_n en la penúltima ecuación del conjunto triangular para obtener un valor X_{n-2} y así sucesivamente. Este es el procedimiento de sustitución inversa al que nos referimos previamente.

Ejemplo 4.1.1. Para ilustrar el método con un conjunto numérico, apliquemos estos procedimientos a la solución del siguiente sistema de ecuaciones:

$$\begin{aligned} X1 + 4\,X2 + X3 &= 7 \\ X1 + 6\,X2 - X3 &= 13 \\ 2\,X1 - X2 + 2\,X3 &= 5 \end{aligned}$$

Utilizando como ecuación pivote la primera ecuación (el coeficiente pivote es unitario), obtenemos:

$$X1 + 4\,X2 + X3 = 7$$
$$2\,X2 - 2\,X3 = 6$$
$$9\,X2 + (0)\,X3 = -9$$

A continuación, utilizando la segunda ecuación del sistema anterior como ecuación pivote y repitiendo el procedimiento, se obtiene el siguiente sistema triangular de ecuaciones:

$$X1 + 4\,X2 + X3 = 7$$
$$2\,X2 - 2\,X3 = 6$$
$$-9\,X3 = 18$$

Finalmente mediante sustitución inversa, comenzando con la última de las ecuaciones del sistema anterior se obtienen los siguientes valores:

$$X3 = -2$$
$$X2 = 1$$
$$X1 = 5$$

Para obtener la solución de un sistema de ecuaciones lineales Ax=b y el determinante de A, proporcionar los:

DATOS : N numero de ecuaciones, A matriz coeficiente y b vector de términos independientes.

RESULTADOS: El vector solución X y el determinante de A o un mensaje de falla "HAY UN CERO EN LA DIAGONAL PRINCIPAL".

PASO 1: Hacer DET=1

PASO 2: Hacer I = 1

PASO 3: Mientras I <= N-1, repetir los pasos 4 a 14.

PASO 4: Hacer DET = DET*A(I,I)

PASO 5: Si DET = 0 IMPRIMIR mensaje "HAY UN CERO EN LA DIAGONAL PRINCIPAL" y TERMINAR. De otro modo CONTINUAR.

PASO 6: Hacer K = I + 1.

PASO 7: Mientras K<=N, repetir los pasos 8 a 13.

PASO 8: Hacer J = I+1

PASO 9: Mientras J<=N, repetir los pasos 10 y 11.

PASO 10: Hacer A(K,J)=A(K,J)-A(K,I)*A(I,J)/A(I,I)

PASO 11: Hacer K=K+1

PASO 12: Hacer b(K) = b(K)-A(K,I)*b(I)/A(I,I)

PASO 13: Hacer K = K+1

PASO 14: Hacer I = I+1

PASO 15: Hacer DET = DET * A(N,N)

PASO 16: Si DET = 0 IMPRIMIR mensaje "HAY UN CERO EN LA DIAGONAL PRINCIPAL" y TERMINAR. De otro modo CONTINUAR.

PASO 17: Hacer x(N) = b(N)/A(N,N)

PASO 18: Hacer I = N-1

PASO 19: Mientras I >= 1, repetir los pasos 20 a 26.

PASO 20: Hacer x(I) = b(I)

PASO 21: Hacer J = 1

PASO 22: Mientras J <= N, repetir los pasos 23 y 24.

PASO 23: Hacer x(I) = x(I)-A(I,J)*x(J)

PASO 24: Hacer J = J+1

PASO 25: Hacer x(I) = x(I)/A(I,I)

PASO 26: Hacer I = I-1

PASO 27: IMPRIMIR x y DET y TERMINAR.

B.- ALGORITMO DEL METODO DE CHOLESKY

Se considera aquí un procedimiento de factorización desarrollado para aquellos sistemas de ecuaciones en donde la matriz de coeficientes es simétrica y definida positiva. Este tipo de matrices resultan muy usuales en la solución de numerosos problemas de ingeniería mediante técnicas numéricas como elementos finitos.

El método de Cholesky utiliza el preconocimiento de las características mencionadas para realizar una descomposición más eficaz.

En líneas generales, la secuencia paso a paso considerada por Cholesky es que la matriz $\boldsymbol{A}$ es factorizada como $\boldsymbol{A=L.L^T}$, es decir, ahora solamente los coeficientes de $\boldsymbol{L}$ son incógnitas ya que su traspuesta hace las

veces de matriz ***U*** en el proceso de descomposición.

Para factorizar una matriz positiva definida en la forma $L\,L^T$, proporcionar los:

DATOS : **N**, el orden de la matriz y sus elementos.

RESULTADOS : La matriz ***L.***

PASO 1. Hacer $L(1,1) = A(1,1) ** 0.5$

PASO 2. Hacer $I = 2$

PASO 3. Mientras $I \leq N$, repetir los pasos 4 y 5.

PASO 4. Hacer $L(I,1) = A(I,1)/L(1,1)$

PASO 5. Hacer $I = I + 1$

PASO 6. Hacer $I = 2$

PASO 7. Mientras $I \leq N$, repetir los pasos 8 a 24

PASO 8. Hacer $S = 0$

PASO 9. Hacer $K = 1$

PASO 10 Mientras $K \leq I - 1$, repetir los pasos 11 y 12.

PASO 11 Hacer $S = S + L(I,K) ** 2$

PASO 12. Hacer $K = K + 1$

PASO 13. Hacer $L(I,I) = (A(I,I) - S) ** 0.5$

PASO 14. Si $I = N$ ir al paso 25

PASO 15. Hacer $J = I + 1$

PASO 16. Mientras $J \leq N$, repetir los pasos 17 a 23.

PASO 17. Hacer $S = 0$

PASO 18. Hacer $K = 1$

PASO 19. Mientras $K \leq I - 1$, repetir los pasos 20 y 21

PASO 20. Hacer $S = S + L(I,K) * L(J,K)$

PASO 21. Hacer $K = K + 1$

PASO 22. Hacer $L(J,I) = (A(J,I) - S)/L(I,I)$

PASO 23. Hacer $J = J + 1$

PASO 24. Hacer $I = I + 1$

PASO 25 IMPRIMIR $L\,y$ TERMINAR.

4.1.3.- SISTEMAS DE ECUACIONES NO LINEALES

A.- ALGORITMO DEL METODO DE NEWTON-RAPHSON MULTIVARIABLE

Este método es similar al de la Secante, la diferencia esencial radica en que en la Secante se utiliza el método de diferencias divididas para aproximar f '(x). El método de Newton-Raphson asume que la función f(x) es derivable sobre un intervalo cerrado [a,b]. Entonces f(x) tiene una pendiente definida y una única línea tangente en cada punto en [a,b]. La tangente en $(x_0,f(x_0))$ es una aproximación a la curva de f(x) cerca del punto $(x_0,f(x_0))$. En consecuencia, el cero de la línea tangente es una aproximación del cero de f(x). Calculamos la primera aproximación, x_1, como el cero de la línea tangente en un punto inicial x_0 dado.

Calculamos la segunda aproximación, x_2, como el cero de la línea tangente en la primera aproximación x_1.

Siguiendo el esquema mostrado más abajo, las primeras dos aproximaciones de raíces usando el método Newton-Raphson, se buscan con el mismo criterio del método de la Bisección:

Repitiendo el mismo proceso obtenemos cada vez mejores aproximaciones de la raíz de la función f(x).

Sea x_0 el valor inicial. La pendiente en x_0 está dada por $f'(x_0)$. La ecuación de la línea tangente en x_0 está dada por:

$$y - f(x_0) = f'(x_0)\,(x - x_0) \quad \ldots\ldots\ldots\ldots \quad (1)$$

La primera aproximación x_1 es obtenida como la raíz de (1). Así $(x_1,0)$ es un punto sobre la ecuación (1).

De aquí se tiene, $0 - f(x_0) = f'(x_0)\,(x_1 - x_0)$

Esto es, $x_1 - x_0 = - f(x_0) / f'(x_0)$ despejando tenemos, $x_1 = x_0 - f(x_0) / f'(x_0)$

Por construcción similar obtenemos:

$$x_{n+1} = x_n - f(x_n) / f'(x_n)$$

Para encontrar una solución aproximada de un sistema de ecuaciones no lineales f(x)=0, proporcionar la matriz jacobiana ampliada con el vector de funciones y los

DATOS : El numero de ecuaciones N, el vector de valores iniciales x, el número máximo de iteraciones MAXIT y el criterio de convergencia EPS.

RESULTADOS: El vector solución xn o mensaje "NO CONVERGE"

PASO 1: Hacer K=1

PASO 2: Mientras K <= MAXIT, repetir los pasos 3 a 9.

PASO 3: Evaluar la matriz jacobiana aumentada.

PASO 4: Resolver el sistema lineal del paso 3.

PASO 5: Hacer xn = x + h.

PASO 6: Si |xn - x |>EPS ir al paso 8. De otro modo continuar.

PASO 7: IMPRIMIR xn y TERMINAR

PASO 8: Hacer x=xn

PASO 9: Hacer K=K+1

PASO 10: IMPRIMIR "NO CONVERGE" Y TERMINAR

B.- ALGORITMO DEL METODO DE BROYDEN

El método de Broyden considera el método de la secante y establece una generalización de él para el espacio multidimensional. El método de la secante sustituye la primera derivada $f'(x_n)$ por la aproximación de diferencia finita

$$f'(x_n) \simeq \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

y procede según el método de Newton:

$$x_{n+1} = x_n - \frac{1}{f'(x_n)} f(x_n)$$

Broyden establece una generalización de esa fórmula para un sistema de ecuaciones mediante una sustitución de la derivada f' por el jacobiano J.

Éste se determina por medio de la ecuación de la secante (la aproximación de diferencia finita):

$$J_n \cdot (x_n - x_{n-1}) \simeq F(x_n) - F(x_{n-1})$$

Sin embargo, esta ecuación está determinada por más de una dimensión.

Broyden sugiere un procedimiento que consta de los siguientes 3 pasos:

1.- Emplear la aproximación del jacobiano J_{n-1}

2.- Tomar la solución de la ecuación de la secante que suponga la modificación mínima de J_{n-1}(entendiendo por mínima que se dé una minimización de la norma de Frobenius $\|J_n - J_{n-1}\|_F$)

$$J_n = J_{n-1} + \frac{\Delta F_n - J_{n-1}\Delta x_n}{\|\Delta x_n\|^2}\Delta x_n^T$$

3.- Continuar según el método de Newton:

$$x_{n+1} = x_n - J_n^{-1} F(x_n).$$

En esa última fórmula,

$$x_n = (x_1[n], ..., x_k[n])$$

y

$$F_n(x) = (f_1(x_1[n], ..., x_k[n]), ..., f_k(x_1[n], ..., x_k[n])$$

son vectores columna de k elementos en un sistema de k dimensiones.

Así:

$$\Delta x_n = \begin{bmatrix} x_1[n] - x_1[n-1] \\ ... \\ x_k[n] - x_k[n-1] \end{bmatrix} \quad \text{y} \quad \Delta F_n = \begin{bmatrix} f_1(x_1[n], ..., x_k[n]) - f_1(x_1[n-1], ..., x_k[n-1]) \\ ... \\ f_k(x_1[n], ..., x_k[n]) - f_k(x_1[n-1], ..., x_k[n-1]) \end{bmatrix}.$$

Para encontrar una solución aproximada de un sistema de ecuaciones no lineales f(x)=0, proporcionar la matriz jacobiana ampliada con el vector de funciones y los

DATOS : El numero de ecuaciones N, dos vectores de valores iniciales: x0 y x1, el número máximo de iteraciones MAXIT y el criterio de convergencia EPS.

RESULTADOS: Una aproximación a una solución: xn o el mensaje "NO CONVERGE"

PASO 1: Calcular AK, la matriz inversa de la matriz Jacobiana evaluada en x0.

PASO 2: Hacer K=1

PASO 3: Mientras K <= MAXIT, repetir los pasos 4 a 10.

PASO 4: Calcular f0 y f1, el vector de funciones evaluado en x0 y x1, respectivamente.

PASO 5: Calcular aplicando operaciones matriciales dx = x1 - x0; df = f1 - f0.

PASO 6: Calcular AK1, la matriz que aproxima a la inversa de la matriz Jacobiana con la ecuación, usando como $(A^{(k+1)})^{-1}$ a AK.

PASO 7: Calcular aplicando operaciones matriciales xn=x1 – AK1*f1.

PASO 8: Si $|xn - x1|$ <=EPS ir al paso 11. De otro modo continuar.

PASO 9: Hacer x0=x1;x1=xn;AK=AK1 (Actualización de x0,x1 y AK).

PASO 10: Hacer K=K+1

PASO 11: Si K<=MAXIT, IMPRIMIR el vector xn Y TERMINAR.

De otro modo IMPRIMIR "NO CONVERGE" Y TERMINAR

4.1.4.- APROXIMACION FUNCIONAL E INTERPOLACIÓN

En este tipo de aproximación se trata de encontrar la ecuación de una curva que, aunque no pase por todos los puntos, tenga pocas variaciones, es decir sea suave y pase lo más cerca posible de todos ellos, para ello es necesario aplicar el criterio de mínimos cuadrados. Antes de aplicar este criterio, debe escogerse la forma de la curva que se va a ajustar al conjunto de puntos dado y su ecuación puede obtenerse desde un conocimiento previo del

problema, es decir por su interpretación física o en forma arbitraria observando que ecuación conocida describe aproximadamente a esta curva.

A.-ALGORITMO DEL METODO DE APROXIMACIÓN POLINOMIAL SIMPLE

Si se desea aproximar una función con un polinomio de grado n, se necesitan n+1 puntos, que sustituidos en la ecuación polinomial de grado n:

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_n x^n$$

Generan un sistema de n+1 ecuaciones lineales en las incógnitas $a_{i,j}$=0,1,2,....,n.

Una vez resuelto el sistema se sustituyen los valores de a_i en la ecuación anterior con lo cual se obtiene el polinomio de aproximación. A este método se le conoce como aproximación polinomial simple.

Para obtener los (n+1) coeficientes del polinomio de grado n (n>0) que pasa por (n+1) puntos, proporcionar los

DATOS : El grado del polinomio N y las N+1 parejas de valores (X(I),F(I), I=0,1,....,N).

RESULTADOS: Los coeficientes A(0), A(1),....., A(N) del polinomio de aproximación.

PASO 1: Hacer I=0

PASO 2: Mientras K <= MAXIT, repetir los pasos 3 a 9.

PASO 3: Hacer B(I,0)=1

PASO 4: Hacer J=1

PASO 5: Mientras J<=N, repetir los pasos 6 y 7.

PASO 6: Hacer B(I,J) = B(I,J-1)*X(I)

PASO 7: Hacer J=J+1

PASO 8: Hacer B(I,N+1) = FX(I)

PASO 9: Hacer I=I+1

PASO 10: Resolver el sistema de ecuaciones lineales B a =fx de orden N+1 con alguno de los algoritmos anteriores.

PASO 11: IMPRIMIR A(0), A(1),....., A(N) y TERMINAR.

B.-ALGORITMO DEL METODO DE APROXIMACIÓN POLINOMIAL DE LAGRANGE

Este método es similar al de la aproximación polinomial simple pero no se requiere resolver un sistema de ecuaciones lineales y los cálculos se realizan directamente.

El polinomio está representado por la siguiente ecuación:

$$P_n(x) = \sum_{i=0}^{n} Li(x) f(x_i)$$

Donde:

$$Li(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$

Al combinarse linealmente con f(xi), los polinomios Li(x), denominados polinomios de Lagrange, generan la aproximación polinomial de Lagrange a la información dada en forma tabular.

Ejemplo 4.1.2. Úsese un polinomio de interpolación de Lagrange de primer y segundo orden para evaluar **ln 2** en base a los datos:

i	X	f(X)
0	1.0	0.000 0000
1	4.0	1.386 2944
2	6.0	1.791 7595

Solución:

El polinomio de primer orden es:

$$f_1(X) = \frac{X - X_1}{X_0 - X_1} f(X_0) + \frac{X - X_0}{X_1 - X_0} f(X_1)$$

y, por lo tanto, la aproximación en X = 2 es

$$f_1(2) = \frac{2 - 4}{1 - 4} (0) + \frac{2 - 1}{4 - 1} (1.3862944) = 0.462\,0981$$

de manera similar, el polinomio de segundo orden se desarrolla como:

$$f_2(2) = \frac{(2-4)(2-6)}{(1-4)(1-6)}(0) + \frac{(2-1)(2-6)}{(4-1)(4-6)}(1.3862944) + \frac{(2-1)(2-4)}{(6-1)(6-4)}(1.7917595) = 0.5658$$

Para aproximar con polinomios de lagrange de grado N proporcionar los:

DATOS : El grado del polinomio N y las N+1 parejas de valores (X(I),F(I), I=0,1,....,N) y el valor para el que se desea la interpolación XINT.

RESULTADOS: La aproximación FXINT, el valor de la función en XINT.

PASO 1: Hacer IFXINT=0

PASO 2: Hacer I=0

PASO 3: Mientras K <= N, repetir los pasos 4 a 10.

PASO 4: Hacer L=1

PASO 5: Hacer J=0

PASO 6: Mientras J<=N, repetir los pasos 7 y 8.

PASO 7: Si I <> J Hacer L=L*(XINT-X(J))/(X(I)-X(J))

PASO 8: Hacer J=J+1

PASO 9: Hacer FXINT= FXINT + L*FX(I)

PASO 10: Hacer I=I+1

PASO 11: IMPRIMIR FXINT y TERMINAR.

C.-ALGORITMO DEL METODO DE INTERPOLACIÓN POLINOMIAL DE NEWTON

Al asumir que los valores de una función(x) son aproximadamente lineales, dentro de un rango de valores, es equivalente a decir que la razón:

$$\frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

es aproximadamente independiente de x0 y x1 en el rango. Esta razón se conoce con el nombre de primera diferencia dividida de f(x), relativa a x1 y x0, y se designa por medio de f[x1 ,x0]. Se puede inferir de la ecuación fue f[x1 ,x0] = f[x0 ,x1].

Por tanto, la linealidad aproximada se puede expresar en la forma

$$f[x0, x] \approx f[x1, x0]$$

lo que nos lleva a la ecuación e interpolación

$$f(x) \approx f(x0) + (x - x0).f[x0, x1]$$

o

$$f(x) \approx f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0} \cdot \left[f(x_1) - f(x_0) \right]$$

o la función equivalente,

$$f(x) \approx \frac{1}{x_1 - x_0} \cdot \left[(x_1 - x) \cdot f(x_0) - (x_0 - x) \cdot f(x_1) \right]$$

Las diferencias divididas de orden 0,1,2,...,n se pueden deducir recursivamente por medio de las relaciones siguientes:

$$f[x_0] = f(x_0)$$

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

$$f[x_0, x_1, x_2] = \frac{f[x_1, x_2] - f[x_0, x_1]}{x_2 - x_0} = \frac{1}{x_1 - x_0} \cdot \left[\frac{f(x_2) - f(x_1)}{x_2 - x_1} - \frac{f(x_1) - f(x_0)}{x_1 - x_0} \right]$$

$$\vdots$$

$$f[x_0, \ldots, x_2] = \frac{f[x_1, \ldots, x_k] - f[x_0, \ldots, x_{k-1}]}{x_k - x_0}$$

Ejemplo 4.1.3. Usando la siguiente tabla de datos, calcúlese ***ln 2*** con un polinomio de interpolación de Newton con diferencias divididas de tercer orden:

X	f(X)
1	0.000 0000
4	1.386 2944
6	1.791 7595
5	1.609 4379

Solución:

El polinomio de tercer orden con ***n* = 3**, es.

$$f_3(X) = b_0 + b_1(X - X_0) + b_2(X - X_0)(X - X_1) + b_3(X - X_0)(X - X_1)(X - X_3)$$

Las primeras diferencias divididas del problema son:

$$f[X_1, X_0] = \frac{1.386\,2944 - 0}{4 - 1} = 0.462\,0981$$

$$f[X_2, X_1] = \frac{1.791\,7595 - 1.386\,2944}{6 - 4} = 0.202\,7326$$

$$f[X_3, X_2] = \frac{1.609\,4379 - 1.791\,7595}{5 - 6} = 0.182\,3216$$

Las segundas diferencias divididas son:

$$f[X_2, X_1, X_0] = \frac{f[X_2, X_1] - f[X_1, X_0]}{X_2 - X_0} = \frac{0.202\,7326 - 0.462\,0981}{6 - 1} = -0.051\,8731$$

$$f[X_3, X_2, X_1] = \frac{f[X_3, X_2] - f[X_2, X_1]}{X_3 - X_1} = -0.020\,4110$$

La tercera diferencia dividida es:

$$f[X_3, X_2, X_1, X_0] = \frac{f[X_3, X_2, X_1] - f[X_2, X_1, X_0]}{X_3 - X_0} = 0.007\,8655$$

Los resultados para **f(X1, X0), f(X2, X1, X0) y f(X3, X2, X1, X0)** *representan los coeficientes* **b1, b2** *y* **b3** *Junto con* **b0** *= f (X0) = 0.0, la ecuación da:*

f3(X)=0+0.46209813 (X-1)-0.0518731(X-1)(X-4)+0.0078655415(X-1)(X-4)(X-6)

Arreglando la tabla de diferencias

X	f [X]	f 1 [ ]	f 2 [ ]	f 3 [ ]
1.0	0.00000000	0.46209813	- 0.051873116	0.0078655415
4.0	1.3862944	0.20273255	- 0.020410950	
6.0	1.7917595	0.18232160		
5.0	1.6094379			

Con la ecuación anterior se puede evaluar para ***X = 2***

$$f\,3\,(2) = 0.62876869$$

lo que representa un error relativo porcentual del e% = 9.3%.

Para interpolar con polinomios de Newton en diferencias divididas de grado N, proporcionar los

DATOS : El grado del polinomio N, las N+1 parejas de valores (X(I),FX(I), I=0,1,2,...,N) y el valor para el que se desea interpolar XINT.

RESULTADOS: La aproximación FXINT al valor de la función en XINT.

PASO 1: Hacer I=0
PASO 2: Mientras I <= N-1, repetir los pasos 4 y 5.
PASO 3: Hacer T(I,0)=(FX(I+1)-FX(I))/(X(I+1)-X(I))
PASO 4: Hacer I=I+1
PASO 5: Hacer J=1
PASO 6: Mientras J<=N-1,repetir los pasos 7 a 11.
PASO 7: Hacer I=J
PASO 8: Mientras I<=N-1, repetir los pasos 10 y 11.
PASO 9: Hacer T(I,J)=(T(I,J-1)-T(I-1,J-1))/(X(I+1)-X(I-J))
PASO 10: Hacer I=I+1
PASO 11: Hacer J=J+1
PASO 12: Hacer FXINT=FX(0)
PASO 13: Hacer I=0
PASO 14: Mientras I<=N-1, repetir los pasos 15 a 21
PASO 15: Hacer P=1
PASO 16: Hacer J=0
PASO 17: Mientras J<=1, repetir los pasos 18 y 19
PASO 18: Hacer P=P*(XINT-X(J))
PASO 19: Hacer J=J+1
PASO 20: Hacer FXINT=FXINT+T(I,I)*P
PASO 21: Hacer I=I+1
PASO 22: IMPRIMIR FXINT y TERMINAR

D.- ALGORITMO DEL METODO DE INTERPOLACIÓN POLINOMIAL CON MINIMOS CUADRADOS

Supongamos que existe una relación funcional $y = f(x)$ entre dos cantidades x y y, con f desconocida y se conocen valores y_k que aproximan a $f(x_k)$, es decir, $f(x_k) = y_k + \in_k$ donde $k = 0,1, 2, ...n$ con $\in_k$ desconocido.

Se trata de recuperar la función f a partir de los datos aproximados y_k, para k = 0, 1,.., n. Este problema se conoce como un problema de "ajuste de datos" o "ajuste de curvas" (caso discreto). Trabajaremos básicamente el caso en el que f es una función polinómica.

Si f es una función polinómica, digamos $f(x) = p_m(x)$, entonces el problema se convierte en: Dados n+1 puntos $(x_0, y_0),(x_1, y_1),.....(x_n, y_n)$ con x_0, x_1, ..., x_n números reales distintos, se trata de encontrar un polinomio:
$P_m(x) = a_0 + a_1x + a_2x^2 + ... + a_mx^m$, con m<n que "mejor se ajuste" a los datos. Lo de "mejor ajuste" se entenderá en el sentido de que:

$$\left[\sum_{k=0}^{n}\left(p_m(x_k)-y_k\right)^2\right]^{1/2}$$

Sea mínimo, es decir, que

$$\sum_{k=0}^{n}\left(p_m(x_k)-y_k\right)^2$$

Este criterio de mejor ajuste, como ya se mencionó antes, se conoce como mínimos cuadrados, y el método para obtener los polinomios que mejor se ajustan según mínimos cuadrados se llama Regresión polinomial.

Ejemplo 4.1.4. Calcular el polinomio de grado 1.

Solución:

1.- En el caso particular en que m = 1, $p_1(x) = a_0 + a_1x$ es la **recta de mínimos cuadrados** donde a_0 y a_1 se obtienen resolviendo el sistema lineal de dos ecuaciones con dos incógnitas:

$$SEL_{2x2}\begin{cases}\underbrace{\left(\sum_{k=0}^{n}x_k^0\right)}_{(n+1)}a_0 + \left(\sum_{k=0}^{n}x_k^1\right)a_1 = \sum_{k=0}^{n}y_k \\ \left(\sum_{k=0}^{n}x_k^1\right)a_0 + \left(\sum_{k=0}^{n}x_k^2\right)a_1 = \sum_{k=0}^{n}x_k y_k\end{cases}$$

2.- Que equivale a la vista en mínimos cuadrados, si se intercambia $R_1 \leftrightarrow R_2$, seguida de un intercambio de $C_1 \leftrightarrow C_2$ queda exactamente la misma ecuación:

$$\begin{pmatrix}a\\b\end{pmatrix}=\begin{pmatrix}\sum_{i=1}^{n}x_i^2 & \sum_{i=1}^{n}x_i\\ \sum_{i=1}^{n}x_i & n\end{pmatrix}^{-1}\begin{pmatrix}\sum_{i=1}^{n}x_i y_i\\ \sum_{i=1}^{n}y_i\end{pmatrix}$$

Si no se realizan los intercambios queda:

$$\begin{pmatrix}a\\b\end{pmatrix}=\begin{pmatrix}n & \sum_{i=1}^{n}x_i\\ \sum_{i=1}^{n}x_i & \sum_{i=1}^{n}x_i^2\end{pmatrix}^{-1}\begin{pmatrix}\sum_{i=1}^{n}y_i\\ \sum_{i=1}^{n}x_i y_i\end{pmatrix}$$

3.- Con m = 1:

$$p_m(x_k)=a_0+a_1x_k+a_2x_k^2+,.....a_mx_k^m=y_k \Rightarrow$$

$$p_1(x_k)=a_0+a_1x_k^1=y_k \Rightarrow p_1(x_k)=a_0+a_1x_k=y_k$$

Para obtener los N+1 coeficientes del polinomio optimo de grado N que pasa entre M parejas de puntos, proporcionar los

DATOS : El grado del polinomio de aproximación N, el número de parejas de valores (X(I),FX(I),I=1,2,...,M).

RESULTADOS: Los coeficientes A(0),A(1),...,A(N) del polinomio de aproximación.

PASO 1: Hacer J=0

PASO 2: Mientras J<=(2*N-1),repetir los pasos 3 a 5.

PASO 3: Si J<=N Hacer SS(J)=0. De otro modo continuar.

PASO 4: Hacer S(J)=0

PASO 5: Hacer J=J+1

PASO 6: Hacer I=1

PASO 7: Mientras I<=N, repetir los pasos 8 a 15.

PASO 8: Hacer XX=1

PASO 9: Hacer J=0

PASO 10: Mientras J<=(2*N-1),repetir los pasos 11 a 14.

PASO 11: Si J<=N hacer SS(J)=SS(J)+XX*FX(I).De otro modo continuar.

PASO 12: Hacer XX=XX*X(I)

PASO 13: Hacer S(J)=S(J)+XX

PASO 14: Hacer J=J+1

PASO 15: Hacer I=I+1

PASO 16: Hacer B(0,0)=M

PASO 17: Hacer I=0

PASO 18: Mientras I<=N,repetir los pasos 19 a 24.

PASO 19: Hacer J=0

PASO 20: Mientras J<=N,repetir los pasos 21 y 22.

PASO 21: Si I<>0 y J<>0. Hacer B(I,J)=S(J-1+I).

PASO 22: Hacer J=J+1

PASO 23: Hacer B(I,N+1)=SS(I)

PASO 24: Hacer I=I+1

PASO 25: Resolver el sistema de ecuaciones lineales Ba=ss de orden N+1.

PASO 26: IMPRIMIR A(0),A(1),....A(N) y TERMINAR.

4.1.5.- INTEGRALES

Dada una función f definida sobre un intervalo [a,b], estamos interesados en calcular

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

Suponiendo que esta integral tenga sentido para la función f. La cuadratura o integración numérica consiste en obtener fórmulas aproximadas para calcular la integral J(f) de f. Estos métodos son de gran utilidad cuando la integral no se puede calcular por métodos analíticos, su cálculo resulta muy costoso y estamos interesados en una solución con precisión finita dada o bien

sólo disponemos de una tabla de valores de la función (es decir, no conocemos la forma analítica de f).

A.-ALGORITMO DEL METODO TRAPEZOIDAL COMPUESTO

Un enfoque simple para evaluar la integral de una función es considerarla como el área bajo la línea de la recta que conecta los valores de la función en los extremos del intervalo de integración, tal como se aprecia en la figura (4.1.4), la fórmula para determinar esta área es:

$$I \cong (b-a)\frac{f(a)+f(b)}{2}$$

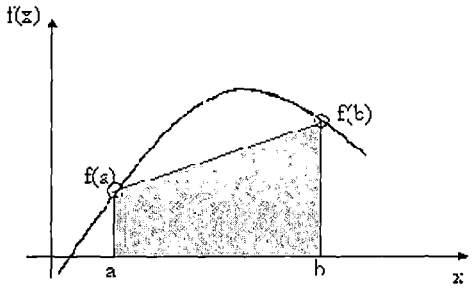


Figura 4.1.4. Representación geométrica de la integral.

A causa de que el área bajo la línea es un trapecio, a esta expresión se le conoce como *regla trapezoidal*. Recuérdese de geometría que la fórmula para calcular el área de un trapecio es multiplicar la altura por el promedio de las bases (vease figura 4.1.5). En esta ocasión el concepto es el mismo pero el trapecio se encuentra sobre uno de sus lados (vease figura 4.1.6); por esta razón la razón de la estimación de la integral se presenta como:

$$I \cong \text{anchura x altura promedio}$$

Donde para la regla trapezoidal la altura media es el promedio de los valores de la función en los extremos, o (f(a) + f(b))/2.

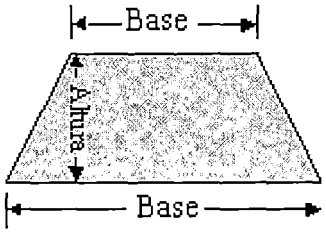


Figura 4.1.5. Representación geométrica del trapecio.

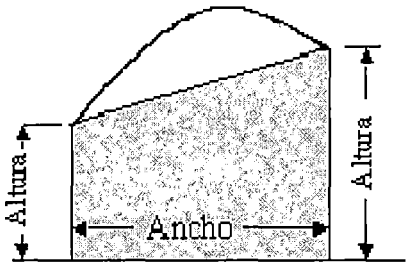


Figura 4.1.6. Representación geométrica del trapecio sobre uno de los lados.

La regla del trapecio compuesto o regla de los trapecios es una forma de aproximar una integral definida utilizando n trapecios. En la formulación de este método se supone que f es continua y positiva en el intervalo [a,b]. De tal modo la integral definida

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

representa el área de la región delimitada por la gráfica de *f* y el eje *x*, desde *x=a* hasta *x=b*. Primero se divide el intervalo [*a*,*b*] en *n* subintervalos, cada uno de ancho

$$\Delta x = (b-a)/n$$

Después de realizar todo el proceso matemático se llega a la siguiente fórmula:

$$\int_a^b f(x)\,dx \sim \frac{h}{2}[f(a)+2f(a+h)+2f(a+2h)+\ldots+f(b)]$$

Dónde:

$$h = \frac{b-a}{n}$$

y n es el número de divisiones. La expresión anterior también se puede escribir como:

$$\int_a^b f(x)dx \sim \frac{b-a}{n}\left(\frac{f(a)+f(b)}{2}+\sum_{k=1}^{n-1} f\left(a+k\frac{b-a}{n}\right)\right)$$

Ejemplo 4.1.5. La velocidad de caída de un paracaidista está definida por la siguiente función del tiempo:

$$v(t) = \frac{gm}{c}\left(1 - e^{-(c/m)t}\right)$$

donde *v(t)* es la velocidad en metros por segundo, *g* es la constante gravitacional de 9.8 m/s^2, *m* es la masa del paracaidista igual a 68.1 Kg, y *c* es el coeficiente de arrastre de 12.5 kg/s. Puede desarrollarse una gráfica dela función sustituyendo los parámetros y varios valores de *t* en la igualdad para obtener:

t,s	v, m/s
0	0.00
2	16.40;
4	27.77
6	35.64
8	41.10
10	44.87
12	47.49
14	49.30
16	50.56

Utilícese el método trapezoidal para determinar cuánto ha caído el paracaidista después de 12 s.

Solución:

Podemos calcular la velocidad integrando como sigue:

$$y(t) = \int_0^t v(t)\,dt$$

o, sustituyendo la función y los valores establecidos,

$$y(t) = \int_0^{12} 53.39(1 - e^{-0.18355t})\,dt$$

Empleando el cálculo puede evaluarse analíticamente esta integral para obtener el resultado exacto de 381.958 m, el cual permite juzgar la precisión de la aproximación por la regla trapezoidal.

A fin de aplicar la regla trapezoidal, los valores de los puntos extremos pueden sustituirse en la ecuación (1) con lo cual se obtiene:

$$I \cong (12 - 0)\frac{0 + 47.49}{2} = 284.94\,\text{m}$$

que representa un porcentaje relativo de error de

$$\text{Error} = \frac{381.958 - 248.94}{381.958} 100\% = 25.4\%$$

Para aproximar el área bajo la curva de una función analítica f(x) en el intervalo [a,b], proporcionar la función por integrar F(x) y los

DATOS : El numero de trapecios N, el límite inferior A y limite superior B.

RESULTADOS: El área aproximada AREA.

PASO 1: Hacer X=A

PASO 2: Hacer S=0

PASO 3: Hacer H=(B-A)/N

PASO 4: Si N=1, ir al paso 10. De otro modo continuar.

PASO 5: Hacer I=1

PASO 6: Mientras I<=N-1, repetir los pasos 7 a 9.

PASO 7: Hacer X=X+H

PASO 8: Hacer S=S+F(X)

PASO 9: Hacer I=I+1

PASO 10: Hacer AREA=(H/2)*(F(A)+2*S+F(B))

PASO 11: IMPRIMIR AREA y TERMINAR

B.-ALGORITMO DEL METODO DE SIMPSON

Además de aplicar la regla trapezoidal con segmentos cada vez más finos, otra manera de obtener una estimación más exacta de una integral, es la de usar polinomios de orden superior para conectar los puntos. Por ejemplo, si hay un punto medio extra entre *f(a)* y *f(b)*, entonces los tres puntos se pueden conectar con un polinomio de tercer orden.

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

REGLA DE SIMPSON 1/3

La Regla de Simpson de 1/3 proporciona una aproximación más precisa, ya que consiste en conectar grupos sucesivos de tres puntos sobre la curva mediante parábolas de segundo grado, y sumar las áreas bajo las parábolas para obtener el área aproximada bajo la curva. Por ejemplo, el área contenida en dos franjas, bajo la curva *f(X)* en la figura 4.1.7, se aproxima mediante el área sombreada bajo una parábola que pasa por los tres puntos:

(Xi , Yi)
(Xi+1, Yi+1)
(Xi+2, Yi+2)

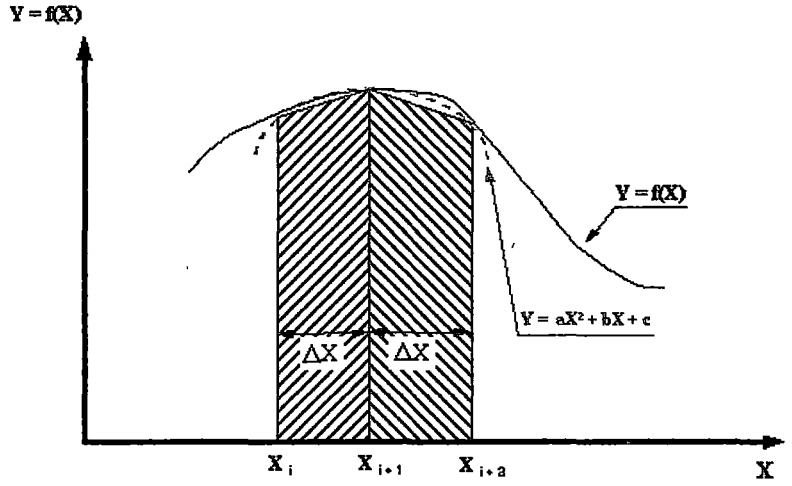


Figura 4.1.7. Representación geométrica de la Regla de Simpson de 1/3

Por conveniencia al derivar una expresión para esta área, supongamos que las dos franjas que comprenden el área bajo la parábola se encuentran en lados opuestos del origen, como se muestra en la Figura 4.1.8. Este arreglo no afecta la generalidad de la derivación.

La forma general de la ecuación de la parábola de segundo grado que conecta los tres puntos es:

$$Y = aX^2 + bX + c \quad (1)$$

La integración de esta ecuación desde - ΔX hasta ΔX proporciona el área contenida en las dos franjas mostradas bajo la parábola. Por lo tanto:

$$\int_{-\Delta X}^{\Delta X} (aX^2 + b\,X + c)\,dx = \left[\frac{aX^3}{3} + \frac{bX^2}{2} + cX\right]_{-\Delta X}^{\Delta X} \qquad (2)$$

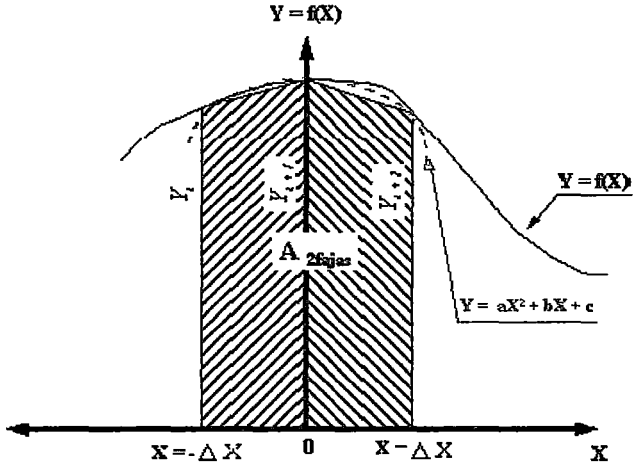


Figura 4.1.8. Representación geométrica de la Regla de Simpson de 1/3 en lados opuestos del origen.

La sustitución de los límites en la ecuación (1) produce:

$$A_{2\text{fajas}} = \frac{2}{3}\,a\,(\Delta X)^3 + 2\,c\,(\Delta X) \qquad (3)$$

Las constantes *a* y *c* se pueden determinar sabiendo que los puntos

$$(-\Delta X, Y_i), (0, Y_{i+1})\ y\ (\Delta X, Y_{i+2})$$

deben satisfacer la ecuación (1). La sustitución de estos tres pares de coordenadas produce:

$$\begin{aligned} Y_i &= a\,(-\Delta X)^2 + b\,(-\Delta X) + c \\ Y_{i+1} &= c \\ Y_{i+2} &= a\,(\Delta X)^2 + b\,(\Delta X) + c \end{aligned} \qquad (4)$$

La solución simultánea de estas ecuaciones para determinar las constantes *a, b, c*, nos lleva a:

$$a = \frac{Y_i - 2Y_{i+1} + Y_{i+2}}{2(\Delta X)^2}$$

$$b = \frac{Y_{i+2} - Y_i}{2\Delta X} \qquad (5)$$

$$c = Y_{i+1}$$

La sustitución de la primera y tercera partes de la ecuación (5) en la ecuación. (3) produce:

$$A_{2fajas} = \frac{\Delta X}{3}(Y_i + 4Y_{i+1} + Y_{i+2}) \quad (6)$$

que nos da el área en función de tres ordenadas Y_i, Y_{i+1}, Y_{i+2} y el ancho ΔX de una franja.

Esto constituye la regla de Simpson para determinar el área aproximada bajo una curva contenida en dos franjas de igual ancho.

Si el área bajo una curva entre dos valores de *X* se divide en *n* franjas uniformes (*n* par), la aplicación de la ecuación (6) muestra que:

$$A_1 = \frac{\Delta X}{3}(Y_1 + 4Y_2 + Y_3)$$

$$A_2 = \frac{\Delta X}{3}(Y_3 + 4Y_4 + Y_5) \qquad (7)$$

$$\cdots$$

$$A_{n/2} = \frac{\Delta X}{3}(Y_{n-1} + 4Y_n + Y_{n+1})$$

Sumando estas áreas, podemos escribir:

$$\int_{X_1=0}^{X_{n+1}} f(X)\,dx = \sum_{i=1}^{n/2} A_i = \frac{\Delta X}{3}(Y_1 + 4Y_2 + 2Y_3 + 4Y_4 + 2Y_5 + \ldots + 2Y_{n-1} + 4Y_n + Y_{n+1}) \quad (8)$$

o bien

$$\int_{X_1=0}^{X_{n+1}} f(X)\,dx = \sum_{i=1}^{n/2} A_i = \frac{\Delta X}{3}\left(Y_1 + 4\sum_{i=1}^{n/2} Y_{2i} + 2\sum_{i=1}^{(n-2)/2} Y_{2i-1} + Y_{n+1}\right) \quad (9)$$

en donde *n* es par.

La ecuación (9) se llama *Regla de Simpson de un Tercio* para determinar el área aproximada bajo una curva. Se puede utilizar cuando el área se divide en un número par de franjas de ancho ΔX .

Para aproximar el área bajo la curva de una función analítica f(x) en el intervalo [a,b], proporcionar la función por integrar F(x) y los

DATOS : El numero par de subintervalos N, el límite inferior A y limite superior B.

RESULTADOS: El área aproximada AREA.

PASO 1: Hacer S1=0

PASO 2: Hacer S2=0

PASO 3: Hacer X=A

PASO 4: Hacer H=(B-A)/N

PASO 5: Si N=2, ir al paso 13. De otro modo continuar.

PASO 6: Hacer I=1

PASO 7: Mientras I<=N/2-1, repetir los pasos 8 a 12.

PASO 8: Hacer X=X+H

PASO 9: Hacer S1=S1+F(X)

PASO 10: Hacer X=X+H

PASO 11: Hacer S2=S2+F(X)

PASO 12: Hacer I=I+1

PASO 13: Hacer X=X+H

PASO 14: Hacer S1=S1+F(X)

PASO 15: Hacer AREA=(H/3)*(F(A)+4*S1+2*S2+F(B))

PASO 16: IMPRIMIR AREA y TERMINAR

4.1.6.- DERIVADAS

Sea f(x) la función cuya derivada queremos aproximar. Asumimos que f es conocida en los puntos x0, x1, x2, . . . xn. Usando el polinomio de Lagrange, se puede f expresar como:

$$f(x) = \sum_{i=0}^{n} f(x_i).L_i(x) + R(x)$$

donde las funciones L_i estan definidos por

$$L_i(x) = \prod_{j=0, j\neq i}^{n} \frac{(x - x_j)}{(x_i - x_j)}$$

y el residuo R(x) se obtiene por

$$R(x) = \frac{1}{(n+1)!} \cdot \prod_{j=1}^{n} (x - x_j).f^{(n+1)}(\varepsilon_x)$$

Si se deriva la función, expresado como polinomio de Lagrange se obtiene:

$$f'(x) = \sum_{i=0}^{n} f(x_i).L'_i(x) + R'(x)$$

que se puede expresar como

$$f'(x) - \sum_{i=0}^{n} f(x_i).L'_i(x) + \frac{1}{(n+1)!} \cdot \left(\prod_{j=1}^{n} (x - x_j) \right)' f^{(n+1)}(\varepsilon_x)$$
$$+ \frac{1}{(n+1)!} \cdot \prod_{j=1}^{n} (x - x_j).(f^{(n+1)}(\varepsilon_x))'$$

Para obtener una aproximación a la primera derivada de una función tabular f(x) en un punto x, proporcionar los

DATOS: El grado N del polinomio de Lagrange por usar, las (N+1) parejas de Valores (X (I), FX (I), I=0, 1,2,..., N) y el punto XD en que se desea la Evaluación.

RESULTADOS: Aproximación a la primera derivada en XD: DP.

PASO 1: Hacer DP = 0

PASO 2: Hacer I = 0

PASO 3: Mientras I ≤ N, repetir los pasos 4 a 21

PASO 4: Hacer P = 1

PASO 5: Hacer J = 0

PASO 6: Mientras J ≤ N, repetir los pasos 7 a 8

PASO 7: Si I < > J

Hacer P = P* (X (I) – X (J))

PASO 8: Hacer J = J + 1

PASO 9: Hacer S = 0

PASO 10: Hacer K = 0

PASO 11: Mientras K ≤ N, repetir los pasos 12 a 19.

PASO 12: Si I < > K, realizar los pasos 13 a 18

PASO 13: Hacer P1 = 1

PASO 14: Hacer J = 0

PASO 15: Mientras J ≤ N, repetir los pasos 16 a 17

PASO 16: Si J < > I y J < > K

Hacer P1 = P1 * (XD – X (J))

PASO 17: Hacer J = J + 1

PASO 18: Hacer S = S + P1

PASO 19: Hacer K = K + 1

PASO 20: Hacer DP = DP + FX (I)/P * S

PASO 21: Hacer I = I + 1

PASO 22: IMPRIMIR DP Y TERMINAR

4.1.7.- ECUACIONES DIFERENCIALES ORDINARIAS

Si y es una función desconocida:

$$y : \mathbb{R} \rightarrow \mathbb{R}$$

de x siendo $y^{(n)}$ la enésima derivada de y, entonces una ecuación de la forma

$$F(x, y, y', \ldots, y^{(n-1)}) = y^{(n)} \quad \text{.....(1)}$$

es llamada una ecuación diferencial ordinaria (EDO) de orden n. Para funciones vectoriales

$$y : \mathbb{R} \to \mathbb{R}^m,$$

la ecuación (1) es llamada un sistema de ecuaciones lineales diferenciales de dimensión m.

Cuando una ecuación diferencial de orden n tiene la forma

$$F\left(x, y, y', y'', \ \ldots, \ y^{(n)}\right) = 0$$

es llamada una ecuación diferencial implícita, mientras que en la forma

$$F\left(x, y, y', y'', \ \ldots, \ y^{(n-1)}\right) = y^{(n)}$$

es llamada una ecuación diferencial explícita.

Una ecuación diferencial que no depende de x es denominada autónoma.

Se dice que una ecuación diferencial es lineal si F puede ser escrita como una combinación lineal de las derivadas de y

$$y^{(n)} = \sum_{i=0}^{n-1} a_i(x) y^{(i)} + r(x)$$

Siendo, tanto $a_i(x)$ como r(x) funciones continuas de x. La función r(x) es llamada el término fuente; si r(x)=0 la ecuación diferencial lineal es llamada homogénea, de lo contrario es llamada no homogénea.

A.-ALGORITMO DEL METODO DE EULER

Consiste en dividir los intervalos que va de x_o a x_f en n subintervalos de ancho h; o sea:

$$h = \frac{x_f - x_o}{n}$$

de manera que se obtiene un conjunto discreto de $n+1$ puntos: $x_o, x_1, x_2, \ldots\ldots, x_n$ del intervalo de interés $[x_o, x_f]$. Para cualquiera de estos puntos se cumple que:

$$x_i = x_0 + ih, 0 \leq i \leq n.$$

La condición inicial $y(x_o) = y_o$, representa el punto $P_o = (x_o, y_o)$ por donde pasa la curva solución de la ecuación del planteamiento inicial, la cual se denotará como $F(x) = y$.

Ya teniendo el punto P_o se puede evaluar la primera derivada de $F(x)$ en ese punto; por lo tanto:

$$F'(x) = \left.\frac{dy}{dx}\right|_{P_o} = f(x_o, y_o)$$

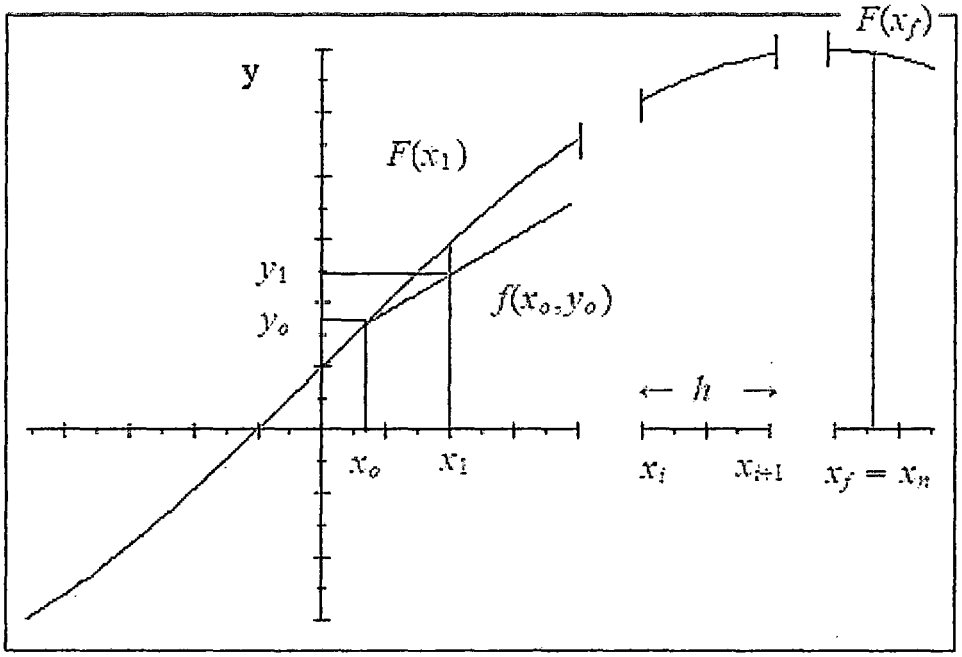


Figura 4.1.9. Representación gráfica del método de Euler

Con esta información se traza una recta, aquella que pasa por P_o y de pendiente $f(x_o, y_o)$. Esta recta aproxima $F(x)$ en una vecindad de x_o. Tómese la recta como reemplazo de $F(x)$ y localícese en ella (la recta) el valor de y correspondiente a x_1. Entonces, podemos deducir de la Gráfica anterior:

$$\frac{y_1 - y_o}{x_1 - x_o} = f(x_o, y_o)$$

Se resuelve para y_1:

$$y_1 = y_o + (x_1 - x_o) f(x_o, y_o) = y_o + h f(x_o, y_o)$$

Es evidente que la ordenada y_1 calculada de esta manera no es igual a $F(x_1)$, pues existe un pequeño error. Sin embargo, el valor y_1 sirve para que se aproxime $F'(x)$ en el punto $P = (x_1, y_1)$ y repetir el procedimiento anterior a fin de generar la sucesión de aproximaciones siguiente:

$$\begin{aligned} y_1 &= y_o + h f(x_o, y_o) \\ y_2 &= y_1 + h f(x_1, y_1) \\ &\vdots \\ y_{i+1} &= y_i + h f(x_i, y_i) \\ &\vdots \\ y_n &= y_{n-1} + h f(x_{n-1}, y_{n-1}) \end{aligned}$$

Ejemplo 4.1.6. Resolver la ecuación diferencial siguiente por el método de Euler **y'-2 x y = x** con condiciones iniciales y(0.5) = 1, en el intervalo $0.5 \leq x \leq 1$ con h = 0.1.

Solución:

Tenemos que:

$$y'(x) = f(x, y)$$

por lo tanto

$$f(x, y) = x + 2xy$$

Ahora usando la ecuación (1) para i=0,

$$y_1 = y_0 + h f(x_0, y_0)$$

en donde:

$y_0 = 1$ *(condición inicial)*

$h_0 = 0.1$

Ahora bien:

$$f(x_0, y_0) = x_0 + 2x_0y_0 = 0.5 + 2(0.5)(1) = 1.5$$

Al sustituir estos valores:

$$y_1 = 1 + 0.1(1.5) = 1.15000$$

Ahora se hace el proceso para i=1, obteniéndose.

$$y_1 = 1.15000$$
$$h = 0.1$$

$$f(x_1, y_1) = x_1 + 2x_1y_1 = 0.6 + 2(0.6)(1.15000) = 1.98000$$

Al sustituir valores:

$$y_2 = 1.15000 + 0.1(1.98) = 1.34800$$

Se sigue realizando el mismo procedimiento para i= 2, 3, 4, tal como se ve a continuación:

$i = 2;$

$$y_3 = y_2 + hf(x_2, y_2)$$
$$y_3 = 1.34800 + 0.1[0.7 + 2(0.7)(1.34800)]$$
$$y_3 = 1.60672$$

$i = 3;$

$$y_4 = y_3 + hf(x_3, y_3)$$
$$y_4 = 1.60672 + 0.1[0.8 + 2(0.8)(1.60672)]$$
$$y_4 = 1.94380$$

$i = 4;$

$$y_5 = y_4 + hf(x_4, y_4)$$
$$y_5 = 1.94380 + 0.1[0.9 + 2(0.9)(1.94380)]$$
$$y_5 = 2.38368$$

Por lo que la solución de la ecuación diferencial en el intervalo $0.5 \leq x \leq 1$ es:

x	y(x)
0.5	1.00000
0.6	1.15000
0.7	1.34800
0.8	1.60672
0.9	1.94380
1.0	2.38368

Para obtener la aproximación YF a la solución de un problema de valor inicial o PVI, proporcionar la función F(X, Y) y los:

DATOS: La condición inicial X0, Y0, el valor XF donde se desea conocer el valor de YF y el numero N de subintervalos por emplear.

RESULTADOS: Aproximación a YF : Y0.

PASO 1. Hacer H = (XF – X0)/N

PASO 2. Hacer I = 1

PASO 3. Mientras I ≤ N, repetir los pasos 4 a 6.

PASO 4. Hacer Y0 = Y0 + H * F(X0, Y0)

PASO 5. Hacer X0 = X0 + H

PASO 6. Hacer I = I + 1

PASO 7. IMPRIMIR Y0 y TERMINAR

B.-ALGORITMO DEL METODO DE EULER MODIFICADO

En el método de Euler se tomó como válida para todo el intervalo la derivada encontrada en un extremo de éste. Para obtener una exactitud razonable se utiliza un intervalo muy pequeño, a cambio de un error de redondeo mayor (ya que se realizarán más cálculos).

El método de Euler modificado trata de evitar este problema utilizando un valor promedio de la derivada tomada en los dos extremos del intervalo. en lugar de la derivada tomada en un solo extremo.

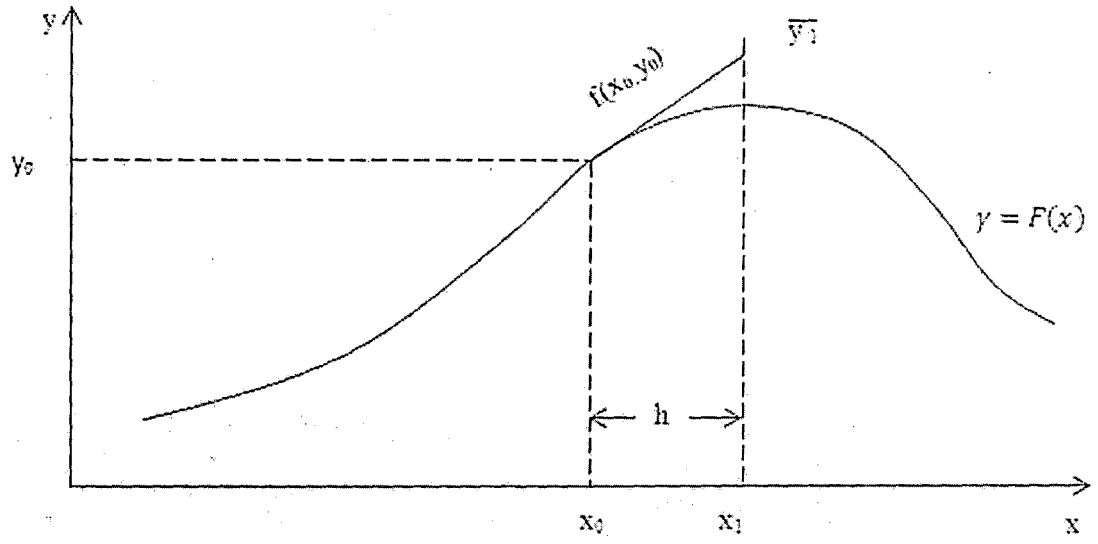


Figura 4.1.10. Representación gráfica del método de Euler modificado

EL METODO DE EULER MODIFICADO CONSTA DE DOS PASOS BASICOS:

1. Se parte de (x_0,y_0) y se utiliza el método de Euler a fin de calcular el valor de y correspondiente a x_1' Este valor de y se denotará aquí como y_1' ya que sólo es un valor transitorio para y_1'.Esta parte del proceso se conoce como paso predictor.
2. El segundo paso se llama corrector, pues trata de corregir la predicción. En el nuevo punto obtenido (x_1, y_1) se evalúa la derivada $f(x_1',y_1)$ usando la ecuación diferencial ordinaria del PVI que se esté resolviendo; se obtiene la media aritmética de esta derivada y la derivada en el punto inicial (x_0',y_0).

$$1/2\ [f(x_0,y_0) + F(x_1,y_1)] = \textit{derivada promedio}$$

Se usa la derivada promedio para calcular un nuevo valor de y_1, con la ecuación $y_1=y_0+hf(x_0,y_0)$, que deberá ser más exacto que y_1

$$y_1 = y_0 + \frac{(x_1 - x_0)}{2}[f(x_0,y_0) + f(x_1,y_1)]$$

y se tomara como valor definitivo de y_1. Este procedimiento se repite hasta llegar a y_n.

$$y_{i+1} = y_i + hf(x_i, y_i)$$

Una vez obtenida y_{i+1} se calcula $f(x_{i+1}, y_{i+1})$, la derivada en el punto (x_{i+1}, y_{i+1}), y se promedia con la derivada previa (x_i, y_i) para encontrar la derivada promedio

$$\frac{1}{2}[f(x_i, y_i) + f(x_{i+1}, y_{i+1})]$$

Se sustituye $f(x_i, y_i)$ con este valor promedio en la ecuación de iteración de euler y se obtiene:

$$y_{i+1} = y_i + \frac{1}{2}[f(x_i, y_i) + f(x_{i+1}, y_{i+1})]$$

Para obtener la aproximación YF a la solución de un PVI, proporcionar la función F(X, Y) y los

DATOS: La condición inicial X0, Y0, el valor XF donde se desea conocer el valor de YF y el numero N de subintervalos por emplear.

RESULTADOS: Aproximación a YF: Y0.

PASO 1. Hacer H = (XF – X0) / N

PASO 2. Hacer I = 1

PASO 3. Mientras I ≤ N, repetir los pasos 4 a 7.

- **PASO 4.** Hacer Y1 = Y0 + H * F(X0, Y0)
- **PASO 5.** Hacer Y0 = Y0 + H/2 * (F(X0, Y0) + F(X0+H, Y1))
- **PASO 6.** Hacer X0 = X0 + H
- **PASO 7.** Hacer I = I + 1

PASO 8. IMPRIMIR Y0 y TERMINAR.

C.-ALGORITMO DEL METODO DE RUNGE-KUTTA DE CUARTO ORDEN

Los métodos de Runge-Kutta (RK) es un conjunto de métodos iterativos (implícitos y explícitos) para la aproximación de soluciones de ecuaciones diferenciales ordinarias, concretamente, del problema del valor inicial.

Sea $y'(t) = f(t, y(t))$

Una ecuación diferencial ordinaria, con $f : \Omega \subset \mathbb{R} \times \mathbb{R}^n \to \mathbb{R}^n$ donde Ω es un conjunto abierto, junto con la condición de que el valor inicial de f sea $(t_0, y_0) \in \Omega$.

Entonces el método RK (de orden *s*) tiene la siguiente expresión, en su forma más general:

$$y_{n+1} = y_n + h \sum_{i=1}^{s} b_i k_i ,$$

donde *h* es el paso por iteración, o lo que es lo mismo, el incremento Δt_n entre los sucesivos puntos t_n y t_{n+1}. Los coeficientes k_i son términos de aproximación intermedios, evaluados en *f* de manera local

$$k_i = f\left(t_n + h\,c_i\,, y_n + h \sum_{j=1}^{s} a_{ij} k_j\right) \quad i = 1, ..., s.$$

con a_{ij}, b_i, c_i coeficientes propios del esquema numérico elegido, dependiente de la regla de cuadratura utilizada. Los esquemas Runge-Kutta pueden ser explícitos o implícitos dependiendo de las constantes a_{ij} del esquema. Si esta matriz es triangular inferior con todos los elementos de la diagonal principal iguales a cero; es decir, $a_{ij} = 0$ para $j = i, ..., s$, los esquemas son explícitos.

Un miembro de la familia de los métodos Runge-Kutta es usado tan comúnmente que a menudo es referenciado como «RK4» o como «el método Runge-Kutta».

Definiendo un problema de valor inicial como:

$$y' = f(x, y), \quad y(x_0) = y_0$$

Entonces el método RK4 para este problema está dado por la siguiente ecuación:

$$y_{i+1} = y_i + \frac{1}{6}(k_1 + 2k_2 + 2k_3 + k_4)\,h$$

Donde

$$\begin{cases} k_1 &= f(x_i, y_i) \\ k_2 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_1 h\right) \\ k_3 &= f\left(x_i + \frac{1}{2}h, y_i + \frac{1}{2}k_2 h\right) \\ k_4 &= f(x_i + h, y_i + k_3 h) \end{cases}$$

Así, el siguiente valor (y_{n+1}) es determinado por el presente valor (y_n) más el producto del tamaño del intervalo (h) por una pendiente estimada. La pendiente es un promedio ponderado de pendientes, donde k_1 es la pendiente al principio del intervalo, k_2 es la pendiente en el punto medio del intervalo, usando k_1 para determinar el valor de y en el punto $x_n + \frac{h}{2}$ usando el método de Euler. k_3 es otra vez la pendiente del punto medio, pero ahora usando k_2 para determinar el valor de y; k_4 es la pendiente al final del intervalo, con el valor de y determinado por k_3. Promediando las cuatro pendientes, se le asigna mayor peso a las pendientes en el punto medio:

$$\text{pendiente} = \frac{k_1 + 2k_2 + 2k_3 + k_4}{6}.$$

Esta forma del método de Runge-Kutta, es un método de cuarto orden lo cual significa que el error por paso es del orden de $O(h^5)$, mientras que el error total acumulado tiene el orden $O(h^4)$. Por lo tanto, la convergencia del método es del orden de $O(h^4)$, razón por la cual es usado en los métodos computaciones.

Para obtener la aproximación YF a la solución de un PVI, proporcionar la función F(X, Y) y los

DATOS: La condición inicial X0, Y0, el valor XF donde se desea conocer el valor de YF y el numero N de subintervalos a emplear.

RESULTADOS: Aproximación a YF : Y0

PASO 1. Hacer H = (XF – X0) / N

PASO 2. Hacer I = 1

PASO 3. Mientras I ≤ N, repetir los pasos 4 a 10.

PASO 4. Hacer K1 = F(X0, Y0)

PASO 5. Hacer K2 = F(X0 + H/2, Y0 + H * K1/2)

PASO 6. Hacer K3 = F(X0 + H/2, Y0 +H * K2/2)

PASO 7. Hacer K4 = F(X0 + H, Y0 + H * K3)

PASO 8. Hacer

Y0 = Y0 + H/6 *(K1 + 2*K2 + 2*K3 + K4)

PASO 9. Hacer X0 = X0 + H

PASO 10. Hacer I = I + 1

PASO 11. IMPRIMIR Y0 y TERMINAR.

4.1.8.- ECUACIONES DIFERENCIALES PARCIALES

Toda ecuación diferencial que contiene derivadas parciales de una o más variables dependientes con respecto a dos o más variables independientes se denomina ecuación diferencial parcial (EDP) y aparecen frecuentemente en problemas de física, química, ingeniería, etc.

$$\frac{\partial^2 u}{\partial x \partial y} - 3\left(\frac{\partial u}{\partial x}\right)\left(\frac{\partial u}{\partial y}\right) = 0$$

$$3x\left(\frac{\partial u}{\partial x}\right) - \left(\frac{\partial^2 u}{\partial y^2}\right) + \left(\frac{\partial^3 u}{\partial x \partial y \partial z}\right) = xyz$$

La forma general de una Ecuación diferencial parcial, lineal de orden 2 es:

$$A\frac{\partial^2 u}{\partial x^2} + B\frac{\partial^2 u}{\partial x \partial y} + C\frac{\partial^2 u}{\partial y^2} + D\frac{\partial u}{\partial x} + E\frac{\partial u}{\partial y} + Fu = G$$

Donde los coeficientes *A,B,C,D,E,F,G* son funciones que están en términos de *x* y *y;* si además se cumple que G(x,y)=0 se dice que la EDP es homogénea de lo contrario es no homogénea.

Una forma equivalente de escritura es:

$$\frac{\partial^2 u}{\partial x^2} = U_{xx} \quad \frac{\partial^2 u}{\partial y^2} = U_{yy} \quad \frac{\partial^2 u}{\partial x \partial y} = U_{yx} \quad \frac{\partial^2 u}{\partial y \partial x} = U_{xy} \quad \frac{\partial u}{\partial x} = U_x \quad \frac{\partial u}{\partial y} = U_y$$

Definiremos el discriminante $d = B^2 - 4AC$

Inicialmente tomando la ecuación del calor se tiene: $U_t = U_{xx}$, se tiene que el discriminante es igual a cero, viendo la forma de la ecuación del calor notamos una similitud con $t = x^2$ (Que se llega mediante la aplicación de la transformada de Fourier a la ecuación) esto motiva a llamar a este tipo de ecuaciones Parabólicas.

Definición 4.1.1. Una Ecuación Diferencial Parcial con coeficientes, lineal y de orden 2 se dice parabólica si $d = B^2 - 4AC = 0$.

A continuación tomando la ecuación de onda se tiene: $U_{tt} - U_{xx} = 0$, se tiene que el discriminante es mayor que cero, viendo la forma de la ecuación de onda notamos una similitud con $t^2 - x^2 = 0$ (que se llega mediante la aplicación de la transformada de Fourier a la ecuación) esto motiva a llamar a este tipo de ecuaciones Hiperbólicas.

Definición 4.1.2. Una Ecuación Diferencial Parcial con coeficientes, lineal y de orden 2 se dice hiperbólica si $d = B^2 - 4AC > 0$.

Finalmente tomando la ecuación de Laplace se tiene: $U_{tt} + U_{xx} = 0$, se tiene que el discriminante es menor que cero, viendo la forma de la ecuación de Laplace notamos una similitud con $t^2 + x^2 = 0$(que se llega mediante la aplicación de la transformada de Fourier a la ecuación) esto motiva a llamar a este tipo de ecuaciones Elípticas.

Definición 4.1.3. Una ecuación Diferencial Parcial con coeficientes, lineal y de orden 2 se dice Elíptica si $d = B^2 - 4AC < 0$.

Las Ecuaciones de tipo Parabólico se presentan al estudiar los procesos de conductibilidad térmica y difusión.

$$k\frac{d^2u}{dx^2} = \frac{dy}{dt}; \quad \text{k>0}$$

Las Ecuaciones de tipo Hiperbólico se presentan en fenómenos oscilatorios: vibraciones de cuerda, membranas, oscilaciones electromagnéticas.

$$a^2\frac{d^2u}{dx^2} = \frac{d^2y}{dt^2}$$

Las Ecuaciones de tipo Elíptico se presentan al estudiar procesos estacionarios, o sea que no cambian con el tiempo.

$$\frac{d^2u}{dx^2} + \frac{d^2u}{dy^2} = 0$$

A.-ALGORITMO DEL METODO EXPLÍCITO

Para aplicar este método se resuelve el problema de valores en la frontera (ecuación del calor unidimensional) con los datos siguientes:

$$\begin{cases} \propto \frac{\partial^2 T}{\partial x^2} = \frac{\partial T}{\partial t} \\ T(x,0) = 20^{\circ}F \quad 0 < x < L \\ T(0,t) = 100^{\circ}F \\ T(1,t) = 100^{\circ}F \qquad t > 0 \end{cases}$$

$\propto$ = 1 pie^2/h
L = 1 pie

$T_{máx}$= 1h

Primero se construye la malla en el dominio de definición dividiendo la longitud de la barra (1 pie) en cuatro subintervalos y el intervalo de tiempo (1 h) en 100 subintervalos

Las condiciones frontera proporcionan la temperatura en cualquier punto del eje t y de la vertical x=1 a cualquier tiempo, mientras que la condición inicial proporciona la temperatura en cualquier punto del eje horizontal x al tiempo cero.

Cada nodo de la malla queda definido por dos coordenadas (i,j); por ejemplo el nodo de coordenadas (3,4) representa la temperatura en el punto x=0.75 pies de la barra al tiempo t=0.04 horas, y el nodo (4,1) la temperatura de la barra en x=1 pies(su frontera) y a t=0.01 horas.

Notar que en el nodo de coordenadas (0,0) (esquina inferior de la malla), la temperatura debería ser 20°F atendiendo la condición inicial, mientras que la condición frontera T(0,t) establece que debería ser de 100°F.

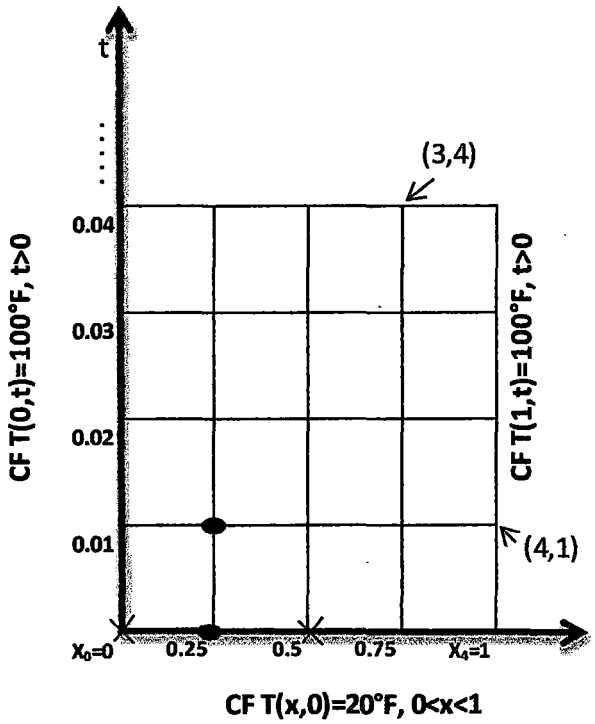


Figura 4.1.10. Representación de la malla en el dominio de definición.

Los puntos que representan estas características se llaman puntos singulares; se acostumbra tomar en ellos un valor de temperatura igual a la media aritmética de las temperaturas sugeridas por la condición inicial y la condición frontera correspondientes. La temperatura tomada para el nodo (0,0) es 60°F. DE igual manera se trata el punto (4,0), cuya temperatura también es 60°F.

Hechas estas consideraciones, el segundo paso consiste en aproximar la ecuación diferencial parcial del problema de valor en la frontera en el nodo (1,0) por la ecuación de diferencias finitas hacia delante y diferencias centrales; entonces queda

$$\frac{T_{1,1}-T_{1,0}}{b}=\propto\frac{T_{0,0}-2T_{1,0}+T_{2,0}}{a^2}$$

Los nodos involucrados en esta ecuación están marcados por círculos y cruces en la figura. De estos, solamente en el nodo (1,1) la temperatura es desconocida, por lo que puede despejarse; entonces resulta

$$T_{1,1}=\propto\frac{b}{a^2}\left(T_{0,0}-2T_{1,0}+T_{2,0}\right)+T_{1,0}$$

al sustituir valores queda

$$T_{1,1}=1\frac{0.01}{(0.25)^2}(60-2(20)+20)+20=26.4$$

Si ahora se aproxima la ecuación del calor en el nodo (*i,j*) = (2,0), mediante la ecuación de diferencias finitas hacia delante y diferencias centrales, se obtiene

$$\frac{T_{2,1}-T_{2,0}}{b}=\propto\frac{T_{1,0}-2T_{2,0}+T_{3,0}}{a^2}$$

Ahora solo se desconoce la temperatura del punto (2,1), ya que todos los demás están dados por la condición inicial; despejando se tiene

$$T_{2,1}=\propto\frac{b}{a^2}\left(T_{1,0}-2T_{2,0}+T_{3,0}\right)+T_{2,0}$$

Se sustituyen valores

$$T_{2,1}=1\frac{0.01}{(0.25)^2}(20-2(20)+20)+20=20$$

Se repiten las mismas consideraciones y cálculos para el punto (3,0) y se obtiene

$$T_{3,1} = 1\frac{0.01}{(0.25)^2}(T_{2,0} - 2T_{3,0} + T_{4,0}) + T_{3,0} = 26.4$$

De esta manera se han obtenido aproximaciones a la temperatura en los tres puntos seleccionados de la barra a un tiempo de 0.01 horas. Al momento se tiene la temperatura de todos los nodos de las dos primeras líneas horizontales (filas) de la malla y se procederá, siguiendo el razonamiento anterior, a calcular la temperatura en todos los nodos intermedios de la tercera fila (1,2), (2,2) y (3,2).

Se empieza con el punto (*i,j*) = (1,1) y se aplica la ecuación de diferencias finitas hacia delante y diferencias centrales, con lo que se obtiene

$$\frac{T_{1,2} - T_{1,1}}{b} = \propto \frac{T_{0,1} - 2T_{1,1} + T_{2,1}}{a^2}$$

de la que

$$T_{1,2} = \propto \frac{b}{a^2}(T_{0,1} - 2T_{1,1} + T_{2,1}) + T_{1,1}$$

Con la sustitución de valores queda

$$T_{1,2} = 1\frac{0.01}{(0.25)^2}(100 - 2(26.4) + 20) + 26.4 = 37.152$$

Al proceder análogamente para los otros puntos se llega a

$$T_{2,2} = 22.048$$

$$T_{3,2} = 37.152$$

Con esto se obtiene la temperatura en los tres puntos seleccionados de la barra cuando hayan transcurrido 0.02 horas.

Este procedimiento se repite para la cuarta, quinta, etc. filas, con lo cual se obtienen las temperaturas en los puntos seleccionados de la barra a tiempo t = 0.03, t = 0.04, etc, hasta llegar al tiempo fijado como $t_{máx}$ = 1 hora.

De esta forma se pueden generar los cien conjuntos de temperaturas del Problema de Valor de Frontera de conducción de calor en una barra metálica.

Este método también se conoce como **método de diferencias hacia delante.**

Para aproximar la solución al problema de valor en la frontera

$$\begin{cases} \propto \dfrac{\partial^2 T}{\partial x^2} = \dfrac{\partial T}{\partial t} \\ T(x,0) = f(x) \quad 0 < x < x_F \\ T(0,t) = g_1(t) \\ T(x_F,t) = g_2(t)\ t > 0 \end{cases}$$

Proporcionar las funciones CI(X), CF1 (T) y CF2 (T) y los

DATOS: El número NX de puntos de la malla en el eje x ,el número NT de puntos de la malla en el eje *t*, la longitud total XF del eje *x*, el tiempo máximo TF por considerar y el coeficiente ALFA de la derivada de segundo orden.

RESULTADOS: Los valores de la variable dependiente T a lo largo del eje *x* a distintos tiempos *t*: T.

PASO 1. Hacer DX = XF/(NX-1)

PASO 2. Hacer DT = TF/(NT-1)

PASO 3. Hacer LAMBDA = ALFA*DT/DX**2

PASO 4. Hacer I=2

PASO 5. Mientras I ≤ NX-1, repetir los pasos 6 y 7.

PASO 6. Hacer T(I) = CI(DX*(I-1))

PASO 7. Hacer I = I+1

PASO 8. Hacer T(1) = (CI(DX) + CF1(DT))/2

PASO 9. Hacer T(NX) = (CI(XF-DX) + CF2(DT))/2

PASO 10. IMPRIMIR T

PASO 11. Hacer J = 1

PASO 12. Mientras J≤ NT repetir los pasos 13 a 24.

PASO 13. Hacer I=2

PASO 14. Mientras I ≤ NX-1, repetir los pasos 15 y 16.

PASO 15. Hacer T1(I) = LAMBDA*T(I-1) +
(1-2*LAMBDA)*T(I)+LAMBDA*T(I+1)

PASO 16. Hacer I = I+1

PASO 17. Hacer I=2

PASO 18. Mientras I ≤ NX-1, repetir los pasos 19 y 20.

PASO 19. Hacer T(I) = T1(I)

PASO 20. Hacer I = I+1

PASO 21. Hacer T(1) = CF1(DT*J)

PASO 22. Hacer T(NX) = CF2(DT*J)

PASO 23. IMPRIMIR T.

PASO 24. Hacer J = J+1

PASO 25. TERMINAR.

4.2 LENGUAJE DE PROGRAMACION C++

El Lenguaje de Programación C++ fue creado por Bjarne Stroustrup en 1985 como una extensión del Lenguaje C, el mismo que es mas consistente y conocido que otros lenguajes de programación científicos.

El Lenguaje de Programación C++ es de propósito general porque ofrece control de flujo, estructuras sencillas y un buen conjunto de operadores. Se puede utilizar en varios tipos de aplicación.

Este Lenguaje ha sido creado estrechamente ligado al sistema operativo UNIX, puesto que fueron desarrollados conjuntamente. Sin embargo, este lenguaje no esta ligado a ningún sistema operativo ni a ninguna máquina concreta.

Sus características principales son:

- Es un Lenguaje que Incluye a versiones anteriores de Lenguaje C
- Abstracción de Datos

- Permite definir nuevos tipos de Datos
- Permite la programación orientada a objetos
- Varias funciones pueden compartir el mismo nombre.
- Permite definir una función como miembro de una estructura.
- Incluye Operadores para reservar y liberar memoria.

4.2.1 OPERADORES

Definición 4.2.1 Son símbolos que se utiliza para manipular datos.

Existen los siguientes tipos de operadores:

A) OPERADORES DE ASIGNACIÓN

Definición 4.2.2 Es el símbolo "=" que se utiliza para dar un valor a una variable.

Ejemplo 4.2.1

A = 3 ;	coloca directamente un valor
A = b ;	se le da un valor de otra variable
A=b=c=3;	da un valor a varias variables
A=b=c=d;	todos toman el valor de la variable d

B) OPERADORES ARITMÉTICOS

Definición 4.2.3 Son símbolos que indican los cálculos que se deben realizar sobre una o más variables dentro de una expresión.

+ suma	suma = a + b
++ Incrementar un uno	x = x + 1; // x = x++; // x +=1;
- resta	resta = a – b

-- Decremento en uno	x = x - 1;	// x = x - -; // x -=1;
* Multiplicación	i = i * j ;	// i * = j ;
/ División	x= x / 2;	// x /=2;
% modulo	Resto de la división de enteros.	

C) OPERADORES DE COMPARACIÓN

Definición 4.2.4 Son símbolos para indicar la relación que hay entre dos o mas valores.

=	==	igual que, cumple sin son iguales
!=		distinto que , lo contrario de igual
> , <, >=, <=		mayor, menor, mayor o igual y menor o igual

D) OPERADORES LÓGICOS

Definición 4.2.5 Son símbolos que sirven para unir varias comparaciones

&&	&	and	(alt + 38)
||	|	or	(alt + 124)
!		not	

E) PRIORIDAD DE LOS OPERADORES

( )

++ -- &

* / %

+ -

< <= > >=

== ¡=

&&

||

=

4.2.2 TIPOS DE DATOS

Definición 4.2.6 Son los atributos de una parte de los datos que indica al ordenador (y/o el programador) algo sobre la clase de datos sobre los que se va a procesar.

A) ENTEROS

Para números enteros con el siguiente rango:

int	-32768 .. 32767
long	-2147483648 .. 2147483647

B) REALES (Coma Flotante)

Para números reales con el siguiente rango:

float	-3.40E+38... -1.17E-37 para negativos
float	1.17E-37 ... 3.40E+38 para positivos
double	-1.79E+308 ... –2.22E-307 para negativos
double	2.22E-307 ... 1.79E+308 para positivos

C) **CARACTER**

Para caracteres solos y cadenas de caracteres:

char [cantidad]

4.2.3 CONSTANTES

Definición 4.2.7 Son aquellos datos que no pueden cambiar a lo largo de la ejecución de un programa.

Sintaxis:

Dentro de la Función Principal

1. Declarar la Variable
2. Variable = valor_según_declaración

Dentro de las Librerías de Cabecera

#define nombre _ constante Valor _ constante

Ejemplo 4.2.2

1) float pi ; pi = 3.14159

2) #define pi 3.14159

4.2.4 VARIABLES

Definición 4.2.8 Son aquellos datos que puede cambiar su valor dentro de la ejecución del programa.

PRIMERA FORMA (Declaración Global): Estas se declaran después de las Librerías y sirven para todo el programa, incluyendo las funciones definidas por el usuario.

```
# include < conio.h >

# include < stdio.h >

tipo_dato  variables globales;

void main()

{

}
```

SEGUNDA FORMA (Declaración Local): Estas se declaran dentro de la función principal ó las funciones definidas por el usuario en el programa.

```
# include < conio.h >

# include < stdio.h >

void main()

{

tipo_dato  variables locales;

}
```

4.2.5 ENTRADA Y SALIDA DE DATOS

Las Librerías que se activa para la entrada y salida de datos son:

- #include < stdio.h >
- #include < bcd.h >

ENTRADA

- **FUNCIÓN CIN : (# include < stdio.h > # include < bcd.h >)**

Sintaxis:

```
cin >>apuntador de Variable;

cin >>Variable 1>>Variable 2;
```

Ejemplo 4.2.3

cin >> n1 >>n2 >>n3 ;

NOTA: *No importa que tipo de datos sea, no se tiene que especificar.*

SALIDA

- **FUNCIÓN COUT: (# include < stdio.h > - # include< bcd.h >)**

  Forma:

```
cout  << " mensaje de Salida " ;

cout  << " mensaje de Salida "  <<Variable <<Variable;

cout  << " mensaje de Salida  \n " <<Variable << " \n" <<Variable;
```

Ejemplo 4.2.4

Cout << "los números son " << a << b <<c;

Para una mejor ubicación del texto en la pantalla colocar el comando:

GOTOXY (Coordenadas_x, Coordenada_y) ; cout ...

4.2.6 INSTRUCCIONES DE CONTROL

Definición 4.2.9 Las Instrucciones de Control permiten que los programas sean más estructurados y puedan de esta forma solucionar todo tipo de problemas (Joyanes Aguilar, 1994).

Estas instrucciones pueden ser:

- Selectivas (**if, if..else, switch)**
- Repetitivas (**for, while y do..while)**
- De Salto (**break, continue, goto)**

SELECTIVAS

Definición 4.2.10 Se realizan mediante las instrucciones **if, if..else, switch** y permiten evaluar una condición o expresión y en función del resultado de la misma se realizara una acción u otra.

A) IF

Toma una decisión referente al bloque de sentencias a ejecutar en un programa, basándose en el resultado (Verdadero o falso) de una Condición.

1ra Sintaxis:

```
if ( Condición _ verdadera )
        Sentencias Ejecutables ;
```

2da Sintaxis:

```
if ( Condición)
{       .....
        Sentencias Ejecutables;
        .....
}
```

```
else
{       .....;
        Sentencias Ejecutables
        .....;}
```

Observación 4.2.1

Si la condición a evaluar en el IF necesita mas opciones colocar :

&& (and) || (or)

Ejemplo 4.2.5

1) CONDICION PARA ENCONTRAR NÚMEROS PARES

```
if (N % 2 ! = 1)
        cout <<" NUMERO PAR"
```

2) CONDICION PARA LOS NUMEROS NEGATIVOS O NÚMEROS CERO

```
if ((N < 0) || (N==0))
        cout <<" NUMERO NEGATIVO O CERO"
```

3) CONDICION PARA LAS PERSONAS MAYORES DE EDAD Y QUE GANEN 500 SOLES, AUMENTARLE 100 SOLES DE LO CONTRARIO DISMINUIRLE 200.

```
if ((SUELDO==500) && (EDAD<17))
{       SUELDO=SUELDO + 100 ;
        cout<<" SU NUEVO SUELDO ES : " << SUELDO ;
```

```
}
else
{       SUELDO=SUELDO - 200 ;
        cout<<" SU SUELDO SIGUE SIENDO :" << SUELDO ; }
```

B) SWITCH

Esta sentencia permite ejecutar una de varias acciones, en función del valor de una expresión.

Sintaxis:

```
switch (EXPRESION)
        {
                case 1 : Sentencias Ejecutables;
                        getch(); break;
                case 2 : Sentencias Ejecutables;
                        getch(); break;
                case 3 : Sentencias Ejecutables;
                        getch(); break;
                case 4 : Sentencias Ejecutables;
                        getch(); break;
        }
```

REGLAS PARA EL USO DEL switch :

1) La EXPRESION puede ser una variable o constante
2) No funciona con datos Flotantes (Reales)
3) El valor en cada CASE debe ser Entero o ' Carácter '
4) C++ no soporta varios casos en el mismo lugar paras esto se utiliza :

   Case 1 :

   Case 2 :

5) Necesita utilizar la sentencia BREAK al finalizar cada CASE. Break hace que la ejecución del programa continúe después del SWITCH.
6) Si no se coloca SWITCH la ejecución del programa se reanuda en las demás CASE.
7) El conjunto de sentencias de cada CASE no necesita encerrarse entre llaves.

REPETITIVAS

Definición 4.2.11 Estas instrucciones hacen que una sección del programa se repita un cierto número de veces. La repetición continua mientras una condición sea verdadera, cuando la decisión es falsa el bucle termina y el control pasa a las sentencias a continuación del bucle, existen tres clases de Bucles **for, while y do..while.**

A) FOR

El bucle FOR, ejecuta una sección de código un número fijo de veces.

Sintaxis:

for (exp1 ; exp2 ; exp3)

Sentencia Ejecutables;

for (exp1 ; exp2 ; exp3)

{ .....;

Sentencia Ejecutables;

.....;

}

for (exp1, exp A; exp2, exp B; exp3, Exp B)

{ .....;

Sentencia Ejecutables;

.....;

}

Donde:

- exp1 : Es el Valor inicial (Signo =)
- exp2 : Condición(Signo <=, >=, <, >)
- exp3 : Es el Operador de incremento o decremento

(i + + , i - -)

Ejemplo 4.2.6

1) IMPRIMIR LOS 10 PRIMEROS NÚMEROS.

```
int i ;

    for ( i = 1 ; i < = 10 ; i + + )

        cout << "\n" << i ;
```

2) IMPRIMIR LOS 20 PRIMEROS NUMERO EN FORMA DESCENDENTE

```
int i ;

    for ( i = 20  ; i >= 1 ; i - - )
```

```
cout << "\n" << i ;
```

B) WHILE

El bucle while ejecuta un bloque de sentencias ejecutables mientras su condición sea verdad.

Sintaxis:

```
while ( condicion_verdadera )
{       .....;
        Sentencias Ejecutables;
        ..... ;
}
```

Ejemplo 4.2.7

1) IMPRIMIR LOS 20 PRIMEROS NUMERO ENTEROS ASCENDENTE

```
int c=1;
        while ( c < = 20 )
        {       cout << "\n"  << c ;
                c + +  ;
        }
```

2) IMPRIMIR LOS 20 PRIMEROS ENTEROS EN FORMA DESCENDENTE

```
int  c = 20 ;
        while ( c > = 1 )
```

```
{       cout << "\n"  << c ;

        c - - ;

}
```

Observación 4.2.2

¿COMO INGRESAR UNA CADENA, SI VARIABLE SE DECLARA COMO CHAR?

UTILIZAR **GETLINE** (VARIABLE, CANTIDAD_DECLARADA);

```
char  nombre[40] ;

      cout << " ingrese su nombre completo :" ;

      cin.getline(nombre,40);
```

¿ COMO CONTROLAR LA CANTIDAD DE DECIMALES DE UNA RESPUESTA ?

UTILIZAR #include < iomanip . h >

SETPRECISION (CANTIDAD_DE_DECIMALES)

```
float  raiz_cuadrada , n ;

cout << " ingrese un numero entero:" ; cin >> n;

raiz_cuadrada =  pow ( n , 0.5 );

cout << "La raiz es :" << setprecision (2) << raiz_cuadrada ;
```

¿COMO IMPRIMIR VARIAS REPUESTAS SEPARADOS POR UN ESPACIO DETERMINADO?

UTILIZAR #include < iomanip . h >

SETW (CANTIDAD_DE_ESPACIOS)

```
Cout << a << setw(5)<< b << setw(5) << c ;  // Ancho
```

```
Cout << a << " \t " << b << " \t " << c ;        //tabulador
```

DO... WHILE

Ejecuta un bloque de sentencias, una o más veces, dependiendo de la condición que tiene que ser verdadera.

Sintaxis:

```
do
{
        ..... ;
        SENTENCIAS EJECUTABLES;
        ..... ;
}
while ( CONDICIÓN _ VERDADERA);
```

Ejemplo 4.2.8

1) INGRESAR NUMEROS Y TERMINAR CUANDO SE INGRESE UN NUMERO NEGATIVO DEVOLVIENDO LA SUMA.

```
SUMA = 0 ;
cin >> N ;
do
{       SUMA + = N ;
        cin >> N ;
} while ( N  > 0 );
```

2) COLOCAR ¿DESEA CONTINUAR S/N? PARA RETORNAR A UN PROGRAMA

```
char OP;
do
{       clrscr();
        cout <<" INGRESE UN NUMERO : " ;
        cin >> N ;
        cout <<" DESEA SEGUIR S/N : " ;
        cin >> OP ;
} WHILE ( OP == ' S ' ) ;
```

DE SALTO

Definición 4.2.12 Estas instrucciones hacen que en un determinado momento de la ejecución del programa se traslade a otra parte o abandone una instrucción repetitiva y pueda continuar con el resto del programa, existen tres clases: **break, continue y goto.**

A) BREAK

Hace finalizar la ejecución del Ciclo ó bucle : DO, FOR, SWITCH, WHILE mas interno que lo contenga.

Sintaxis:

```
break;
```

BREAK solamente finaliza la ejecución de la sentencia de donde esta incluida.

Ejemplo 4.2.9

Imprimir los 4 primeros números enteros

```
n=1;
while( n < 5 )
{
        cout<< n ;
        if ( n == 4)
                break;
        else
                n++;
}
```

B) CONTINUE

Traslada la ejecución del programa a la ultima línea del Ciclo ó bucle : DO, FOR, SWITCH, WHILE mas interno que lo contenga.

Sintaxis:

```
continue;
```

Ejemplo 4.2.10

Imprimir los números entre 1 y 50 que no sean múltiplos de 5

```
n=1;
while( n < 50 )
{
        if ( n % 5 == 0)
                continue;
        else
                cout<< n ;
n++;
}
```

C) GOTO

Traslada la ejecución del programa a una línea específica identificada por una etiqueta.

Sintaxis:

```
goto etiqueta;
.....
.....
Etiqueta: bloque de SENTENCIAS;
```

Ejemplo 4.2.11

Calcular el área de un triangulo

```
Inicio: { cout<<" Ingrese la base del triangulo:";
        cin >>b;
        cout<<"Ingrese la altura del Triangulo:";
        cin >>a;
        cout<<"El área del triangulo es : "<<(b*a)/2;
        }
        if (a>0)  goto  Inicio;
        cout<<"FIN DEL PROGRAMA";
```

4.2.7 FUNCIONES

Definición 4.2.13 Es una colección independiente de declaraciones y sentencias, generalmente realiza una tarea específica, En C++ existe una función por naturaleza y es el programa principal:

```
void main ( )
{
.....
}
```

Cuando se llama a una función el control pasa a el mismo para su ejecución, una vez finalizado el trabajo de la función devuelve el control a la función que lo llamó.

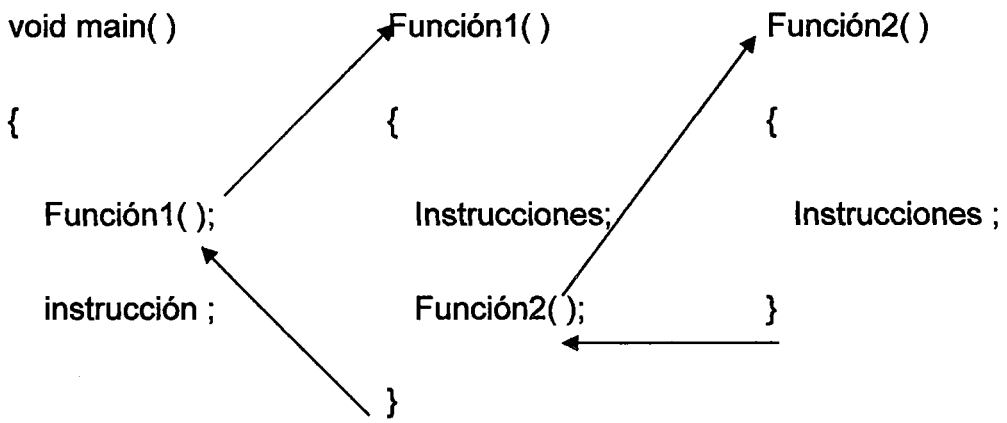


Grafico 4.2.1

Observación 4.2.3

- Una función no pertenece al programa
- En el programa sólo se le llama
- El cuerpo de una función es la parte funcional de un programa.
- Una función siempre debe estar fuera de la función principal main( ).

PASOS PARA DEFINIR UNA FUNCIÓN:

1. Tener en cuenta las variables globales o locales
2. En C++ se debe declarar una función antes de utilizarla, colocándolo el encabezado de la función

int suma (int a, int b) ;

3. Comenzar con la función

Sintaxis:

```
TIPO  NOMBRE (TIPO VARIABLE DE ENTRADA)
{
        DECLARACIONES DE VARIABLES LOCALES;
        SENTENCIAS EJECUTABLES;
        RETURN [VARIABLE;]    // Variables de Salida
}
```

Donde:

TIPO : Indica el tipo de Datos que devolverá la función luego de su uso (no es necesario, solo en casos que se quiera hacer referencia a la salida).

NOMBRE : Es el nombre de la función, trata de no tenerlo como variable.

TIPO VARIABLE DE ENTRADA

Aquí se colocan las Variables con su respectivo tipo de Datos que entraran a la función para su posterior uso, reemplazando a la variable verdadera.

Si dentro de la función se hace uso de variables que no pertenecen a la función MAIN solo se declararan dentro de la función, como una variable local.

FORMAS:

```
        SUMA ( );          → Función Sin Argumentos
INT     SUMA ( ) ;
```

INT SUMA (INT A) ; → Funcion con 1 entrada

INT SUMA (INT A,INT B) ; → Funcion con 2 entradas

INT SUMA (INT A, FLOAT C, INT B) → Funcion con 3 entradas

VALOR RETORNADO POR UNA FUNCIÓN – SENTENCIA RETURN

Indicara que variable retornara con el valor, solo puede haber un solo retorno y en algunos casos no se coloca valor de retorno cuando se trata de varios valores.

Sintaxis:

return variable;

Observación 4.2.4

- Esta variable debe estar declarado dentro de la función o como variable global.

- En el valor de retorno también se le puede alterar su devolución

return (N)

return (N + 2)

return (N * 2)

LLAMADA A LA FUNCIÓN

Una llamada se realiza para que la función tenga efecto.

Sintaxis:

[Variable =] Nombre_de_la_función (parámetros o argumentos)

Ejemplo 4.2.12

Suma ( )

Cout << suma ( )

Su = suma ( )

4.2.8 LIBRERÍA DE FUNCIONES CREADAS POR EL USUARIO

Definición 4.2.15 Son un programa fichero que tiene las funciones que serán compiladas en forma separada y pueda ser utilizado en cualquier programa de C++.

Pasos:

a) Se crean solo funciones dentro de un programa, indicando en la parte superior los prototipos de las funciones y los ficheros.
b) Luego, una vez terminado la función grabarlo con un nombre y extensión (**.hpp**)

Ejemplo 4.2.13

Sumatoria. hpp

Factorial. hpp

c) Se pueden colocar varias funciones dentro de este programa(**.hpp**)
d) No ejecutar un programa **hpp**, mas bien cerrarlo.
e) Dentro de un programa **cpp**, llamar a la función que se encuentra dentro del programa **hpp**.

Sintaxis:

En la Cabecera del programa CPP :

 # include " NombredelFichero.hpp "

4.2.9 ARREGLOS

Definición 4.2.16 Los arreglos son estructuras de datos, que permiten el almacenamiento masivo de información.

Un arreglo esta compuesto por varios componentes, todas del mismo tipo y almacenados consecutivamente en la memoria de la PC.

Ventajas de usar arreglos:

- Almacenamiento de datos en una sola variable
- A la vez cada dato es independiente de los demás
- Se puede acceder a cada elemento indicando el número de índice.
- Se pueden manipular con operaciones de cualquier tipo

Los Arreglos se presentan en dos formas:

VECTORES O ARREGLOS UNIDIMENSIONALES

Definición 4.2.17 Son listas de datos que pueden almacenar un número determinado de datos.

	0	1	....	n
NOMBRE DEL ARREGLO	Dato 1	Dato 2	....	Dato N

Un vector esta compuesto:

Nombre del Arreglo

Índice : 0, 1, 2, 3,... n (Utilizar Estructuras Repetitivas)

Datos : Son los datos ingresados al arreglo de un mismo tipo.

Ejemplo 4.2.14

	0	1	2	3	4
NOTAS	08	09	10	11	10
NOMBRES	ANA	LUISA	LIZA	JANET	ELISA
PROMEDIOS	10.5	11.50	3.45	19.55	0.025

Para acceder a cada elemento solo se escribirá el nombre de la variable arreglo y el índice.

- Para acceder a Janet → NOMBRES [3]
- Para acceder a 3.45 → PROMEDIOS [2]

Sintaxis:

TIPO_DE_DATOS NOMBRE_DEL_ARREGLO [TAMAÑO_APROX] ;

Observación 4.2.6

Si el tipo de datos del arreglo es de caracteres su declaración será:

CHAR nombre_del_arreglo **[tamaño_aprox] [cantidad_de_espacios];**

Ejemplo 4.2.15

- Declarar el ingreso de 5 notas enteras

  int notas [5] ;

- Declarar el ingreso de 5 nombres de personas

  char nombres [5] [20];

MATRICES O ARREGLOS BIDIMENSIONALES

Definición 4.2.18 Son Arreglos que tienen 2 dimensiones.

	1	2	3	4
1	10	8	13	11
2	10	8	13	11
3	10	8	13	11
4	10	8	13	11

Una Matriz esta compuesta:

Nombre del Arreglo

Índice filas :0, 1, 2, 3, ... n(utilizar estructuras repetitivas)

Índice Columnas :0, 1, 2, 3, ... n(utilizar estructuras repetitivas)

Datos :Son los datos ingresados al arreglo de un mismo tipo.

Ejemplo 4.2.16

Nombre del Arreglo: NÚMERO

	1	2	3	4
1	A	E	I	M
2	B	F	J	N
3	C	G	K	O
4	D	H	L	P

Para acceder a cada elemento solo se escribirá el nombre de la variable arreglo y los índices.

- Para acceder a la Letra A (NÚMERO [1][1])
- Para acceder a la Letra L (NÚMERO [4][3])

Sintaxis:

TIPO_DE_DATOS NOMBRE_DEL_ARREGLO [TAMAÑO_FILAS] [TAMAÑO_COLUMNAS] ;

Observación 4.2.7

Al hacer referencia de cada elemento de un arreglo Bidimensional se coloca el nombre del arreglo luego la fila y la columna.

Ejemplo 4.2.17

Ingresar un arreglo Bidimensional de 3 x 3 y determinar la suma de todos sus Elementos.

```
For( I = 1 ; I < = 3 ; I + + )
        For( j = 1 ;j < = 3 ; j + + )
                Cin>> números [I] [j];
For( I = 1 ; I < = 3 ; I + + )
        For( j = 1 ;j < = 3 ; j + + )
                suma=suma+números[ I ] [ j ] ;
Cout<<"la suma es " << suma ;
```

V. MATERIALES Y MÉTODOS

- **MATERIALES**

- Material de Oficina: papel bond carta, lapiceros, liquid-paper, calculadora.
- Material bibliográfico: Libros de la especialidad de Análisis Numérico y el Lenguaje de programación C++.
- Material de cómputo: BORLAND C++, computadora e impresora.

- **MÉTODO**

La elaboración del presente trabajo de investigación demando del suscrito ordenar información recopilada durante su vida profesional como docente del curso de programación de computadoras en la facultad de ciencias naturales y matemática.

Para cada método de análisis numérico materia de esta investigación se ha resuelto ejercicios donde se puede visualizar como trabaja este sistema computacional.

El método empleado en este trabajo de investigación es inductivo y deductivo que ha hecho posible interpretar el formalismo de la teoría, así como también para analizar las soluciones y poder implementar un sistema computacional.

VI. RESULTADOS

El resultado de la presente investigación es un sistema computacional para resolver problemas de análisis numérico, el mismo que se adjunta con este informe.

6.1 DESARROLLO DE LAS APLICACIONES EN C++

6.1.1 APLICACIÓNES PARA LAS ECUACIONES NO LINEALES

A.- ALGORITMO DEL METODO DEL PUNTO FIJO

PROGRAMA (PUNTOFIJO.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>

void main()
{float X0,EPS,X;
int MAXIT,I;
clrscr();
cout<<"Ingresar valor inicial:";
cin>>X0;
cout<<"Ingresar el criterio de convergencia:";
cin>>EPS;
cout<<"Ingresar el número máximo de iteraciones:";
cin>>MAXIT;
I=1;
while (I<MAXIT)
{X= G(X0);
if (abs(X-X0)<=EPS)
   {cout<<"La raíz aproximada es:"<<X;
   break;
   }
I=I+1;
X0=X;}
if (I>=MAXIT)
        cout<<"EL METODO NO CONVERGE A UNA RAIZ";
getch();
}
```

B.- ALGORITMO DEL METODO DE NEWTON-RAPHSON

PROGRAMA (NEWTON.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>
void main()
{float X0,EPS,EPS1,X;
int MAXIT,I;
clrscr();
cout<<"Ingresar valor inicial:";
cin>>X0;
cout<<"Ingresar el criterio de convergencia:";
cin>>EPS;
cout<<"Ingresar el criterio de exactitud:";
cin>>EPS1;
cout<<"Ingresar el número máximo de iteraciones:";
cin>>MAXIT;
I=1;
while (I<MAXIT)
{
X= (X0-F(X0))/DF(X0);
if (abs(X-X0)<EPS)
   {cout<<"La raíz aproximada es:"<<X;
   break;
   }
if (abs(F(X))<EPS1)
   {cout<<"La raíz aproximada es:"<<X;
   break;
   }
I=I+1;
X0=X;
}
if (I>=MAXIT)
        cout<<"EL METODO NO CONVERGE A UNA RAIZ";
getch();
}
```

C.- ALGORITMO DEL METODO DE LA SECANTE

PROGRAMA (SECANTE.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>
```

```
void main()
{float X0,X1,EPS,EPS1,X;
int MAXIT,I;
clrscr();
cout<<"Ingresar dos valores iníciales:";
cin>>X0>>X1;
cout<<"Ingresar el criterio de convergencia:";
cin>>EPS;
cout<<"Ingresar el criterio de exactitud:";
cin>>EPS1;
cout<<"Ingresar el número máximo de iteraciones:";
cin>>MAXIT;
I=1;
while (I<MAXIT)
{
X= (X0-(X1-X0)*F(X0))/((F(X1)-F(X0));
if (abs(X-X1)<EPS)
   {cout<<"La raíz aproximada es:"<<X;
   break;
   }
if (abs(F(X))<EPS1)
   {cout<<"La raíz aproximada es:"<<X;
   break;
   }
X0=X1;
X1=X;
I=I+1;
}
if (I>=MAXIT)
        cout<<"EL METODO NO CONVERGE A UNA RAIZ";
getch();
}
```

D.- ALGORITMO DEL METODO DE POSICION FALSA

PROGRAMA (POSICIONF.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>

void main()
{float XI,XD,EPS,EPS1,X FX,FD;
int MAXIT,I;
clrscr();
cout<<"Ingresar dos valores iníciales que forman un intervalo:";
cin>>XI>>XD;
```

```
cout<<"Ingresar el criterio de convergencia:";
cin>>EPS;
cout<<"Ingresar el criterio de exactitud:";
cin>>EPS1;
cout<<"Ingresar el número máximo de iteraciones:";
cin>>MAXIT;
I=1;
FI = F(XI);
FD = F(XD);
while (I<MAXIT)
{
XM= (XI*FD-XD*FI)/(FD-FI);
FM = F(XM);
if (abs(FM)<EPS1)
   {cout<<"La raíz aproximada es:"<<XM;
   break;
   }
if (abs(XD-XI)<EPS)
   {XM=(XD+XI)/2;
    cout<<"La raíz aproximada es:"<<XM;
   break;
   }
if (FD*FM>0)
  XD=XM;
if (FD*FM<0)
  XI=XM;
I=I+1;
}
if (I>=MAXIT)
        cout<<"EL METODO NO CONVERGE A UNA RAIZ";
getch();
}
```

6.1.2. APLICACIÓNES PARA LOS SISTEMAS DE ECUACIONES LINEALES

A.- ALGORITMO DEL METODO DE ELIMINACION DE GAUSS

PROGRAMA (ELIMGAUSS.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>

void main()
{float A(10,10),b(10),X(10),DET;
int N,I,J,K;
clrscr();
```

```
cout<<"Ingresar numero de ecuaciones:";
cin>>N;
cout<<"Ingresar la matriz de coeficientes:";
for (I=0;I<N;I++)
for (J=0;J<N;J++)
cin>>A(I,J);
cout<<"Ingresar el vector de términos independientes:";
for (I=0;I<N;I++)
cin>>b(I);
DET=1;
I=1;
while (I<=N-1)
{
DET=DET*A(I,I);
if (DET==0)
{
cout<<"HAY UN CERO EN LA DIAGONAL PRINCIPAL"
break;
}
K=I+1;
while (K<=N)
{
while(J<=N)
{
A(K,J)=A(K,J)-A(K,I)*A(I,J)/A(I,I);
J=J+1;
}
b(K)=b(K)-A(K,I)*b(I)/A(I,I);
K=K+1;
}
I=I+1;
}
DET=DET*A(N,N);
if (DET==0)
{
cout<<"HAY UN CERO EN LA DIAGONAL PRINCIPAL"
break;
}
X(N)=b(N)/A(N.N);
I=N-1;
while(I>=1)
{
X(I)=b(I);
J=1;
while(J<=N)
{
```

```
X(I)=X(I)-A(I,J)*X(J);
J=J+1;
}
X(I)=X(I)/A(I,I);
I=I-1;
}
cout<<"El Vector solución es:\n"
for (I=0;I<N;I++)
  cout<<X(I)<<"\t";
cout<<"El Determinante es:"<<DET;
getch();
}
```

B.- ALGORITMO DEL METODO DE CHOLESKY

PROGRAMA (CHOLESKY.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>
void main()
{float L(10,10),A(10,10),b(10),X(10);
int N,I,J,K,S;
clrscr();
cout<<"Ingresar el orden de la matriz:";
cin>>N;
cout<<"Ingresar la matriz de coeficientes:";
for (I=0;I<N;I++)
for (J=0;J<N;J++)
cin>>A(I,J);
cout<<"Ingresar el vector de términos independientes:";
for (I=0;I<N;I++)
cin>>b(I);
L(1,1)=A(1,1)*0.5;
I=1;
while (I<=N)
{
L(I,1)=A(I,1)/L(1,1);
I=I+1;
}
I=2;
while (I<=N)
{
S=0;
K=1;
while(K<=I-1)
```

```
{
S=S+L(I,K)*2;
K=K+1;
}
L(I,I)=(A(I,I)-S)*0.5;
If (I==N)
{cout<<"La matriz L es:";
for (I=0;I<N;I++)
for (J=0;J<N;J++)
cin>>L(I,J);
}
else
J=I+1;
while(J<=N)
{
S=0;
K=1;
while (K<=I-1)
{
S=S+L(I,K)*L(J,K);
K=K+1;
}
L(J,I)=(A(J,I)-S)/L(I,I);
J=J+1;
}
I=I+1;
}

cout<<"La matriz L es:";
for (I=0;I<N;I++)
for (J=0;J<N;J++)
cin>>L(I,J);
```

6.1.3 APLICACIÓNES PARA LOS SISTEMAS DE ECUACIONES NO LINEALES

A.- ALGORITMO DEL METODO DE NEWTON-RAPHSON MULTIVARIABLE

PROGRAMA (NRMULTIVAR.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>

void main()
{float A(10,10),b(10,10),X0(10),EPS,DET;
int N,I,J,MAXIT;
```

```
clrscr();
cout<<"Ingresar numero de ecuaciones:";
cin>>N;
cout<<"Ingresar el vector de valores iniciales:";
for (I=0;I<N;I++)
cin>>X0[I];
cout<<"Ingresar el criterio de convergencia:";
cin>>EPS;
cout<<"Ingresar el número máximo de iteraciones:";
cin>>MAXIT;
K=1;
while (K<=MAXIT)
{
cout<<"Ingresar la matriz de coeficientes:";
for (I=0;I<N;I++)
for (J=0;J<N;J++)
cin>>A(I,J);
cout<<"Ingresar el vector de términos independientes:";
for (I=0;I<N;I++)
cin>>b(I);
DET=1;
I=1;
while (I<=N-1)
{
DET=DET*A(I,I);
if (DET==0)
{
cout<<"HAY UN CERO EN LA DIAGONAL PRINCIPAL"
break;
}
K=I+1;
while (K<=N)
{
while(J<=N)
{
A(K,J)=A(K,J)-A(K,I)*A(I,J)/A(I,I);
J=J+1;
}
b(K)=b(K)-A(K,I)*b(I)/A(I,I);
K=K+1;
}
I=I+1;
}
DET=DET*A(N,N);
if (DET==0)
{
```

```
cout<<"HAY UN CERO EN LA DIAGONAL PRINCIPAL"
break;
}
X0(N)=b(N)/A(N.N);
I=N-1;
while(I>=1)
{
X0(I)=b(I);
J=1;
while(J<=N)
{
X0(I)=X0(I)-A(I,J)*X0(J);
J=J+1;
}
X0(I)=X0(I)/A(I,I);
I=I-1;
}
cout<<"El Vector solución es:\n"
for (I=0;I<N;I++)
  cout<<X0(I)<<"\t";
getch();
}
X=X0;
K=K+1;
cout<<"NO CONVERGE";
getch();
}
```

B.- ALGORITMO DEL METODO DE BROYDEN

PROGRAMA (BROYDEN.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>

void main()
{float A(10,10),b(10,10),X0(10),X1(10),EPS,DET;
int N,I,J,MAXIT;
clrscr();
cout<<"Ingresar numero de ecuaciones:";
cin>>N;
cout<<"Ingresar el vector X0 de valores iniciales:";
for (I=0;I<N;I++)
cin>>X0(I);
cout<<"Ingresar el vector X1 de valores iniciales:";
for (I=0;I<N;I++)
```

```
cin>>X1(I);
cout<<"Ingresar el criterio de convergencia:";
cin>>EPS;
cout<<"Ingresar el número máximo de iteraciones:";
cin>>MAXIT;
K=1;
while (K<=MAXIT)
{
cout<<"Ingresar la matriz de coeficientes:";
for (I=0;I<N;I++)
for (J=0;J<N;J++)
cin>>A(I,J);
cout<<"Ingresar el vector de términos independientes:";
for (I=0;I<N;I++)
cin>>b(I);
DET=1;
I=1;
while (I<=N-1)
{
DET=DET*A(I,I);
if (DET==0)
{
cout<<"HAY UN CERO EN LA DIAGONAL PRINCIPAL"
break;
}
K=I+1;
while (K<=N)
{
while(J<=N)
{
A(K,J)=A(K,J)-A(K,I)*A(I,J)/A(I,I);
J=J+1;
}
b(K)=b(K)-A(K,I)*b(I)/A(I,I);
K=K+1;
}
I=I+1;
}
DET=DET*A(N,N);
if (DET==0)
{
cout<<"HAY UN CERO EN LA DIAGONAL PRINCIPAL"
break;
}
X0(N)=b(N)/A(N.N);
I=N-1;
```

```
while(I>=1)
{
X0(I)=b(I);
J=1;
while(J<=N)
{
X0(I)=X0(I)-A(I,J)*X0(J);
J=J+1;
}
X0(I)=X0(I)/A(I,I);
I=I-1;
}
cout<<"El Vector solución es:\n"
for (I=0;I<N;I++)
  cout<<X0(I)<<"\t";
getch();
}
X=X0;
K=K+1;
cout<<"NO CONVERGE";
getch();
}
```

6.1.4. APLICACIÓNES PARA LA APROXIMACIÓN FUNCIONAL E INTERPOLACIÓN

A.-ALGORITMO DEL METODO DE APROXIMACION POLINOMIAL SIMPLE

PROGRAMA (POLSIMPLE.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>
void main()
{float A(10),B(10,10),X(10),FX(10);
int N,I,J,XINT,C,F,K;
clrscr();
cout<<"Ingresar grado del polinomio:";
cin>>N;
cout<<"Ingresar el vector X de valores iniciales:";
for (I=0;I<=N;I++)
cin>>X(I);
cout<<"Ingresar el vector FX de valores iniciales:";
for (I=0;I<=N;I++)
cin>>FX(I);
cout<<"Ingresar el número para la interpolacion:";
cin>>XINT;
```

```
I=0;
while(I<=N)
{
B(I,0)=1;
J=1;
while(J<=N)
{
B(I,J)=B(I,J-1)*X(I)
J=J+1;
}
B(I,N+1)=FX(I);
I=I+1;
}
**se resuelve el sistema de ecuaciones por el método de Gauss Jordan**
C=N+1;
F=N;
for(K=1;C-1;K++)
{
  B(K,:)=B(K,:)/B(K,K);
   for(J=K+1;F;J++)
{
  B(J,:)=B(J,:)-B(K,:)*B(J,K);
  J=J+1;
  }
  K=K+1;
}
for(K=F;2;K--)
{
   for(J=K-1;1;J--)
{
    B(J,:)=B(J,:)-B(K,:)*B(J,K);
    J=J-1;
 }
  K=K-1;
}
for(I=0;N-1;I++)
  A(I)=A(I)+((B(I+1,N+1))*pow(XINT,I));
cout<<"Los coeficientes del polinomio de aproximación es:\n"
for (I=0;I<N;I++)
  cout<<A(I)<<"\t";
getch();
}
```

B.-ALGORITMO DEL METODO DE APROXIMACION POLINOMIAL DE LAGRANGE

PROGRAMA (LAGRANGE.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>
void main()
{float X(10),FX(10);
int N,I,J,XINT,FXINT;
clrscr();
cout<<"Ingresar grado del polinomio:";
cin>>N;
cout<<"Ingresar el vector X de valores iniciales:";
for (I=0;I<=N;I++)
cin>>X(I);
cout<<"Ingresar el vector FX de valores iniciales:";
for (I=0;I<=N;I++)
cin>>FX(I);
cout<<"Ingresar el número para la interpolacion:";
cin>>XINT;
FXINT=0;
I=0;
while(I<=N)
{
L=1;
J=0;
while(J<=N)
{
if (I != J)
{
L=L*(XINT-X(J))/(X(I)-X(J));
J=J+1;
}
FXINT=FXINT+L*FX(I);
I=I+1;
}
cout<<"La aproximación es:"<<FXINT;
getch();
}
```

C.-ALGORITMO DEL METODO DE INTERPOLACION POLINOMIAL DE NEWTON

PROGRAMA (POLNEWTON.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>
void main()
{float X(10),FX(10),T(10,10),P,FXINT;
int N,I,J,XINT;
clrscr();
cout<<"Ingresar el grado del polinomio:";
cin>>N;
cout<<"Ingresar el vector de valores iníciales X:";
for (I=0;I<=N;I++)
cin>>X[I];
cout<<"Ingresar el vector de valores iníciales FX:";
for (I=0;I<=N;I++)
cin>>FX[I];
I=0;
while (I<=N-1)
{
T(I,0)=(FX(I+1)-FX(I))/(X(I+1)-X(I));
I=I+1;
}
J=1;
while (J<=N-1)
{
I=J;
while (I<=N-1)
{
T(I,J)=(T(I,J-1)-T(I-1,J-1))/(X(I+1)-X(I-J));
I=I+1;
}
J=J+1;
}
FXINT=FX(0);
I=0;
while (I<=N-1)
{
P=1;
J=0;
while (J<=1)
{
P=P*(XINT-X(J));
```

```
J=J+1;
}
FXINT=FXINT+T(I,I)*P;
I=I+1;
}
cout<<"La Aproximación es:";
cout<<FXINT;
getch();
}
```

D.- ALGORITMO DEL METODO DE INTERPOLACION POLINOMIAL CON MINIMOS CUADRADOS

PROGRAMA (MINCUAD.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>
void main()
{float X(10),FX(10),S(10),SS(10),XX,B(10,10);
int N,I,J,M;
clrscr();
cout<<"Ingresar el grado del polinomio:";
cin>>N;
cout<<"Ingresar el numero de valores:";
cin>>M;
cout<<"Ingresar el vector de valores iníciales X:";
for (I=0;I<=N;I++)
cin>>X[I];
cout<<"Ingresar el vector de valores iníciales FX:";
for (I=0;I<=N;I++)
cin>>FX[I];
J=0;
while (J<=2*N-1)
{
if(J<=N)
SS(J);
S(J)=0;
J=J+1;
I=1;
while (I<=N)
{
XX=1;
J=0;
while (J<=2*N-1)
{
```

```
if(J<=N)
SS(J)=SS(J)+XX*FX(I);
XX=XX*X(I);
S(J)=S(J)+XX;
J=J+1;
}
I=I+1;
}
B(0,0)=M;
I=0;
while (I<=N)
{
J=0;
while (J<=N)
{
if ((I<>0)&&(J<>0))
B(I,J)=S(J-1+I);
J=J+1;
}
B(I,N+1)=SS(I);
I=I+1;
}
//resolver el sistema de ecuaciones lineales BA=SS de orden N+1
cout<<"Los coeficientes del polinomio de aproximación es:";
for(I=0;I<=N;I++)
cout<<A(I)<<"\t";
getch();
}
```

6.1.5 APLICACIÓNES PARA LAS INTEGRALES

A.-ALGORITMO DEL METODO TRAPEZOIDAL COMPUESTO

PROGRAMA (TRAPCOMP.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>
void main()
{float A,B,H,AREA,S,X;
int N,I;
clrscr();
cout<<"Ingresar el límite inferior:";
cin>>A;
cout<<"Ingresar el límite superior:";
cin>>B;
```

```
cout<<"Ingresar el numero de trapecios:";
cin>>N;
X=A;
S=0;
H=(B-A)/N;
if(N==1)
{
I=1;
while(I<=N-1)
{
X=X+H;
S=S+F(X);
I=I+1;
}
}
AREA=(H/2)*(F(A)+2*S+F(B));
cout<<"EL AREA ES:"
getch();
}
```

B.-ALGORITMO DEL METODO DE SIMPSON

PROGRAMA (SIMPSON.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>
void main()
{float A,B,H,AREA,S1,S2,X;
int N,I;
clrscr();
cout<<"Ingresar el límite inferior:";
cin>>A;
cout<<"Ingresar el límite superior:";
cin>>B;
cout<<"Ingresar el numero de trapecios:";
cin>>N;
S1=0;
S2=0;
X=A;
S=0;
H=(B-A)/N;
if(N==1)
{
I=1;
while(I<=N/2-1)
```

```
{
X=X+H;
S1=S1+F(X);
X=X+H;
S2=S2+F(X);
I=I+1;
}
}
X=X+H;
S1=S1+F(X);
AREA=(H/2)*(F(A)+ 4*S1+2*S2+F(B));
cout<<"EL AREA ES:";
cout<<AREA;
getch();
}
```

6.1.6 APLICACIÓN PARA LAS DERIVADAS

A.-ALGORITMO DEL METODO DE DERIVACIÓN CON POLINOMIOS DE LAGRANGE

PROGRAMA (DERPOLAGRA.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>
void main()
{float XD,DP,PT[10][10];
int N,P,S,K,I,J;
clrscr();
cout<<"Ingresar el grado del polinomio de lagrange:";
cin>>N;
cout<<"Ingresar los N+1 valores de interpolación (X,F(X)):";
for(i=0;i<=N;i++)
cin>>PT[i][i];
cout<<"Ingresar el punto donde se va a encontrar la derivada:";
cin>>XD;
DP=0;
I=0;
while(I<=N)
{
P=1;
J=0;
while(J<=N)
{
if (I!=J)
```

```
P=P*(X(I)-X(J));
J=J+1;
}
S=0;
K=0;
while(K<=N)
{
if (I!=K)
P1=1;
J=0;
while(J<=N)
{
if (J!=I)&&(J!=K)
P1=P1*(XD-X(J));
J=J+1;
}
S=S+P1;
K=K+1;
}
DP=DP+FX(I)/P*S;
I=I+1;
}
cout<<"LA APROXIMACION DE LA DERIVADA ES:";
cout<<DP;
getch();
}
```

6.1.7. APLICACIÓNES PARA LAS ECUACIONES DIFERENCIALES ORDINARIAS

A.-ALGORITMO DEL METODO DE EULER

PROGRAMA (EULER.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>
void main()
{float X0,Y0,XF,YF,H;
int N,I;
clrscr();
cout<<"Ingresar valor X0 de la condición inicial:";
cin>>X0;
cout<<"Ingresar valor Y0 de la condición inicial:";
cin>>Y0;
cout<<"Ingresar valor XF:";
```

```
cin>>XF;
cout<<"Ingresar el número de subintervalos:";
cin>>N;
H=(XF-X0)/N;
I=1;
while(I<=N)
{
Y0=Y0+H*F(X0,Y0);
X0=X0+H;
I=I+1;
}
cout<<"EL VALOR Y0 ES:"
cout<<Y0;
getch();
}
```

B.-ALGORITMO DEL METODO DE EULER MODIFICADO

PROGRAMA (EULERMOD.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>
void main()
{float X0,Y0,XF,Y1,YF,H;
int N,I;
clrscr();
cout<<"Ingresar valor X0 de la condición inicial:";
cin>>X0;
cout<<"Ingresar valor Y0 de la condición inicial:";
cin>>Y0;
cout<<"Ingresar valor XF:";
cin>>XF;
cout<<"Ingresar el número de subintervalos:";
cin>>N;
H=(XF-X0)/N;
I=1;
while(I<=N)
{
Y1=Y0+H*F(X0,Y0);
Y0=Y0+H/2*(F(X0,Y0)+F(X0+H,Y1));
X0=X0+H;
I=I+1;
}
cout<<"EL VALOR Y0 ES:"
cout<<Y0;
getch(); }
```

C.-ALGORITMO DEL METODO DE RUNGE-KUTTA DE CUARTO ORDEN

PROGRAMA (RKUTTA.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>
void main()
{float X0,Y0,XF,YF,H;
int N,I;
clrscr();
cout<<"Ingresar valor X0 de la condición inicial:";
cin>>X0;
cout<<"Ingresar valor Y0 de la condición inicial:";
cin>>Y0;
cout<<"Ingresar valor XF:";
cin>>XF;
cout<<"Ingresar el número de subintervalos:";
cin>>N;
H=(XF-X0)/N;
I=1;
while(I<=N)
{
K1=F(X0,Y0);
K2=F(X0+H/2,Y0+H*K1/2);
K3=F(X0+H/2,Y0+H*K2/2);
K4=F(X0+H,Y0+H*K3);
Y0=Y0+H/6*(K1+2*K2+2*K3+K4);
X0=X0+H;
I=I+1;
}
cout<<"EL VALOR Y0 ES:"
cout<<Y0;
getch();
}
```

6.1.8. APLICACIÓN PARA LAS ECUACIONES DIFERENCIALES PARCIALES

A.-ALGORITMO DEL METODO EXPLÍCITO

PROGRAMA (EXPLICITO.CPP)

```
#include <bcd.h>
#include <conio.h>
#include <math.h>
void main()
{float DT,DX,T(100),ALFA,LAMBDA;
```

```
int NX,NT,XF,TF,I,J;
clrscr();
cout<<"Ingresar el número NX de puntos de la malla en el eje x:";
cin>>NX;
cout<<"Ingresar el numero NT de puntos de la malla en el eje t:";
cin>>NT;
cout<<"Ingresar valor XF longitud total del eje x:";
cin>>XF;
cout<<"Ingresar valor TF tiempo máximo:";
cin>>TF;
cout<<"Ingresar el coeficiente alfa de la derivada de segundo orden:";
cin>>ALFA;
DX=XF/(NX-1);
DT=TF/(NT-1);
LAMBDA=ALFA*DT/DX**2;
I=2;
while (I<=NX-1)
{
T(I)=CI(DX*(I-1));
I=I+1;
}
T(1)=(CI(DX)+CF1(DT))/2;
T(NX)=(CI(XF-DX)+CF2(DT))/2;
cout<<"EL VALOR DE T ES:"
cout<<T;
J=1;
while (J<=NT)
{
I=2;
while (I<=NX-1)
{
T1(I)=LAMBDA*T(I-1)+(I-2*LAMBDA)*T(I)+LAMBDA*T(I+1);
I=I+1;
}
I=2;
while (I<=NX-1)
{
T(I)=T1(I);
I=I+1;
}
T(1)=CF1(DT*J);
T(NX)=CF2(DT*J);
cout<<"EL VALOR DE T ES:"
cout<<T;
J=J+1;
getch(); }
```

VII. DISCUSIÓN

En los compiladores del Lenguaje de Programación C++, tales como: Borland C++, Microsoft C++ o Visual C++ no tienen una librería para resolver problemas de análisis numérico, la bibliografía acerca de este tema es muy escasa.

Existen software para aplicar al área del análisis numérico tales como: MATHEMATICA, MATLAB Y SCILAB que pueden resolver estos métodos numéricos.

En el caso del MATLAB se aplica con funciones ya contenidas en el paquete de software, en cambio el sistema propuesto en este trabajo de investigación está desarrollado a partir de la codificación de los algoritmos de métodos numéricos en el lenguaje de programación C++ y por consiguiente tendrán una mejor consistencia y ciclo de vida útil.

El resultado obtenido en este trabajo de investigación es un sistema computacional que tiene una librería para resolver problemas de análisis numérico.

VIII. REFERENCIAS BIBLIOGRAFICAS

[1] BURDEN, RICHARD L; FAIRES, J. DOUGLAS. ***Análisis Numérico,*** México: Thomson Learning, 7ma edición, 2002.

[2] MARON, MELVIN J; LOPEZ, ROBERT J. **Análisis Numérico,** México: Continental,1ra reimpresión, 1998.

[3] KINCAID, D.; CHENEY W. **Análisis Numérico: Las matemáticas del cálculo científico,** México: Addison-Wesley Iberoamérica, 2da edición, 1994.

[4] NIEVES, ANTONIO; DOMINGUEZ, FEDERICO C. **Métodos numéricos aplicados a la Ingeniería,** México: Continental S.A., 1ra edición, 1995.

[5] MATHEWS, J. **Numerical Methods,** México: Prentice-Hall, 1ra edición, 1992.

[6] SCHEID, FRANCIS. **Análisis Numérico,** México: McGraw-Hill, 1ª edición, 1972.

[7] NAKAMURA., CHOICHIRO. **Métodos Numéricos aplicados con Software,** México: Prentice–Hall, 1ª edición, 1992.

[8] STROUSTRUP, BJARNE. ***El lenguaje de programación C++,*** Madrid: Addison-Wesley, 3ª edición, 2002.

[9] STROUSTRUP, BJARNE. ***The C++ Programming Language,*** Madrid: Addison-Wesley, 3ª edición, 2000.

[10] STROUSTRUP, BJARNE. ***The Design and Evolution of C++,*** Madrid: Addison-Wesley, 1ª edición, 1994.

[11] COPLIEN, JAMES. ***Advanced C++: Programming Styles and Idioms,*** Madrid: Addison-Wesley, 3ª edición, 1992.

[12] MUSSER, D.R. ***Effective C++,*** Madrid: Addison – Wesley, 2ª edición, 1997.

[13] JOYANES AGUILAR, LUIS. ***C++ a su alcance,*** Madrid: Editora McGraw-Hill, 2ª edición 1994.

IX. APÉNDICE

9.1 FUNCIONES DEL LENGUAJE DE PROGRAMACIÓN C++

Son funciones predefinidas por el lenguaje de programación C++ y tienen una tarea específica para ejecutar.

FUNCIONES	ACCIÓN A EJECUTAR
clrscr()	Borra pantalla
cin	Lectura de datos
cout	Escritura de datos
if	Instrucción de control selectiva simple
If....else	Instrucción de control selectiva doble
switch	Instrucción de control selectiva múltiple
for	Instrucción de control repetitiva fija
while	Instrucción de control repetitiva condicional al inicio
do.....while	Instrucción de control repetitiva condicional al final
break	Instrucción de salto fuera del ciclo mas interno que lo contiene
continue	Instrucción de salto al final del ciclo mas interno que lo contiene
goto	Instrucción de salto hacia una posición determinada
getch()	Retención para visualizar la solución del programa

ANEXOS

9.2 CADENA DE CARACTERES

Definición 9.2.1 Una cadena de caracteres es un Arreglo Unidimensional, en el cual todos los elementos contenidos en el son de tipo carácter.

LEER UNA CADENA DE CARACTERES

Definición 9.2.2 (GETS) Lee caracteres y lo almacena en una variable especificada, a diferencia de la función cin, la función gets permite la entrada de una cadena de caracteres formada por palabras separadas por espacios en blanco sin ningún tipo de formato.

Sintaxis:

```
GETS (CADENA);
```

ESCRIBIR UNA CADENA DE CARACTERES

Definición 9.2.3 (PUTS) Escribe una cadena de caracteres en la salida estándar y reemplazar el carácter nulo de terminación de la Cadena (\0) por el carácter nueva línea (\n).

Sintaxis:

```
PUTS(CADENA);
```

ARREGLO DE CADENA DE CARACTERES

Definición 9.2.4 En un arreglo de cadenas de caracteres cada elemento es un arreglo de caracteres. Es un arreglo de 2 dimensiones de tipo char.

Sintaxis:

CHAR CADENA [Nº ELEMENTOS][LONGITUD DE LA CADENA]

FUNCIONES DE CADENAS DE CARACTERES

Definición 9.2.5 Son funciones de tipo cadena y para utilizarlas se tiene que activar la librería <STRING.H>

STRCAT._ Agrega toda la cadena2 en la cadena1 y el resultado es devuelto en la cadena1

Sintaxis:

STRCAT(CADENA1,CADENA2);

Ejemplo 9.2.1

1ra Forma: CHAR NOMBRE1 (30), NOMBRE2 (30);

GETS (NOMBRE1) ; GETS (NOMBRE2);

cout<<"La unión de las cadenas es:"<< STRCAT (NOMBRE1, NOMBRE2):

2da Forma: CHAR NOMBRE1 (30) , NOMBRE2(30);

GETS (NOMBRE1); GETS (NOMBRE2);

STRCAT (NOMBRE1, NOMBRE2);

cout<<" La unión de las cadenas es: << NOMBRE1;

STRCPY.-Esta función copia cadena2 en la cadena 1 y el resultado lo devuelve en la cadena 1

Sintaxis:

STRCPY (CADENA 1, CADENA2);

Ejemplo 9.2.2

1ra FORMA: CHAR NOMBRE1 (30), NOMBRE2 (30);

GETS (NOMBRE1); GETS (NOMBRE2);

cout <<"La Nueva cadena es :"<< STRCPY (NOMBRE1, NOMBRE2):

2da Forma: CHAR NOMBRE1 (30), NOMBRE2 (30);

GETS (NOMBRE1); GEST (NOMBRE2);

SRTCPY (NOMBRE1, NOMBRE2);

cout<<" LA NUEVA CADENA ES; "<< NOMBRE1;

STRLEN .- Esta función devuelve la longitud de la cadena

Sintaxis:

STRLEN (CADENA 1);

Ejemplo 9.2.3

CHAR NOMBRE1 (30),

GETS (NOMBRE1);

cout <<"La Longitud de la cadena es :"<< STRLEN(NOMBRE1)

STRLWR .- Convierte la Cadena de mayúsculas a minúsculas

Sintaxis:

STRLWR (CADENA 1);

Ejemplo 9.2.4

CHAR NOMBRE 1 (30),

GETS (NOMBRE1);

cout <<"En minúsculas es:" << STRLWR (NOMBRE 1)

STRUPR.- Convierte la cadena de minúsculas a mayúsculas

Sintaxis:

STRUPR (CADENA 1);

Ejemplo 9.2.5

CHAR NOMBRE1(30),

GETS (NOMBRE1);

cout <<"En mayúsculas es :"<< STRPR (NOMBRE1):

PRINCIPALES FUNCIONES PARA CONVERSION DE DATOS

Definición 9.2.6 Son funciones que sirven para convertir datos a otros tipos de datos.

ATOF .- Convierte una cadena caracteres a un valor de doble precisión.

Sintaxis:

ATOF (CADENA);

ATOL .- Convierte una cadena de caracteres aun valor entero o entero largo.

Sintaxis:

ATOL (CADENA);

ITOA .- Convierte un numero entero en uno cadena de caracteres.

Sintaxis:

ITOA (VALOR, CADENA, BASE);

Donde :

Valor - Es el valor a transformar a cadena.

Cadena - Cadena donde ira el resultado

Base - Colocar a la base en la que se transformara el número

Ejemplo 9.2.6

Realizar un programa que ingrese un numero entero y determine su base binaria utilizar la función itoa.

```
# Include <coino.h>
# Include < bcd.h>
# Include< stdio.h>
# Include <string.h>
# Include < stdlib.h>
void main()
{clrscr();
char a (15);
int n;
puts(" ingrese un número a cambiar de base:");
cin>> n;
itoa ( n, a, 16);
cout<<" en base 2 es : " <<a<<"/n";
getch() ;}
```