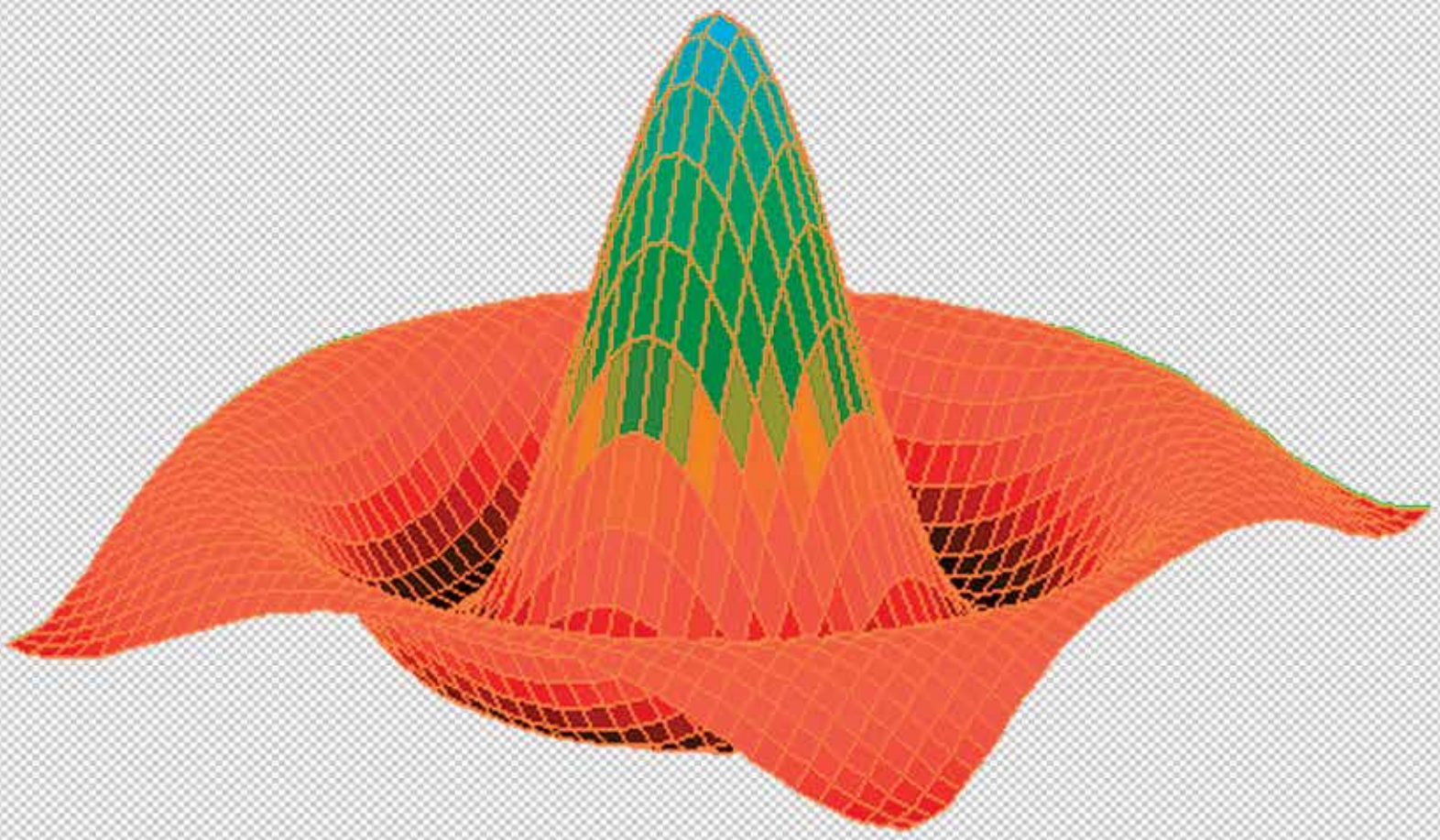
MÉTODOS NUMÉRICOS EN LOS LENGUAJES MATLAB Y MICROSOFT VISUAL C#.NET



Fausto Meneses Becerra

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

MÉTODOS NUMÉRICOS EN LOS LENGUAJES MATLAB Y MICROSOFT VISUAL C#.NET

Fausto Meneses Becerra

Métodos numéricos en los lenguajes Matlab y Microsoft Visual C#.NET
Fausto Meneses Becerra

Primera edición electrónica. Junio 2015
ISBN: 978-9978-301-77-7
Par revisor: Msc. Miguel Chavez Manosalvas; Msc. Hernán Aules Centeno

Universidad de las Fuerzas Armadas ESPE
Grab. Roque Moreira Cedeño
Rector

Publicación autorizada por:
Comisión Editorial de la Universidad de las Fuerzas Armadas ESPE

Edición y producción
David Andrade Aguirre
daa06@yahoo.es

Diseño
Pablo Zavala A.

Universidad de las Fuerzas Armadas ESPE
Av. General Rumiñahui s/n, Sangolquí, Ecuador.
htpp//www.espe.edu.ec

Prologo

Este libro ha sido diseñado para los estudiantes y profesionales, especialmente de las áreas de la Ingeniería, Economía, Ciencias Administrativas, Ciencias Exactas, entre otras que tengan conocimientos básicos de Matlab y/o sólidos en Microsoft Visual C#.NET.

Para el caso de Microsoft Visual C#.NET., se ha puesto énfasis en la reutilización de código con el empleo de una librería común de clases.

No pretende cubrir temas nuevos que vienen con Microsoft Visual Studio .NET como:

Librerías .NET (que trabajan con Framework)
ADO.NET en Visual C # .NET.
Aplicaciones Windows Form.NET
Bibliotecas de controles.NET
Servicio de Windows.NET
Servicio de Web.NET
Debug and Release.NET

Pues se considera que es materia de otra obra con temas avanzados.

Esta obra, se encuentra formada por ocho capítulos, y dos apéndices, estructurados de la siguiente manera:

En el capítulo I, se consideran temas relacionados con teoría y propagación de errores, iniciando con definiciones básicas, se pasa a ilustrar con ejemplos para luego complementar con programas en Matlab y Microsoft Visual C#.NET mostrando los resultados de la ejecución de los programas.

En el capítulo II, se describen métodos gráficos y numéricos para calcular raíces reales de funciones trascendentes por los métodos de la Bisección, Aproximaciones Sucesivas, Aproximaciones Sucesivas Modificado, Régula - Falsi, Secante, Newton - Raphson y Muller. En la siguiente sección se presenta el Método de Horner para calcular raíces reales de funciones. En la última sección de este capítulo se presenta el Método de Newton - Bairstow para calcular raíces reales y complejas de funciones polinómicas de la forma

F(x) = 0. Todos los métodos se describen con un análisis matemático, algoritmo en seudocódigo, ejemplos ilustrativos e implementación en Matlab y Microsoft Visual C#.NET mostrando los resultados de la ejecución de los programas.

En el capítulo III, se desarrollan métodos para resolver sistemas de ecuaciones lineales y simultáneas por Gauss, Gauss – Jordan, Gauss – Seidel y Crout. En la segunda sección se propone un método para calcular la inversa de una matriz a través de la aplicación de la resolución de una secuencia de sistemas de ecuaciones lineales y simultáneas por Crout. En la tercera sección se presenta una variante del Método de Crout para resolver sistemas simétricos de ecuaciones lineales y simultáneas. En la cuarta sección se presenta otra variante del Método de Crout para resolver sistemas simétricos en banda de ecuaciones lineales y simultáneas. En la sección final de este capítulo se presenta otra variante del Método de Crout para resolver sistemas simétricos Sky - Line de ecuaciones lineales y simultáneas. Todos los métodos se describen con un análisis matemático, algoritmo en seudocódigo, ejemplos ilustrativos e implementación en Matlab y Microsoft Visual C#.NET mostrando los resultados de la ejecución de los programas.

En el capítulo IV, se describen algoritmos para interpolar por los métodos de Interpolación Lineal, Polinomios de Lagrange y Newton mediante diferencias divididas. Todos los métodos se describen con un análisis matemático, algoritmo en seudocódigo, ejemplos ilustrativos e implementación en Matlab y Microsoft Visual C#.NET mostrando los resultados de la ejecución de los programas.

En el capítulo V, se aplican los algoritmos descritos en los capítulos anteriores para ajustar una muestra estadística a una función polinómica mediante los métodos de los Mínimos Cuadrados; Polinomios de Lagrange y Newton, mostrando el diagrama de dispersión y el gráfico de la familia de polinomios procesados. Los métodos se describen con un análisis matemático, algoritmo en seudocódigo, ejemplos ilustrativos e implementación en Matlab y Microsoft Visual C#.NET mostrando los resultados de la ejecución de los programas.

En el capítulo VI, se describen métodos para calcular derivadas de primer orden y de orden superior en un punto dado, especificando la longitud

de paso; cuando la función es discreta (por ejemplo una muestra de puntos), se utiliza interpolación. Aplicando las reglas de derivación se calcula la derivada de cualquier orden para polinomios obteniendo adicionalmente la expresión simbólica del polinomio resultante. Se realiza un análisis de los errores absolutos y relativos de los métodos descritos. Todos los métodos se describen con un análisis matemático, ejemplos ilustrativos e implementación en Matlab empleando cálculo simbólico y Microsoft Visual C#.NET mostrando los resultados de la ejecución de los programas.

En el capítulo VII, se describen métodos para calcular integrales definidas aplicando Series de Taylor y las reglas de los rectángulos, trapecios, Simpson, Tres Octavos y Dos Cuarentaycincoavos; se aplica la regla de integración en funciones polinómicas para obtener integrales de cualquier dimensión, con la correspondiente expresión simbólica, especificando en todos los casos el intervalo de integración; cuando la función es discreta (por ejemplo una muestra de puntos), se utiliza interpolación. Se realiza un análisis de los errores relativos de los métodos descritos. Todos los métodos se describen con un análisis matemático, ejemplos ilustrativos e implementación en Matlab empleando cálculo simbólico y Microsoft Visual C#.NET mostrando los resultados de la ejecución de los programas.

En el capítulo VIII, se desarrollan algoritmos para resolver ecuaciones diferenciales aplicando los métodos de Euler, Taylor, Euler Modificado, Runge – Kutta de orden dos y Runge – Kutta de orden cuatro. Se realiza un análisis de los errores relativos de los métodos descritos. Todos los métodos se describen con el algoritmo en pseudocódigo, ejemplos ilustrativos e implementación en Matlab empleando cálculo simbólico y Microsoft Visual C#.NET mostrando los resultados de la ejecución de los programas.

El apéndice A, está dirigido a los lectores con pocos conocimientos de Matlab, en el se plantean algunos ejemplos del uso del control del flujo en Matlab y de la graficación de sólidos de revolución. En algunos de estos ejercicios se describe con un análisis matemático, y todos con la implementación en Matlab mostrando los resultados de la ejecución de los programas.

El apéndice B, está dirigido a los lectores con algunos conocimientos de Microsoft Visual C#.NET, en el se plantean algunos ejemplos del uso del control del flujo en Microsoft Visual C#.NET, casi todos los descritos en el

apéndice A. En todos los ejemplos se los implementa en Microsoft Visual C#.NET mostrando los resultados de la ejecución de los programas. En la siguiente sección se describe el código fuente de la librería común de clases, misma que es reutilizada en la mayoría de programas escritos en Microsoft Visual C#.NET. En la sección final de este apéndice se describen los diagramas UML de clases y de secuencia cuyo diseño permite implantar los métodos para calcular raíces reales de funciones trascendentes y polinómicas.

Se agradece a la Facultad de Ingeniería de la PUCE, Institución que ha hecho posible la realización de una parte de la primera edición de esta obra. A las autoridades de la ESPE que han hecho posible la realización de la primera edición de esta obra

De una manera particular:

A mi esposa Nancy y a mis hijos Lenín, Carolina, Milton y Mikaela y a mi querida nieta Pía Marcel; quienes me han incentivado para seguir adelante y culminar con este trabajo.

Al Dr. Manuel Corrales S.J., Rector de la PUCE; al Ing. Galo Cevallos, Ex Director General Académico; Al Ing. Diego Andrade, Decano de la Facultad de Ingeniería; Al Ing. Francisco Rodríguez, Subdecano de la Facultad de Ingeniería y a otros distinguidos compañeros de la Pontificia Universidad Católica del Ecuador.

Al Grab. Carlos Rodríguez, Rector de la ESPE; al Vicerector Académico de la ESPE; al Ing. Walter Fuertes, Ph. D. Director de Postgrados; al Ing. Luis Guerra, Msc, Director de la Carrera de Ingeniería del Software, Sede Latacunga; al Crnl (SP) Jorge Vergara, Director del Departamento de Ciencias Exactas; al Ing. Ignacio Dávila por sus valiosos comentarios en la revisión de esta obra, al Mat. Paúl Medina Ph. D., por haber facilitado importante bibliografía relacionada con esta obra; al Ing. Marcelo Romo, Msc, por haber facilitado su valioso artículo para la resolución de sistemas de ecuaciones lineales y simultáneas del tipo SKY-LINE.

A mis apreciados estudiantes de Métodos Numéricos, por sus valiosos aportes a la realización de esta obra.

1

NÚMEROS APROXIMADOS Y ERRORES

Generalidades

Número Aproximado .- Se dice que un número es aproximado cuando difiere ligeramente del número exacto.
Sean: R = número exacto y
r = número aproximado de R
Entonces se presentan dos casos:
a) cuando r > R, r es aproximado por exceso, y
b) cuando r < R, r es aproximado por defecto.

Error de un número aproximado.-Es la diferencia que existe entre el número exacto y el número aproximado, es decir:
Error de r = Er = R - r (1.1).

Error absoluto de un número aproximado.-Es igual al valor absoluto del error de un número aproximado.
Matemáticamente se define así: Error absoluto de r = EAr = $|R - r|$ (1.2).

Existen dos casos en los números exactos:
a) R es conocido.
b) R es desconocido.
Para el caso (a), el error absoluto de un número aproximado, se lo puede determinar aplicando la fórmula (1.2).
En la práctica se presenta el caso (b), por consiguiente debe introducirse el concepto de un error superior estimado, denominado:
Cota de error absoluto.-Que se define como un número cualquiera Cr no menor que el error absoluto de un número aproximado,
Entonces $Cr \geq EAr$ (1.3).
De (1.2) se puede deducir que: $R = r \pm Er$ (1.4).
Ejemplos 1.1:
a) si se conoce que $A = \sqrt{2}$ tiene una variación de $1.41 \leq A \leq 1.42$ entonces
Ea = 1.42 - 1.41 = 0.01
b) si se sabe que $A = \sqrt{2}$ tiene una variación de $1.411 \leq A \leq 1.414$ entonces
Ea = 1.414 - 1.411 = 0.003
Por consiguiente, se puede tener un número infinito de cotas de error absoluto, se debe seleccionar entonces el menor de los valores, según las circunstancias.
Cuando se escribe un número aproximado, procedente de una medida, es común dar su cota de error absoluto, por ejemplo si la longitud de un segmento es l = 235 cm

con un error de 0.5 cm, se escribe L = 235 cm ± 0.5 cm. En este caso la cota de error absoluto es CI = 0.5 cm y la magnitud exacta de la longitud del segmento cae dentro del margen 234.5 cm ≤ L ≤ 235.5 cm
La cota de error absoluto no indica las características de la calidad de una medida; suponiendo que al medir las longitudes de dos segmentos se tiene L1 = 80.8 cm ± 0.2 cm y L2 = 5.2 cm ±
0.2 cm, independientemente de que los dos errores coincidan, la primera es mejor que la segunda medida, por consiguiente, debe introducirse el criterio de:

Error relativo.-Que es la relación entre el error absoluto y el módulo (valor absoluto) del número exacto: δr = EAr / │R│ (1.5).
Aplicando el mismo criterio de la cota de error absoluto, se define la:
Cota de Error relativo.-De un número aproximado como cualquier CRr no menor que el error relativo de dicho número. Por consiguiente δr ≤ CRr (1.6), en la práctica se utiliza frecuentemente:
Er = │r│CRr (1.7).
Ejemplo:
Conocida la cota de error relativo, determinar los límites del número exacto.
Solución:
De (1.4) R = r ± Er, reemplazando en esta ecuación 1.7 se obtiene: R = r ± rCRr, por consiguiente R = r(1 ± CRr) (1.8) o también r(1 - CRr) ≤ R ≤ r(1 + CRr) (1.9).
Ejemplo 1.2:
Dado el número aproximado a = 42.53 y la cota de error relativo
CRa = 2 por mil, averiguar entre que límites se encuentra el número exacto.
Solución:
Aplicando (1.9) y reemplazando valores:
42.53(1 - 0.002) ≤ A ≤ 42.53(1 + 0.002); 42.53*0.998 ≤ A ≤ 42.53*1.002; entonces:
42.445 ≤ A ≤ 42.615

Principales fuentes de errores:

Las fuentes más importantes de errores que se presentan son:
1. Errores del problema o del método.-El planteamiento matemático raramente ofrece una presentación exacta de los fenómenos reales, pues en la mayoría de los casos son modelos idealizados. Al estudiar los fenómenos de la naturaleza nos vemos forzados, por regla general a aceptar ciertas condiciones que simplifican el problema, lo cual produce a errores que se les conoce como del problema. Asi mismo ocurre a veces que es difícil o imposible resolver un problema formulado en forma exacta, por lo que se vuelve necesario reemplazarlo por una solución aproximada, la cual debe ofrecer el mismo resultado, esto nuevamente ocasiona errores que se los conoce con el nombre de errores del método.
2. Error Residual.-Se presentan al realizar el cálculo de funciones a través del uso de series matemáticas, puesto que un proceso infinito no puede completarse en un número finito de pasos, entonces se debe forzar a pararlo en cierto término de la

secuencia y considerarla como una aproximación de la solución requerida, teneniendo como consecuencia en error que se define como residual.

3. Error Inicial.-Se producen al efectuar cálculos en los que intervienen ciertos valores que ya tienen errores, tales como por ejemplo las constantes físicas, la constante π, mediciones, etc.

4. Errores asociados a un sistema de numeración.-Cuando se representan números racionales en el sistema decimal, o cualquier otro, puede haber una serie infinita de dígitos, resulta evidente que en los cálculos, sólo se emplea una serie finita.

Ejemplo 1.3:
al representar 2/3 = 0.66667, se comete un error = 0.00001.

5. Errores de operación.-Debido a las operaciones en que se consideran números aproximados, al realizarse cálculos intermedios, lo cual repercute en el resultado final.

En ciertos problemas específicos, no existen errores y en otros ejercen efecto despreciable; mas, por lo general, en un análisis completo, se deben tener en cuenta todos los tipos de errores; sin embargo los más comunes son los de operación y los del método.

6. Errores de propagación.- Son los errores en los resultados de un proceso provocados por los errores en las entradas.

Digitos significativos.

Cualquier número positivo *a* puede representarse en forma decimal con una cantidad finita o infinita de dígitos:

$$a = \alpha_m 10^m + \alpha_{m-1} 10^{m-1} + \alpha_{m-2} 10^{m-2} + \ldots \alpha_{m-n+1} 10^{m-n+1} + \ldots \quad (1.10)$$

donde los α_i son los dígitos del sistema numérico decimal,
$\alpha_m \neq 0$, es el dígito más significativo, m es la potencia más elevada de diez del número *a* y n el número de dígitos.

Ejemplo 1.4:

$$14325.98 = 1*10^4 + 4*10^3 + 3*10^2 + 2*10^1 + 5*10^0 + 9*10^{-1} + 8*10^{-2}$$

Se denomina dígito significativo de un número aproximado a cualquier dígito diferente de cero o cualquier cero situado entre dígitos significativos; son además aquellos ceros que se utilizan para indicar posiciones a la derecha de un número o de otro dígito significativo. Todos aquellos ceros que están a la izquierda del número y que solamente se indican para la posición del punto decimal no se deben considerar como dígitos significativos.

Ejemplos 1.5:
645.8004 tiene 7 dígitos significativos.
0.0000038 tiene 2 dígitos significativos.
78.4000 tiene 6 dígitos significativos.
78.4 tiene 3 dígitos significativos.

$9.4500*10^{-2}$ tiene 5 dígitos significativos.
$9.45*10^{-2}$ tiene 3 dígitos significativos.

Dígitos Significativos Exactos.-Se dice que los n primeros dígitos significativos de un número aproximado son exactos si la cota de error absoluto de dicho número no sobrepasa la media unidad situada en la posición n-ésima, contando de izquierda a derecha. Por lo tanto si *a* es el número aproximado de A, entonces
$Ca = |A-a| \leq 0.5 * 10^{m-n+1}$ (1.11), entonces de 1.10 se tiene que α_m, α_{m-1}, α_{m-2}, ..., α_{m-n+1} son los dígitos significativos de *a*.
Ejemplos 1.6:
a) Si A = 43.9996 y a = 44.0000, entonces *a* es una aproximación con cinco dígitos significativos exactos, puesto que:
$Ca = |43.9996-44.0000| = 0.4*10^{-3} < 0.5 * 10^{1-n+1}$, por tanto
$-3 = 1-n+1$, entonces n = 5.
b) Si se sabe que 247.6125 tiene cuatro dígitos significativos exactos, calcular la cota de error absoluto.
Solución:
De (1.11) $Ca \leq 0.5 * 10^{2-4+1}$, por tanto $Ca \leq 0.05$, tomando el máximo valor Ca= 0.05.

c) Si el número aproximado es 0.023474 con una cota de error absoluto de 0.00005, calcular la cantidad de dígitos significativos exactos y determinar cuáles son.
Solución:
De (1.11) $0.5*10^{-4} \leq 0.5 * 10^{-2-n+1}$, por tanto $-4 = -2-n+1$, entonces n = 3, por lo que los dígitos significativos exactos son 2,3,4.

d) Si el número aproximado es 74.6139 con una cota de error absoluto de 0.08, determinar cuáles son los dígitos significativos exactos de ese número.
Solución:
De (1.11) $0.8*10^{-1} < 0.5*10^{0} \leq 0.5*10^{1-n+1}$, por consiguiente $0 = 1-n+1$, entonces n = 2 y los dígitos significativos exactos son: el 7 y el 4.

Errores en las operaciones:

Errores en la suma.-Sea $U = X_1 + X_2 + X_3 + ... + X_n$ y $u = x_1 + x_2 + x_3 + ... + x_n$
Por definición $EAu = |U-u|$
Efectuando la operación U - u y agrupando términos:
$U - u = (X_1-x_1) + (X_2-x_2) + (X_3-x_3) + ... + (X_n-x_n)$
tomando el valor absoluto en ambos miembros:
$|U-u| = |(X_1-x_1) + (X_2-x_2) + (X_3-x_3) + ... + (X_n-x_n)|$
De acuerdo a la desigualdad triangular:
$|U-u| \leq |(X_1-x_1)| + |(X_2-x_2)| + |(X_3-x_3)| + ... + |(X_n-x_n)|$
Tomando el máximo valor: $EAu = EAx_1 + EAx_2 + EAx_3 + ... + EAx_n$ (1.12)

Errores en la resta.-El criterio de calcular errores a la suma, se extiende a la resta. Sea $u = x_1 - x_2 = x_1 + (-x_2)$ y $x_3 = -x_2$ entonces: $u = x_1 + x_3$, aplicando la fórmula de errores en la suma: $EAu = EAx_1 + EAx_3$; puesto que $x_3 = -x_2 : Ex_3 = EAx_2$, entonces $EAu = EAx_1 + EAx_3$ (1.13)

Ejemplos 1.7:
a) Sea $f(x) = e^x sen(x^2 – 3x + 4)$. Para X = 3.141593. Calcular el error absoluto y relativo de f(x) que se comete al truncar a X a dos decimales.
Solución:
De acuerdo a las condiciones del problema x = 3.14;
$f(X) = f(3.141593) = e^{3.141593} sen(3.141593^2 - 3*3.141593 + 4) = -22.31731936$
$f(x) = f(3.14) = e^{3.14} sen(3.14^2 - 3*3.14 + 4) = -22.24956073$
$EAf(x) = |-22.31731936 - (-22.24956073)| = 0.067758635$
$\delta f(x) = EAf(x) / \left|f(X)\right| = 0.067758635 / \left|-22.31731936\right| = 0.003036146 = 0.3036\ \%$

b) Calcular el error absoluto y relativo de f(x) = tan(x) que se comete al tomar los cuatro primeros términos de la serie de Taylor de sen(x) y cos(x); tan(x) = sen(x) / cos(x); para
x = PI/8 = 0.392699082
Solución:
La serie de Taylor de la función seno es $sen(x) = x - x^3/3! + x^5/5! - x^7/7! + ...$
El detalle del cálculo de sen(PI/8) para los primeros cuatro términos, se detalla en la siguiente tabla:

# TERMINO	TERMINO	SENO
1	0.392699082	0.392699082
2	-0.010093189	0.382605893
3	0.000077825	0.382683718
4	-0.000000286	0.382683432

La serie de Taylor de la función coseno es $\cos(x) = 1 - x^2/2! + x^4/4! - x^6/6! + ...$
El detalle del cálculo de cos(PI/8) para los primeros cuatro términos, se detalla en la siguiente tabla:

# TERMINO	TERMINO	COSENO
1	1	1
2	-0.077106284	0.922893716
3	0.000990897	0.923884612
4	-0.000005094	0.923879519

Considerando los cuatro primeros términos:
tan(PI/8) = sen(PI/8)/cos(PI/8) = 0.382683432 / 0.923879519 = 0.414213568
TAN(PI/8) = 0.414213562
$EAtan(x) = |0.414213562 - 0.414213568| = 6e\text{-}9$
$\delta tan(x) = EAtan(x) / \left|TAN(x)\right| = 6e\text{-}9 / 0.414213562 = 1.36e\text{-}8 = 1.36e\text{-}6\ \%$

c) Desarrollar una aplicación Matlab y otra en Visual C#, para que parametrice el ejemplo anterior, ingresando el ángulo en radianes x; y el número de términos; la aplicación debe desplegar como resultados: sen(x); cos(x); tan(x), EAtan(x)
Solución:

Programas en MATLAB y VISUAL C#:

Para cada lenguaje, se elabora un solo programa, desde el cual se ingresan el ángulo en grados y el número de términos. El programa convierte el ángulo a radianes y calcula y despliega el seno, coseno, tangente, tangente exacta y los errores absoluto y relativo.

Archivo fuente en MATLAB:

```
% ErroresTangente.m
% Calculo de errores de la tangente usando series de Taylor.
while 1
  ang = input('Ingrese el ángulo en grados: ');
  % Validación:
  nTer = input('Ingrese el número de términos (salir < 1 o > 12: ');
  if (nTer < 1 | nTer > 12) break; end;
  % Conversión de grados a radianes:
  x = pi * ang / 180;
  % Cálculo del seno mediante la serie de Taylor:
  senx = 0;
  aux = 1;
  signo = 1;
  for i = 1 : nTer
     fac = 1;
     for j = 2 : aux
        fac = fac * j;
     end;
     senx = senx + signo * x ^ aux / fac;
     signo = -signo;
     aux =  aux + 2;
  end;
  % Cálculo del coseno mediante la serie de Taylor:
  cosx = 0;
```

```
  aux = 0;
  signo = 1;
  for i = 1 : nTer
     fac = 1;
     for j = 2 : aux
        fac = fac * j;
     end;
     cosx = cosx + signo * x ^ aux / fac;
     signo = -signo;
     aux =  aux + 2;
  end;
  % Calcular las tangentes aproximada y exacta:
  tanx = senx / cosx;
  TAN = tan(x);
  % Calcular los errores:
  erAbs = abs(TAN - tanx);
  erRel = erAbs / abs(TAN);
  % Desplegar resultados:
  disp(sprintf('Angulo (radianes): %f', x));
  disp(sprintf('Tangente (exacta): %f', TAN));
  disp(sprintf('Seno: %f', senx));
  disp(sprintf('Coseno: %f', cosx));
  disp(sprintf('Tangente (aproximada): %f', tanx));
  disp(sprintf('Error absoluto: %f', erAbs));
  disp(sprintf('Error relativo: %f\n', erRel));
end;
```

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa:

Este libro no pretende, incorporar un curso de Matlab o Visual C#, sin embargo otras aplicaciones de estas herramientas de desarrollo, se las describe más adelante y en el Apéndice A para Matlab; Apéndice B para Visual C#.

```
Command Window

  To get started, select MATLAB Help or Demos from the Help menu.

Ingrese el ángulo en grados: 22.5
Ingrese el número de términos (salir < 1 o > 12: 4
Angulo (radianes): 0.392699
Tangente (exacta): 0.414214
Seno: 0.382683
Coseno: 0.923880
Tangente (aproximada): 0.414214
Error absoluto: 0.000000
Error relativo: 0.000000

Ingrese el ángulo en grados: 60
Ingrese el número de términos (salir < 1 o > 12: 3
Angulo (radianes): 1.047198
Tangente (exacta): 1.732051
Seno: 0.866295
Coseno: 0.501796
Tangente (aproximada): 1.726389
Error absoluto: 0.005662
Error relativo: 0.003269

Ingrese el ángulo en grados:
```

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 1.1.

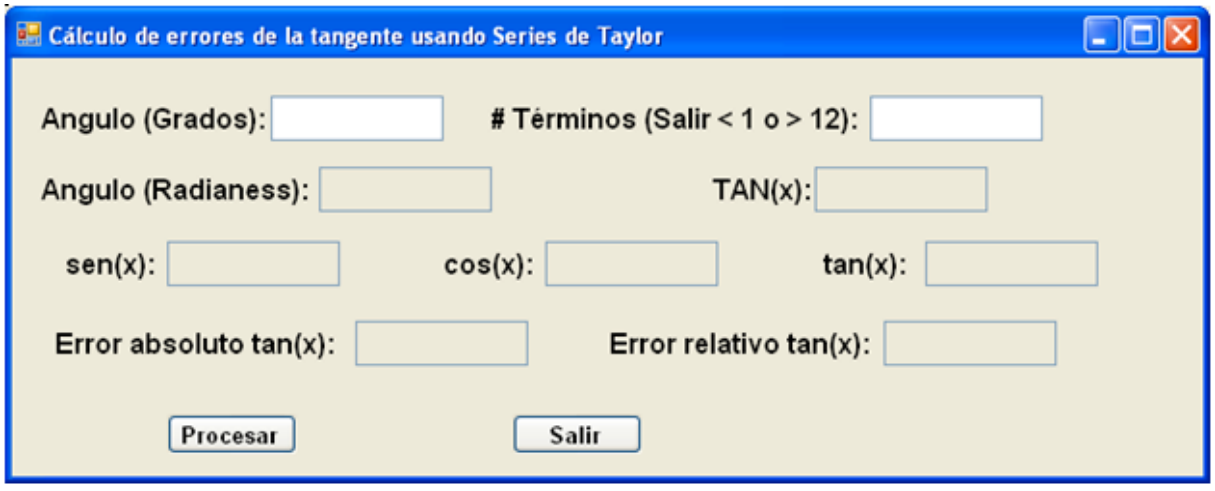


Figura 1.1 Ventana para cálculo de errores en Visual C#

Como se puede apreciar, existen nueve cuadros de edición, mismos que permitirán ingresar los datos; y presentar los resultados; y dos botones, mismos que tienen la funcionalidad:

- Procesar: Ejecuta el método para calcular los errores y desplegar los resultados. Y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Botón Procesar:

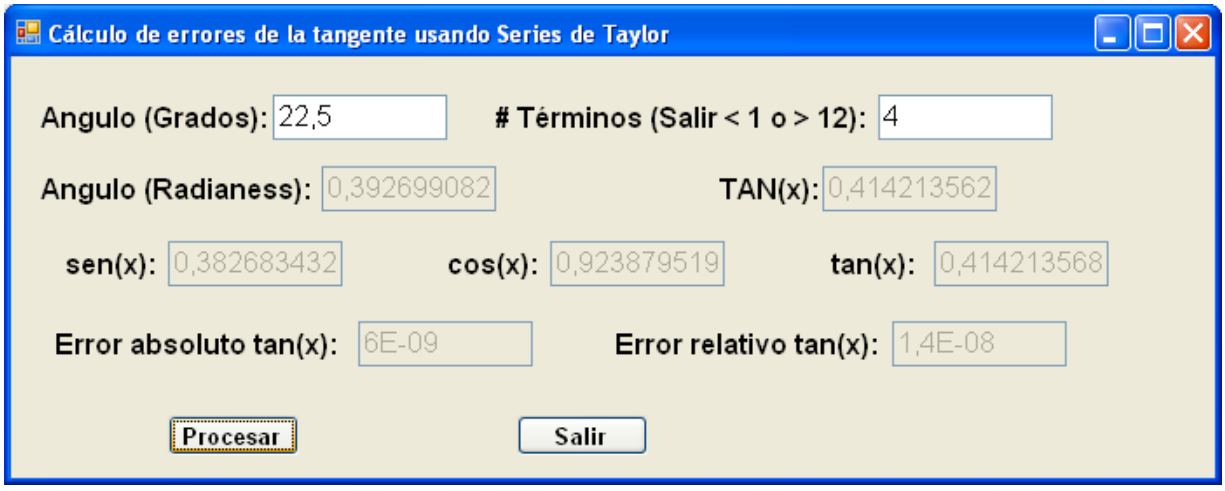
```
private void b_procesar_Click(object sender, EventArgs e)
{
  // Declaracion de variables
  double ang, x, sen = 0, cos = 0, tan, TAN, erAbs, erRel, fac;
  int nTer, signo = 1, aux, i, j;
  try
  {
    // Transferencia de captura de datos a variables:
    ang = double.Parse(txtAngGrad.Text);
    nTer = int.Parse(txtNTer.Text);
    // Validación:
    if (nTer < 1 || nTer > 12) return;
    // Conversión de grados a radianes:
    x = Math.PI * ang / 180;
    // Cálculo del seno mediante la serie de Taylor:
    aux = 1;
    for (i = 1; i <= nTer; i ++) {
      fac = 1;
      for (j = 2; j <= aux; j ++) fac *= j;
      sen += signo * Math.Pow(x, aux) / fac;
      signo = -signo;
      aux += 2;
    }
    // Cálculo del coseno mediante la serie de Taylor:
    aux = 0;
    signo = 1;
    for (i = 1; i <= nTer; i ++) {
       fac = 1;
       for (j = 2; j <= aux; j++) fac *= j;
       cos += signo * Math.Pow(x, aux) / fac;
       signo = -signo;
       aux += 2;
    }
    // Calcular las tangentes aproximada y exacta:
    tan = sen / cos;
    TAN = Math.Tan(x);
    // Calcular los errores:
    erAbs = Math.Abs(TAN - tan);
    erRel = erAbs / Math.Abs(TAN);
    // Transferir variables a pantalla:
    txtAngRad.Text = Math.Round(x, 9).ToString();
    txtTanExacto.Text = Math.Round(TAN, 9).ToString();
```

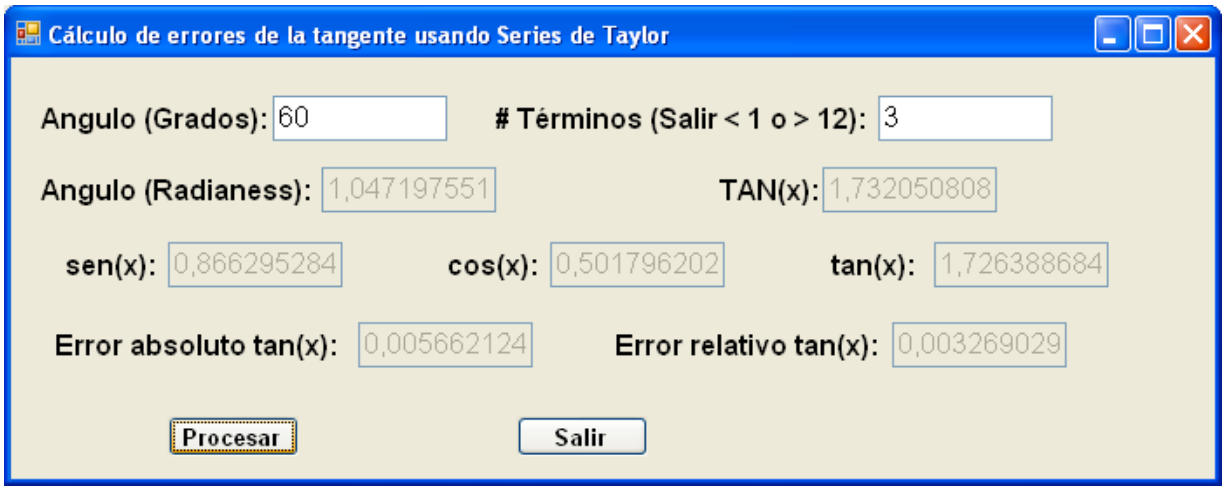
```
        txtSen.Text = Math.Round(sen, 9).ToString();
        txtCos.Text = Math.Round(cos, 9).ToString();
        txtTan.Text = Math.Round(tan, 9).ToString();
        txtErAbs.Text = Math.Round(erAbs, 9).ToString();
        txtErRel.Text = Math.Round(erRel, 9).ToString();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Botón Salir:

```
private void cb_salir_Click(object sender, EventArgs e)
{
    this.Close();
    this.Dispose();
}
```

Se presenta a continuación los resultados que produce la ejecución de este programa:




Cálculo de errores de la tangente usando Series de Taylor
Angulo (Grados): 60
Términos (Salir < 1 o > 12): 3
Angulo (Radianess): 1,047197551
TAN(x): 1,732050808
sen(x): 0,866295284
cos(x): 0,501796202
tan(x): 1,726388684
Error absoluto tan(x): 0,005662124
Error relativo tan(x): 0,003269029
Procesar
Salir

EJERCICIOS DEL CAPITULO I

1.-Encontrar la cota de error absoluto de los siguientes números, conociendo que tienen todos sus dígitos significativos exactos:
a) 0.00825 b) 26.1245 c) $7.23*10^{-5}$ d) 0.0023000

2,- Sea $f(x) = 3x^4 + 6x^3 - 2x^2 – 5x + 2$. Para X = 5.03. Calcular el error absoluto y relativo de f(x) que se comete al truncar a X a los decimales.

3.- La función arco seno, se encuentra definida por la siguiente serie de Taylor:
$\text{sen}^{-1}(x) = x + (x^3/3)(1/2) + (x^5/5)(1*3)/(2*4) + (x^7/7)(1*3*5)/(2*4*6) + \ldots$
Calcular el error absoluto y relativo cuando se toman los cuatro primeros términos de la serie.

4.- Implementar un programa en Matlab y otro en Visual C# que permita ingresar el argumento x y el número de términos y que calcule y despliegue por pantalla el arco seno de la serie de Taylor y de la correspondiente función de biblioteca; además de los errores absoluto y relativo.

2

CÁLCULOS DE RAICES REALES DE ECUACIONES DE LA FORMA F (X) = 0

Generalidades

Este capítulo se centra en la localización de los ceros de funciones en el plano cartesiano.
Ejemplos de ecuaciones de este tipo son:

1) $A*X + B = 0$
2) $A*X^2 + B*X + C = 0$
3) $A_0*X^n + A_1*X^{n-1} + A_2*X^{n-2} + \ldots + A_{n-1}*X + A_n = 0$
4) $X^2*\log_{10}(2*X) - 8 = 0$
5) $X^5 * \cos 2*X + e^x * \text{sen } X + 3 = 0$

Dependiendo del caso, se puede aplicar un método para resolver el problema, tales métodos pueden ser exactos o algebraicos, aproximados (gráficos o numéricos), etc.
Así se tiene que para resolver la ecuación del ejemplo 1, bastará con despejar X de esta ecuación, con lo que X = -B/A. En el ejemplo 2, se aplica la fórmula para el cálculo de raíces de ecuaciones cuadráticas, con lo que

$$X = \frac{-B \pm \sqrt{B^2 - 4AC}}{2A}$$

En el caso de los otros ejemplos, resulta difícil o casi imposible encontrar las raíces reales, aplicando un método exacto, por lo que se recurren a métodos aproximados, los cuales se analizan ampliamente en este capítulo.

Métodos Gráficos:

Se revisa a continuación, un método gráfico, el cual consiste en hacer Y = F(X), seguidamente se elabora una tabla, dando valores en forma ascendente a la variable independiente X, paralelamente se calcula el valor Y. Este grupo de pares ordenados, se lo representa en un sistema de coordenadas rectangulares y finalmente se unen tales puntos, generándose un diagrama de dispersión. La intersección entre la función F(X) y el eje horizontal, constituyen los ceros de la función o las raíces reales de la ecuación F(X) = 0.

Ejemplo 2.2:

Dada la ecuación: $X^2*\log_{10}(2*X) - 8 = 0$, encontrar una raíz real, utilizando un método gráfico.
Solución:
Se hace $Y = F(X) = X^2*\log_{10}(2*X) - 8 = 0$ (1)
Luego se da un rango de valores para la variable independiente X y aplicando la ecuación 1, se calcula los valores de Y, formándose la tabla:

X	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	5.5
Y = F(X)	-8.0	-7.7	-6.9	-5.6	-3.6	-1.0	2.4	6.4	11.3	17.0	23.5

Este conjunto de puntos se los representa en el plano cartesiano, formándose el siguiente diagrama de dispersión:

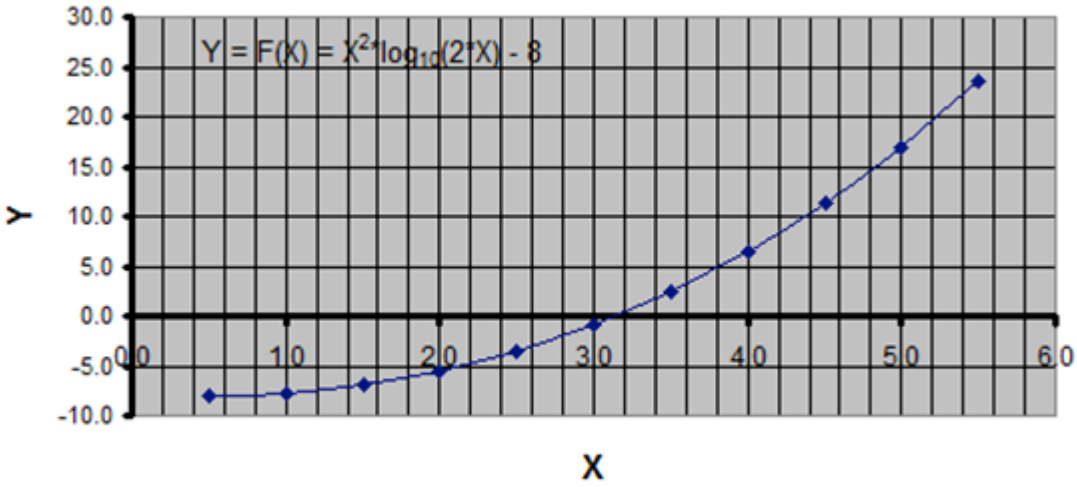


Figura 2.1, Método gráfico para calcular raíces reales.

Como puede observarse, la función F(X), corta el eje horizontal, en el rango de las abscisas 3 y 3.5, lo que significa que la raíz es aproximadamente 3.2.

Otro método gráfico consiste en transformar la ecuación F(X) = 0 en f1(X) = f2(X) y haciendo Y1 = f1(X), Y2 = f2(X), se elabora una tabla, asignando un grupo de valores a X, calculándose Y1 y Y2. Los puntos de esta tabla, se los representa en un diagrama de dispersión, en el cual se dibujan las funciones f1(X) y f2(X). Las intersecciones de éstas representan las raíces reales de la ecuación F(X) = 0.

Ejemplo 2.3:

Dada la ecuación: $X^2*\log_{10}(2*X) - 8 = 0$, encontrar una raíz real, utilizando el método gráfico alternativo.

Solución:
Se hace $8/X = X*\log_{10}(2*X)$ (2)
Entonces Y1 = f1(X) = 8/X, Y2 = f2(X) = $X*\log_{10}(2*X)$
Asignando a X los valores del ejemplo anterior, se tiene la tabla:

X	0.50	1.00	1.50	2.00	2.50	3.00	3.50	4.00	4.50	5.00	5.50
y1 = f1(X)	16.00	8.00	5.33	4.00	3.20	2.67	2.29	2.00	1.78	1.60	1.45
y2 = f2(X)	0.00	0.30	0.72	1.20	1.75	2.33	2.96	3.61	4.29	5.00	5.73

Los puntos que representan a las dos funciones, se los lleva al sistema de ejes de coordenadas en el plano de una manera independiente, obteniéndose así el diagrama:

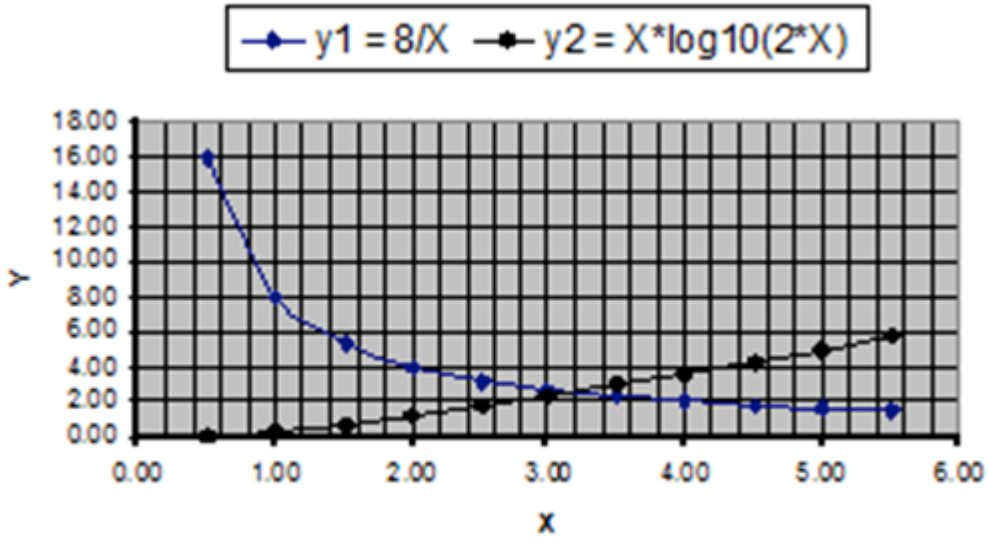


Figura 2.2, Método alternativo gráfico para calcular raíces reales.

Al observar el diagrama de dispersión, se tiene que la intersección de las funciones f1(X) y f2(X), se encuentra en la abscisa 3.2 aproximadamente.

Los métodos gráficos son bastante engorrosos de implementar, aparte de que las raíces producen un error considerable en la determinación de las mismas, por lo que se descartan y se procede a analizar seguidamente los métodos numéricos.

Métodos Numéricos:

Estos involucran la implementación de un algoritmo y se separan en dos fases:

- Localización del cambio de signo de la función F(X), y

- Afinamiento de los valores de las abscisas (XI y XD), entre las que se encuentra la raíz real.

Primera fase: Localización del cambio de signo de la función F(X)

Consiste en tomar un intervalo cerrado de valores para las abscisas y para dos valores consecutivos, se calculan las respectivas ordenadas, luego se establece el producto entre dichos valores, mismo que si es menor o igual a cero entonces se ha localizado la raíz, de lo contrario, se toma el siguiente par de valores consecutivos en las ordenadas y así sucesivamente hasta alcanzar el límite superior del intervalo.

Algoritmo para la localización del cambio de signo de la función F(X):

Entradas: F(X), LIX , LSX, XD.
Donde F(X) es la función continua en el intervalo [LIX, LSX]; LIX y LSX, son los límites inferior y superior, respectivamente, del intervalo en las abscisas, entre los que pueden encontrarse un grupo de raíces reales. DX incremento de variación de las abscisas.
Salidas: XI, XD, P
Donde XI y XD son las abscisas izquierda y derecha, respectivamente, entre las que puede estar una raíz real. P = F(XI)*F(XD) = estado que regresa el algoritmo, luego de su procesamiento y puede tomar los valores: menor que cero indica que efectivamente entre XI y XD por lo menos existe una raíz real; igual a cero, significa que XI o XD es una raíz real exacta o mayor que cero implica que en ese intervalo no existen raíces reales.
Se describen en seguida la secuencia de pasos:

a) XI = LIX
b) XD = XI
c) YD = F (XD)
d) XI = XD
e) YI = YD
f) XD = XD +DX
g) YD = F (XD)
h) P = YI * YD
i) REPETIR DESDE EL PASO (d) MIENTRAS (P > 0) Y (XD < LSX)
j) SI P = 0 ENTONCES
    SI YI = 0 ENTONCES
        XI ES RAIZ EXACTA
    CASO CONTRARIO
        XD ES RAIZ EXACTA
        XD = XD + DX
    FIN SI
FIN SI

Ejemplo 2.4:
Localizar los cambios de signo de la ecuación: $F(X) = X^2 * \log_{10}(2*X) - 8 = 0$

Solución:
Se adoptan los valores LIX = 0.5, LSX = 5, DX = 0.5, realizando el seguimiento del algoritmo, se genera la tabla siguiente:

XI	XD	YI = F(XI)	YD = F(XD)	P = YI*YD
0.50	0.50		-8.00	
0.50	1.00	-8.00	-7.70	(+)
1.00	1.50	-7.70	-6.93	(+)
1.50	2.00	-6.93	-5.59	(+)
2.00	2.50	-5.59	-3.63	(+)
2.50	3.00	-3.63	-1.00	(+)
3.00	3.50	-1.00	2.35	(-)
3.50	4.00	2.35	6.45	(+)
4.00	4.50	6.45	11.32	(+)
4.50	5.00	11.32	17.00	(+)
5.00	5.50	17.00	23.50	(+)

Entonces en el intervalo cerrado [3, 3.5], se encuentra una raíz real.

Ejemplo 2.5:
Localizar los cambios de signo de la ecuación: $F(X) = X^3 - 7*X + 4 = 0$
Solución:
Se asumen los valores LIX = -4, LSX = 4, DX = 1, mediante el uso del algoritmo, se genera la tabla:

XI	XD	YI = F(XI)	YD = F(XD)	P = YI*YD
-4.00	-4.00		-32.00	
-4.00	-3.00	-32.00	-2.00	(+)
-3.00	-2.00	-2.00	10.00	(-)
-2.00	-1.00	10.00	10.00	(+)
-1.00	0.00	10.00	4.00	(+)
0.00	1.00	4.00	-2.00	(-)
1.00	2.00	-2.00	-2.00	(+)
2.00	3.00	-2.00	10.00	(-)
3.00	4.00	10.00	40.00	(+)
4.00	5.00	40.00	94.00	(+)

Por consiguiente, existen raíces en los intervalos [-3, -2], [0, 1] y [2, 3].

Segunda fase: Afinamiento de los valores que se encuentran cercanos a la raíz.

Criterios para encontrar una raíz en base a una precisión requerida.

1) F (x) es conocido en el cambio de signo

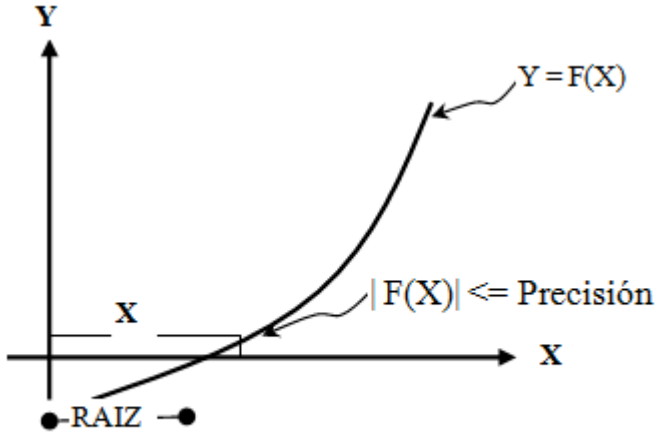
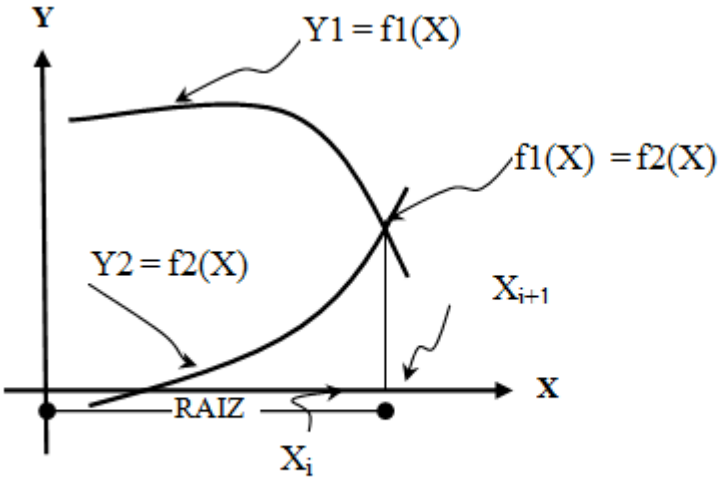


Figura 2.3 El valor de F(X) es conocido

2) El valor de F (x) es desconocido



$\Delta X = X_{i+1} - X_i$

$|\Delta X| \leq$ Precisión

Figura 2.4. El valor de F(x) es desconocido

Teorema 2.1:

Dada una función continua F(X) en el intervalo cerrado [A, B] en las abscisas. Existe por lo menos una raíz real, si F(A)*F(B) <= 0. Si además F´(X), mantiene el signo en ese intervalo, entonces existe una sola raíz (ver figuras 2.3 y 2.5).

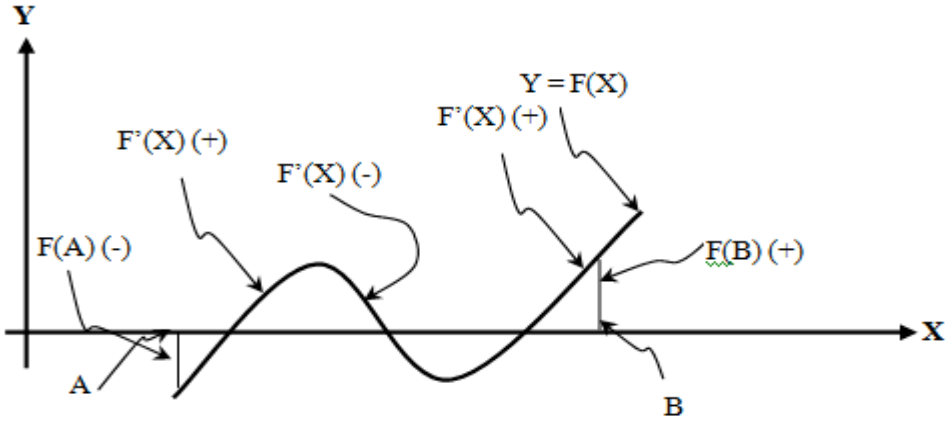


Figura 2.5. Función continua en el intervalo [A, B]

Existen varios métodos para calcular raíces reales, los que analizaremos en este capítulo son:

- Bisección.
- Aproximaciones sucesivas.
- Aproximaciones sucesivas modificado.
- Régula - falsi.
- Secante y.
- Newton - Raphson.
- Muller.

Cuyos algoritmos, necesitan las siguientes entradas: F(X), XI, XD, PRECISION
Donde F(X) es la función continua; XI y XD son las abscisas izquierda y derecha, respectivamente, entre las que puede estar por lo menos una raíz real; PRECISION es el margen de error que se produce al efectuar el cálculo de una raíz.
Seguidamente se revisan cada uno de ellos.

Método de la Bisección:

Consiste en acortar el intervalo de las abscisas, calculando su valor medio Xm, luego se calcula F(Xm) y se aproxima a la abscisa cuya ordenada tiene el mismo signo de F(Xm). (Ver la figura 2.6).

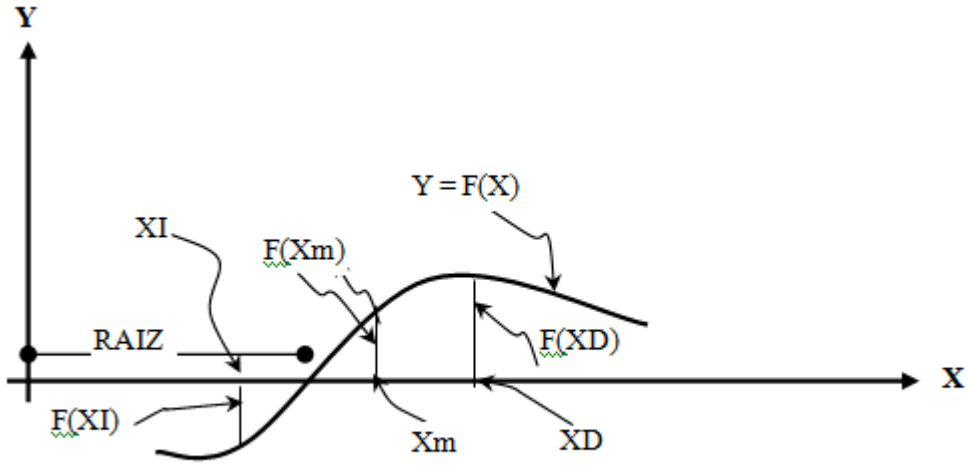


Figura 2.6. Método de la Bisección

Algoritmo para el método de la Bisección

a) YI = F(XI)
b) Xm = (XI+XD) / 2
c) Ym= F(Xm)
d) P = YI * Ym
e) SI P > 0 ENTONCES
   XI = Xm
   YI = Ym
   CASO CONTRARIO
   XD = Xm
   FIN SI
f) REPETIR DESDE EL PASO (b) MIENTRAS |Ym| > PRECISION
g) ESCRIBIR RESULTADOS

Ejemplo 2.6:
Calcular la raíz de la ecuación del ejemplo 2.5, para XI = -3, XD = -2, PRECISION = 1E-6,
usando el algoritmo del método de la bisección.

Solución:
Para calcular la raíz, se elabora la tabla:

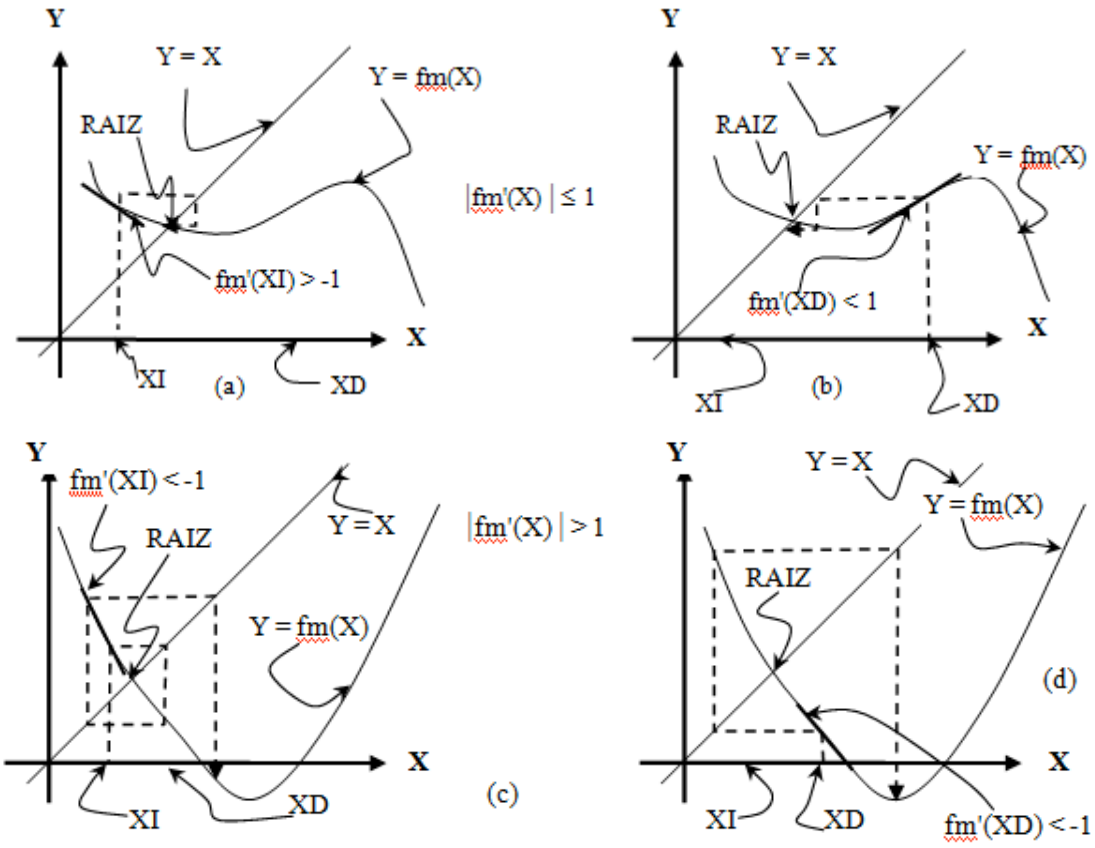
ITERACION	XI	XD	Xm=(XI+XD)/2	YI = F(XI)	Ym= F(Xm)	P = YI*Ym
1	-3.000000	-2.000000	-2.500000	-2.000000	5.875000	(-)
2	-3.000000	-2.500000	-2.750000	-2.000000	2.453125	(-)
3	-3.000000	-2.750000	-2.875000	-2.000000	0.361328	(-)
4	-3.000000	-2.875000	-2.937500	-2.000000	-0.784912	(+)
5	-2.937500	-2.875000	-2.906250	-0.784912	-0.203278	(+)
6	-2.906250	-2.875000	-2.890625	-0.203278	0.081142	(-)
7	-2.906250	-2.890625	-2.898438	-0.203278	-0.060537	(+)
8	-2.898438	-2.890625	-2.894531	-0.060537	0.010435	(-)
9	-2.898438	-2.894531	-2.896484	-0.060537	-0.025018	(+)
10	-2.896484	-2.894531	-2.895508	-0.025018	-0.007283	(+)
11	-2.895508	-2.894531	-2.895020	-0.007283	0.001578	(-)
12	-2.895508	-2.895020	-2.895264	-0.007283	-0.002852	(+)
13	-2.895264	-2.895020	-2.895142	-0.002852	-0.000637	(+)
14	-2.895142	-2.895020	-2.895081	-0.000637	0.000471	(-)
15	-2.895142	-2.895081	-2.895111	-0.000637	-0.000083	(+)
16	-2.895111	-2.895081	-2.895096	-0.000083	0.000194	(-)
17	-2.895111	-2.895096	-2.895103	-0.000083	0.000056	(-)
18	-2.895111	-2.895103	-2.895107	-0.000083	-0.000014	(+)
19	-2.895107	-2.895103	-2.895105	-0.000014	0.000021	(-)
20	-2.895107	-2.895105	-2.895106	-0.000014	0.000004	(-)
21	-2.895107	-2.895106	-2.895107	-0.000014	-0.000005	(+)
22	-2.895107	-2.895106	-2.895107	-0.000005	-0.000001	(+)
23	-2.895107	-2.895106	-2.895106	-0.000001	0.000001	(-)
24	-2.895107	-2.895106	-2.895106	-0.000001	0.000000	(-)
25	-2.895107	-2.895106	-2.895107	-0.000001	0.000000	(+)

De acuerdo a la tabla, se han necesitado 25 iteraciones para calcular la raíz = -2.895107

Método de aproximaciones sucesivas:

Para este método, se transforma la ecuación F(X) = 0, en Y = X = fm(X), a continuación se aplican las fórmulas recurrentes:
$X_1 = fm(X_0)$; $X_2 = fm(X_1)$; $X_3 = fm(X_2)$; ...; $X_{i+1} = fm(X_i)$, tal que se cumpla:
$| X_{i+1} - X_i | <= PRECISION$

Antes de efectuar el planteamiento algorítmico de este método, se analiza el comportamiento del mismo, cuando la función fm(X), varía en su derivada. (Ver la figuras 2.7).



Figuras 2.7. Comportamiento de fm(X)

Conforme a lo que se observa en las figuras 2.7 (a) y (b) existe convergencia para cuando |fm'(X) | ≤ 1; en (c) y (d) no existe convergencia porque |fm'(X) | > 1, por consiguiente hay que considerar esta situación en el algoritmo.

Algoritmo para el método de aproximaciones sucesivas:

a) Ym = (XI+XD)/2
b) Xm = Ym
c) Ym = fm(Xm)
d) REPETIR DESDE EL PASO (b) MIENTRAS (|Ym - Xm| > PRECISION) Y
(|fm'(Ym)| ≤ 1)
e) ESCRIBIR RESULTADOS

Ejemplo 2.7:
Calcular la raíz de la ecuación del ejemplo 2.5, para XI = -3, XD = -2, PRECISION = 1E-6,
usando el algoritmo del método de aproximaciones sucesivas.

Solución:
Debe elaborarse una ecuación Ym = Xm = fm(Xm), tal que $|fm'(-2.78)| \leq 1$. Despejando X, del primer término de la ecuación F(X) = 0, se tiene que:
$Ym = Xm = fm(Xm) = (7*X - 4)^{(1/3)}$, por consiguiente, $fm'(Xm) = (7/3)*(7*X - 4)^{(-2/3)}$.
Para calcular la raíz, se elabora la tabla:

ITERACION	Xm	Ym = fm(Xm)	YP = fm'(Ym)	|Ym - Xm|
0		-2.500000		
1	-2.500000	-2.780649	0.284688	0.280649
2	-2.780649	-2.862886	0.280125	0.082237
3	-2.862886	-2.886109	0.278869	0.023223
4	-2.886109	-2.892600	0.278520	0.006491
5	-2.892600	-2.894408	0.278424	0.001809
6	-2.894408	-2.894912	0.278397	0.000504
7	-2.894912	-2.895052	0.278389	0.000140
8	-2.895052	-2.895091	0.278387	0.000039
9	-2.895091	-2.895102	0.278386	0.000011
10	-2.895102	-2.895105	0.278386	0.000003
11	-2.895105	-2.895106	0.278386	0.000001

Realizando una comparación de este método, respecto del de la bisección, se observa que las iteraciones se reducen a menos de la mitad, sin embargo, debe desarrollarse un esfuerzo para elaborar la transformación de F(X) = 0 a X = fm(X), verificando que se produzca la convergencia. Para superar este inconveniente, se analiza a continuación el método de aproximaciones sucesivas modificado.

Método de aproximaciones sucesivas modificado:

Este método, aplica el mismo principio de su antecesor, pero introduce un factor de corrección (β). (Ver figura 2.8)

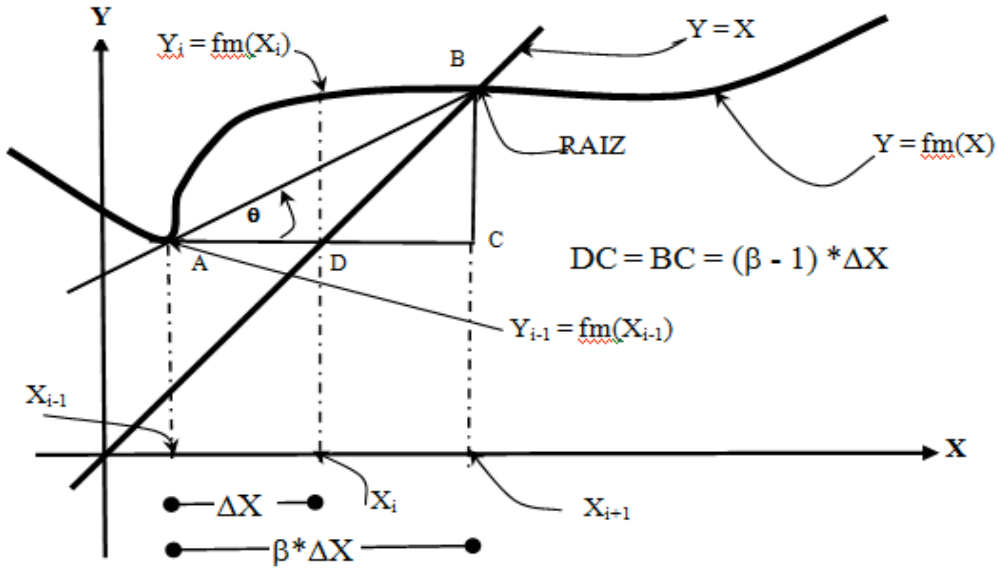


Figura 2.8 Método de Aproximaciones Sucesivas Modificado

En la figura 2.8: $\Delta X = X_i - X_{i-1} = fm(X_{i-1}) - X_{i-1}$ (2.1)
Despejando X_i de (1): $X_i = X_{i-1} + \Delta X$ (2.2)
Aplicando el factor de corrección en (2.2), se convierte en: $X_i = X_{i-1} + \beta * \Delta X$ (2.3)
Reemplazando el miembro de la derecha de (2.1) en (2.3), se obtiene la fórmula de recurrencia de este método: $X_i = X_{i-1} + \beta * (fm(X_{i-1}) - X_{i-1})$ (2.4)
En el triángulo ABC: $Tg(\theta) = (\beta - 1) * \Delta X / (\beta * \Delta X) = (\beta - 1) / \beta$ (2.5)
Despejando de (5), el valor β: $\beta = 1 / (1 - Tg(\theta))$ (2.6)
En el triángulo ABC: $Tg(\theta) = (fm(X_i) - fm(X_{i-1})) / (X_i - X_{i-1})$ (2.7)
Con estos elementos, se puede plantear a continuación el algoritmo.

Algoritmo para el método de aproximaciones sucesivas modificado:

a) YI = fm(XI)
b) YD = fm(XD)
c) TGT = (YD - YI) / (XD - XI)
d) BETA = 1 / (1 - TGT)
e) XI = XD
f) YI = YD
g) XD = XD + BETA* (YD - XD)
h) REPETIR DESDE EL PASO (b) MIENTRAS |XD - XI| > PRECISION
i) ESCRIBIR RESULTADOS

Ejemplo 2.8:
Calcular la raíz de la ecuación del ejemplo 2.5, para XI = -3, XD = -2, PRECISION = 1E-6,
usando el algoritmo del método de aproximaciones sucesivas modificado.
Solución:
Despejando X, del segundo término de la ecuación F(X) = 0, se tiene que:
$Y = X = fm(X) = (X^3 + 4) / 7$. Para calcular la raíz, se elabora la tabla:

ITERACION	XI	XD	YI = fm(XI)	YD = fm(XD)	TGT	BETA	|XD - XI|
0	-3.000000	-2.000000	-3.285714	-0.571429			
1	-2.000000	-2.833333	-0.571429	-2.677910	2.714286	-0.583333	0.833333
2	-2.833333	-2.935065	-2.677910	-3.040633	2.527778	-0.654545	0.101732
3	-2.935065	-2.893916	-3.040633	-2.890831	3.565486	-0.389790	0.041149
4	-2.893916	-2.895084	-2.890831	-2.895026	3.640455	-0.378723	0.001168
5	-2.895084	-2.895107	-2.895026	-2.895107	3.590627	-0.386007	0.000022
6	-2.895107	-2.895107	-2.895107	-2.895107	3.592104	-0.385787	0.000000

Comparando este método con su antecesor, se observan dos ventajas:
a) Puede transformarse la ecuación F(X) = 0 a X = fm(X), sin ninguna restricción.
b) El número de iteraciones, para alcanzar la convergencia, se ha reducido a casi la mitad.

Método de Regula - Falsi:

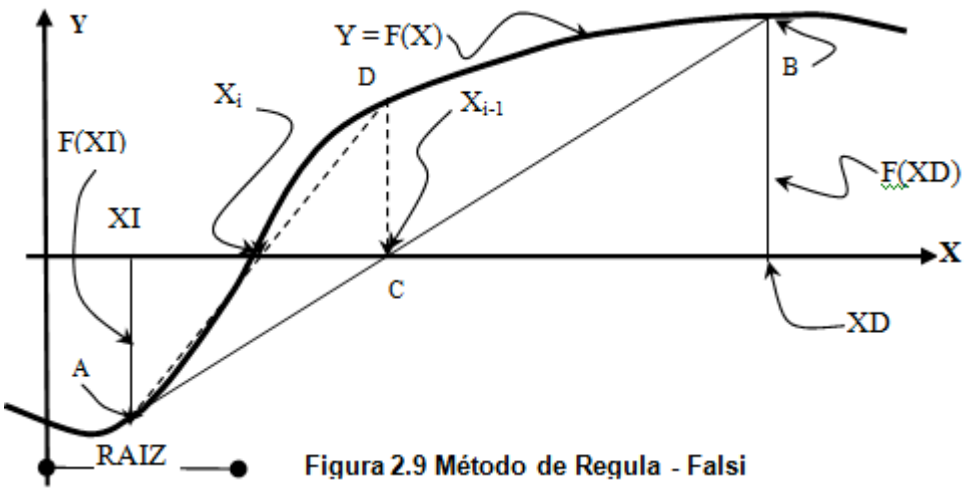


Figura 2.9 Método de Regula - Falsi

Consiste en trazar una secante que pasa por los puntos A(XI, F(XI)) y B(XD, F(XD)), (Ver figura 2.9); luego, mediante la ecuación de la línea recta, se calcula la abscisa en base a la intersección de esta secante con el eje de las X, con lo que se obtiene el punto C(X_{i-1}, 0). Seguidamente se deduce la fórmula de recurrencia:

La ecuación de recta AB es:

(F(XD) - F(XI)) / (XD - XI) = (F(X_{i-1}) - F(XI)) / (X_{i-1} - XI) (2.8);

además F(X_{i-1}) = 0 (2.9);

Reemplazando (2.9) en (2.8) y despejando de (1) X_{i-2}:

X_{i-1} = XI - F(XI) * (XD - XI) / (F(XD) - F(XI)) (2.10)

Reemplazando X_{i-1} por X_m, se obtiene la fórmula de recurrencia:

X_m = XI - F(XI) * (XD - XI) / (F(XD) - F(XI)) (2.11).

Este método se parece al de la Bisección, en el sentido de que se va acortando el intervalo por el extremo izquierdo o el derecho.

Algoritmo para el método de Regula - Falsi:

a) YI = F(XI)
b) YD = F(XD)
c) Xm = XI - YI * (XD - XI) / (YD - YI)
d) Ym= F(Xm)
e) P = YI * Ym
f) SI P > 0 ENTONCES
   XI = Xm
   YI = Ym
   CASO CONTRARIO
   XD = Xm
   YD = Ym
   FIN SI
g) REPETIR DESDE EL PASO (c) MIENTRAS |Ym| > PRECISION
h) ESCRIBIR RESULTADOS.

Ejemplo 2.9:

Calcular la raíz de la ecuación del ejemplo 2.5, para XI = -3, XD = -2, PRECISION = 1E-6,
usando el algoritmo del método de Régula - Falsi.
Solución:
Para calcular la raíz, se elabora la tabla:

ITERACION	XI	XD	YI = F(XI)	YD = F(XD)	Xm	Ym = F(Xm)	P= YI*Ym
0	-3.000000	-2.000000	-2.000000	10.000000			
1	-3.000000	-2.833333	-2.000000	1.087963	-2.833333	1.087963	(-)
2	-3.000000	-2.892054	-2.000000	0.055307	-2.892054	0.055307	(-)
3	-3.000000	-2.894959	-2.000000	0.002681	-2.894959	0.002681	(-)
4	-3.000000	-2.895099	-2.000000	0.000130	-2.895099	0.000130	(-)
5	-3.000000	-2.895106	-2.000000	0.000006	-2.895106	0.000006	(-)
6	-3.000000	-2.895106	-2.000000	0.000000	-2.895106	0.000000	(-)

Método de la Secante:

Este método utiliza la fórmula de recurrencia del de Regula - Falsi, calculándose valores consecutivos en las abscisas de uno de los extremos y manteniendo fijo el otro extremo. El método tiene la restricción de que la función F(X) en el intervalo donde se encuentra la raíz debe mantener F'(X) el mismo signo, como lo indica la figura 2.10.

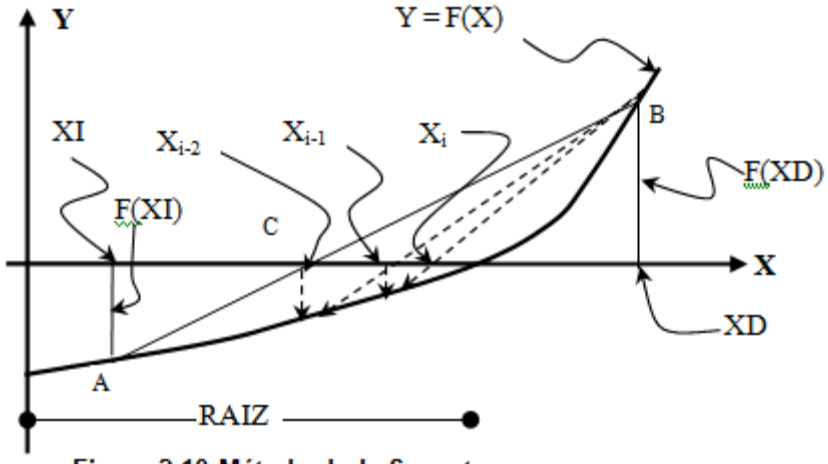


Figura 2.10 Método de la Secante

Algoritmo para el método de la Secante:

a) YI = F(XI)
b) YD = F(XD)
c) XI = XI - YI * (XD - XI) / (YD - YI)
d) YI = F(XI)
e) REPETIR DESDE EL PASO (c) MIENTRAS |F(XI)| > PRECISION
h) ESCRIBIR RESULTADOS.

Ejemplo 2.10:
Calcular la raíz de la ecuación del ejemplo 2.5, para XI = -3, XD = -2, PRECISION = 1E-6,
usando el algoritmo del método de la Secante.
Solución:
Se hará variar el punto que se encuentra a la izquierda y se mantendrá fijo el punto de la derecha. Los resultados del proceso se muestran en la siguiente tabla:

ITERACION	XI	YI = F(XI)	XD	YD
0	-3.000000	-2.000000	-2.000000	10.000000
1	-2.833333	1.087963	-2.000000	10.000000
2	-2.935065	-0.738974	-2.000000	10.000000
3	-2.870721	0.437324	-2.000000	10.000000
4	-2.910541	-0.282132	-2.000000	10.000000
5	-2.885557	0.172490	-2.000000	10.000000
6	-2.9011	-0.109059	-2.000000	10.000000
7	-2.891378	0.067524	-2.000000	10.000000
8	-2.897438	-0.042358	-2.000000	10.000000
9	-2.893653	0.026356	-2.000000	10.000000
10	-2.896015	-0.016483	-2.000000	10.000000
11	-2.89454	0.010275	-2.000000	10.000000
12	-2.89546	-0.006418	-2.000000	10.000000
13	-2.894886	0.004004	-2.000000	10.000000
14	-2.895244	-0.002500	-2.000000	10.000000
15	-2.895021	0.001560	-2.000000	10.000000
16	-2.89516	-0.000974	-2.000000	10.000000
17	-2.895073	0.000608	-2.000000	10.000000
18	-2.895127	-0.000379	-2.000000	10.000000
19	-2.895093	0.000237	-2.000000	10.000000
20	-2.895115	-0.000148	-2.000000	10.000000
21	-2.895101	0.000092	-2.000000	10.000000
22	-2.89511	-0.000058	-2.000000	10.000000
23	-2.895105	0.000036	-2.000000	10.000000
24	-2.895108	-0.000022	-2.000000	10.000000
25	-2.895106	0.000014	-2.000000	10.000000
26	-2.895107	-0.000009	-2.000000	10.000000
27	-2.895106	0.000005	-2.000000	10.000000
28	-2.895107	-0.000003	-2.000000	10.000000
29	-2.895106	0.000002	-2.000000	10.000000
30	-2.895107	-0.000001	-2.000000	10.000000

Como se puede observar en la tabla anterior, se han requerido demasiadas iteraciones (30), para alcanzar la convergencia, por lo que se puede introducir la siguiente modificación la cual consistiría en intercambiar los extremos a fin mejorar el citado algoritmo. Tal cambio se sugiere que lo haga como ejercicio, el lector.

Método de Newton - Raphson:

Se basa en el cálculo de la derivada. (Observar la figura 2.11)

En la figura 2.11:
$\Delta X = X_i - X_{i-1}$ (2.12),
entonces: $X_i = X_{i-1} + \Delta X$ (2.13)
En el triángulo ABC:
$Tg(\Phi) = -F(X_{i-1})/\Delta X = F'(X_{i-1})$ (2.14)
Despejando de (2.14) ΔX:
$\Delta X = - F(X_{i-1}) / F'(X_{i-1})$ (2.15).
Sustituyendo (2.15) en (2.13):
$X_i = X_{i-1} - F(X_{i-1}) / F'(X_{i-1})$ (2.16),
la cual es la fórmula de recurrencia para calcular raíces reales por este método.

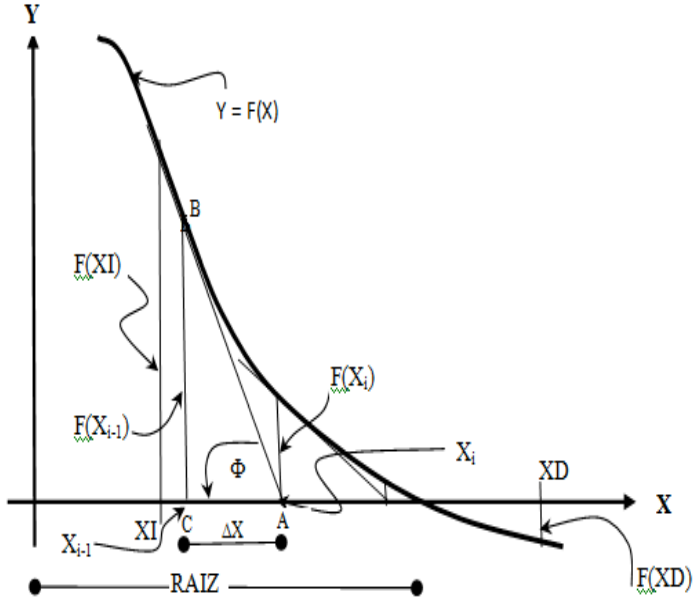


Figura 2.11. Método de Newton Raphson

Algoritmo para el método de Newton - Raphson:

a) Xm = (XI + XD) / 2
b) Ym = F(Xm)
c) YP = F'(Xm)
d) Xm = Xm - Ym / YP
e) REPETIR DESDE EL PASO (b) MIENTRAS |Ym| > PRECISION
d) ESCRIBIR RESULTADOS.

Ejemplo 2.11:
Calcular la raíz de la ecuación del ejemplo 2.5, para XI = -3, XD = -2, PRECISION = 1E-6,
usando el algoritmo del método de Newton - Raphson.
Solución:
$F(X) = X^3 - 7*X + 4$; por consiguiente $F'(X) = 3 * X^2 - 7$; la tabla siguiente muestra los resultados del proceso de este algoritmo:

ITERACION	Xm	Ym = F(Xm)	YP = F'(Xm)
0	-2.500000		
1	-3	5.875000	11.750000
2	-2.9	-2.000000	20.000000
3	-2.895118	-0.089000	18.230000
4	-2.895107	-0.000207	18.145124
5	-2.895107	0.000000	18.144925

Como se puede observar hasta aquí, este es uno de los métodos más simples porque involucra pocas operaciones, y a la vez el más eficiente.

Ejemplo 2.12:
Calcular la raíz de la ecuación del ejemplo 2.4, para XI = 3, XD = 3.5, PRECISION = 1E-6,
usando el algoritmo del método de Newton - Raphson.
Solución:
$F(X) = X^2*\log_{10}(2*X) - 8$; por consiguiente $F'(X) = 2*X*\log_{10}(2*X) + 2*X / \ln(10)$; la tabla siguiente muestra los resultados del proceso de este algoritmo:

ITERACION	Xm	Ym = F(Xm)	YP = F'(Xm)
0	3.250000		
1	3.177666	0.586397	8.106851
2	3.163712	0.109740	7.864297
3	3.161213	0.019541	7.817668
4	3.160772	0.003443	7.809322
5	3.160694	0.000606	7.807850
6	3.160681	0.000106	7.807591
7	3.160678	0.000019	7.807545
8	3.160678	0.000003	7.807537
9	3.160678	0.000001	7.807536

Para las condiciones indicadas, se obtiene una RAIZ = 3.160678, en 9 iteraciones.

Método de Muller:

Se basa en el método de la Secante, reemplazando la línea recta por una parábola de la forma:
Y = Parabola(x) = $a(x - x_2)^2 + b(x - x_2) + c$.
(Observar la figura 2.12)
En la figura 2.12: En la RAIZ: Parabola(X) = F(X) (2.17);
Para encontrar los coeficientes a, b, c, se plantea el siguiente sistema de ecuaciones:
$F(x_0) = a(x_0 - x_2)^2 + b(x_0 - x_2) + c$ (2.18)
$F(x_1) = a(x_1 - x_2)^2 + b(x_1 - x_2) + c$ (2.19)
$F(x_2) = c$ (2.20)

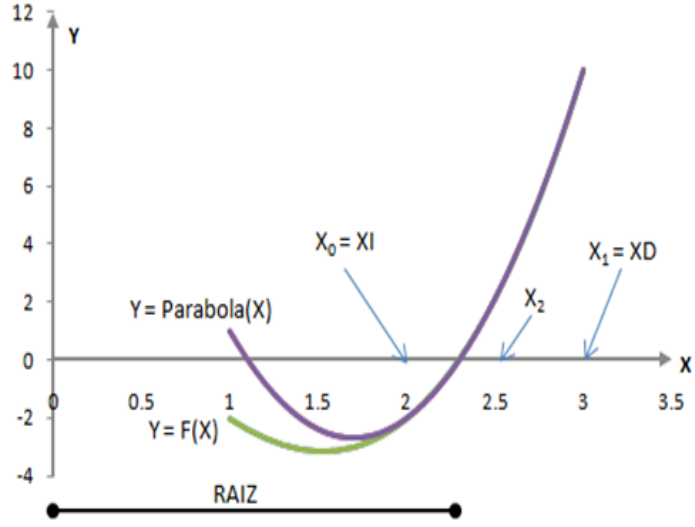


Figura 2.12. Método de Muller

Resolviendo el sistema:
$a = (d_1 – d_0)/(h_1 + h_0)$ (2.21);
$b = ah_1 + d_1$ (2.22), y
$c = F(x_2)$ (2.23).
Donde:
$h_0 = x_1 – x_0$ (8); $h_1 = x_2 – x_1$ (2.24);
$d_0 =(F(x_1) – F(x_0)) / h_0$ (2.25);
$d_1 =(F(x_2) – F(x_1)) / h_1$ (2.26);
Para encontrar la fórmula de recurrencia se puede aplicar la expresión para el cálculo de raíces de la ecuación cuadrática; sin embargo, por problemas de redondeo se aplica la relación:

$$x_3 = x_2 - \frac{2c}{b \pm \sqrt{b^2 - 4ac}}; (2.27)$$

Como se puede apreciar en la fórmula de recurrencia, se tienen dos valores en el denominador de la fracción, en este caso, para lograr la convergencia, se debe tomar el máximo en valor absoluto de los dos. Seguidamente se describe el correspondiente algoritmo.

Algoritmo para el método de Muller:

a) X0 = XI; X1 = XD; X2 = (XI + XD) / 2
b) Y0 = F(X0); Y1 = F(X1); Y2 = F(X2)
c) H0 = X1 – X0; H1 = X2 – X1
d) D0 = (Y1 – Y0) / H0; D1 = (Y2 – Y1) / H1
e) A = (D1 – D0) / (H1 + H0)
f) B = A * H1 + D1
g) C = Y2
h) R = RCUAD(B^2 – 4 * A * C)
i) DEN = B + R
j) Si ABS(B + R) < ABS(B – R) Entonces DEN = B – R
k) X3 = X2 - 2 * C / DEN
l) Y3 = F'(X3)
m) X0 = X1; X1 = X2; X2 = X3
n) Y0 = Y1; Y1 = Y2; Y2 = Y3
o) REPETIR DESDE EL PASO (c) MIENTRAS |Y3| > PRECISION
p) ESCRIBIR RESULTADOS

Ejemplo 2.13:
Calcular la raíz de la ecuación del ejemplo 2.5, para XI = -3, XD = -2, PRECISION = 1E-6,
usando el algoritmo del método de Muller.

Solución:
$F(X) = X^3 - 7*X + 4$; las tablas siguientes muestran los resultados del proceso de este algoritmo:

Iter	x0	x1	x2	x3	y0 = f(x0)	y1 = f(x1)	y2 = f(x2)	y3 = f(x3)
1	-3	-2	-2.5	-2.89303534	-2	10	5.875	0.03754400
2	-2	-2.5	-2.89303534	-2.89514769	10	5.875	0.03754400	-0.00074717
3	-2.5	-2.89303534	-2.89514769	-2.89510651	5.875	0.037544	-0.00074717	0.00000003

Iter	h0	h1	d0	d1	a	b	c	R	DEN
1	1.000000	-0.500000	12.000000	8.250000	-7.500000	12.000000	5.875000	17.895530	29.895530
2	-0.500000	-0.393035	8.250000	14.852242	-7.393035	17.757966	0.037544	17.789199	35.547165
3	-0.393035	-0.002112	14.852242	18.127298	-8.288183	18.144806	-0.000747	18.144123	36.288929

Raiz = -2.89510651 en 3 iteraciones; como se puede observar, este es uno de los métodos más eficientes aunque involucra bastantes operaciones, en pocas iteraciones, se logra la convergencia.

Ejemplo 2.14:
Calcular la raíz de la ecuación del ejemplo 2.4, para XI = 3, XD = 3.5, PRECISION = 1E-6,
usando el algoritmo del método de Muller.
Solución:
$F(X) = X^2*\log_{10}(2*X) - 8$; las tablas siguientes muestran los resultados del proceso de este algoritmo:

Iter	x0	x1	x2	x3	y0 = f(x0)	y1 = f(x1)	y2 = f(x2)	y3 = f(x3)
1	3	3.5	3.25	3.16071155	-0.9966387	2.3524510	0.5863973	0.0002185
2	3.5	3.25	3.16071155	3.16067758	2.3524510	0.5863973	0.0002185	0.0000000

Iter	h0	h1	d0	d1	a	b	c	R	DEN
1	0.500000	-0.250000	6.698179	7.064215	1.464141	6.698179	0.586397	6.436718	13.134897
2	-0.250000	-0.089288	7.064215	6.565001	1.471353	6.433627	0.000219	6.433527	12.867153

Para las condiciones indicadas, se obtiene una RAIZ = 3.16067758, en 2 iteraciones.

Implementación de los algoritmos en MATLAB y VISUAL C#:

Para cada lenguaje, se elabora un solo programa, desde el cual se ingresan el intervalo en el que se encuentran todas las raíces reales de la ecuación; y el incremento de variación en las abscisas. El programa llama a cada función o subrutina, de una manera adecuada para localizar el cambio de signo y luego prueba a cada uno de los métodos para calcular la raíz y la despliega por pantalla. Se describen a continuación una lista de funciones y subrutinas.

1) Una función que reciba el valor de X y calcule y devuelva F(X).
2) Una función que reciba el valor de X y calcule y devuelva F´(X).
3) Una función que reciba límite inicial (XI), el límite final (LSX), y el incremento (DX) y calcule y devuelva el intervalo del cambio de signo XI y XD, y además el estado en el nombre de la función.
4) Una subrutina que reciba el intervalo del cambio de signo XI y XD y calcule e imprima la raíz en ese intervalo.

Archivos fuente en MATLAB:

```
% RAICES1.M
% Calculo de raices reales de funciones trascendentes.
% FUNCION PRINCIPAL
clc;
nf = 'FPOL';
%nfp = 'FDPOL';
nfp = 'derivada';
nfm = 'fm';
%nfmp = 'fmp';
nfmp = 'derivadafm';
li = input('Ingrese el límite izquierdo: ');
ld = input('Ingrese el límite derecho: ');
dx = input('Ingrese el incremento de variación: ');
xi = li;
while 1
  [p, xi, xd] = Localiza_Raiz(nf, xi, dx, ld);
  if (p < 0)
    x = Biseccion(nf, xi, xd);
    x = AproxSuc(nfm, nfmp, xi, xd);
    x = AproxMod(nfm, xi, xd);
    x = RegulaFalsi(nf, xi, xd);
    x = Secante(nf, xi, xd);
    x = Newton_Raphson(nf, nfp, xi, xd);
```

```
    x = Muller(nf, xi, xd);
    fprintf('USO DEL METODO MATLAB:');
    fzero(nf, xi)
  end
  xi = xd; xd = xd + dx;
  if (xd > ld) break; end;
end;
% FPOL.m
function y = FPOL(x)
y = x ^ 3 - 7 * x + 4;
return

%derivada.m
%Por definicion de la derivada
function y = derivada(x)
  h = 1e-12;
  y = (FPOL(x + h) - FPOL(x)) / h;
return

% fm.m
function y = fm(x)
signo = 1;
r = 7 * x - 4;
if r < 0
    r = -r;
    signo = -1;
end;
y = signo * r ^ (1 / 3);
return

%derivadafm.m
%Por definicion de la derivada
function y = derivadafm(x)
  h = 1e-12;
  y = (fm(x + h) - fm(x)) / h;
return

% Localiza_Raiz.m
function [p, xi, xd] = Localiza_Raiz(nombre_f, li, dx, b)
xd = li;
yd = feval(nombre_f, xd);
while 1
   xi = xd;
   yi = yd;
   xd = xd + dx;
   yd = feval(nombre_f, xd);
   p = yi * yd;
   if (p <= 0) | (xd >= b)
```

```
      break;
   end;
end;
if (p == 0)
   if (xi == 0)
      fprintf('Raiz exacta = %12.5f\n', xi);
   elseif (xd == 0)
      fprintf('Raiz exacta = %12.5f\n', xd);
      xd = xd + dx;
   end;
end;
return

% Biseccion.m
function b = Biseccion(nombre_f, xi, xd)
NIter = 100; Precision = 1e-7;
fprintf('Càlculo de la raiz entre %12.5f y %12.5f, por el método de la bisección:\n', xi,
xd);
yi = feval(nombre_f, xi);
for Iter = 1 : NIter
    x = (xi + xd) / 2;
    y = feval(nombre_f, x);
    if (abs(y) <= Precision)
       break;
    elseif (y * yi > 0)
       xi = x; yi = y;
    else
       xd = x;
    end;
 end;
y = abs(y);
if (y > Precision)
   fprintf('No se alcanza la precisiòn requerida: %12.5f   ', y);
end;
fprintf('Raíz =: %12.5f, en %d iteraciones\n', x, Iter);
b = x;
return

% AproxSuc.m
function b = AproxSuc(nombre_f, n_fp, xi, xd)
NIter = 100; Precision = 1e-7;
fprintf('Càlculo de la raiz entre %12.5f y %12.5f, por el método de aproximaciones
sucesivas:\n', xi, xd);
x = (xi + xd) / 2;
for Iter = 1 : NIter
   yp = feval(n_fp, x);
```

```
   if abs(yp) >= 1
      fprintf('No existe convergencia\n');
      break;
   end;
   x0 = x;
   x = feval(nombre_f, x);
   if (abs(x - x0) <= Precision)
       break;
   end;
 end;
if abs(yp) < 1
  x0 = abs(x - x0);
  if (x0 > Precision)
     fprintf('No se alcanza la precisiòn requerida: %12.5f   ', x0);
  end;
  fprintf('Raíz =: %12.5f, en %d iteraciones\n', x, Iter);
  b = x;
else
    b = -11111;
end;
return

% AproxMod.m
function b = AproxMod(nombre_f, xi, xd)
NIter = 100; Precision = 1e-7;
fprintf('Raiz entre %12.5f y %12.5f, por el método de aproximaciones sucesivas
modificado:\n', xi, xd);
dif = xd - xi;
yi = feval(nombre_f, xi);
for Iter = 1 : NIter
   yd = feval(nombre_f, xd);
   tgt = (yd - yi) / dif;
   alfa = 1 / (1 - tgt);
   xi = xd;
   yi = yd;
   xd = xd + alfa * (yd - xd);
   dif = xd  - xi;
   if (abs(dif) <= Precision)
       break;
   end;
end;
dif = abs(dif);
if (dif > Precision)
   fprintf('No se alcanza la precisiòn requerida: %12.5f   ', dif);
end;
fprintf('Raíz =: %12.5f, en %d iteraciones\n', xd, Iter);
```

```
b = xd;
return

% RegulaFalsi.m
function x = RegulaFalsi(nombre_f, xi, xd)
NIter = 100; Precision = 1e-7;
fprintf('Raiz entre %12.5f y %12.5f, por el método de Régula - Falsi:\n', xi, xd);
yi = feval(nombre_f, xi);
yd = feval(nombre_f, xd);
for Iter = 1 : NIter
   x = xi - yi * (xd - xi) / (yd - yi);
   y = feval(nombre_f, x);
   if y * yi > 0
      xi = x;
      yi = y;
   else
      xd = x;
      yd = y;
   end;
   y = abs(y);
   if (y <= Precision)
       break;
   end;
end;
if (y > Precision)
   fprintf('No se alcanza la precisiòn requerida: %12.5f   ', y);
end;
fprintf('Raíz =: %12.5f, en %d iteraciones\n', x, Iter);
return

% Secante.m
function xi = Secante(nombre_f, xi, xd)
NIter = 100; Precision = 1e-7;
fprintf('Raiz entre %12.5f y %12.5f, por el método de la Secante:\n', xi, xd);
yi = feval(nombre_f, xi);
yd = feval(nombre_f, xd);
for Iter = 1 : NIter
   xi = xi - yi * (xd - xi) / (yd - yi);
   yi = feval(nombre_f, xi);
   if (abs(yi) <= Precision)
       break;
   end;
end;
yi = abs(yi);
if (yi > Precision)
   fprintf('No se alcanza la precisiòn requerida: %12.5f   ', yi);
```

```
end;
fprintf('Raíz =: %12.5f, en %d iteraciones\n', xi, Iter);
return

% Newton_Raphson.m
function nr = Newton_Raphson(nombre_f, nombre_fp, xi, xd)
NIter = 100; Precision = 1e-7;
fprintf('Càlculo de la raiz entre %12.5f y %12.5f, por el método de Newton -
Raphson:\n', xi, xd);
x = (xi + xd) / 2;
for Iter = 1 : NIter
    x = x - feval(nombre_f, x) / feval(nombre_fp, x);
    y = abs(feval(nombre_f, x));
    if (y <= Precision)
       break;
    end;
end;
if (y > Precision)
   fprintf('No se alcanza la precisiòn requerida: %12.5f   ', y);
end;
fprintf('Raíz =: %12.5f, en %d iteraciones\n', x, Iter);
nr = x;
return

% Muller.m
function x3 = Muller(nombre_f, x0, x1)
NIter = 100; Precision = 1e-7;
fprintf('Raiz entre %12.5f y %12.5f, por el método de Muller:\n', x0, x1);
x2 = (x0 + x1) / 2;
y0 = feval(nombre_f, x0);
y1 = feval(nombre_f, x1);
y2 = feval(nombre_f, x2);
for Iter = 1 : NIter
   h0 = x1 - x0;
   h1 = x2 - x1;
   d0 = (y1 - y0) / h0;
   d1 = (y2 - y1) / h1;
   a = (d1 - d0) / (h1 + h0);
   b = a * h1 + d1;
   c = y2;
   rc = sqrt(b^2 - 4 * a * c);
   den = b - rc;
   if abs(b + rc) > abs(b - rc)
      den = b + rc;
   end;
   x3 = x2 - 2 * c / den;
```

```
   y3 = feval(nombre_f, x3);
   if (abs(y3) <= Precision)
      break;
   end;
   x0 = x1;
   x1 = x2;
   x2 = x3;
   y0 = y1;
   y1 = y2;
   y2 = y3;
end;
y3 = abs(y3);
if (y3 > Precision)
   fprintf('No se alcanza la precisiòn requerida: %12.5f   ', y3);
end;
fprintf('Raíz =: %12.5f, en %d iteraciones\n', x3, Iter);
return
```

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa:

```
Ingrese el límite izquierdo: -5
Ingrese el límite derecho: 5
Ingrese el incremento de variación: 1
Càlculo de la raiz entre     -3.00000 y     -2.00000, por el método de la bisección:
Raíz =:     -2.89511, en 27 iteraciones
Càlculo de la raiz entre     -3.00000 y     -2.00000, por el método de aproximaciones sucesivas:
Raíz =:     -2.89511, en 13 iteraciones
Raiz entre     -3.00000 y     -2.00000, por el método de aproximaciones sucesivas modificado:
Raíz =:     -2.89511, en 4 iteraciones
Raiz entre     -3.00000 y     -2.00000, por el método de Régula - Falsi:
Raíz =:     -2.89511, en 7 iteraciones
Raiz entre     -3.00000 y     -2.00000, por el método de la Secante:
Raíz =:     -2.89511, en 36 iteraciones
Càlculo de la raiz entre     -3.00000 y     -2.00000, por el método de Newton - Raphson:
Raíz =:     -2.89511, en 4 iteraciones
Raiz entre     -3.00000 y     -2.00000, por el método de Muller:
Raíz =:     -2.89511, en 3 iteraciones
USO DEL METODO MATLAB:
ans =

   -2.8951

Càlculo de la raiz entre      0.00000 y      1.00000, por el método de la bisección:
Raíz =:      0.60270, en 24 iteraciones
Càlculo de la raiz entre      0.00000 y      1.00000, por el método de aproximaciones sucesivas:
No existe convergencia
Raiz entre      0.00000 y      1.00000, por el método de aproximaciones sucesivas modificado:
Raíz =:      0.60270, en 9 iteraciones
Raiz entre      0.00000 y      1.00000, por el método de Régula - Falsi:
Raíz =:      0.60270, en 8 iteraciones
Raiz entre      0.00000 y      1.00000, por el método de la Secante:
Raíz =:      0.60270, en 10 iteraciones
Càlculo de la raiz entre      0.00000 y      1.00000, por el método de Newton - Raphson:
Raíz =:      0.60270, en 3 iteraciones
Raiz entre      0.00000 y      1.00000, por el método de Muller:
Raíz =:      0.60270, en 3 iteraciones
USO DEL METODO MATLAB:
ans =

    0.6027

Càlculo de la raiz entre      2.00000 y      3.00000, por el método de la bisección:
Raíz =:      2.29240, en 25 iteraciones
Càlculo de la raiz entre      2.00000 y      3.00000, por el método de aproximaciones sucesivas:
Raíz =:      2.29240, en 19 iteraciones
Raiz entre      2.00000 y      3.00000, por el método de aproximaciones sucesivas modificado:
Raíz =:      2.29240, en 5 iteraciones
Raiz entre      2.00000 y      3.00000, por el método de Régula - Falsi:
Raíz =:      2.29240, en 18 iteraciones
Raiz entre      2.00000 y      3.00000, por el método de la Secante:
Raíz =:      2.29240, en 18 iteraciones
Càlculo de la raiz entre      2.00000 y      3.00000, por el método de Newton - Raphson:
Raíz =:      2.29240, en 4 iteraciones
Raiz entre      2.00000 y      3.00000, por el método de Muller:
Raíz =:      2.29240, en 3 iteraciones
USO DEL METODO MATLAB:
ans =

    2.2924
```

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 2.13.

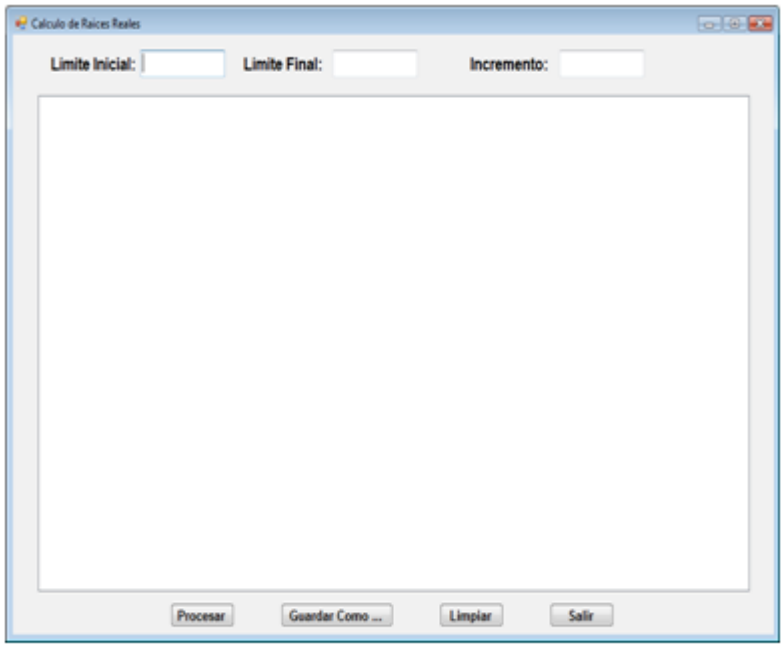


Figura 2.13 Ventana para cálculo de raíces reales en Visual C#

Como se puede apreciar, existen tres cuadros de edición, mismas que permitirán ingresar los datos; un cuadro de lista, el cual permitirá presentar los resultados; y cuatro botones, mismos que tienen la funcionalidad:

- Procesar: Ejecuta los métodos para calcular las raíces reales y despliega los resultados.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Funciones Externas:

```
// Funcion a ser analizada:
static double F(double x)
  {
     return Math.Pow(x, 3) - 7 * x + 4;
  }

// Derivada de la funcion a ser analizada:
static double FP(double x)
  {
     double h = 1e-12;
     return (F(x + h) - F(x)) / h;
  }

// Funcion modificada de F:
static double fm(double x)
  {
    return RaizCubica(7 * x - 4);
  }
```

```
// Funcion Raiz Cubica:
static double RaizCubica(double x)
  {
    return x < 0 ? - Math.Pow(Math.Abs(x), 1.0 / 3): Math.Pow(x, 1.0 / 3);
  }

// Derivada de la funcion modificada de F:
static double fmp(double x)
  {
     double h = 1e-12;
     return (fm(x + h) - fm(x)) / h;
  }
```

Botón Procesar:

```
private void cb_procesar_Click(object sender, EventArgs e)
      {
         double liminf, limsup, delta, estado, xi, xd = 0;
         try
         {
            liminf = double.Parse(txtInicial.Text);
            limsup = double.Parse(txtFinal.Text);
            delta = double.Parse(txtIncremento.Text);
            CRaicesReales.a = liminf;
            CRaicesReales.b = limsup;
            CRaicesReales.dx = delta;
            CRaicesReales.maxIter = 100;
            CRaicesReales.precision = 1e-8;
            CRaicesReales.lb = listBox1;
            CRaicesReales.punf = F;
            CRaicesReales.punfp = FP;
            CRaicesReales.punfm = fm;
            CRaicesReales.punfpm = fmp;
            xi = liminf;
            xd = xi;
            do {
               CRaicesReales.xi = xi;
               CRaicesReales.xd = xd;
               estado = CRaicesReales.LocCambioSigno();
               xi = CRaicesReales.xi;
               xd = CRaicesReales.xd;
               if (estado < 0)
               {
                 CRaicesReales.Biseccion();
                 CRaicesReales.xi = xi;
                 CRaicesReales.xd = xd;
                 CRaicesReales.AproxSucesivas();
                 CRaicesReales.xi = xi;
                 CRaicesReales.xd = xd;
                 CRaicesReales.AproxSucesivasMod();
                 CRaicesReales.xi = xi;
                 CRaicesReales.xd = xd;
```

```
                CRaicesReales.NewtonRaphson();
                CRaicesReales.xi = xi;
                CRaicesReales.xd = xd;
                CRaicesReales.RegulaFalsi();
                CRaicesReales.xi = xi;
                CRaicesReales.xd = xd;
                CRaicesReales.Secante();
                CRaicesReales.xi = xi;
                CRaicesReales.xd = xd;
                CRaicesReales.Muller();
              }
              xi = xd;
            }
            while ((xd += delta) < limsup);
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }

    }
```

Botón Guardar Como …:

```
    private void b_guardar_Click(object sender, EventArgs e)
    {
        CListBox.lb = listBox1;
        CListBox.GuardarComo("c:\\");
    }
```

Botón Limpiar:

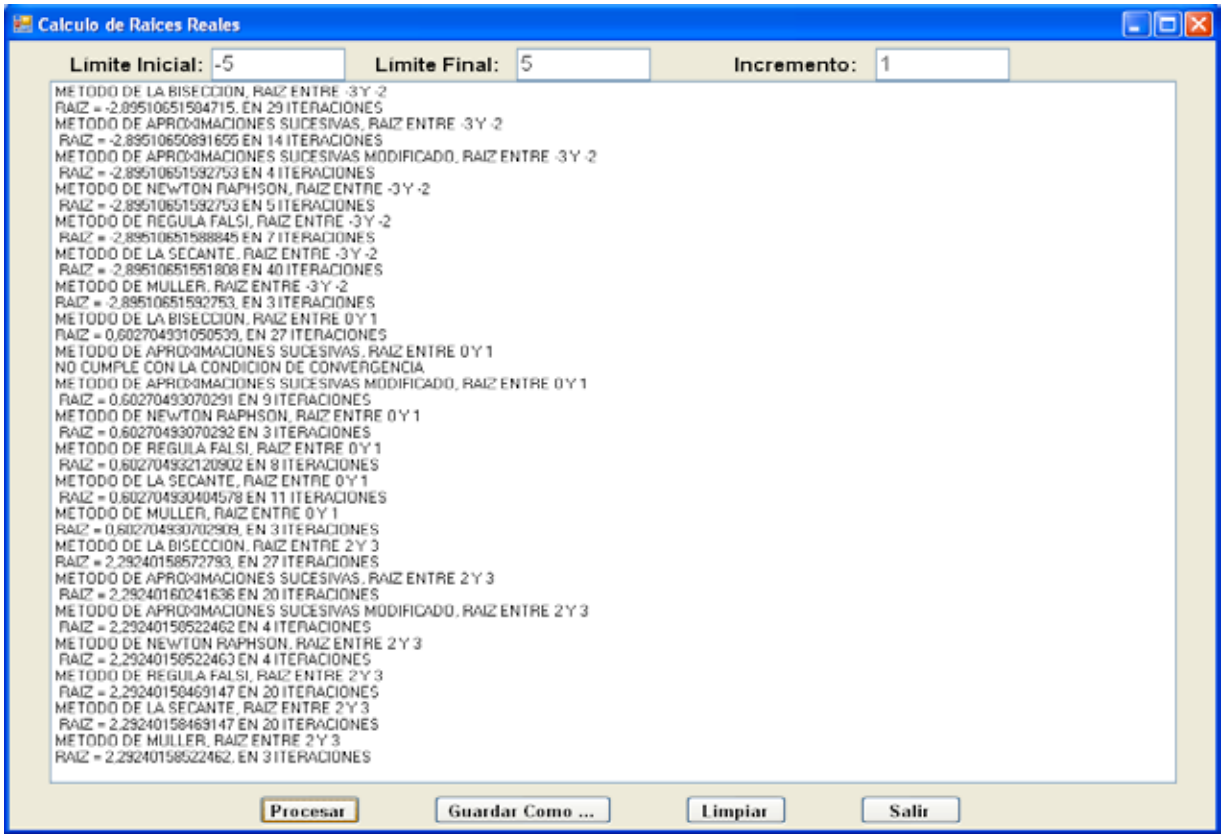
```
    private void b_limpiar_Click(object sender, EventArgs e)
    {
        CListBox.lb = listBox1;
        CListBox.Limpiar();
    }
```

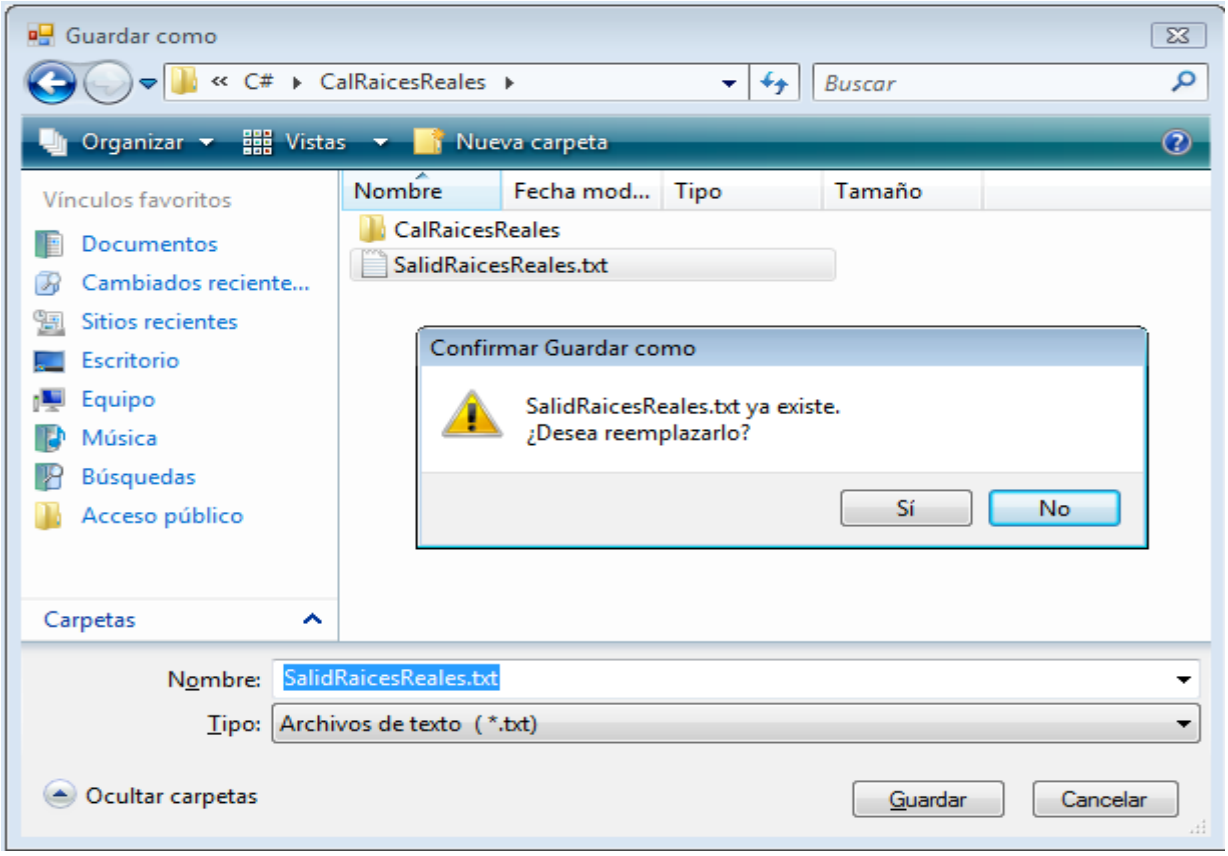
Botón Salir:

```
    private void cb_salir_Click(object sender, EventArgs e)
    {
        this.Close();
        this.Dispose();
    }
```

Las referencias a las clases CRaicesReales y CListBox, provienen de la definición de las respectivas clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:


Calculo de Raices Reales
Limite Inicial: -5
Limite Final: 5
Incremento: 1
METODO DE LA BISECCION, RAIZ ENTRE -3 Y -2
RAIZ = -2,89510651594715, EN 29 ITERACIONES
METODO DE APROXIMACIONES SUCESIVAS, RAIZ ENTRE -3 Y -2
RAIZ = -2,89510650891655 EN 14 ITERACIONES
METODO DE APROXIMACIONES SUCESIVAS MODIFICADO, RAIZ ENTRE -3 Y -2
RAIZ = -2,89510651592753 EN 4 ITERACIONES
METODO DE NEWTON RAPHSON, RAIZ ENTRE -3 Y -2
RAIZ = -2,89510651592753 EN 5 ITERACIONES
METODO DE REGULA FALSI, RAIZ ENTRE -3 Y -2
RAIZ = -2,89510651588845 EN 7 ITERACIONES
METODO DE LA SECANTE, RAIZ ENTRE -3 Y -2
RAIZ = -2,89510651551808 EN 40 ITERACIONES
METODO DE MULLER, RAIZ ENTRE -3 Y -2
RAIZ = -2,89510651592753, EN 3 ITERACIONES
METODO DE LA BISECCION, RAIZ ENTRE 0 Y 1
RAIZ = 0,602704931050539, EN 27 ITERACIONES
METODO DE APROXIMACIONES SUCESIVAS, RAIZ ENTRE 0 Y 1
NO CUMPLE CON LA CONDICION DE CONVERGENCIA
METODO DE APROXIMACIONES SUCESIVAS MODIFICADO, RAIZ ENTRE 0 Y 1
RAIZ = 0,60270493070291 EN 9 ITERACIONES
METODO DE NEWTON RAPHSON, RAIZ ENTRE 0 Y 1
RAIZ = 0,60270493070292 EN 3 ITERACIONES
METODO DE REGULA FALSI, RAIZ ENTRE 0 Y 1
RAIZ = 0,602704932120902 EN 8 ITERACIONES
METODO DE LA SECANTE, RAIZ ENTRE 0 Y 1
RAIZ = 0,602704930404578 EN 11 ITERACIONES
METODO DE MULLER, RAIZ ENTRE 0 Y 1
RAIZ = 0,602704930702909, EN 3 ITERACIONES
METODO DE LA BISECCION, RAIZ ENTRE 2 Y 3
RAIZ = 2,29240158572793, EN 27 ITERACIONES
METODO DE APROXIMACIONES SUCESIVAS, RAIZ ENTRE 2 Y 3
RAIZ = 2,29240160241636 EN 20 ITERACIONES
METODO DE APROXIMACIONES SUCESIVAS MODIFICADO, RAIZ ENTRE 2 Y 3
RAIZ = 2,29240158522462 EN 4 ITERACIONES
METODO DE NEWTON RAPHSON, RAIZ ENTRE 2 Y 3
RAIZ = 2,29240158522463 EN 4 ITERACIONES
METODO DE REGULA FALSI, RAIZ ENTRE 2 Y 3
RAIZ = 2,29240158469147 EN 20 ITERACIONES
METODO DE LA SECANTE, RAIZ ENTRE 2 Y 3
RAIZ = 2,29240158469147 EN 20 ITERACIONES
METODO DE MULLER, RAIZ ENTRE 2 Y 3
RAIZ = 2,29240158522462, EN 3 ITERACIONES
Procesar
Guardar Como ...
Limpiar
Salir


Guardar como
« C# ▸ CalRaicesReales ▸
Buscar
Organizar
Vistas
Nueva carpeta
Vínculos favoritos
Documentos
Cambiados reciente...
Sitios recientes
Escritorio
Equipo
Música
Búsquedas
Acceso público
Carpetas
Nombre
Fecha mod...
Tipo
Tamaño
CalRaicesReales
SalidRaicesReales.txt
Confirmar Guardar como
SalidRaicesReales.txt ya existe.
¿Desea reemplazarlo?
Sí
No
Nombre: SalidRaicesReales.txt
Tipo: Archivos de texto (*.txt)
Ocultar carpetas
Guardar
Cancelar

CASO PARTICULAR: Cálculo de Raíces Reales de Funciones Polinómicas.

En este apartado, se calcularán raíces reales de ecuaciones F(X) = 0, cuyas funciones tendrán la forma:

$F(X) = A_0X^N + A_1X^{N-1} + A_2X^{N-2} + \ldots + A_{N-1}X + A_N$ (2.51)

Donde: A_0, A_1, A_2, . . . , A_{N-1}, A_N son valores reales, correspondientes a los coeficientes del polinomio; N > 0 es un valor entero que corresponde al grado del polinomio.

Antes de efectuar el planteamiento matemático, se revisan algunos teoremas.

Teorema 2.2: ***(de los cambios de signo o Descartes)***

- El número de raíces reales positivas de la función polinómica es igual al número de cambios de signo de esa función o menor a este en una cantidad par.

- El número de raíces reales negativas de la función polinómica es igual al número de los cambios de signo para cuando se evalúa para F(-X) o menor a este en una cantidad par.

Ejemplo 2.15:

Dada la función: $F(X) = 3X^4 + 2X^3 – 5X^2 + 2X – 5$, se observa 3 cambios de signo, por consiguiente, máximo existirán tres raíces reales positivas.

Para $F(-X) = 3X^4 - 2X^3 – 5X^2 - 2X – 5$, se observa un cambio de signo, por lo que se deduce que existe una raíz real negativa.

De acuerdo a este ejemplo, se pueden presentar las siguientes posibilidades:

	RAICES REALES	RAICES COMPLEJAS
POSITIVAS	3	0
NEGATIIVAS	1	0
POSITIVAS	1	2
NEGATIIVAS	1	0

Teorema 2.3: ***(del resíduo)***

El residuo de la división del polinomio F(X) para un binomio de la forma X – K es igual al valor que resulta de la evaluación de ese polinomio cuando X es igual a K.

Ejemplo 2.16:

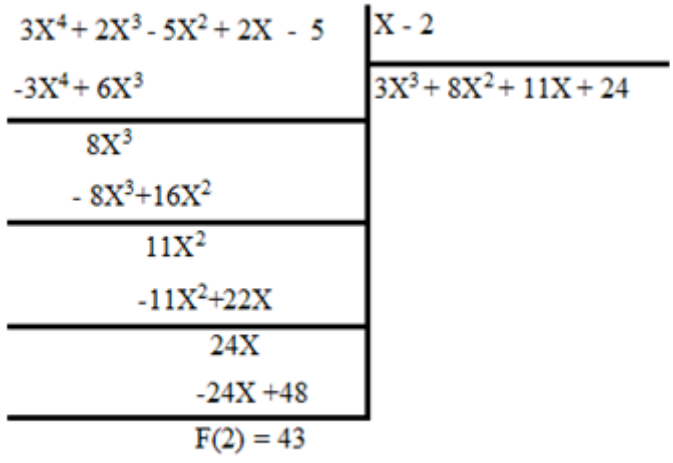


Figura 2.14 Esquema división polinómica

Encontrar el valor de F(K) de la función del ejemplo 2.15, cuando K es igual a 2.
La Solución se muestra en la figura 2.14.

Para verificar el resultado bastará con reemplazar el valor X = 2, en la función:
$F(2) = 3(2)^4+2(2)^3-5(2)^2+2(2)-5 = 48 + 16 - 20 + 4 - 5 = 43$

Por consiguiente, el teorema puede resumirse mediante el esquema:

F(X)	X - K
R = F(K)	Q(X)

Del esquema: $F(X)= Q(X) * (X - K) + R$ (2.52)
Donde: $Q(X)= B_0X^{N-1} + B_1X^{N-2} + B_2X^{N-3} + \ldots + B_{N-2}X + B_{N-1}$ (2.53)
Reemplazando los miembros derechos de (2.51) y (2.53) en (2.52):
$A_0X^N+ A_1X^{N-1}+ A_2X^{N-2}+ \ldots + A_{N-1}X+ A_N = (B_0X^{N-1}+ B_1X^{N-2}+ B_2X^{N-3}+\ldots + B_{N-2}X+ B_{N-1}) * (X - K)+ R$ (2.54)
Efectuando las operaciones en (2.54) e igualando potencias iguales de X:
$A_0 = B_0$; $A_1 = B_1 - B_0*K$; $A_2 = B_2 - B_1*K$; ... ;$A_{N-1} = B_{N-1} - B_{N-2} * K$; $A_N = R - B_{N-1} * K$; (2.55)
Despejando de las fórmulas (2.55), los valores de B_i:
$B_0 = A_0$; $B_1 = A_1 + B_0*K$; $B_2 = A_2 + B_1*K$; ...; $B_{N-1} = A_{N-1} + B_{N-2}*K$; $R = A_N + B_{N-1} * K$; (2.56)
Las fórmulas (2.56), se las puede representar mediante el esquema denominado de HORNER o de división sintética:

A_0	A_1	A_2	. . .	A_{N-1}	A_N	K
	$+ B_0 * K$	$+ B_1 * K$	. . .	$+ B_{N-2} * K$	$+ B_{N-1} * K$	
B_0	B_1	B_2	. . .	B_{N-1}	$R = F(K)$	

Algoritmo para calcular F(X), mediante el esquema de HORNER:

Entradas: Coeficientes del polinomio: A_i, para $0 \le i \le N$; grado del polinomio: N;
valor de la variable independiente: X.
Salidas: valor de la variable dependiente: F(X).

a) $B = A_0$
b) PARA I = 1 HASTA N HACER
c) $B = A_I + B * X$
d) FIN PARA
e) RETORNAR B

Para calcular F'(K), se deriva la fórmula (2.52):
$F´(X) = Q´(X) * (X - K) + Q(K)$ (2.57);
reemplazando en (2.57) X por K:

F´(K) = Q(K) (2.58), esto significa que para calcular F'(K) debe aplicarse otra división sintética sobre el esquema de HORNER, con lo que se tiene:

A_0	A_1	A_2	. . .	A_{N-1}	A_N	K
	$+ B_0 * K$	$+ B_1 * K$	. . .	$+ B_{N-2} * K$	$+ B_{N-1} * K$	
B_0	B_1	B_2	. . .	B_{N-1}	R = F(K)	
	$+ C_0 * K$	$+ C_1 * K$	. . .	$+ C_{N-2} * K$		
C_0	C_1	C_2	. . .	F'(K)		

Algoritmo para calcular F'(X), mediante el esquema de HORNER:

Entradas: Coeficientes del polinomio: A_i, para $0 \le i \le N$; grado del polinomio: N;
valor de la variable independiente: X.
Salidas: valor de la derivada: F'(X).

a) $B = A_0$
b) $FP = A_0$
c) PARA I = 1 HASTA N - 1 HACER
d) $B = A_I + B * X$
e) $FP = B + FP * X$
f) FIN PARA
g) RETORNAR FP

Ejemplo 2.17:
Dado el polinomio del ejemplo 2.15, calcular F(2) y F'(2) mediante ele esquema de HORNER:

Solución:

3	2	-5	2	-5	2
	6	16	22	48	
3	8	11	24	F(2) = 43	
	6	28	78		
3	14	39	F'(2) = 102		

Si $F(X) = 3X^4 + 2X^3 - 5X^2 + 2X - 5$, entonces:
$F´(X) = 12x^3 + 6x^2 - 10x + 2$
$F´(2) = 12\ (2)^3 + 6\ (2)^2 – 10\ (2) + 2$
$F´(2) = 96 + 24 – 20 + 2 = 102$

Ejemplo 2.18:
Calcular la raíz entre XI = 2 y XD = 3, de la ecuación del ejemplo 2.5, con una precisión de 0.0001, empleando los esquemas de HORNER y el método de Newton - Raphson.

Solución:
A este método combinado se lo conoce también como de Birge - Vieta. los resultados del proceso de estos algoritmos, se presentan a continuación:
Xm = (2 +3) / 2 = 2.5

Iteración 1:

1	0	-7	4	2.5
	2.5	6.25	F(X) =-1.875	
1	2.5	-0.75	2.125	
	2.5	12.5		
1	5	F'(X) =11.75		

Xm = 2.5 - 2.125 / 11.75 = 2.3191

Iteración 2:

1	0	-7	4	2.3191
	2.3191	5.3785	-3.7606	
1	2.3191	-1.6215	F(X) = 0.2394	
	2.3191	10.7569		
1	4.6383	F'(X) = 9.1354		

Xm = 2.3191 - 0.2394 / 9.1354 = 2.2929

Iteración 3:

1	0	-7	4	2.2929
	2.2929	5.2576	-3.9952	
1	2.2929	-1.7424	F(X) = 0.0048	
	2.2929	10.5152		
1	4.5859	F'(X) = 8.7728		

Xm = 2.2929 - 0.0048 / 8.7728 = 2.2924

Iteración 4:

1	0	-7	4	2.2924
	2.2924	5.2551	-4.0000	
1	2.2924	-1.7449	F(X) = 0.0000	
	2.2924	10.5102		
1	4.5848	F'(X) = 8.7653		

Como en esta iteración se ha alcanzado la precisión requerida, la RAIZ = 2.2924

Implementación de los algoritmos en MATLAB y VISUAL C#:

Archivos fuente en MATLAB:

El esquema de HORNER, se adapta a los algoritmos que calculan raíces reales, los cuales emplean la ecuación F(X) = 0, es decir, Bisección, Regula - Falsi, Secante, Newton – Raphson y Muller. Se incorporan entonces, estos cinco métodos e los programas respectivos. Para cada lenguaje, se elabora un solo programa, desde el cual se ingresan el grado del polinomio, los coeficientes del polinomio, almacenándolos en un arreglo, el intervalo en el que se encuentran todas las raíces reales de la ecuación; y el incremento de variación en las abscisas. El programa llama a cada función o subrutina, de una manera adecuada para localizar el cambio de signo y luego prueba a cada uno de los métodos para calcular la raíz y la despliega por pantalla. Se describen a continuación una lista de funciones y subrutinas.

1) Una función que reciba el valor de X y calcule y devuelva F(X), usando el método de la división sintética o esquema de HORNER.
2) Una función que reciba el valor de X y calcule y devuelva F´(X) , usando el método de la división sintética o esquema de HORNER..
3) Una función que reciba límite inicial (XI), el límite final (LSX), y el incremento (DX) y calcule y devuelva el intervalo del cambio de signo XI y XD, y además el estado en el nombre de la función.
4) Una subrutina que reciba el intervalo del cambio de signo XI y XD y calcule e imprima la raíz en ese intervalo.
5) Una función para ingresar desde el teclado el grado y los coeficientes del polinomio y almacenarlos a estos últimos en un arreglo.
6) Una subrutina para desplegar sobre la pantalla los coeficientes del polinomio.

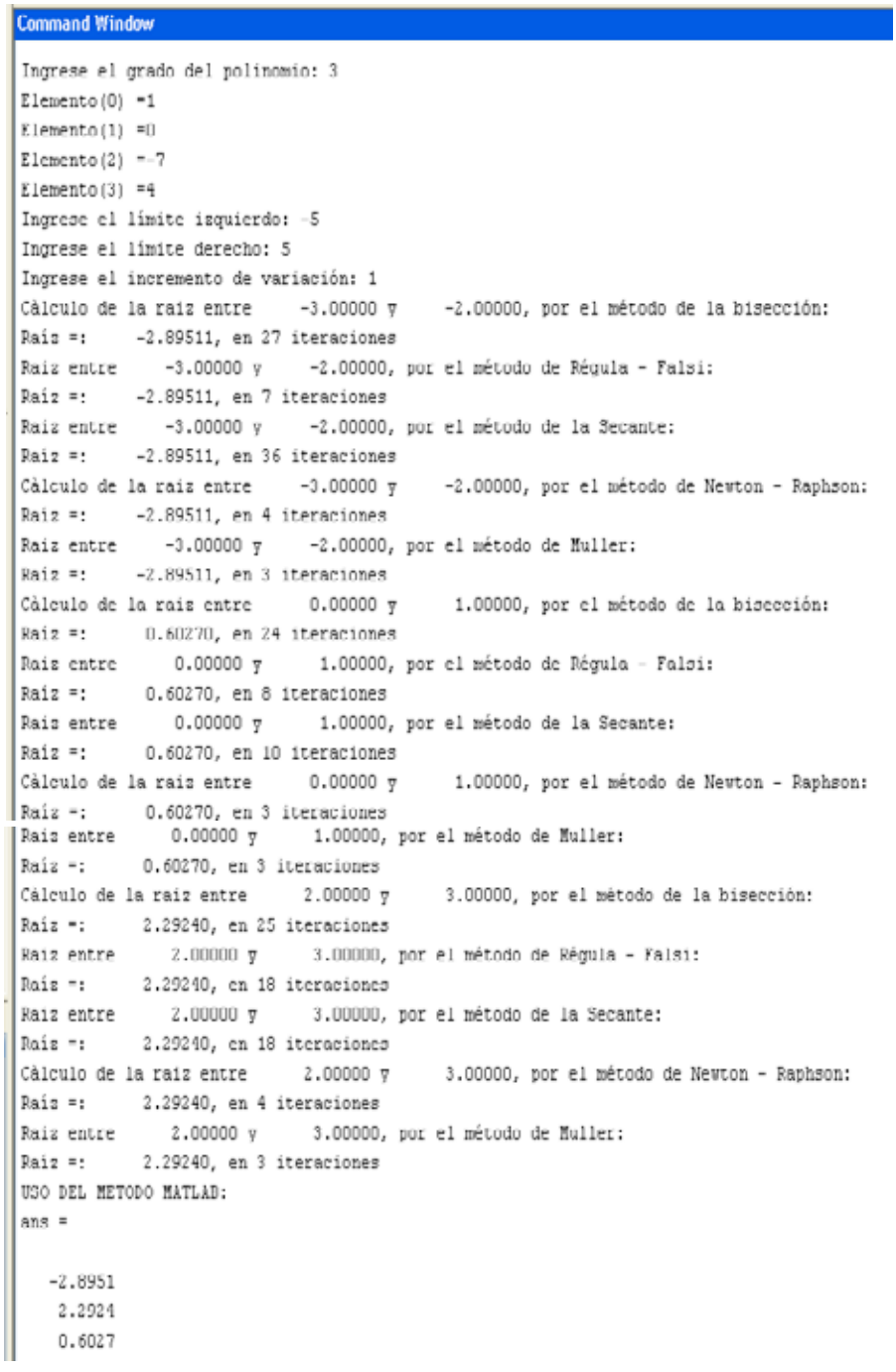
Se describe a continuación el código fuente necesario y los resultados en Matlab:

```
% raices2.m
% Calculo de raices reales de funciones polinomicas.
% FUNCION PRINCIPAL:
global A N
nf = 'HORNER';
nfp = 'HORNERP';
[N, A] = LeePolinomio;
li = input('Ingrese el límite izquierdo: ');
ld = input('Ingrese el límite derecho: ');
dx = input('Ingrese el incremento de variación: ');
xi = li;
while 1
```

```
  [p, xi, xd] = Localiza_Raiz(nf, xi, dx, ld);
  if (p < 0)
    x = Biseccion(nf, xi, xd);
    x = RegulaFalsi(nf, xi, xd);
    x = Secante(nf, xi, xd);
    x = Newton_Raphson(nf, nfp, xi, xd);
    x = Muller(nf, xi, xd);
  end
  xi = xd; xd = xd + dx;
  if (xd > ld) break; end;
end;
fprintf('USO DEL METODO MATLAB:');
roots(A)

% LeePolinomio.m
function [L, x] = LeePolinomio
while 1
  L = input('Ingrese el grado del polinomio: ');
  if (L >= 0) break; end;
end
for i = 1 : L + 1
  x(i) = input(strcat('Elemento(', int2str(i - 1), ') = '));
end
return

% HORNER.M
function b = HORNER(x)
global A N;
b = A(1);
for i = 1 : N
 b = A(i + 1) + b * x;
end
return

% Hornerp.m
function yp = HORNERP(x)
global A N;
b = A(1);
yp = b;
for i = 1 : N - 1
  b = A(i + 1) + b * x;
  yp = b + yp * x;
end
return
```

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa:



```
Command Window

Ingrese el grado del polinomio: 3
Elemento(0) =1
Elemento(1) =0
Elemento(2) =-7
Elemento(3) =4
Ingrese el límite izquierdo: -5
Ingrese el límite derecho: 5
Ingrese el incremento de variación: 1
Cálculo de la raiz entre     -3.00000 y     -2.00000, por el método de la bisección:
Raiz =:     -2.89511, en 27 iteraciones
Raiz entre     -3.00000 y     -2.00000, por el método de Régula - Falsi:
Raiz =:     -2.89511, en 7 iteraciones
Raiz entre     -3.00000 y     -2.00000, por el método de la Secante:
Raiz =:     -2.89511, en 36 iteraciones
Cálculo de la raiz entre     -3.00000 y     -2.00000, por el método de Newton - Raphson:
Raiz =:     -2.89511, en 4 iteraciones
Raiz entre     -3.00000 y     -2.00000, por el método de Muller:
Raiz =:     -2.89511, en 3 iteraciones
Cálculo de la raiz entre      0.00000 y      1.00000, por el método de la bisección:
Raiz =:      0.60270, en 24 iteraciones
Raiz entre      0.00000 y      1.00000, por el método de Régula - Falsi:
Raiz =:      0.60270, en 8 iteraciones
Raiz entre      0.00000 y      1.00000, por el método de la Secante:
Raiz =:      0.60270, en 10 iteraciones
Cálculo de la raiz entre      0.00000 y      1.00000, por el método de Newton - Raphson:
Raiz =:      0.60270, en 3 iteraciones
Raiz entre      0.00000 y      1.00000, por el método de Muller:
Raiz =:      0.60270, en 3 iteraciones
Cálculo de la raiz entre      2.00000 y      3.00000, por el método de la bisección:
Raiz =:      2.29240, en 25 iteraciones
Raiz entre      2.00000 y      3.00000, por el método de Régula - Falsi:
Raiz =:      2.29240, en 18 iteraciones
Raiz entre      2.00000 y      3.00000, por el método de la Secante:
Raiz =:      2.29240, en 18 iteraciones
Cálculo de la raiz entre      2.00000 y      3.00000, por el método de Newton - Raphson:
Raiz =:      2.29240, en 4 iteraciones
Raiz entre      2.00000 y      3.00000, por el método de Muller:
Raiz =:      2.29240, en 3 iteraciones
USO DEL METODO MATLAB:
ans =

   -2.8951
    2.2924
    0.6027
```

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 2.15.

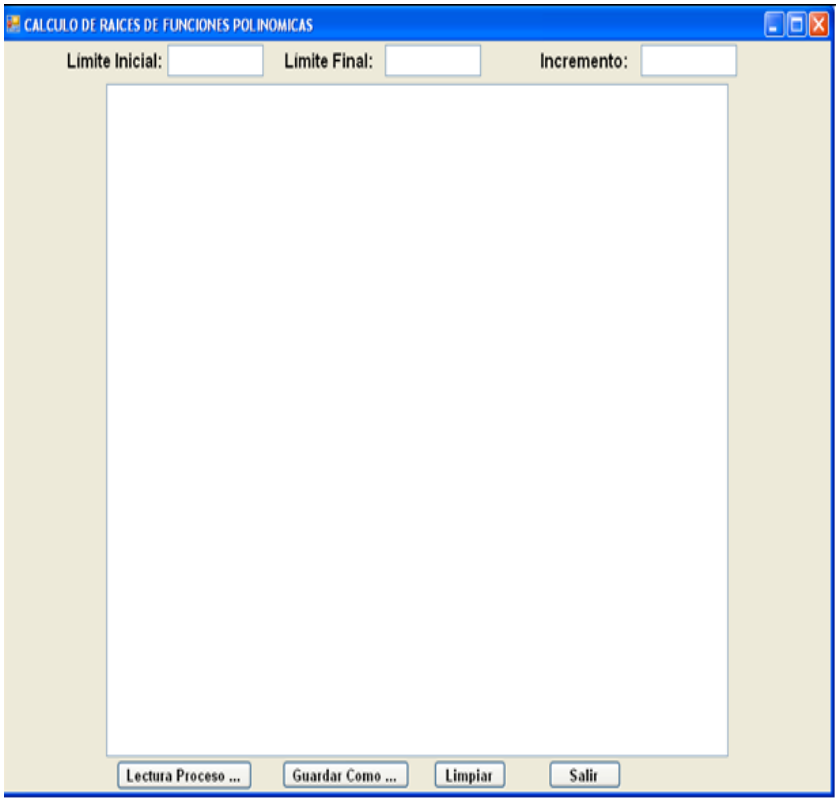


Figura 2.15 Ventana para cálculo de raíces reales de func. Polin. en Visual C#

Como se puede apreciar, existen tres cuadros de edición, mismas que permitirán ingresar los datos; un cuadro de lista, el cual permitirá presentar los resultados; y cuatro botones, mismos que tienen la funcionalidad:

- Lectura Proceso…: Valida las entradas; abre una ventana para registrar el nombre del archivo el cual permite leer los datos del polinomio desde el medio de almacenamiento permanente; ejecuta los métodos para calcular las raíces reales y despliega los resultados.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Botón Lectura Proceso…:

```
private void b_lectura_Click(object sender, EventArgs e)
  {
    double liminf, limsup, delta, estado, xi, xd = 0;
    try
    {
      // Transferir valores de los texbox a memoria:
      liminf = double.Parse(txtInicial.Text);
      limsup = double.Parse(txtFinal.Text);
      delta = double.Parse(txtIncremento.Text);
      // Abrir el archivo de entrada a traves del cuadro de dialogo:
      CEntSalArchivos objes = new CEntSalArchivos("c:\\");
      objes.AbrirArchivoEnt();
      // Leer el grado del polinomio:
      CHorner.n = Int16.Parse(objes.LeerLinea());
```

```
        if (CHorner.n > 0)
        {
          // Definir el arreglo de coeficientes de la funcion:
          CHorner.a = new double[CHorner.n + 1];
         // Leer el arreglo de coeficientes y generar los resultados en el listbox:
          CLectEscrMatrices.n = CHorner.n + 1;
          CLectEscrMatrices.b = CHorner.a;
          CLectEscrMatrices.obent = objes;
          CLectEscrMatrices.lb = listBox1;
          CLectEscrMatrices.LeeVector();
          listBox1.Items.Add("CALCULO DE RAICES DE LA FUNCION POLINOMICA DE GRADO " +
CHorner.n.ToString());
          CLectEscrMatrices.EscribeVector(" DE COEFICIENTES");
          CRaicesReales.a = liminf;
          CRaicesReales.b = limsup;
          CRaicesReales.dx = delta;
          CRaicesReales.maxIter = 100;
          CRaicesReales.precision = 1e-8;
          CRaicesReales.lb = listBox1;
          CRaicesReales.punf = CHorner.horner;
          CRaicesReales.punfp = CHorner.fphorner;
          xi = liminf;
          xd = xi;
          do
          {
            CRaicesReales.xi = xi;
            CRaicesReales.xd = xd;
            estado = CRaicesReales.LocCambioSigno();
            xi = CRaicesReales.xi;
            xd = CRaicesReales.xd;
            if (estado < 0)
            {
              CRaicesReales.Biseccion();
              CRaicesReales.xi = xi;
              CRaicesReales.xd = xd;
              CRaicesReales.NewtonRaphson();
              CRaicesReales.xi = xi;
              CRaicesReales.xd = xd;
              CRaicesReales.RegulaFalsi();
              CRaicesReales.xi = xi;
              CRaicesReales.xd = xd;
              CRaicesReales.Secante();
              CRaicesReales.xi = xi;
              CRaicesReales.xd = xd;
              CRaicesReales.Muller();
            }
            xi = xd;
          }
          while ((xd += delta) < limsup);
          // Cerrar flujo de entrada:
          objes.CerrarLectura();
        }
       }
       catch (Exception ex)
       {
```

```
    MessageBox.Show(ex.Message);
  }
}
```

Botón Guardar Como …:

```
  private void b_guardar_como_Click(object sender, EventArgs e)
      {
          CListBox.lb = listBox1;
          CListBox.GuardarComo("c:\\");
      }
```

Botón Limpiar:

```
  private void b_limpiar_Click(object sender, EventArgs e)
      {
          CListBox.lb = listBox1;
          CListBox.Limpiar();
      }
```

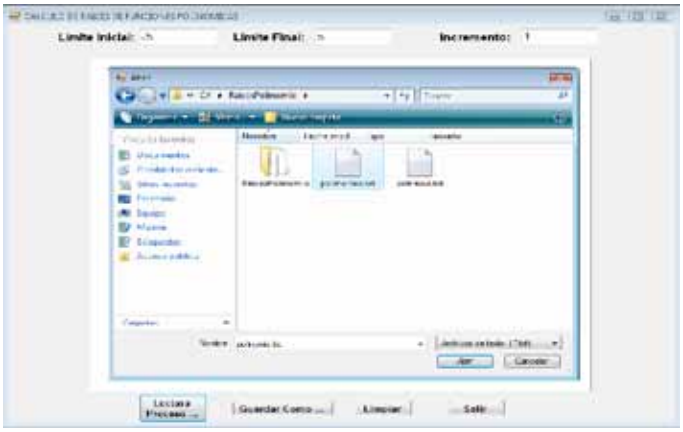
Botón Salir:

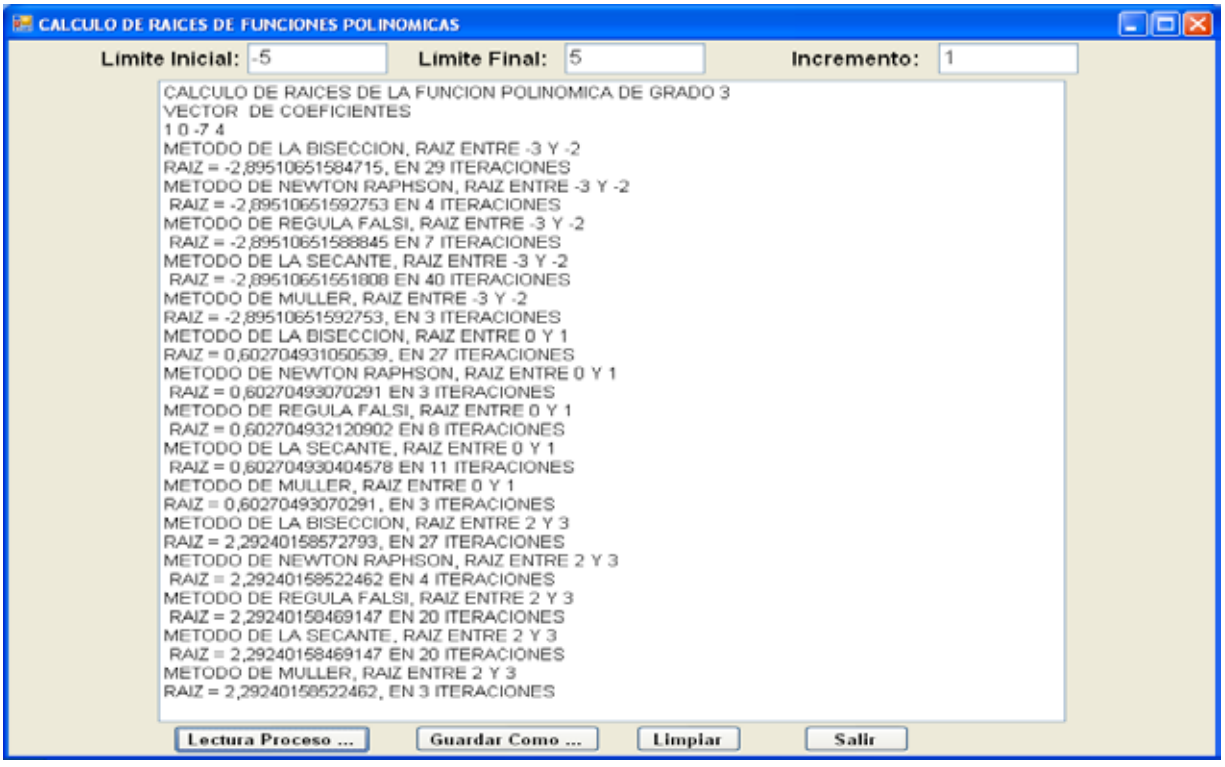
```
   private void b_salir_Click(object sender, EventArgs e)
      {
          this.Close();
          this.Dispose();
      }
```

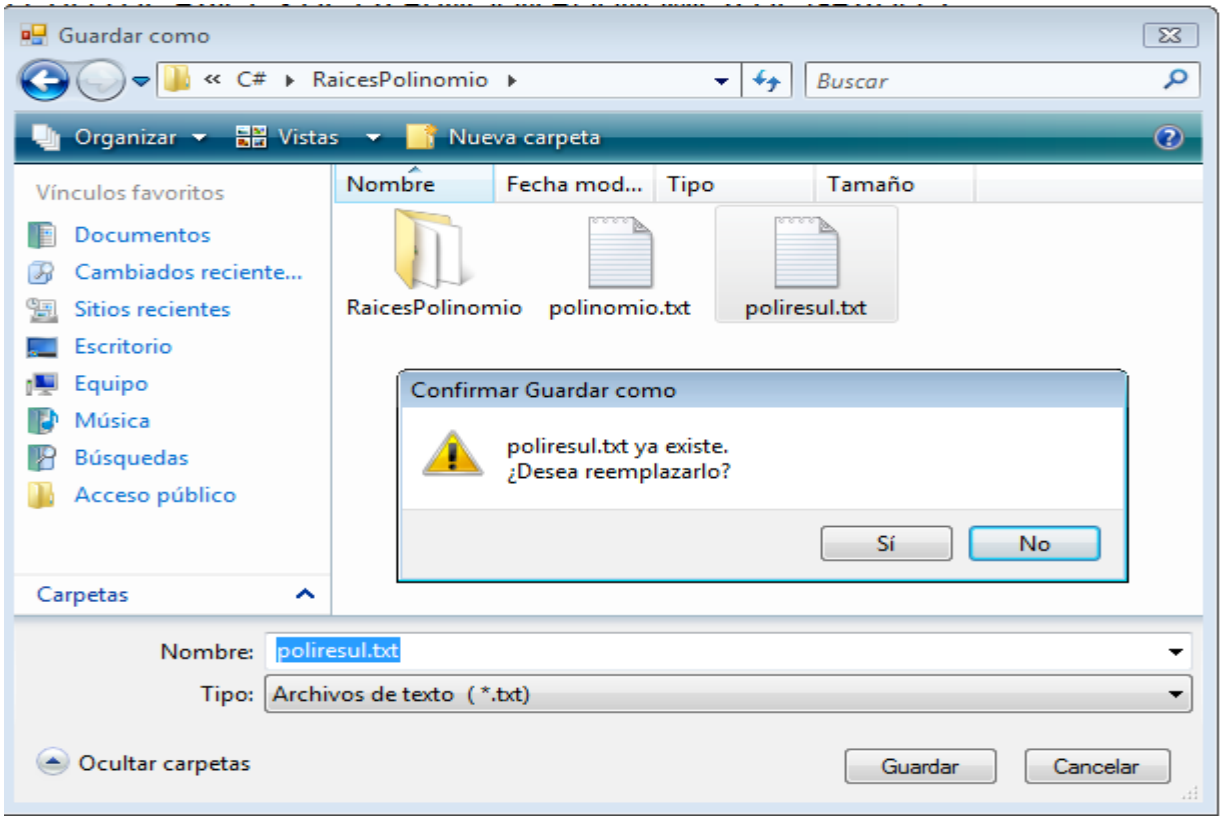
Las referencias a las clases CEntSalArchivos, CRaicesReales, CHorner, CLectEscrMatrices y CListBox, provienen de la definición de las respectivas clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:


Limite Inicial:
Limite Final:
Incremento:
Lectura Proceso ...
Guardar Como ...
Limpiar
Salir


Limite Inicial:
Limite Final:
Incremento:
Lectura Proceso ...
Guardar Como ...
Limpiar
Salir


CALCULO DE RAICES DE FUNCIONES POLINOMICAS
Limite Inicial: -5
Limite Final: 5
Incremento: 1
CALCULO DE RAICES DE LA FUNCION POLINOMICA DE GRADO 3
VECTOR DE COEFICIENTES
1 0 -7 4
METODO DE LA BISECCION, RAIZ ENTRE -3 Y -2
RAIZ = -2,89510651584715, EN 29 ITERACIONES
METODO DE NEWTON RAPHSON, RAIZ ENTRE -3 Y -2
RAIZ = -2,89510651592753 EN 4 ITERACIONES
METODO DE REGULA FALSI, RAIZ ENTRE -3 Y -2
RAIZ = -2,89510651588845 EN 7 ITERACIONES
METODO DE LA SECANTE, RAIZ ENTRE -3 Y -2
RAIZ = -2,89510651551808 EN 40 ITERACIONES
METODO DE MULLER, RAIZ ENTRE -3 Y -2
RAIZ = -2,89510651592753, EN 3 ITERACIONES
METODO DE LA BISECCION, RAIZ ENTRE 0 Y 1
RAIZ = 0,602704931050539, EN 27 ITERACIONES
METODO DE NEWTON RAPHSON, RAIZ ENTRE 0 Y 1
RAIZ = 0,60270493070291 EN 3 ITERACIONES
METODO DE REGULA FALSI, RAIZ ENTRE 0 Y 1
RAIZ = 0,602704932120902 EN 8 ITERACIONES
METODO DE LA SECANTE, RAIZ ENTRE 0 Y 1
RAIZ = 0,602704930404578 EN 11 ITERACIONES
METODO DE MULLER, RAIZ ENTRE 0 Y 1
RAIZ = 0,60270493070291, EN 3 ITERACIONES
METODO DE LA BISECCION, RAIZ ENTRE 2 Y 3
RAIZ = 2,29240158572793, EN 27 ITERACIONES
METODO DE NEWTON RAPHSON, RAIZ ENTRE 2 Y 3
RAIZ = 2,29240158522462 EN 4 ITERACIONES
METODO DE REGULA FALSI, RAIZ ENTRE 2 Y 3
RAIZ = 2,29240158469147 EN 20 ITERACIONES
METODO DE LA SECANTE, RAIZ ENTRE 2 Y 3
RAIZ = 2,29240158469147 EN 20 ITERACIONES
METODO DE MULLER, RAIZ ENTRE 2 Y 3
RAIZ = 2,29240158522462, EN 3 ITERACIONES
Lectura Proceso ...
Guardar Como ...
Limpiar
Salir


Guardar como
C# ▸ RaicesPolinomio ▸
Buscar
Organizar
Vistas
Nueva carpeta
Vínculos favoritos
Documentos
Cambiados reciente...
Sitios recientes
Escritorio
Equipo
Música
Búsquedas
Acceso público
Carpetas
Nombre
Fecha mod...
Tipo
Tamaño
RaicesPolinomio
polinomio.txt
poliresul.txt
Confirmar Guardar como
poliresul.txt ya existe.
¿Desea reemplazarlo?
Sí
No
Nombre: poliresul.txt
Tipo: Archivos de texto (*.txt)
Ocultar carpetas
Guardar
Cancelar

CALCULO DE RAICES REALES Y/O COMPLEJAS DE ECUACIONES DE LA FORMA F (X) = 0, PARA FUNCIONES POLINOMICAS

En este apartado, se resolverán problemas para determinar tanto las raíces reales, como las complejas.

En todo caso, el teorema 2.2, sigue siendo empleado en esta sección, al que se agregan los siguientes:

Teorema 2.4

Si una función polinómica tiene grado impar, entonces existe por lo menos una raíz real.

Teorema 2.5

El residuo de la división entre una función polinómica F(X) y un polinomio de la forma $X^2 + p*X + q$ es $B_{N-1} * X + B_{N-1} * p + B_N$, donde B_{N-1} y B_N son los coeficientes de X^{-1} y X^{-2}, respectivamente del polinomio divisor (Q(X)) de esa división.

Ejemplo 2.19:
Dado el polinomio: $F(X) = 3X^6 + 2X^5 - 4X^4 + 2X^3 + 3X^2 - X + 2$ y el divisor
$D(X) = X^2 + p*X + q = X^2 + 2X + 1$, determinar el cuociente Q(X) y B_{N-1} y B_N.

Solución:
Efectuando la división de F(X) para de D(X):

$$
\begin{array}{rrrrrrrr|l}
3X^6 & +2X^5 & -4X^4 & +2X^3 & +3X^2 & -X & +2 & X^2+2X+1 \\
-3X^6 & -6X^5 & -3X^4 & & & & & 3X^4-4X^3+X^2+4X-6 \\
\hline
 & -4X^5 & -7X^4 & & & & & \\
 & 4X^5 & +8X^4 & +4X^3 & & & & \\
\hline
 & & X^4 & +6X^3 & & & & \\
 & & -X^4 & -2X^3 & -X^2 & & & \\
\hline
 & & & 4X^3 & +2X^2 & & & \\
 & & & -4X^3 & -8X^2 & -4X & & \\
\hline
 & & & & -6X^2 & -5X & & \\
 & & & & 6X^2 & +12X & +6 & \\
\hline
 & & & & & 7X & +8 &
\end{array}
$$

Por consiguiente: $Q(X) = 3X^4 - 4X^3 + X^2 + 4X - 6$ y,
$7X + 8 = B_{N-1} * X + B_{N-1} * p + B_N$;
igualando potencias de X: $B_{N-1} = 7$; $B_{N-1} * p + B_N = 8$; entonces $B_N = 8 - 7 * 2 = -6$.

Del teorema 2.4, se deduce que $F(X) = Q(X) * (X^2 + p*X + q) + B_{N-1} * (X + p) + B_N$ (2.81)
Por el teorema 2.4, se pueden encontrar raíces complejas para polinomios F(X), de par, por consiguiente, para encontrar la relación existente entre los coeficientes A_I de F(X), y B_I de Q(X), se los examina para el grado de F(X) par (2m, $0 < m \leq N / 2$), lo que significa que el grado de Q(X) es 2m - 2, entonces:
$F(X) = A_0X^{2m} + A_1X^{2m-1} + \ldots + A_kX^{2m-k} + \ldots + A_{N-2}X^2 + A_{N-1}X + A_N$ (2.82)
$Q(X) = B_0X^{2m-2} + B_1X^{2m-3} + \ldots + B_kX^{2m-k-2} + \ldots + B_{N-4}X^2 + B_{N-3}X + B_{N-2}$ (2.83)
Sustituyendo (2.82) y (2.83) en (2.81):
$A_0X^{2m} + A_1X^{2m-1} + \ldots + A_kX^{2m-k} + \ldots + A_{N-2}X^2 + A_{N-1}X + A_N =$
$(X^2 + p*X + q) * (B_0X^{2m-2} + B_1X^{2m-3} + \ldots + B_kX^{2m-k-2} + \ldots + B_{N-4}X^2 + B_{N-3}X + B_{N-2}) +$
$B_{N-1} * (X + p) + B_N$ (2.84)
Efectuando las operaciones en (2.84), e igualando potencias de X:
$A_0 = B_0$; $A_1 = B_1 + B_0 * p$; $A_2 = B_2 + B_1 * p + B_0 * q$; . . .; $A_{N-1} = B_{N-1} + B_{N-2} * p + B_{N-3} * q$;
$A_N = B_N + B_{N-1} * p + B_{N-2} * q$; (2.85)
Despejando de las fórmulas (2.85) B_I:
$B_0 = A_0$; $B_1 = A_1 - B_0 * p$; $B_2 = A_2 - B_1 * p - B_0 * q$; . . .; $B_{N-1} = A_{N-1} - B_{N-2} * p - B_{N-3} * q$;
$B_N = A_N - B_{N-1} * p - B_{N-2} * q$; (2.86).
Las fórmulas (2.86), generan el esquema de Newton - Bairstow, cuya representación es:

A_0	A_1	A_2	. . .	A_{N-2}	A_{N-1}	A_N	p
		$- B_0 * q$	. . .	$- B_{N-4} * q$	$- B_{N-3} * q$	$- B_{N-2} * q$	q
	$- B_0 * p$	$- B_1 * p$	. . .	$- B_{N-3} * p$	$- B_{N-2} * p$	$- B_{N-1} * p$	
B_0	B_1	B_2	. . .	B_{N-2}	B_{N-1}	B_N	

Ejemplo 2.20:
Dado el polinomio del ejemplo 2.19, mediante el esquema de Newton - Bairstow, determinar B_{N-1} y B_N.

Solución:
El esquema que se genera es:

3	2	-4	2	3	-1	2	p = 2
		-3	4	-1	-4	6	q = 1
	-6	8	-2	-8	12	-14	
3	-4	1	4	-6	$B_{N-1} = 7$	$B_N = -6$	

Las siguientes fórmulas permiten determinar las cotas, para raíces complejas:
$||RAIZ| \leq (|A_0| + |A_{máx}|) / |A_0|$ (2.87); $|RAIZ| \geq |A_N| / (|A_N| + |A_{máx}|)$ (2.88)
Se presenta a continuación, un método general para calcular los ceros de una función polinómica F(X).

ALGORITMO PARA EL CALCULO DE RAICES REALES Y COMPEJAS DE ECUACIONES DE LA FORMA F(X), PARA FUNCIONES POLINOMICAS.

A) Calcular todas las raíces reales deflacionando al mismo tiempo el polinomio.
B) Si el grado del polinomio es mayor que 2 entonces
C) Determinar las cotas de las raíces del polinomio, utilizando las fórmulas 2.87 y 2.88.
D) Si el valor absoluto de la cota superior es mayor que 100 entonces
D.1) Calcular los valores iniciales de P y Q de esta manera:
$P_0 = A_1 / A_0$; $Q_0 = A_2 / A_0$
E) Caso contrario
E.1) Calcular los valores iniciales de P y Q de esta manera:
$P_0 = A_{n-1} / A_{n-2}$; $Q_0 = A_n / A_{n-2}$
E) Aplicar el método de Newton – Bairstow para el polinomio deflacionado de grado 2M de esta manera se obtiene los valores de B_{n-1} y B_n y el polinomio deflacionado de grado 2m–2.
F) Aplicar por segunda vez el método de Newton – Bairstow para el polinomio deflacionado de grado 2m –2, con los mismos valores de P_0 y Q_0; de esta forma se determinan C_{n-3}, C_{n-2} y C_{n-1}.
G) Resolver el sistema de ecuaciones:
$C_{n-1} * \Delta P + C_{n-2} * \Delta Q = B_n$
$C_{n-2} * \Delta P + C_{n-3} * \Delta Q = B_{n-1}$
H) Calcular en la iteración K, $P_k = P_{k-1} + \Delta P$; $Q_k = Q_{k-1} + \Delta Q$.

aplicar la fórmula : $X = (-P \pm \sqrt{P^2 - 4 * Q}) / 2$

caso contrario, las raices son complejas y se aplica : $X = (-P \pm \sqrt{4 * Q - P^2} * I) / 2$

I) Repetir desde el paso (E), mientras $\Delta P >$ PRECISION y $\Delta Q >$ PRECISION.
J) Calcular e imprimir las raíces que resultan del polinomio:
$X^2 + P * X + Q = 0$,

aplicar la fórmula : $X = (-A_1 \pm \sqrt{A_1^2 - 4 * A_0 * A_2}) / 2 / A_0$, caso contrario,

las raices son complejas, aplicar la fórmula : $X = (-A_1 \pm \sqrt{4 * A_0 * A_2 - A_1^2} * I) / 2 / A_0$

donde $I = \sqrt{-1}$

o sea , si el discriminante $P^2 - 4 * Q \geq 0$, las raíces son reales y se procede a
K) Obtener el siguiente polinomio deflacionado.
L) Repetir desde el paso (B).
M) Calcular e imprimir las raíces del polinomio restante:
$A_0 * X^2 + A_1 * X + A_2 = 0$,
o sea , si el discriminante $A_1^2 - 4 * A_0 * A_2 \geq 0$, las raíces son reales y se procede a
Para ilustrar el algoritmo, se procede a continuación a resolver el problema siguiente:

Ejemplo 2.3:

Calcular las raíces de la ecuación:
$F(X) = 2X^5 - 4X^4 + 7X^3 - 14X^2 + 6X - 10 = 0$; usar una PRECISION DE 1E-7
Solución:
Cálculo de raíces reales:
Localización del cambio de signo:

2	-4	7	-14	6	-10	1
	2	-2	5	-9	-3	
2	-2	5	-9	-3	F(1) = -13	

2	-4	7	-14	6	-10	2
	4	0	14	0	12	
2	0	7	0	6	F(2) = 2	

Como F(1) = -13 y F(2) = 2, una raíz está real entre 1 y 2, procedemos al calculo, para un
X0 = (1 + 2)/2 = 1.5:

2	-4	7	-14	6	-10	X0=1.5
	3	-1.5	8.25	-8.625	-3.938	
2	-1	5.5	-5.75	-2.625	F(X0) =-13.94	
	3	3	12.75	10.5		
2	2	8.5	7	F'(X0) =7.875		

X1 = X0 - F(X0)/F'(X0) = 1.5 - (-13.938)/ 7.875 = 3.26984127
Como $|F(X0)| >$ PRECISION, se procede a la iteración siguiente:

Iteración 2:

2	-4	7	-14	6	-10	3.26984
	6.53968	8.30436	50.0428	117.854	404.984	
2	2.53968	15.3044	36.0428	123.854	F(X)=394.984	
	6.53968	29.6881	147.118	598.907		
2	9.07937	44.9924	183.161	F'(X)=722.762		

X = 3.26984 - 394.984 / 722.762 = 2.723348559

Iteración 3:

2	-4	7	-14	6	-10	2.72335
	5.4467	3.93986	29.7931	43.01	133.471	
2	1.4467	10.9399	15.7931	49.01	F(X)=123.471	
	5.4467	18.7731	80.9188	263.38		
2	6.89339	29.713	96.7118	F'(X)=312.39		

X = 2.72335 - 123.471 / 312.39 = 2.328101386

Iteración 4:

2	-4	7	-14	6	-10	2.3281
	4.6562	1.52771	19.8534	13.6272	45.6942	
2	0.6562	8.52771	5.85337	19.6272	F(X)=35.6942	
	4.6562	12.3678	48.6469	126.882		
2	5.31241	20.8955	54.5003	F'(X)=146.509		

X = 2.3281 - 35.6942 / 146.509 = 2.084470714

Iteración 5:

2	-4	7	-14	6	-10	2.08447
	4.16894	0.35215	15.3253	2.76265	18.2655	
2	0.16894	7.35215	1.32535	8.76265	F(X)=8.26549	
	4.16894	9.04219	34.1735	73.9964		
2	4.33788	16.3943	35.4989	F'(X)=82.759		

X = 2.08447 - 8.26549 / 82.759 = 1.98459655

Iteración 6:

2	-4	7	-14	6	-10	1.9846
	3.96919	-0.0611	13.7708	-0.4548	11.005	
2	-0.0308	6.93886	-0.2292	5.54521	F(X)=1.005	
	3.96919	7.81611	29.2827	57.6595		
2	3.93839	14.755	29.0535	F'(X)=63.2047		

X = 1.9846 - 1.005 / 63.2047 = 1.968695818

Iteración 7:

2	-4	7	-14	6	-10	1.9687
	3.93739	-0.1233	13.5382	-0.9091	10.0224	
2	-0.0626	6.87674	-0.4618	5.09089	F(X)=0.02241	
	3.93739	7.62827	28.556	55.3089		
2	3.87478	14.505	28.0942	F'(X)=60.3998		

X = 1.9687 - 0.02241 / 60.3998 = 1.968324829

Iteración 8:

2	-4	7	-14	6	-10	1.96832
	3.93665	-0.1247	13.5328	-0.9195	10	
2	-0.0634	6.87531	-0.4672	5.08047	F(X)=1.2E-05	
	3.93665	7.62391	28.5392	55.2548		
2	3.8733	14.4992	28.072	F'(X)=60.3353		

X = 1.96832 - 1.2E-05 / 60.3353= 1.968324631

Iteración 9:

2	-4	7	-14	6	-10	1.96832
	3.93665	-0.1247	13.5328	-0.9195	10	
2	-0.0634	6.87531	-0.4672	5.08046	F(X)=3.4E-12	
	3.93665	7.62391	28.5392	55.2548		
2	3.8733	14.4992	28.072	F'(X)=60.3353		

X = 1.96832 - 3.4E-12 / 60.3353 = 1.968324631
Entonces la raíz es 1.968324631, calculada en 9 iteraciones
CALCULO DE RAICES COMPLEJAS:

$F(X) = (X - 1.9683246)*(2X^4 - 0.063351X^3 + 6.8753052X^2 - 0.467167X + 5.0804628)$

De acuerdo a las fórmulas (2.87) y (2.88), |RAIZ| es pequeño, por lo que se toman los tres últimos términos del polinomio deflacionado:
$6.8753052X^2 - 0.467167X + 5.0804628$
Entonces:
$P0 = A_{N-1} / A_{N-2} = - 0.467167 / 6.8753052 = -0.067949$
$Q0 = A_N / A_{N-2} = 5.0804628 / 6.8753052 = 0.7389436$
A continuación se aplica el método de Newton - Bairstow:
Iteración 1:

2	-0.063351	6.8753052	-0.467167468	5.080462767	p=-0.067948547
		-1.477887	-0.053607666	-3.992030051	q=0.738943604
	0.1358971	0.0049294	0.367081657	-0.010443248	
2	0.0725464	5.4023474	B_{N-1}=-0.153693476	B_N=1.077989468	

Aplicando por segunda ocasión el método para el nuevo polinomio:

2	0.0725464	5.4023474	-0.153693476	p=-0.067948547
		-1.477887	-0.154027954	q=0.738943604
	0.1358971	0.0141634	0.267623753	
2	C_{N-3}=0.2084435	C_{N-2}=3.9386236	C_{N-1}=-0.040097678	

Resolviendo el sistema:

$C_{N-1}*\Delta P + C_{N-2}*\Delta Q = B_N$ $\quad -0.040098*\Delta P + 3.938624*\Delta Q = 1.077989$

$C_{N-2}*\Delta P + C_{N-3}*\Delta Q = B_{N-1}$ $\quad 3.938624*\Delta P + 0.208443*\Delta Q = -0.153693$

$\Delta P = -0.053478159$; $\Delta Q = 0.273152559$

Iteración 2:

$P = P + \Delta P = -0.121426706$; $Q = Q + \Delta Q = 1.012096162$

2	-0.063351	6.8753052	-0.467167468	5.080462767	p=-0.121426706
		-2.024192	-0.181673967	-4.931852776	q=1.012096162
	0.2428534	0.0217964	0.591701321	-0.006938336	
2	0.1795027	4.8729093	B_{N-1}=-0.057140113	B_N=0.141671655	

Aplicando por segunda ocasión el método para el nuevo polinomio:

2	0.1795027	4.8729093	-0.057140113	p=-0.121426706
		-2.024192	-0.427464972	q=1.012096162
	0.2428534	0.0512853	0.352137721	
2	C_{N-3}=0.4223561	C_{N-2}=2.9000023	C_{N-1}=-0.132467365	

Resolviendo el sistema:

$C_{N-1}*\Delta P + C_{N-2}*\Delta Q = B_N$ $\quad -0.132467365*\Delta P + 2.9000023*\Delta Q = 0.141671655$

$C_{N-2}*\Delta P + C_{N-3}*\Delta Q = B_{N-1}$ $\quad 2.9000023*\Delta P + 0.4223561*\Delta Q = -0.057140113$

$\Delta P = -0.026641078$; $\Delta Q = 0.047635336$

Iteración 3:

$P = P + \Delta P = -0.148067784$; $Q = Q + \Delta Q = 1.059731498$

2	-0.063351	6.8753052	-0.467167468	5.080462767	p=-0.148067784
		-2.119463	-0.246689416	-5.076442519	q=1.059731498
	0.2961356	0.0344679	0.709290604	-0.000676119	
2	0.2327848	4.7903101	B_{N-1}=-0.00456628	B_N=0.003344129	

Aplicando por segunda ocasión el método para el nuevo polinomio:

2	0.2327848	4.7903101	-0.00456628	p=-0.148067784
		-2.119463	-0.560513606	q=1.059731498
	0.2961356	0.0783161	0.407062502	
2	C_{N-3}=0.5289204	C_{N-2}=2.7491632	C_{N-1}=-0.158017383	

Resolviendo el sistema:

$C_{N-1}*\Delta P + C_{N-2}*\Delta Q = B_N$ $\quad -0.158017383*\Delta P + 2.7491632*\Delta Q = 0.003344129$

$C_{N-2}*\Delta P + C_{N-3}*\Delta Q = B_{N-1}$ $\quad 2.7491632*\Delta P + 0.5289204*\Delta Q = -0.00456628$

ΔP = -0.001874275; ΔQ = 0.001108687

Iteración 4:

P = P + ΔP = -0.149942059; Q = Q + ΔQ = 1.060840185

2	-0.063351	6.8753052	-0.467167468	5.080462767	p=-0.14994206
		-2.12168	-0.250924113	-5.080460302	q=1.060840185
	0.2998841	0.0354663	0.718086181	-8.09751E-07	
2	0.2365334	4.7890911	B_{N-1}=-5.40042E-06	B_N=1.65473E-06	

Aplicando por segunda ocasión el método para el nuevo polinomio:

2	0.2365334	4.7890911	-5.40042E-06	p=-0.14994206
		-2.12168	-0.569053236	q=1.060840185
	0.2998841	0.0804315	0.412017129	
2	C_{N-3}=0.5364175	C_{N-2}=2.7478423	C_{N-1}=-0.157041507	

Resolviendo el sistema:

$C_{N-1}*\Delta P + C_{N-2}*\Delta Q = B_N$ -0.157041507*ΔP + 2.7478423*ΔQ = 1.65473E-06

$C_{N-2}*\Delta P + C_{N-3}*\Delta Q = B_{N-1}$ 2.7478423*ΔP + 0.5364175*ΔQ = -5.40042E-06

ΔP = -2.05991E-06; ΔQ = 4.84469E-07

Iteración 5:

P = P + ΔP = -0.149944119; Q = Q + ΔQ = 1.06084067

2	-0.063351	6.8753052	-0.467167468	5.080462767	p=-0.14994412
		-2.121681	-0.250928599	-5.080462767	q=1.06084067
	0.2998882	0.0354674	0.718096066	-6.6472E-14	
2	0.2365375	4.7890912	B_{N-1}=-4.43312E-13	B_{N-1}=1.98966E-12	

Aplicando por segunda ocasión el método para el nuevo polinomio:

2	0.2365375	4.7890912	-4.43312E-13	p=-0.14994412
		-2.121681	-0.569062237	q=1.06084067
	0.2998882	0.0804339	0.412023016	
2	C_{N-3}=0.5364257	C_{N-2}=2.7478438	C_{N-1}=-0.157039221	

Resolviendo el sistema:

$C_{N-1}*\Delta P + C_{N-2}*\Delta Q = B_N$ -0.157039221*ΔP + 2.7478438*ΔQ = 1.98966E-12

$C_{N-2}*\Delta P + C_{N-3}*\Delta Q = B_{N-1}$ 2.7478438*ΔP + 0.5364257*ΔQ = -4.43312E-13

ΔP = -2.99344E-13; ΔQ = 7.06973E-13

P = P + ΔP = -0.149944119; Q = Q + ΔQ = 1.06084067

Por consiguiente, el polinomio cuadrático es:
$P(X) = X^2 + P*X + Q = X^2 - 0.149944119*X + 1.0608407 = 0$
Entonces $X = 0.074972059 +/- 1.02723895*I$

COMPROBACION PARA LA RAIZ POSITIVA:
Reemplazando RAIZ = 0.074972059 + 1.02723895*I, en el polinomio original y aplicando la fórmula de Moivre: $[r(\cos \psi + i \operatorname{sen} \psi)]^n = r^n (\cos n\psi + i \operatorname{sen} n\psi)$, se puede generar la tabla:
$r = 1.029971198$; $\psi = 1.497941451$

k	X^k		A_{N-k}	$A_{N-k} * X^k$	
	$r^k \cos k\psi$	$r^k \operatorname{sen} k\psi$		REAL	IMAGINARIA
5	0.41296	1.08305	2	0.82592	2.16611
4	1.07793	-0.3233	-4	-4.3117	1.29334
3	-0.2369	-1.0666	7	-1.6584	-7.4665
2	-1.0496	0.15403	-14	14.6944	-2.1564
1	0.07497	1.02724	6	0.44983	6.16343
			SUMA	10	0

CALCULO DE LAS OTRAS 2 RESTANTES RAICES:
El polinomio deflacionado resultante es cuadratico:
$P(X) = 2X^2 + 0.236537499X + 4.7890912 = 0$
Por consiguiente:
$X = -0.059134375 +/- 1.546301636*I$

COMPROBACION PARA LA RAIZ POSITIVA:
Aplicando el mismo procedimiento para RAIZ = -0.059134375 + 1.546301636*I
$r = 1.547431945$; $\psi = -1.5325725$

k	X^k		A_{N-k}	$A_{N-k} * X^k$	
	$r^k \cos k\psi$	$r^k \operatorname{sen} k\psi$		REAL	IMAGINARIA
5	-1.6854	8.71119	2	-3.3709	17.4224
4	5.66696	0.87327	-4	-22.668	-3.4931
3	0.42397	-3.6811	7	2.96781	-25.767
2	-2.3876	-0.1829	-14	33.4257	2.56031
1	-0.0591	1.5463	6	-0.3548	9.27781
			SUMA	10	0

Implementación de los algoritmos en MATLAB y VISUAL C#:

Archivos fuente en MATLAB:

A fin de ingresar el polinomio se utiliza la función LeePolinomio.El esquema de HORNER, se adapta a los algoritmos para localizar los cambios de signo y que calculan raíces reales, los cuales emplean la ecuación F(X) = 0, en este caso, las funciones HORNER, Localiza_Raiz y Muller, respectivamente. Todas estas funciones ya fueron descritas en apartados anteriores. Para cada lenguaje, se elabora un solo programa, desde el cual se ingresan los datos del polinomio, almacenándolos en un arreglo, el intervalo en el que se encuentran todas las raíces reales de la ecuación; y el incremento de variación en las abscisas. El programa llama a cada función o subrutina, de una manera adecuada para localizar el cambio de signo y luego al método de Muller para calcular la raíz real, la despliega por pantalla y deflaciona el polinomio; este proceso se repite hasta calcular todas las raíces reales. Mientras el grado del polinomio sea mayor que 2, se realizan llamadas a la función NewtonBairstow; finalmente se aplica la fórmula de la ecuación cuadrática para calcular las raíces reales o complejas del polinomio de segundo grado que queda. Se describe a continuación la función NewtonBairstow:
Calcular las cotas de la raíz. Calcular los valores iniciales de p y q. Deflacionar el polinomio al grado 2m - 2. Aplicar las reglas de Newton - Bairstow para calcular Δp y Δq. Calcular los nuevos valores de p y q. Si Δp y Δq alcanzan la precisión entonces finalizar este proceso. Calcular las raíces reales o complejas de la ecuación $X^2 + pX + q = 0$. Transferir los coeficientes B(i) a los A(i). Disminuir en 2 al grado del polinomio.

Se describe a continuación el código fuente necesario y los resultados en Matlab:

```
% raices3.m
% Calculo de raices reales o complejas de funciones polinomicas.
% FUNCION PRINCIPAL:
global A N
nf = 'HORNER';
[N, A] = LeePolinomio;
% Obtener una copia de A:
B = A;
NB = N;
li = input('Ingrese el límite izquierdo: ');
ld = input('Ingrese el límite derecho: ');
dx = input('Ingrese el incremento de variación: ');
xi = li;
```

```
while 1
   [p, xi, xd] = Localiza_Raiz(nf, xi, dx, ld);
   if (p < 0)
      Raiz = Muller(nf, xi, xd);
      % DEFLACIONAR EL POLINOMIO: */
      for i = 2 : N
        A(i) = A(i) + A(i - 1) * Raiz;
      end;
      N = N - 1;
   end
   xi = xd;
   xd = xd + dx;
   if (xd > ld) break; end;
end;
if mod(N, 2) == 1
    disp('Error: NO SE CALCULARON TODAS LAS RAICES REALES');
    return;
end;
if N > 1
   while N > 2
       NewtonBairstow;
   end;
   % POLINOMIO RESTANTE, VERIFICAR SI LAS RAICES SON REALES O
COMPLEJAS */
   P1 = -A(2) / 2 / A(1); P2 = A(2) * A(2) - 4 * A(1) * A(3);
   if (P2 < 0)
     disp(sprintf('RAICES COMPLEJAS = %f +/- %f * I\n', P1, sqrt(-P2)/2/A(1)));
   else
     disp(sprintf('RAICES REALES MULTIPLES O CERCANAS: %f Y %f\n', P1 -
sqrt(P2)/2/A(1), P1 + sqrt(P2)/2/A(1)));
   end;
end;
fprintf('USO DEL METODO MATLAB:');
roots(B)

%NewtonBairstow.m
function NewtonBairstow
  global A N;
  NIter = 100; Precision = 1e-7;
  % CALCULAR LAS COTAS DEL MODULO DE LA RAIZ */
  AMax = abs(A(1));
  for I = 2 : N + 1
   if AMax <  abs(A(I))
      AMax = abs(A(I));
   end;
  end;
```

```
CTS = (abs(A(1)) + AMax) / abs(A(1));
% CALCULAR LOS COEFICIENTES INICIALES P1 Y Q1 */
P1 = A(N) / A(N - 1);
Q1 = A(N + 1) / A(N - 1);
if CTS > 100
  P1 = A(2) / A(1);
  Q1 = A(3) / A(1);
end;
for iter = 1 : NIter
  % DEFLACIONAR EL POLINOMIO AL GRADO 2*M-2: */
  B(1) = A(1); B(2) = A(2) - P1 * B(1);
  for I = 3 : N +1
    B(I) = A(I) - Q1 * B(I - 2) - P1 * B(I - 1);
  end;
  % CALCULAR C1, C2 Y C3: */
  C3 = B(1); C2 = B(2) - P1 * B(1);
  for I = 3 : N
    C1 = B(I) - Q1 * C3 - P1 * C2;
    if I < N
      C3 = C2; C2 = C1;
     end;
  end;
  % RESOLVER EL SISTEMA:
  %   C1*DP+C2*DQ=B(N)
  %   C2*DP+C3*DQ=B(N-1)  */
  DT = C1 * C3 - C2 * C2; DP = (B(N+1) * C3 - B(N) * C2) / DT;
  DQ = (C1 * B(N) - C2 * B(N+1)) / DT;
  % CALCULAR LOS NUEVOS COEFICIENTES P Y Q: */
  P1 = P1 + DP; Q1 = Q1 + DQ;
  DMax = abs(DP);
  if DMax > abs(DQ)
    DMax = abs(DQ);
  end;
  if DMax <= Precision
    break;
  end;
end;
if DMax > Precision
  disp(sprintf('NO SE ALCANZA LA PRECISION REQUERIDA: %f\n', DMax));
end;
% VERIFICAR SI LAS RAICES SON REALES O COMPLEJAS: */
Q1 = P1 * P1 - 4 * Q1;
if Q1 < 0
  disp(sprintf('RAICES COMPLEJAS = %f +/- %f * I\n', -P1 / 2, sqrt(-Q1) / 2));
else
```

```
    disp(sprintf('RAICES REALES MULTIPLES O CERCANAS: %f Y %f\n', (-P1 -
sqrt(Q1)) / 2, (-P1 + sqrt(Q1)) / 2));
  end;
  disp(sprintf('EN %d NITERACIONES\n', iter));
  % TRANSFERIR LOS COEFICIENTES B(I) A LOS A(I): */
  for I = 1 : N - 1
     A(I) = B(I);
  end;
  N = N - 2;
  return;
```

```
Command Window

Ingrese el grado del polinomio: 5
Elemento(0) =2
Elemento(1) =-4
Elemento(2) =7
Elemento(3) =-14
Elemento(4) =6
Elemento(5) =-10
Ingrese el límite izquierdo: -5
Ingrese el límite derecho: 5
Ingrese el incremento de variación: 1
Raiz entre      1.00000 y      2.00000, por el método de Muller:
Raíz =:      1.96832, en 4 iteraciones
RAICES COMPLEJAS = 0.074972 +/- 1.027239 * I

EN 5 NITERACIONES

RAICES COMPLEJAS = -0.059134 +/- 1.546302 * I

USO DEL METODO MATLAB:
ans =

   1.9683
  -0.0591 + 1.5463i
  -0.0591 - 1.5463i
   0.0750 + 1.0272i
   0.0750 - 1.0272i
```

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 2.16.

Como se puede apreciar, existen tres cuadros de edición, mismas que permitirán ingresar los datos; un cuadro de lista, el cual permitirá presentar los resultados; y cuatro botones, mismos que tienen la funcionalidad:

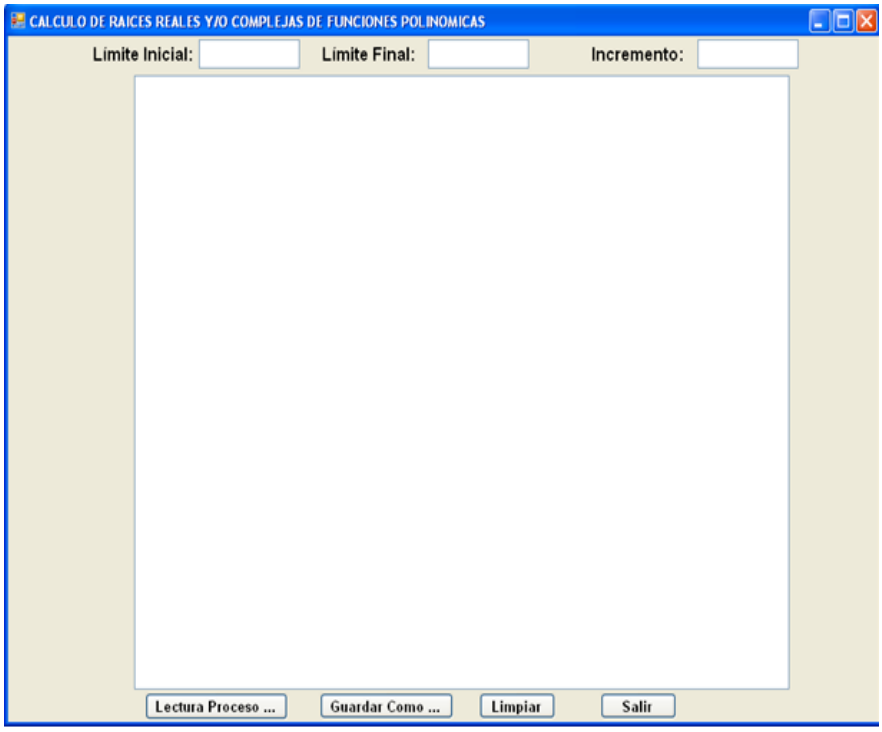


Figura 2.16 Ventana para cálculo de raíces reales y/o complejas de func. Polin. en Visual C#

- Lectura Proceso…: Valida las entradas; abre una ventana para registrar el nombre del archivo el cual permite leer los datos del polinomio desde el medio de almacenamiento permanente; ejecuta los métodos para calcular las raíces reales y/o complejas y despliega los resultados.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Botón Lectura Proceso…:

```
private void b_lectura_Click(object sender, EventArgs e)
 {
    double liminf, limsup, delta, estado, xi, xd = 0, Raiz;
    int I;
    try
    {
      // Transferir valores de los texbox a memoria:
      liminf = double.Parse(txtInicial.Text);
      limsup = double.Parse(txtFinal.Text);
      delta = double.Parse(txtIncremento.Text);
      // Abrir el archivo de entrada a traves del cuadro de dialogo:
      CEntSalArchivos objes = new CEntSalArchivos("c:\\");
      objes.AbrirArchivoEnt();
      // Leer el grado del polinomio:
      CHorner.n = Int16.Parse(objes.LeerLinea());
      if (CHorner.n > 0)
      {
```

```
        // Definir el arreglo de coeficientes de la funcion:
        CHorner.a = new double[CHorner.n + 1];
        // Leer el arreglo de coeficientes y generar los resultados en el listbox:
        CLectEscrMatrices.n = CHorner.n + 1;
        CLectEscrMatrices.b = CHorner.a;
        CLectEscrMatrices.obent = objes;
        CLectEscrMatrices.lb = listBox1;
        CLectEscrMatrices.LeeVector();
        listBox1.Items.Add("CALCULO DE RAICES REALES Y/O COMPLEJAS DE LA FUNCION
POLINOMICA DE GRADO " + CHorner.n.ToString());
        CLectEscrMatrices.EscribeVector(" DE COEFICIENTES");
        CRaicesReales.a = liminf;
        CRaicesReales.b = limsup;
        CRaicesReales.dx = delta;
        CRaicesReales.maxIter = 100;
        CRaicesReales.precision = 1e-8;
        CRaicesReales.lb = listBox1;
        CRaicesReales.punf = CHorner.horner;
        CRaicesReales.punfp = CHorner.fphorner;
        xi = liminf;
        xd = xi;
       do
       {
         CRaicesReales.xi = xi;
         CRaicesReales.xd = xd;
         estado = CRaicesReales.LocCambioSigno();
         xi = CRaicesReales.xi;
         xd = CRaicesReales.xd;
         if (estado < 0)
         {
           Raiz = CRaicesReales.Muller();
           /* DEFLACIONAR EL POLINOMIO: */
           for (I = 1; I < CHorner.n; I++)
             CHorner.a[I] += CHorner.a[I - 1] * Raiz;
           CHorner.n--;
         }
         xi = xd;
       }
       while ((xd += delta) < limsup);
       // Cerrar flujo de entrada:
       objes.CerrarLectura();
       if (CHorner.n % 2 == 1)
       {
        listBox1.Items.Add("NO SE CALCULARON TODAS LAS RAICES REALES");
        return;
       }
       if (CHorner.n > 1)
       {
         CRaicesRealesComplejas.n = CHorner.n;
         CRaicesRealesComplejas.lb = CRaicesReales.lb;
         CRaicesRealesComplejas.maxIter = CRaicesReales.maxIter;
         CRaicesRealesComplejas.precision = CRaicesReales.precision;
         CRaicesRealesComplejas.A = new double[CHorner.n + 1];
         CRaicesRealesComplejas.B = new double[CHorner.n + 1];
         CRaicesRealesComplejas.n = CHorner.n;
```

```
        for (I = 0; I <= CHorner.n; I++)
          CRaicesRealesComplejas.A[I] = CHorner.a[I];
        while (CRaicesRealesComplejas.n > 2)
          CRaicesRealesComplejas.NewtonBairstow();
        /* POLINOMIO RESTANTE, CALCULAR LAS RAICES SON REALES O COMPLEJAS */
        double P1 = -CRaicesRealesComplejas.A[1] / 2 / CRaicesRealesComplejas.A[0];
        double P2 = CRaicesRealesComplejas.A[1] * CRaicesRealesComplejas.A[1]- 4 *
CRaicesRealesComplejas.A[0] * CRaicesRealesComplejas.A[2];
        if (P2 < 0)
          listBox1.Items.Add("RAICES COMPLEJAS = " + P1.ToString() + " +/- " + (Math.Sqrt(-P2) / 2 /
CRaicesRealesComplejas.A[0]).ToString() + " * I");
        else
          listBox1.Items.Add("RAICES REALES MULTIPLES O CERCANAS: " + (P1 - Math.Sqrt(P2) / 2
/ CRaicesRealesComplejas.A[0]).ToString() + " Y " + (P1 + Math.Sqrt(P2) / 2 /
CRaicesRealesComplejas.A[0]).ToString());
        }
    }
  }
  catch (Exception ex)
 {
    MessageBox.Show(ex.Message);
 }
}
```

Botón Guardar Como …:

```
  private void b_guardar_como_Click(object sender, EventArgs e)
      {
          CListBox.lb = listBox1;
          CListBox.GuardarComo("c:\\");
      }
```

Botón Limpiar:

```
  private void b_limpiar_Click(object sender, EventArgs e)
      {
          CListBox.lb = listBox1;
          CListBox.Limpiar();
      }
```

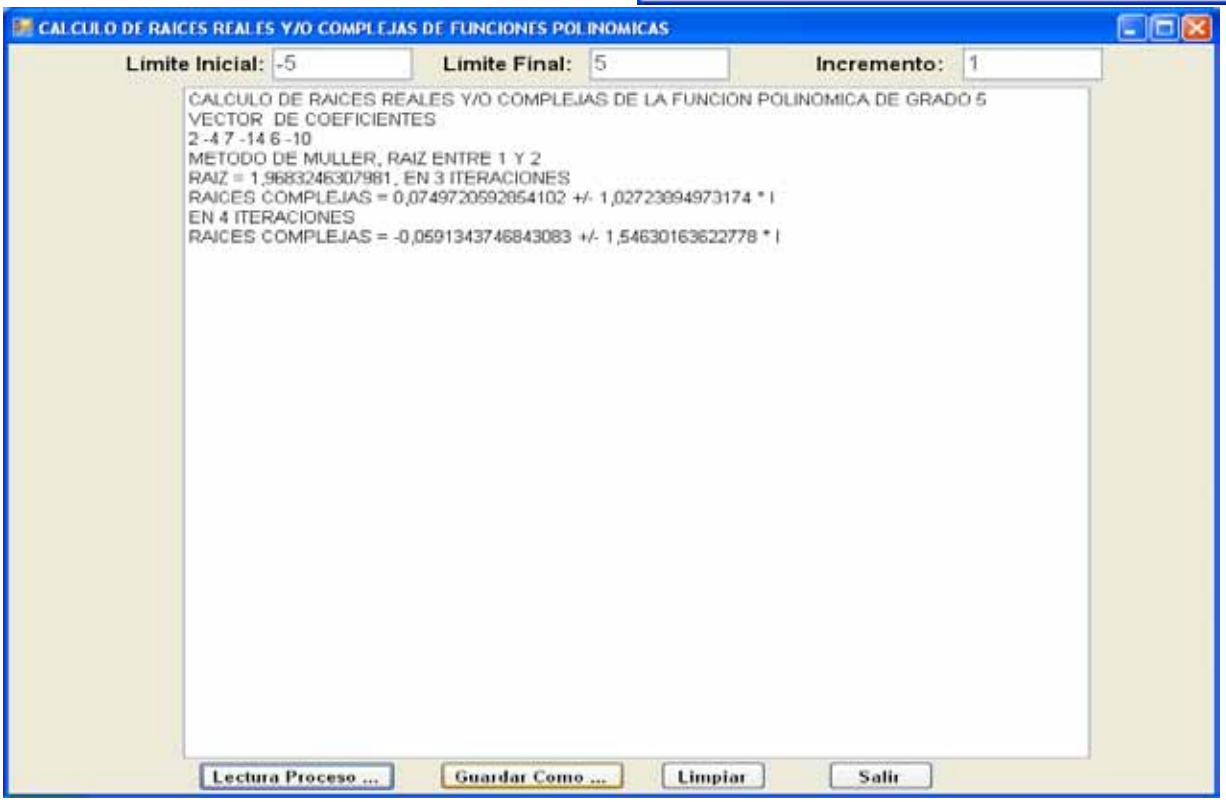
Botón Salir:

```
   private void b_salir_Click(object sender, EventArgs e)
      {
          this.Close();
          this.Dispose();
      }
```

Las referencias a las clases CEntSalArchivos, CRaicesReales, CRaicesRealesComplejas, CHorner, CLectEscrMatrices y CListBox, provienen de la

definición de las respectivas clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:



EJERCICIOS DEL CAPITULO II

1.-Dada la ecuación: $F(x) = 2\cos(3x) - 5e^{-x} * \sin(x) = 0$
Calcular una raíz real en el dominio, -pi <= x <= pi usando un método gráfico.

2.-. Para la ecuación y rango del ejercicio (1), localizar los cambios de signo para un incremento en las abscisas de uno.

3.-. Calcular una raíz real para cualquiera de los cambios de signo del ejercicio 2, por el método de la bisección usando una precisión de 1E-1. Indicar en cuantas iteraciones se produce la convergencia.

4.-. Calcular la raíz real para cualquiera de los cambios de signo del ejercicio 2, por el método de aproximaciones sucecivas usando una precisión de 1E-1. Indicar en cuantas iteraciones se produce la convergencia.

5.-. Calcular una raíz real para el cambio de signo del ejercicio 4, por el método de aproximaciones sucesivas modificado usando una precisión de 1E-2. Indicar en cuantas iteraciones se produce la convergencia.

6.-. Calcular una raíz real para el cambio de signo del ejercicio 5, por el método de regula - falsi usando una precisión de 1E-1. Indicar en cuantas iteraciones se produce la convergencia.

7.-. Calcular una raíz real para el cambio de signo del ejercicio 6, por el método de la secante usando una precisión de 1E-1. Indicar en cuantas iteraciones se produce la convergencia.

8.-. Calcular una raíz real para el cambio de signo del ejercicio 7, por el método de Newton - Raphson usando una precisión de 1E-2. Indicar en cuantas iteraciones se produce la convergencia.

9.-. Calcular una raíz real para el cambio de signo del ejercicio 8, por el método de Muller usando una precisión de 1E-2. Indicar en cuantas iteraciones se produce la convergencia.

10.-. Indicar una ventaja y una desventaja de los métodos para calcular raíces reales (Bisección, Secante, Newton – Raphson y Muller). A su criterio cual método le parece el más eficiente y el menos eficiente; por que?.
.

11.- Para la ecuación indicada en la pregunta 1, ejecutar los aplicativos correspondientes desarrollados en Matlab y Visual C#.

12.- Considerar la siguiente ecuación $F(x) = x^5 - 4x^4 + 2x^3 - 6x + 5 = 0$. Calcular los cambios de signo en el dominio, $-2 <= x <= 4$ con un incremento de 1, usando el método de Horner.

13.- Calcular una raíz real para cualquiera de los cambios de signo del ejercicio 12, por el método de Newton - Raphson - Horner usando una precisión de 1E-2. Indicar en cuantas iteraciones se produce la convergencia.

14.- Considerar la siguiente ecuación $F(x) = (2x^3 - 6x + 5) / (3x^3 + 2x^2 - 7) = 0$. Como se puede observar, la función F(x) tiene discontinuidades (divisiones para cero) y existen raíces reales en el dominio, $-3 <= x <= 2$. Determinar el intervalo en el que se produce la discontinuidad y luego calcular una raíz real por el método de Newton - Raphson - Horner usando una precisión de 1E-2. Indicar en cuantas iteraciones se produce la convergencia.

15.- Desarrollar un programa en Matlab y otro Visual C#.NET para ingresar desde la interface de usuario un intervalo dado por XI y XD y desde archivos, los coeficientes de los polinomios del problema del numeral 14. A continuación el programa debe buscar a) Una discontinuidad, en [XI, XD], en el caso de existir, debe desplegar el mensaje de error apropiado y finalizar. b) localizar la primera raíz real en [XI, XD] y calcularla con una precisión de 1E-8 por el método de Horner – Muller y desplegarla. c) en el caso de no existir la raíz en [XI, XD], desplegar el mensaje apropiado y finalizar. Ejecutar los programas con un grupo de datos que coincidan con los casos (a), (b) y (c).

16.- Considerar la siguiente ecuación $F(x) = 3x^4 + 4x^3 + 2x^2 - 6x + 5 = 0$.Calcular las dos primeras raíces complejas por el método de Newton - Bairstow, empleando una presición de 1e-5. Indicar en cuantas iteraciones se produce la convergencia.

17.- Ejecutar los programas en Matlab y Visual C#.NET para el cálculo de raíces complejas, ingresando los datos del polinomio del numeral 16.

3

ANÁLISIS MATRICIAL

Generalidades

En este capítulo, se analizarán los siguientes temas:

- Generalidades.
- Resolución de sistemas de ecuaciones lineales y simultáneas.
- Cálculo del determinante de una matriz.
- Cálculo de la matriz inversa.
- Caso particular en la resolución de sistemas de ecuaciones lineales y simultáneas: sistemas simétricos.
- Caso particular en la resolución de sistemas de ecuaciones lineales y simultáneas: sistemas simétricos en banda.
- Caso particular en la resolución de sistemas de ecuaciones lineales y simultáneas: sistemas simétricos del tipo Sky - Line.

Ecuación Lineal:

Es una secuencia de términos cuyas variables o incógnitas están elevados a la potencia uno, cero, y no están afectadas por otras variables o incógnitas.
Ejemplos:
$2x + 3y – 2z = 37$ es una ecuación lineal; $-8x + 2xz = 3$ no es una ecuación lineal.

Sistema de Ecuaciones Lineales y Simultáneas:

Es una agrupación de ecuaciones lineales para las que el número de ecuaciones es igual al número de incógnitas
Ejemplo:
$2x – 3y – 2z = 37$
$-18x – 16 y + 12z=18$
$2y – 4z = 3$
Es un sistema lineal de tres ecuaciones con tres incógnitas.

Resolución de Sistemas de Ecuaciones Lineales y Simultáneas.

Se presentan tres casos en la solución de este tipo de sistemas, para ilustrar esto, se proponen los siguientes ejemplos, mismos que se describen en la figura 3.1.

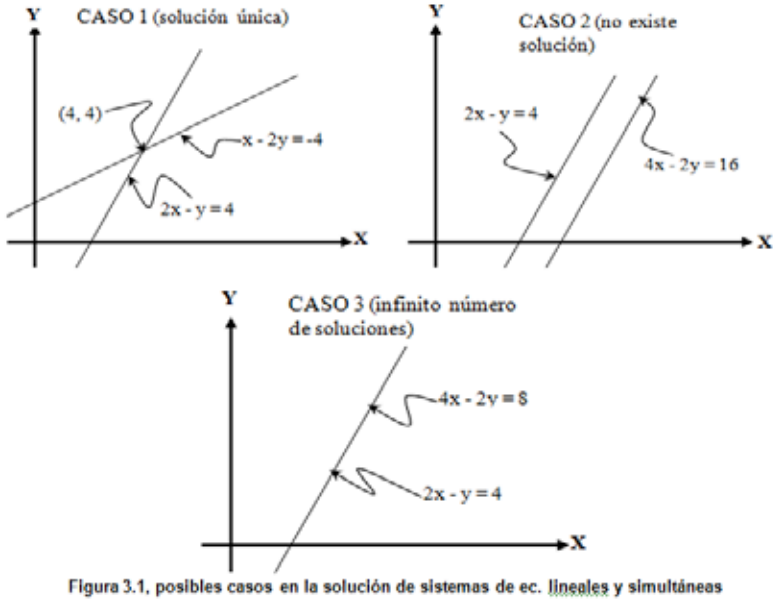


Figura 3.1, posibles casos en la solución de sistemas de ec. lineales y simultáneas

Para el propósito de este capítulo, se centra en el caso 1.

$$\text{Al sistema : } \begin{cases} 2x - y = 4 \\ x - 2y = -4 \end{cases}, \text{ se lo puede plantear : } \begin{pmatrix} 2 & -1 \\ 1 & -2 \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} 4 \\ -4 \end{pmatrix}$$

$$\text{Haciendo : } A = \begin{pmatrix} 2 & -1 \\ 1 & -2 \end{pmatrix};\ X = \begin{pmatrix} x \\ y \end{pmatrix};\ B = \begin{pmatrix} 4 \\ -4 \end{pmatrix}$$

Planteamiento Matricial de un Sistema de Ecuaciones Lineales Simultáneas

AX = B (3.1)

Sea A^{-1} la matriz inversa de A, entonces multiplicando por la izquierda A^{-1} la ecuación (3.1):
$A^{-1}AX = A^{-1}B$, como A^{-1} A = I = matriz identidad y a su vez IX = X, entonces:
$X = A^{-1}B$ (3.2), la cual representa la solución del sistema lineal de ecuaciones

$$\text{Si } A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}, \text{ entonces } A^{-1} = \frac{1}{D}\begin{pmatrix} a_{22} & -a_{12} \\ -a_{21} & a_{11} \end{pmatrix} \quad (3.3)$$

simultáneas.

Donde D = determinante de la matriz A = $a_{11}a_{22} - a_{12}a_{21}$

Volviendo al ejemplo planteado, para una matriz de 2 por 2:

$$\text{Dado que } A = \begin{pmatrix} 2 & -1 \\ 1 & -2 \end{pmatrix} \text{ y } B = \begin{pmatrix} 4 \\ -4 \end{pmatrix}, \text{entonces } A^{-1} = -\frac{1}{3}\begin{pmatrix} -2 & 1 \\ -1 & 2 \end{pmatrix}$$

$$\text{Aplicando la relación (2):}$$

$$X = \begin{pmatrix} x \\ y \end{pmatrix} = -\frac{1}{3}\begin{pmatrix} -2 & 1 \\ -1 & 2 \end{pmatrix}\begin{pmatrix} 4 \\ -4 \end{pmatrix} = -\frac{1}{3}\begin{pmatrix} -12 \\ -12 \end{pmatrix} = \begin{pmatrix} 4 \\ 4 \end{pmatrix}$$

Resolución de sistemas de ecuaciones lineales y simultáneas.

Existen varios métodos para la resolución de sistemas de ecuaciones lineales y simultáneas, los que analizaremos en este capítulo son:

- Método de Gauss.
- Método de Gauss - Jordán.
- Método de Gauss - Seidel.
- Método de Crout.

Método de Gauss

En este método, se puede calcular el determinante de la matriz de coeficientes; se aplican dos fases:

- Eliminación Gaussiana y
- Retrosustitución.

Fase de Eliminación Gaussiana:

Consiste en transformar la matriz de coeficientes aplicando transformaciones elementales, generándose una matriz triangular superior.

Fase de Retrosustitución

Una vez realizada la eliminación gaussiana, se despejan desde la última ecuación, la última incógnita, seguidamente de la penúltima ecuación se despeja la penúltima incógnita y se sustituye el valor de la última incógnita, y así sucesivamente hasta llegar a la primera ecuación.

Ejemplo 3.1.

Dado el sistema:

$a_{11}\, x_1 + a_{12}\, x_2 + a_{13}\, x_3 = b_1$
$a_{21}\, x_1 + a_{22}\, x_2 + a_{23}\, x_3 = b_2$
$a_{31}\, x_1 + a_{32}\, x_2 + a_{33}\, x_3 = b_3$

Resolverlo por el método de Gauss y calcular el determinante de la matriz de coeficientes:

Solución:

Deben darse los siguientes pasos:
- Planteamiento matricial

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

- Eliminación Gaussiana:

Por facilidad, se designa a la fila i como: F_I. En la primera etapa, se aplican las siguientes transformaciones elementales:

$m_1 = -a_{21}/a_{11}$; $F_2 \Rightarrow F_2 + m_1 * F_1$; $m_2 = -a_{31}/a_{11}$; $F_3 \Rightarrow F_3 + m_2 * F_1$

$a_{21} = a_{21} + (-a_{21} / a_{11})a_{11} = 0$; $a_{22} = a_{22} + m_1 * a_{12} = a_{22}^{(1)}$; $b_2 = b_2 + m_1 * b_1 = b_2^{(1)}$

$a_{31} = a_{31} + (-a_{31} / a_{11})a_{11} = 0$; $a_{32} = a_{32} + m_2 * a_{12} = a_{32}^{(1)}$; $b_3 = b_3 + m_2 * b_1 = b_2^{(1)}$

Sustituyendo estos transformaciones en el sistema:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(1)} \\ b_3^{(1)} \end{pmatrix}; \quad a_{ij}^{(k)} = \text{elemento de la fila i, columna j, etapa de cálculo k}$$

En la segunda etapa, se aplican las siguientes transformaciones elementales:

$m_1^{(2)} = -a_{32}^{(1)} / a_{22}^{(1)}$; $F_3 \Rightarrow F_3 + m_1^{(2)} * F_2$

$a_{32}^{(1)} = a_{32}^{(1)} + (-a_{32}^{(1)} / a_{22}^{(1)})\, a_{22}^{(1)} = 0$; $a_{33}^{(1)} = a_{33}^{(1)} + m_1^{(2)} * a_{23}^{(1)} = a_{33}^{(2)}$;

$b_3^{(1)} = b_3^{(1)} + m_1^{(2)} * b_2^{(1)} = b_3^{(2)}$

Sustituyendo estos transformaciones en el sistema:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} \\ 0 & 0 & a_{33}^{(2)} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2^{(1)} \\ b_3^{(2)} \end{pmatrix}$$

-Cálculo del determinante:
Det = $a_{11} * a_{22}^{(1)} * a_{33}^{(2)}$

-Retrosustitución:
$a_{33}^{(2)}x_3 = b_3^{(2)} \Rightarrow x_3 = b_3^{(2)} / a_{33}^{(2)}$
$a_{22}^{(1)}x_2 + a_{23}^{(1)}x_3 = b_2^{(1)} \Rightarrow x_2 = (b_2^{(1)} - a_{23}^{(1)}x_3)/a_{22}^{(1)}$
$a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \quad x_1 = (b_1 - a_{12}x_2 - a_{13}x_3)/a_{11}$

Ejemplo 3.2.

Dado el sistema de ecuaciones lineales y simultáneas:

$2x_1 - x_2 + 3x_3 + x_4 = 1$
$-x_1 + 4x_2 + 2x_3 + x_4 = 2$
$3x_1 + 2x_2 - 2x_3 - x_4 = 3$
$x_1 + x_2 - x_3 + 2x_4 = -2$

Resolverlo por el método de Gauss y calcular el determinante de la matriz de coeficientes. Expresar las operaciones y resultados en números racionales.

Solución:

-Planteamiento matricial:

$$\begin{pmatrix} 2 & -1 & 3 & 1 \\ -1 & 4 & 2 & 1 \\ 3 & 2 & -2 & -1 \\ 1 & 1 & -1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ -2 \end{pmatrix}$$

- Eliminación Gaussiana:

En la primera etapa, se aplican las siguientes transformaciones elementales:
$m_1 = 1/2$; $F_2 \rightarrow F_2 + m_1 * F_1$; $m_2 = -3/2$; $F_3 \rightarrow F_3 + m_2 * F_1$; $m_3 = -1/2$; $F_4 \rightarrow F_4 + m_3 * F_1$
$a_{21} = -1 + (1/2)*2 = 0$; $a_{22} = 4 + (1/2)*(-1) = 7/2$; $a_{23} = 2 + (1/2)*3 = 7/2$;
$a_{24} = 1 + (1/2)*1 = 3/2$; $b_2 = 2 + (1/2)*1 = 5/2$;
$a_{31} = 3 + (-3/2)*2 = 0$; $a_{32} = 2 + (-3/2)*(-1) = 7/2$; $a_{33} = -2 + (-3/2)*3 = -13/2$;
$a_{34} = -1 + (-3/2)*1 = -5/2$; $b_3 = 3 + (-3/2)*1 = 3/2$;
$a_{41} = 1 + (-1/2)*2 = 0$; $a_{42} = 1 + (-1/2)*(-1) = 3/2$; $a_{43} = -1 + (-1/2)*3 = -5/2$;
$a_{44} = 2 + (-1/2)*1 = 3/2$; $b_4 = -2 + (-1/2)*1 = -5/2$;

Sustituyendo estos transformaciones en el sistema:

$$\begin{pmatrix} 2 & -1 & 3 & 1 \\ 0 & 7/2 & 7/2 & 3/2 \\ 0 & 7/2 & -13/2 & -5/2 \\ 0 & 3/2 & -5/2 & 3/2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 5/2 \\ 3/2 \\ -5/2 \end{pmatrix}$$

En la segunda etapa, se aplican las siguientes transformaciones elementales:
$m_1 = -a_{32}/a_{22} = -1$; $F_3 \Rightarrow F_3 + m_1 * F_2$; $m_2 = -a_{42}/a_{22} = -3/7$; $F_4 \Rightarrow F_4 + m_2 * F_2$
$a_{32} = 7/2 - 1*(7/2) = 0$; $a_{33} = -13/2 - 1*(7/2) = -10$; $a_{34} = -5/2 - 1*(3/2) = -4$;
$b_3 = 3/2 - 1*(5/2) = -1$;
$a_{42} = 3/2 + (-3/7)*(7/2) = 0$; $a_{43} = -5/2 + (-3/7)*(7/2) = -4$; $a_{44} = 3/2 + (-3/7)*(3/2) = 6/7$;
$b_4 = -5/2 + (-3/7)*(5/2) = -25/7$;

Sustituyendo estos transformaciones en el sistema:

$$\begin{pmatrix} 2 & -1 & 3 & 1 \\ 0 & 7/2 & 7/2 & 3/2 \\ 0 & 0 & -10 & -4 \\ 0 & 0 & -4 & 6/7 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 5/2 \\ -1 \\ -25/7 \end{pmatrix}$$

En la tercera etapa, se aplican las siguientes transformaciones elementales:
$m_1 = -a_{43}/a_{33} = -2/5$; $F_4 \Rightarrow F_4 + m_1 * F_3$
$a_{43} = -4 + (-2/5)*(-10) = 0$; $a_{44} = 6/7 + (-2/5)*(-4) = 86/35$; $b_4 = -25/7 + (-2/5)*(-1) = -$

$$\begin{pmatrix} 2 & -1 & 3 & 1 \\ 0 & 7/2 & 7/2 & 3/2 \\ 0 & 0 & -10 & -4 \\ 0 & 0 & 0 & 86/35 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 5/2 \\ -1 \\ -111/35 \end{pmatrix}$$

111/35;

-Cálculo del determinante:

Det = 2 * (7/2) * (-10) * (86/35) = -172

-Retrosustitución:

$(86/35) * x_4 = -111/35 \Rightarrow x_4 = (-111/35)/ (86/35) \Rightarrow x_4 = -111/86$
$-10x_3 - 4x_4 = -1 \Rightarrow x_3 = (1 - 4*(-111/86))/10 \Rightarrow x_3 = 53/86$
$(7/2)*x_2 + (7/2)*x_3 + (3/2)*x_4 = 5/2 \Rightarrow x_2 = (5 - 7*(53/86) - 3*(-111/86))/7$
$\Rightarrow x_2 = 28/43$

$2x_1 - x_2 + 3x_3 + x_4 = 1 \Rightarrow x_1 = (1 + 28/43 - 3*(53/86) - (-111/86))/2$
$\Rightarrow x_1 = 47/86$

Por consiguiente el vector de soluciones es:

$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \frac{1}{86} \begin{pmatrix} 47 \\ 56 \\ 53 \\ -111 \end{pmatrix}$$

Deducción de fórmulas para el método de Gauss:
Considerar el siguiente sistema general de ecuaciones lineales y simultáneas:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1k} & \dots & a_{1j} & \dots & a_{1N} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2k} & \dots & a_{2j} & \dots & a_{2N} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3k} & \dots & a_{3j} & \dots & a_{3N} \\ & & & \dots & & & & & \\ a_{k1} & a_{k2} & a_{k3} & \dots & a_{kk} & \dots & a_{kj} & \dots & a_{kN} \\ & & & \dots & & & & & \\ a_{i1} & a_{i2} & a_{i3} & \dots & a_{ik} & \dots & a_{ij} & \dots & a_{iN} \\ & & & \dots & & & & & \\ a_{N1} & a_{N2} & a_{N3} & \dots & a_{Nk} & \dots & a_{Nj} & \dots & a_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_k \\ \dots \\ x_i \\ \dots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_k \\ \dots \\ b_i \\ \dots \\ b_N \end{pmatrix}$$

En la etapa k - 1, de la eliminación gaussiana, el estado del sistema es:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1k} & \dots & a_{1j} & \dots & a_{1N} \\ 0 & a_{22} & a_{23} & \dots & a_{2k} & \dots & a_{2j} & \dots & a_{2N} \\ 0 & 0 & a_{33} & \dots & a_{3k} & \dots & a_{3j} & \dots & a_{3N} \\ & & & \dots & & & & & \\ 0 & 0 & 0 & \dots & a_{kk} & \dots & a_{kj} & \dots & a_{kN} \\ & & & \dots & & & & & \\ 0 & 0 & 0 & \dots & a_{ik} & \dots & a_{ij} & \dots & a_{iN} \\ & & & \dots & & & & & \\ 0 & 0 & 0 & \dots & a_{Nk} & \dots & a_{Nj} & \dots & a_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_k \\ \dots \\ x_i \\ \dots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_k \\ \dots \\ b_i \\ \dots \\ b_N \end{pmatrix}$$

En esta etapa, se aplican las siguientes transformaciones elementales:

$m = -a_{ik}/a_{kk}$; $F_i \Rightarrow F_i + m * F_k$

Con lo que: $a_{ij} = a_{ij} + m * a_{kj}$; $b_i = b_i + m * b_k$

Una vez que se ha transformado la matriz de coeficientes, mediante la fase de la eliminación gaussiana, se puede calcular el determinante, efectuando el producto de todos los elementos de la diagonal principal, con lo que: $Det = a_{11} * a_{22} * a_{33} * \dots * a_{ii} * a_{NN} = \prod a_{ii}$

En la fase de retrosustitución se toma la ecuación k:

$a_{kk}x_k + a_{k(k+1)}x_{(k+1)} + \dots + a_{kj}x_j + \dots + a_{kN}x_N = b_k$

De esta ecuación se despeja x_k:

$$x_k = (b_k - (a_{k(k+1)}x_{k+1} + \dots + a_{kj}x_j + \dots + a_{kN}x_N))/a_{kk} = (b_k - \sum_{j=k+1}^{N} a_{kj}x_j)/a_{kk}$$

A fin de efectuar el planteamiento algorítmico, todas las operaciones se realizan sobre la matriz de coeficientes (A) y vector de términos independientes (B), por lo que el vector de incógnitas (X), se procesa sobre el vector (B). Por consiguiente se tiene el siguiente resumen de fórmulas para el método de Gauss:

Fase de eliminación gaussiana:

$$m = -a_{ik} / a_{kk} \quad (3.4); \quad a_{ij} = a_{ij} + m * a_{kj} \quad (3.5); b_i = b_i + m * b_k \quad (3.6)$$

$$\forall k = 1, 2, 3, \ldots, N-1; \forall i = k+1, k+2, \ldots, N; \forall j = k, k+1, k+2, \ldots, N;$$

$$\text{Deter.} = a_{11} * a_{22} * a_{33} * \ldots * a_{NN} = \prod_{i=1}^{N} a_{ii} \quad (3.7)$$

Fase de retrosustitución :

$$b_k = (b_k - \sum_{j=k+1}^{N} a_{kj} b_j) / a_{kk} \quad (3.8)$$

$$\forall k = N, N-1, \ldots, 3, 2, 1$$

Algoritmo para el método de Gauss:

a) Transformar la matriz de coeficientes (A) y vector de términos independientes (B) aplicando las fórmulas (3.4), (3.5) y (3.6).
b) Calcular el determinante de la matiz, aplicando la fórmula (3.7).
c) Calcular el vector de incógnitas (B), aplicando la fórmula (3.8).

Método de Gauss - Jordan

Consiste en transformar la ecuación matricial: AX = B en IX =C, entonces X = C, a través de operaciones elementales. Este método también permite el cálculo del determinante, efectuando el producto de los elementos de la diagonal principal, conforme se va transformando la matriz de coeficientes a la matriz identidad.

Ejemplo 3.3.

Dado el sistema:
$a_{11} x_1 + a_{12} x_2 + a_{13} x_3 = b_1$
$a_{21} x_1 + a_{22} x_2 + a_{23} x_3 = b_2$
$a_{31} x_1 + a_{32} x_2 + a_{33} x_3 = b_3$

Resolverlo por el método de Gauss - Jordan y calcular el determinante de la matriz de coeficientes:

Solución:

Deben darse los siguientes pasos:

- Planteamiento matricial

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

- Transformación de la matriz de coeficientes a la matriz identidad y del vector de términos independientes (B) al vector de soluciones (C). Cálculo del determinante:

Al igual que en el método de Gauss, se designa a la fila i como: F_I. En la primera etapa, se aplican las siguientes transformaciones elementales:

Det = a_{11}; m = a_{11}; $F_1 \Rightarrow F_1 / m$; $m_1 = a_{21}$: $F_2 \Rightarrow F_2 - m_1 * F_1$;
$m_2 = a_{31}$: $F_3 \Rightarrow F_3 - m_2 * F_1$
$a_{11} = a_{11} / m = 1$; $a_{12} = a_{12} / m = a_{12}^{(1)}$; $a_{13} = a_{13} / m = a_{13}^{(1)}$;$b_1 = b_1 / m = b_1^{(1)}$
$a_{21} = a_{21} - m_1 * 1 = 0$; $a_{22} = a_{22} - m_1 * a_{12}^{(1)} = a_{22}^{(1)}$; $a_{23} = a_{23} - m_1 * a_{13}^{(1)} = a_{23}^{(1)}$;
$b_2 = b_2 - m_1 * b_1^{(1)} = b_2^{(1)}$
$a_{31} = a_{31} - m_2 * 1 = 0$; $a_{32} = a_{32} - m_2 * a_{12}^{(1)} = a_{32}^{(1)}$; $a_{33} = a_{33} - m_2 * a_{13}^{(1)} = a_{33}^{(1)}$;
$b_3 = b_3 - m_2 * b_1^{(1)} = b_3^{(1)}$

Sustituyendo estos transformaciones en el sistema:

$$\begin{pmatrix} 1 & a_{12}^{(1)} & a_{13}^{(1)} \\ 0 & a_{22}^{(1)} & a_{23}^{(1)} \\ 0 & a_{32}^{(1)} & a_{33}^{(1)} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1^{(1)} \\ b_2^{(1)} \\ b_3^{(1)} \end{pmatrix}$$

En la segunda etapa, se aplican las siguientes transformaciones elementales:
Det = Det*$a_{22}^{(1)}$; m = $a_{22}^{(1)}$; $F_2 \Rightarrow F_2 / m$; $m_1 = a_{12}^{(1)}$: $F_1 \Rightarrow F_1 - m_1 * F_2$;
$m_2 = a_{32}^{(1)}$: $F_3 \Rightarrow F_3 - m_2 * F_2$

$a_{22}^{(1)} = a_{22}^{(1)} / m = 1$; $a_{23}^{(1)} = a_{23}^{(1)} / m = a_{23}^{(2)}$; $b_2^{(1)} = b_2^{(1)} / m = b_2^{(2)}$
$a_{12}^{(1)} = a_{22}^{(1)} - m_1 * 1 = 0$; $a_{13}^{(1)} = a_{13}^{(1)} - m_1 * a_{23}^{(2)} = a_{13}^{(2)}$;
$b_1^{(1)} = b_1^{(1)} - m_1 * b_2^{(2)} = b_1^{(2)}$;
$a_{32}^{(1)} = a_{32}^{(1)} - m_2 * 1 = 0$; $a_{33}^{(1)} = a_{33}^{(1)} - m_2 * a_{23}^{(2)} = a_{33}^{(2)}$;
$b_3^{(1)} = b_3^{(1)} - m_2 * b_2^{(2)} = b_3^{(2)}$;

Sustituyendo estos transformaciones en el sistema:

$$\begin{pmatrix} 1 & 0 & a_{13}^{(2)} \\ 0 & 1 & a_{23}^{(2)} \\ 0 & 0 & a_{33}^{(2)} \end{pmatrix}\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1^{(2)} \\ b_2^{(2)} \\ b_3^{(2)} \end{pmatrix}$$

En la tercera etapa, se aplican las siguientes transformaciones elementales:

Det = Det*$a_{33}^{(2)}$; m = $a_{33}^{(2)}$; F_3 ➠ F_3 / m; $m_1 = a_{13}^{(2)}$: F_1 ➠ F_1 - m_1 * F_3;
$m_2 = a_{33}^{(2)}$: F_2 ➠ F_2 - m_2 * F_3

$a_{33}^{(2)} = a_{33}^{(2)} / m = 1$; $b_3^{(2)} = b_3^{(2)} / m = b_3^{(3)}$
$a_{13}^{(2)} = a_{13}^{(2)} - m_1 * 1 = 0$; $b_1^{(2)} = b_1^{(2)} - m_1 * b_3^{(3)} = b_1^{(3)}$;
$a_{23}^{(2)} = a_{32}^{(2)} - m_2 * 1 = 0$; $b_2^{(2)} = b_2^{(2)} - m_2 * b_3^{(3)} = b_3^{(23)}$;

Sustituyendo estos transformaciones en el sistema:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1^{(3)} \\ b_2^{(3)} \\ b_3^{(3)} \end{pmatrix}$$

Ejemplo 3.4.

Resolver el sistema de ecuaciones lineales y simultáneas, propuesto en ejemplo 3.3. Expresar las operaciones y resultados en números racionales.

Solución:

-Planteamiento matricial:

$$\begin{pmatrix} 2 & -1 & 3 & 1 \\ -1 & 4 & 2 & 1 \\ 3 & 2 & -2 & -1 \\ 1 & 1 & -1 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ -2 \end{pmatrix}$$

- Transformación de la matriz de coeficientes a la matriz identidad y del vector de términos independientes (B) al vector de soluciones (C). Cálculo del determinante:

En la primera etapa, se aplican las siguientes transformaciones elementales:
Det = 2; m = 2; F_1 ➠ F_1 / 2; m_1 = -1: F_2 ➠ F_2 + F_1; m_2 = 3: F_3 ➠ F_3 - 3 * F_1;
m_3 = 1: F_4 ➠ F_4 - F_1
$a_{11} = 2 / 2 = 1$; $a_{12} = -1 / 2$; $a_{13} = 3 / 2$; $a_{14} = 1 / 2$; $b_1 = 1 / 2$
$a_{21} = -1 + 1 = 0$; $a_{22} = 4 - 1/2 = 7/2$; $a_{23} = 2 + 3/2 = 7/2$; $a_{24} = 1 + 1/2 = 3/2$;
$b_2 = 2 + 1/2 = 5/2$;

$a_{31} = 3 - 3\ (1) = 0$; $a_{32} = 2 - 3 * (-1/2) = 7/2$; $a_{33} = -2 - 3 * (3/2) = -13/2$;
$a_{34} = -1 - 3 * (1/2) = -5/2$; $b_3 = 3 - 3 * (1/2) = 3/2$;
$a_{41} = 1 - 1 = 0$; $a_{42} = 1 - (-1/2) = 3/2$; $a_{43} = -1 - 3/2 = -5/2$; $a_{44} = 2 - 1/2 = 3/2$;
$b_4 = -2 - 1/2 = -5/2$;
Sustituyendo estos transformaciones en el sistema:

$$\begin{pmatrix} 1 & -1/2 & 3/2 & 1/2 \\ 0 & 7/2 & 7/2 & 3/2 \\ 0 & 7/2 & -13/2 & -5/2 \\ 0 & 3/2 & -5/2 & 3/2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1/2 \\ 5/2 \\ 3/2 \\ -5/2 \end{pmatrix}$$

En la segunda etapa, se aplican las siguientes transformaciones elementales:

Det = Det * (7/2) = 2 * (7/2) = 7; m = 7/2; $F_2 \Rightarrow F_2 * 2/7$; $F_1 \Rightarrow F_1 + F_2/2$;
$F_3 \Rightarrow F_3 - (7/2) * F_2$; $F_4 \Rightarrow F_4 - (3/2) * F_1$
$a_{22} = (2/7) * 7/2 = 1$; $a_{23} = (2/7) * 7/2 = 1$; $a_{24} = (2/7) * 3/2 = 3/7$; $b_2 = (2/7) * 5/2 = 5/7$;
$a_{12} = -1/2 + (1/2) * (1) = 0$; $a_{13} = 3/2 + (1/2)*(1) = 2$; $a_{14} = 1/2 + (1/2)*(3/7) = 5/7$;
$b_1 = 1/2 + (1/2)*(5/7) = 6/7$;
$a_{32} = 7/2 - (7/2) * (1) = 0$; $a_{33} = -13/2 - (7/2) * (1) = -10$; $a_{34} = -5/2 - (7/2) * (3/7) = -4$;
$b_3 = 3/2 - (7/2) * (5/7) = -1$;
$a_{42} = 3/2 + (-3/2) * (1) = 0$; $a_{43} = -5/2 + (-3/2) * (1) = -4$; $a_{44} = 3/2 + (-3/2) * (3/7) = 6/7$;
$b_4 = -5/2 + (-3/2) * (5/7) = -25/7$;

Sustituyendo estos transformaciones en el sistema:

$$\begin{pmatrix} 1 & 0 & 2 & 5/7 \\ 0 & 1 & 1 & 3/7 \\ 0 & 0 & -10 & -4 \\ 0 & 0 & -4 & 6/7 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 6/7 \\ 5/7 \\ -1 \\ -25/7 \end{pmatrix}$$

En la tercera etapa, se aplican las siguientes transformaciones elementales:

Det = Det * (-10) = 7 * (-10) = -70; m = -1/10; $F_3 \Rightarrow (-1/10) * F_3$;
$F_1 \Rightarrow F_1 - 2 * F_3$; $F_2 \Rightarrow F_2 - F_3$; $F_4 \Rightarrow F_4 + 4 * F_3$
$a_{33} = (-1/10) * (-10) = 1$; $a_{34} = (-1/10) * (-4) = 2/5$; $b_3 = (-1/10) * (-1) = 1/10$;
$a_{13} = 2 - 2 * (1) = 0$; $a_{14} = 5/7 - 2 * (2/5) = -3/35$; $b_1 = 6/7 - 2 * (1/10) = 23/35$;
$a_{23} = 1 - 1 = 0$; $a_{24} = 3/7 - 2/5 = 1/35$; $b_2 = 5/7 - 1/10 = 43/70$;
$a_{43} = -4 + 4*(1) = 0$; $a_{44} = 6/7 + 4 * (2/5) = 86/35$; $b_4 = -25/7 + 4 *(1/10) = -111/35$;

Sustituyendo estos transformaciones en el sistema:

$$\begin{pmatrix} 1 & 0 & 0 & -3/35 \\ 0 & 1 & 0 & 1/35 \\ 0 & 0 & 1 & 2/5 \\ 0 & 0 & 0 & 86/35 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 23/35 \\ 43/70 \\ 1/10 \\ -111/35 \end{pmatrix}$$

En la cuarta etapa, se aplican las siguientes transformaciones elementales:

Det = Det * (86/35) = -70 * (86/35) = -172; m = 35/86; $F_4 \Rightarrow (35/86) * F_4$;

$F_1 \Rightarrow F_1 + (3/35)* F_4$; $F_2 \Rightarrow F_2 - (1/35)* F_4$; $F_3 \Rightarrow F_3 - (2/5)* F_4$;
$a_{44} = (35/86) * (86/35) = 1$; $b_4 = (35/86) * (-111/35) = -111/86$;
$a_{14} = -3/35 + (3/35) * (1) = 0$; $b_1 = (23/35) + (3/35) * (-111/86) = 47/86$;
$a_{24} = 1/35 + (-1/35) * (1) = 0$; $b_2 = (43/70) - (1/35) * (-111/86) = 28/43$;
$a_{34} = 2/5 - (2/5) * (1) = 0$; $b_3 = (1/10) - (2/5) * (-111/86) = 53/86$;

Por consiguiente el vector de soluciones es:

$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \frac{1}{86} \begin{pmatrix} 47 \\ 56 \\ 53 \\ -111 \end{pmatrix}$$

Deducción de fórmulas para el método de Gauss - Jordan:

Considerar el siguiente sistema general de ecuaciones lineales y simultáneas:

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1k} & \dots & a_{1j} & \dots & a_{1N} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2k} & \dots & a_{2j} & \dots & a_{2N} \\ a_{31} & a_{32} & a_{33} & \dots & a_{3k} & \dots & a_{3j} & \dots & a_{3N} \\ & & & \dots & & & & & \\ a_{k1} & a_{k2} & a_{k3} & \dots & a_{kk} & \dots & a_{kj} & \dots & a_{kN} \\ & & & \dots & & & & & \\ a_{i1} & a_{i2} & a_{i3} & \dots & a_{ik} & \dots & a_{ij} & \dots & a_{iN} \\ & & & \dots & & & & & \\ a_{N1} & a_{N2} & a_{N3} & \dots & a_{Nk} & \dots & a_{Nj} & \dots & a_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_k \\ \dots \\ x_i \\ \dots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_k \\ \dots \\ b_i \\ \dots \\ b_N \end{pmatrix}$$

En la etapa k - 1, de la transformación de la matriz de coeficientes, el estado del sistema es:

$$\begin{pmatrix} 1 & 0 & 0 & \dots & a_{1k} & \dots & a_{1j} & \dots & a_{1N} \\ 0 & 1 & 0 & \dots & a_{2k} & \dots & a_{2j} & \dots & a_{2N} \\ 0 & 0 & 1 & \dots & a_{3k} & \dots & a_{3j} & \dots & a_{3N} \\ & & & & \dots & & & & \\ 0 & 0 & 0 & \dots & a_{kk} & \dots & a_{kj} & \dots & a_{kN} \\ & & & & \dots & & & & \\ 0 & 0 & 0 & \dots & a_{ik} & \dots & a_{ij} & \dots & a_{iN} \\ & & & & \dots & & & & \\ 0 & 0 & 0 & \dots & a_{Nk} & \dots & a_{Nj} & \dots & a_{NN} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \dots \\ x_k \\ \dots \\ x_i \\ \dots \\ x_N \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \dots \\ b_k \\ \dots \\ b_i \\ \dots \\ b_N \end{pmatrix}$$

En esta etapa, se aplican las siguientes transformaciones elementales:

Det = Det * a_{kk}; m = a_{kk}; F_k ➠ F_k / m; F_i ➠ F_i - a_{ik} * F_k

Con lo que: $a_{kj} = a_{kj} / m$; $a_{ij} = a_{ij} - a_{ik} * a_{kj}$; $b_i = b_i - a_{ik} * b_k$

Por consiguiente, las fórmulas para el método de Gauss - Jordan, quedan así:

$$m = a_{kk} \quad (3.11); \quad a_{kj} = a_{kj} / m \quad (3.12); a_{ij} = a_{ij} - a_{ik} * a_{kj} \quad (3.13); b_i = b_i - a_{ik} * b_k \quad (3.14)$$

$$\forall_{k = 1,2,3,\dots,N}; \forall i_{=1,2,3,\dots,N, i \neq k}; \forall j_{=k,k+1,k+2,\dots,N};$$

$$Deter. = a_{11} * a_{22} * a_{33} * \dots * a_{NN} = \prod_{k=1}^{N} a_{kk} \quad (3.15)$$

Algoritmo para el método de Gauss - Jordan:

a) Determinante = a_{11}
b) Afectar un elemento de la diagonal principal de la matriz de coeficientes como factor al determinante, aplicando la fórmula (3.15); transformar la matriz de coeficientes (A) y vector de términos independientes (B) aplicando las fórmulas (3.11), (3.12), (3.13) y (3.14).

Método de Gauss - Seidel

Es un método iterativo, en el que, de cada ecuación, se despeja una incógnita (la ubicada en la posición de la diagonal principal de la matriz de coeficientes), a continuación, se asignando valores iniciales a todas las incógnitas, se los reemplanza en las fórmulas, comenzando por la primera. Seguidamente los valores que resultan del cálculo de la aplicación de las fórmulas, se van reemplazando en la de la incógnita que sigue. Este último proceso se repite hasta cuando el máximo del valor absoluto de la diferencia entre dos valores consecutivos de la misma incógnita alcanza la precisión requerida.

Ejemplo 3.5.

Dado el sistema del ejemplo 3.2. Resolverlo por el método de Gauss - Seidel.

Solución:

$a_{11}\, x_1 + a_{12}\, x_2 + a_{13}\, x_3 = b_1 \quad (1)$
$a_{21}\, x_1 + a_{22}\, x_2 + a_{23}\, x_3 = b_2 \quad (2)$
$a_{31}\, x_1 + a_{32}\, x_2 + a_{33}\, x_3 = b_3 \quad (3)$

Despejando x_1, x_2 y x_3 de las ecuaciones (1), (2) y (3), respectivamente:

$x_1 = (b_1 - (a_{12}\, x_2 + a_{13}\, x_3))/a_{11} \quad (4)$
$x_2 = (b_2 - (a_{21}\, x_1 + a_{23}\, x_3))/a_{22} \quad (5)$
$x_3 = (b_3 - (a_{31}\, x_1 + a_{32}\, x_2))/a_{33} \quad (6)$

Seguidamente, se asigna un valor inicial a las incógnitas x_2 y x_3 , por ejemplo 0 y se reemplazan en (4), procediéndose al cálculo de $x_1 = b_1 / a_{11} = x_1^{(1)}$, este valor se reemplaza en (5), calculándose $x_2 = (b_2 - a_{21}\, x_1^{(1)})/a_{22} = x_2^{(1)}$, los valores de $x_1^{(1)}$ y $x_2^{(1)}$, se sustituyen en (6) y se calcula $x_3 = (b_3 - (a_{31}\, x_1^{(1)} + a_{32}\, x_2^{(1)}))/a_{33} = x_3^{(1)}$, y así sucesivamente, hasta que se cumpla la condición de convergencia. El proceso completo se muestra en la tabla siguiente:

Iteración	x_1	x_2	x_3	$MAX(\lvert x_i^{(k+1)} - x_i^{k} \rvert)$
0	0	0	0	
1	$x_1^{(1)}$	$x_2^{(1)}$	$x_3^{(1)}$	$valor_1$
2	$x_1^{(2)}$	$x_2^{(2)}$	$x_3^{(2)}$	$Valor_2$
. . .	. . .	. . .	. . .	. . .
k	$x_1^{(k)}$	$x_2^{(k)}$	$x_3^{(k)}$	$valor_k$

Donde $valor_k \leq$ PRECISION

Antes de resolver un ejemplo numérico, se analiza el comportamiento de este método, respecto de la condición de convergencia:

Se analiza el sistema:

x - 2y = -4 (1)
2x - y = 4 (2)

Despejando x de (1) e y de (2): x = 2y - 4; y = 2x - 4 , de esta manera se genera la tabla:

ITERACION	x	y	MAX($\|x_i^{(k+1)} - x_i^k\|$)
0	0	0	
1	-4	-12	24
2	-28	-60	96
3	-124	-252	384
4	-508	-1020	1536
5	-2044	-4092	6144
6	-8188	-16380	24576
7	-32764	-65532	98304
8	-131068	-262140	393216

Como se observa, la solución se aleja de la convergencia, intercambiando el sistema:

2x - y = 4 (1)
x - 2y = -4 (2)

Despejando x de (1) e y de (2): x = (y + 4)/2; y = (x + 4)/2 , de esta manera se genera la tabla:

ITERACION	x	y	MAX($\|x_i^{(k+1)} - x_i^k\|$)
0	0	0	
1	2	3	3
2	3.5	3.75	0.75
3	3.875	3.9375	0.1875
4	3.96875	3.984375	0.046875
5	3.9921875	3.99609375	0.01171875
6	3.998046875	3.999023438	0.002929688
7	3.999511719	3.999755859	0.000732422
8	3.99987793	3.999938965	0.000183105
9	3.999992371	3.999984741	4.57764E-05
10	3.999998093	3.999996185	1.14441E-05
11	3.999999523	3.999999046	2.86102E-06
12	3.999999881	3.999999762	7.15256E-07
13	3.99999997	3.99999994	1.78814E-07
14	3.999999993	3.999999985	4.47035E-08

Se ve que en esta vez, si se alcanza la condición de convergencia.

Como conclusión entonces se puede indicar que para alcanzar la condición de convergencia en la resolución de sistemas de ecuaciones lineales y simultáneas, por el método de Gauss - Seidel, la matriz de coeficientes del sistema, debe estar dispuesta, de una manera, tal que los valores en la diagonal principal deben ser mayores, en valor absoluto, respecto del resto de elementos en esa fila.

Ejemplo 3.6.

Resolver el sistema de ecuaciones lineales y simultáneas, propuesto en ejemplo 3.3, utilizando una precisión de 1e-7.

Solución:

El sistema propuesto es:

$2x_1 - x_2 + 3x_3 + x_4 = 1$
$-x_1 + 4x_2 + 2x_3 + x_4 = 2$
$3x_1 + 2x_2 - 2x_3 - x_4 = 3$
$x_1 + x_2 - x_3 + 2x_4 = -2$

Con el fin de garantizar la convergencia del método, deben intercambiarse las ecuaciones 1 con la 3, de esta manera el sistema queda:

$3x_1 + 2x_2 - 2x_3 - x_4 = 3$ (1)
$-x_1 + 4x_2 + 2x_3 + x_4 = 2$ (2)
$2x_1 - x_2 + 3x_3 + x_4 = 1$ (3)
$x_1 + x_2 - x_3 + 2x_4 = -2$ (4)

Despejando de cada ecuación una incógnta:

$x_1 = (3 - 2x_2 + 2x_3 + x_4)/3$
$x_2 = (2 + x_1 - 2x_3 - x_4)/4$
$x_3 = (1 - 2x_1 + x_2 - x_4)/3$
$x_4 = (-2 - x_1 - x_2 + x_3)/2$

Los resultados se muestran en la siguiente tabla:

ITERACION	x_1	x_2	x_3	x_4	$MAX(\|x_i^{(k+1)} - x_i^k\|)$
0	0	0	0	0	
1	1	0.75	-0.083333333	-1.91666667	1.91666667
2	-0.194444444	0.972222222	1.425925926	-0.67592593	1.50925926
3	1.077160494	0.225308642	-0.08436214	-1.69341564	1.51028807
4	0.229080933	1.022805213	1.086019662	-1.08293324	1.1703818

5	0.681165219	0.398014784	0.37287253	-1.35315374	0.71314713
6	0.532187251	0.784898982	0.691226072	-1.31293008	0.3868842
7	0.499908033	0.607596492	0.640236836	-1.23363384	0.17730249
8	0.610548947	0.64092728	0.55115441	-1.35016091	0.11652706
9	0.490097784	0.684487468	0.684817603	-1.24488382	0.13366319
10	0.585258815	0.615126858	0.563164351	-1.31861066	0.12165325
11	0.525821441	0.67952585	0.64883121	-1.27825804	0.08566686
12	0.553450893	0.633511628	0.601622628	-1.29266995	0.04720858
13	0.547850684	0.659318844	0.618762474	-1.29420353	0.02580722
14	0.541561245	0.649559956	0.620213664	-1.28545377	0.00975889
15	0.551951216	0.649244414	0.61026525	-1.29546519	0.01038997
16	0.54219216	0.654281713	0.621787527	-1.28734317	0.01152228
17	0.549222819	0.648247734	0.612381757	-1.2925444	0.00940577
18	0.545241215	0.653255525	0.618439164	-1.29002879	0.00605741
19	0.546779497	0.649982489	0.615484095	-1.29063895	0.00327304
20	0.546788089	0.651614711	0.616225826	-1.29108849	0.00163222
21	0.546044581	0.651170354	0.616723226	-1.29024585	0.00084263
22	0.546953297	0.650938175	0.615759145	-1.29106616	0.00096408
23	0.546191926	0.65143495	0.616705753	-1.29046056	0.00094661
24	0.546693682	0.650935684	0.61600296	-1.2908132	0.00070279
25	0.54644045	0.651311933	0.616414746	-1.29066882	0.00041179
26	0.546512269	0.651087899	0.61624406	-1.29067805	0.00022403
27	0.546544756	0.651183672	0.616257405	-1.29073551	9.5773E-05
28	0.546470651	0.651172838	0.616322349	-1.29066057	7.4942E-05
29	0.546546151	0.651140505	0.616236258	-1.2907252	8.6092E-05
30	0.546488769	0.651185363	0.616311008	-1.29068156	7.475E-05
31	0.546523243	0.651145697	0.616260258	-1.29070434	5.0751E-05
32	0.54650826	0.651173022	0.616286948	-1.29069717	2.7325E-05

33	0.546510228	0.651158375	0.616278362	-1.29069512	1.4647E-05
34	0.546514951	0.651163337	0.616276185	-1.29070105	5.9304E-06
35	0.546508215	0.651164224	0.616282948	-1.29069475	6.7628E-06
36	0.546514234	0.651160771	0.616275682	-1.29069966	7.2658E-06
37	0.546510054	0.651164588	0.61628138	-1.29069663	5.698E-06
38	0.546512318	0.651161547	0.616277847	-1.29069801	3.5334E-06
39	0.54651153	0.651163461	0.61627947	-1.29069776	1.9143E-06
40	0.546511419	0.65116256	0.616279161	-1.29069741	9.0147E-07
41	0.546511931	0.651162754	0.616278767	-1.29069796	5.5042E-07
42	0.546511355	0.651162945	0.616279398	-1.29069745	6.3091E-07
43	0.546511818	0.651162618	0.616278811	-1.29069781	5.8686E-07
44	0.546511524	0.651162929	0.616279231	-1.29069761	4.1987E-07
45	0.546511665	0.651162703	0.616278995	-1.29069769	2.3586E-07
46	0.546511632	0.651162832	0.616279085	-1.29069769	1.2874E-07
47	0.546511605	0.651162781	0.616279087	-1.29069765	5.0702E-08

Por consiguiente, en 47 iteraciones: $x_1 = 0.546511605$; $x_2 = 0.651162781$; $x_3 = 0.616279087$; $x_4 = -1.29069765$

Deducción de fórmulas para el método de Gauss - Seidel:

Considerar la ecuación k-ésima:

$a_{k1}x_1 + a_{k2}x_2 + a_{k3}x_3 + \ldots + a_{kk}x_k + \ldots + a_{kN}x_N = b_k$
Despejando la incógnita x_k:

$x_k = (b_k - (a_{k1}x_1 + a_{k2}x_2 + a_{k3}x_3 + \ldots + a_{kN}x_N))/a_{kk}$

En consecuencia, deben aplicarse las siguientes fórmulas:

$$x_1^{(0)} = x_2^{(0)} = x_3^{(0)} = \ldots = x_k^{(0)} = \ldots = x_N^{(0)} = 0 \qquad (3.21)$$

$$x_k = \left(b_k - \sum_{j=1, j\neq k}^{N} a_{kj}x_j\right)/a_{kk} \qquad (3.22);\ \forall_{k\ =\ 1,2,3,\ldots,N}$$

Algoritmo para el método de Gauss - Seidel:

a) Inicializar con el valor cero el vector de incógnitas (X), aplicando la fórmula (3.21).
b) Calcular cada incógnita, aplicando la fórmula (3.22)
c) Repetir el paso (b), mientras $\text{MAX}(|x_i^{(k+1)} - x_i^k|) > \text{PRECISION}$

Método de Crout

Consiste en sustituir la matriz de coeficientes A, por el producto matricial LU y el vector de términos independientes B por el producto LC.

Entonces: A = LU (3.31); B = LC (3.32), donde L, es una matriz triangular inferior de

$$L = \begin{pmatrix} l_{11} & 0 & 0 & \cdots & 0 & \cdots & 0 \\ l_{21} & l_{22} & 0 & \cdots & 0 & \cdots & 0 \\ l_{31} & l_{32} & l_{33} & \cdots & 0 & \cdots & 0 \\ & & \cdots & & & & \\ l_{i1} & l_{i2} & l_{i3} & \cdots & l_{ii} & \cdots & 0 \\ & & \cdots & & & & \\ l_{N1} & l_{N2} & l_{N3} & \cdots & l_{Ni} & \cdots & l_{NN} \end{pmatrix}; \quad l_{ij} : \begin{cases} = 0, \text{ para } i < j \\ \neq 0, \text{ para } i \geq j \end{cases}$$

la forma:

U es una matriz triangular superior y C es un vector columna, estos tienen la forma:

$$U = \begin{pmatrix} 1 & u_{12} & u_{13} & \cdots & u_{1i} & \cdots & u_{1j} & \cdots & u_{1N} \\ 0 & 1 & u_{23} & \cdots & u_{2i} & \cdots & u_{2j} & \cdots & u_{2N} \\ 0 & 0 & 1 & \cdots & u_{3i} & \cdots & u_{3j} & \cdots & u_{3N} \\ & & & & \cdots & & & & \\ 0 & 0 & 0 & \cdots & 1 & \cdots & u_{ij} & \cdots & u_{iN} \\ & & & & \cdots & & & & \\ 0 & 0 & 0 & \cdots & 0 & \cdots & 0 & \cdots & 1 \end{pmatrix}; u_{ij} : \begin{cases} \neq 0, \text{ para } i < j \\ = 0, \text{ para } i > j \\ = 1, \text{ para } i = j \end{cases}; C = \begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ \cdots \\ c_i \\ \cdots \\ c_N \end{pmatrix}$$

Deducción de fórmulas para el método de Crout:

Para obtener un elemento genérico del producto matricial: A = LU, se aplica la

$$a_{ij} = \sum_{k=1}^{N} l_{ik} u_{kj} \qquad (3.33)$$

fórmula:

Cálculo de la matriz L:

De (3.33) se tiene que:

$$a_{ij} = \sum_{k=1}^{j-1} l_{ik} u_{kj} + l_{ij} u_{jj} + \sum_{k=j+1}^{N} l_{ik} u_{kj} \qquad (3.34)$$

Pero:

$$u_{jj} = 1 \ (3.35); \quad \sum_{k=j+1}^{N} l_{ik} u_{kj} = l_{i(j+1)} u_{(j+1)j} + l_{i(j+2)} u_{(j+2)j} + \ldots + l_{iN} u_{Nj} = 0 \qquad (3.36)$$

Reemplazando (3.35) y (3.36) en (3.34) y despejando l_{ij}:

$$l_{ij} = a_{ij} - \sum_{k=1}^{j-1} l_{ik} u_{kj} \qquad (3.37)$$

Cuando j = 1, el sumatorio de (3.37) no existe, por tanto:

$$l_{i1} = a_{i1} \qquad (3.38)$$

Cálculo de la matriz U:

De (3.33) se tiene que:

$$a_{ij} = \sum_{k=1}^{i-1} l_{ik} u_{kj} + l_{ii} u_{ij} + \sum_{k=i+1}^{N} l_{ik} u_{kj} \qquad (3.39)$$

Pero:

$$\sum_{k=i+1}^{N} l_{ik} u_{kj} = l_{i(i+1)} u_{(i+1)j} + l_{i(i+2)} u_{(i+2)j} + \ldots + l_{iN} u_{Nj} = 0 \qquad (3.40)$$

Sustituyendo (3.40) en (3.39) y despejando u_{ij}:

$$u_{ij} = (a_{ij} - \sum_{k=1}^{i-1} l_{ik} u_{kj}) / l_{ii} \qquad (3.41)$$

Cuando i = 1, el sumatorio de (3.41) no existe, por tanto:

$$u_{1j} = a_{1j} / l_{11} \qquad (3.42)$$

Cálculo del vector C :

Para obtener un elemento genérico del producto matricial: B = LC, se aplica la

$$b_i = \sum_{j=1}^{N} l_{ij} c_j \qquad (3.43)$$

fórmula:

De (3.43) se tiene que:

$$b_i = \sum_{j=1}^{i-1} l_{ij} c_j + l_{ii} c_i + \sum_{j=i+1}^{N} l_{ij} c_j \qquad (3.44)$$

Pero:

$$\sum_{j=i+1}^{N} a_{ij}c_j = a_{i(i+1)}c_{(i+1)} + a_{i(i+2)}c_{(i+2)} + \ldots + a_{iN}c_N = 0 \quad (3.45)$$

Sustituyendo (3.45) en (3.44) y despejando c_i:

$$c_i = (b_i - \sum_{j=1}^{i-1} a_{ij}c_j)/\ a_{ii} \quad (3.46)$$

Cuando i = 1, el sumatorio de (3.46) no existe, por tanto:

$$c_1 = b_1 /\ a_{11} \quad (3.47)$$

Cálculo del vector X :

Partiendo del sistema general de ecuaciones lineales y simultáneas:

AX = B (3.48)

Sustituyendo (3.31) y (3.32) en (3.48):

LUX = LC (3.49)

Multiplicando por la izquierda, L^{-1} en (3.49):

$L^{-1}LUX = L^{-1}LC$ (3.50)

Sustituyendo $L^{-1}L$ por I, donde I es la matriz identidad:

IUX = IC (3.51)

Simplificando (3.51) y transponiendo miembros:

C = UX (3.52)

Para obtener un elemento genérico del producto matricial: C = UX, se aplica la fórmula:

$$c_i = \sum_{j=1}^{N} u_{ij}x_j \quad (3.53)$$

De (3.53) se tiene que:

$$c_i = \sum_{j=1}^{i-1} u_{ij}x_j + u_{ii}x_i + \sum_{j=i+1}^{N} u_{ij}x_j \quad (3.54)$$

Pero:

$$\sum_{j=i+1}^{N} u_{ij}x_j = u_{i(i+1)}x_{(i+1)} + u_{i(i+2)}x_{(i+2)} + \ldots + u_{iN}x_N \neq 0, \text{ entonces}$$

$$\sum_{j=1}^{i-1} u_{ij}x_j = 0 \quad (3.55), \text{ además} : u_{ii} = 1; (3.56)$$

Sustituyendo (3.55) y (3.56) en (3.54) y despejando x_i:

$$x_i = c_i - \sum_{j=i+1}^{N} u_{ij}x_j \qquad (3.57)$$

Cuando i = N, el sumatorio de (3.57) no existe, por tanto:

$$x_N = c_N \qquad (3.58)$$

Ejemplo 3.7.

Dado el sistema:

$a_{11}\, x_1 + a_{12}\, x_2 + a_{13}\, x_3 + a_{14}\, x_4 = b_1$
$a_{21}\, x_1 + a_{22}\, x_2 + a_{23}\, x_3 + a_{24}\, x_4 = b_2$
$a_{31}\, x_1 + a_{32}\, x_2 + a_{33}\, x_3 + a_{34}\, x_4 = b_3$
$a_{41}\, x_1 + a_{42}\, x_2 + a_{43}\, x_3 + a_{44}\, x_4 = b_4$

Resolverlo por el método de Crout y calcular el determinante de la matriz de coeficientes:

Solución:

Se plantea el esquema:

a11 ***l11***	***a12*** ***u12***	***a13*** ***u13***	***a14*** ***u14***	***b1*** ***c1 / x1***
a21 ***l21***	***a22*** ***l22***	***a23*** ***u23***	***a24*** ***u24***	***b2*** ***c2 / x2***
a31 ***l31***	***a32*** ***l32***	***a33*** ***l33***	***a34*** ***u34***	***b3*** ***c3 / x3***
a41 ***l41***	***a42*** ***l42***	***a43*** ***l43***	***a44*** ***u44***	***b4*** ***c4 / x4***

Para encontrar los valores de L, U, C y X, se emplean las fórmulas deducidas anteriormente, mismas que se encuentran sombreadas; así se tiene que para calcular la primera columna de L, se aplica la fórmula (3.38); para calcular la primera fila de U, se aplica la fórmula (3.42); para calcular la segunda columna de L se aplica la fórmula (3.37) ; para calcular la segunda fila de U, se aplica la fórmula (3.41) y asi sucesivamente hasta transformar completamente la matriz A.

A fin de mecanizar el uso de las fórmulas (3.37) y (3.41), considerar los siguientes

$$_{43} = a_{43} - \sum_{k=1}^{2} {}_{4k}u_{k3} = a_{43} - (\ {}_{41}u_{13} + \ {}_{42}u_{23})$$

casos:
Es decir, para calcular un elemento de L, a partir de la segunda columna, se resta del respectivo elemento de A, la suma de los productos de los elementos de L de esa fila por los de U de esa columna.
Es decir, para calcular un elemento de U, a partir de la segunda fila, se resta del respectivo elemento de A, la suma de los productos de los elementos de L de esa fila por los de U de esa columna, este resultado se divide para el elemento de L de la diagonal principal.

$$u_{34} = (a_{34} - \sum_{k=1}^{2} {}_{3k}u_{k4})/ \ {}_{33} = (a_{34} - (\ {}_{31}u_{14} + \ {}_{32}u_{24}))/ \ {}_{33}$$

Para calcular el determinante de la matriz A, se multiplican los elementos de la diagonal principal de L. es decir:

$$\det = \prod_{i=1}^{N} {}_{ii} \qquad (3.59)$$

Para este ejemplo sería:

$$\det = \prod_{i=1}^{4} u_{ii} = u_{11}\ u_{22}\ u_{33}\ u_{44}$$

Se aplica la formula (3.47) para calcular el primer elemento de C; y la fórmula (3.46) para calcular los demás elementos de C, a partir del segundo. Un cálculo típico sería:

$$c_3 = (b_3 - \sum_{j=1}^{2} l_{3j}c_j)/\ u_{33} = (b_3 - (\ l_{31}c_1 + \ l_{32}c_2))/\ u_{33}$$

Es decir, para calcular un elemento de C, a partir del segundo, se resta del respectivo elemento de B, la suma de los productos de los elementos de L de esa fila por los de C, este resultado se divide para el elemento de L de la diagonal principal.

Se aplica la formula (3.58) para calcular el último elemento de X; y la fórmula (3.57) para calcular los demás elementos de X, partiendo desde el penúltimo hasta el primero. Un cálculo típico sería:

$$x_2 = c_2 - \sum_{j=3}^{4} u_{2j}x_j = c_2 - (u_{23}x_3 + u_{24}x_4)$$

Ejemplo 3.8.

Dado el sistema:

$$\begin{bmatrix} -4 & 1 & -1 & 2 \\ 1 & 5 & -2 & -4 \\ -1 & -2 & 8 & 3 \\ 2 & -4 & 3 & -5 \end{bmatrix} \begin{bmatrix} x1 \\ x2 \\ x3 \\ x4 \end{bmatrix} = \begin{bmatrix} 8 \\ -2 \\ 4 \\ -6 \end{bmatrix}$$

Resolverlo por el método de Crout y calcular el determinante de la matriz de coeficientes:

Solución:

Se plantea el esquema:

-4 -4	1 -1/4	-1 1/4	2 -1/2	8 -2|-178/99
1 1	5 21/4	-2 -3/7	-4 -2/3	-2 0|164/495
-1 -1	-2 -9/4	8 51/7	3 7/51	4 14/51|112/495
2 2	-4 -7/2	3 1	-5 -110/17	-6 58/165|58/165

Los cálculos típicos, se detallan a continuación:

$l_{22} = 5 – 1(-1/4\ l22) = 21/4$
$l_{32} = -2 - (-1)(/1/4) = -9/4$
$l_{42} = -4 - 2(-1/4) = -7/2$
$u_{23} = 4/21(-2 - 1(1/4)) = -3/7$
$u_{24} = 4/21(-4 - 1(-1/2)) = -2/3$
$l_{33} = 8 - (-1(1/4) + (-9/4)(-3/7)) = 51/7$
$l_{43} = 3 - (2(1/4) + (-7/2)(-3/7)) = 1$
$u_{34} = 7/51(3 - (-1(-1/2) + (-9/4)(-2/3)) = 7/51$
$l_{44} = -5(2(-1/2) + (7/2)(-2/3) + 1\ (7/51)) = -110/17$
$c_2 = 4/21(-2 - (1)(-2)) = 0$
$c_3 = 7/51(4 - ((-1)(-2) + (-9/4)(0)))\ = 14/51$
$c_4 = -17/110(-6 - ((2)(-2) + (-7/2)(0) + (1)(14/51))) = 58/165$
$x_4 = c_4 = 58/165$
$x_3 = 14/51 – 7/51(58/165) = 112/495$
$x_2 = 0 – (-3/7(112/495) + (-2/3)(58/165) = 164/495$
$x_1 = -2 – (-1/4(164/495) + 1/4(112/495) + (-1/2)(58/165)) = -178/99$

Cálculo del determinante:

Det = 4 (21/4) (51/7) (-110/17) = -990

Algoritmo para el método de Crout:

1. Aplicando la fórmula 3.38 se determinan los valores de la primera columna de L.
2. Aplicando la fórmula 3.42 se calcula los elementos de la primera fila de la matriz U.
3. Aplicando la fórmula 3.37 se calcula la siguiente columna de la matriz L.

4. Aplicando la fórmula 3.41 se calcula la siguiente fila de la matriz U
5. Repetir desde el paso 3 hasta calcular todas las columnas de la matriz L y todas las filas de la matriz U.
6. Aplicando la fórmula 3.47, se calcula el primer elemento del vector C.
7. Aplicando la fórmula 3.46 se calcula el siguiente elemento del vector C.
8. Repetir el paso 7 hasta calcular todos los elementos del vector C.
9. Aplicando la fórmula 3.58 se determina el último elemento del vector X.
10. Aplicando la fórmula 3.57 se calcula el elemento anterior del vector X.
11. Repetir el paso 10 hasta calcular todos los elementos del vector X.

FÓRMULAS MODIFICADAS PARA CODIFICACIÓN

Para implementar el método de Crout en Matlab o Visual C#.NET, las fórmulas sombreadas, deducidas anteriormente, deben ser modificadas, a fin de que las matrices A, L y U compartan el mismo espacio de memoria; la misma situación debe ocurrir con los vectores B, C y X; además, ciertas operaciones como el cálculo de la primera columna de L, así omo el Último elemento de X, resultan obvias. Con estas indicaciones, se describe el formulario simplificado para este método:

Cálculo de la matriz L:

$$a_{ij} = a_{ij} - \sum_{k=1}^{j-1} a_{ik}a_{kj} \qquad (3.60)$$

Cálculo de la matriz U:

$$a_{1j} = a_{1j}/a_{11} \qquad (3.61); \qquad a_{ij} = (a_{ij} - \sum_{k=1}^{i-1} a_{ik}a_{kj})/a_{ii} \qquad (3.62)$$

Cálculo del vector C:

$$b_1 = b_1/a_{11} \qquad (3.63); b_i = (b_i - \sum_{j=1}^{i-1} a_{ij}b_j)/a_{ii} \qquad (3.64)$$

Cálculo del vector X:

$$b_i = b_i - \sum_{j=i+1}^{N} a_{ij}x b_j \qquad (3.65)$$

Implementación de los algoritmos en MATLAB y VISUAL C#:

Para cada lenguaje, se elabora un solo programa, desde el cual se abren archivos para lectura de datos; se ejecuta cada uno de los métodos de resolución de sistemas de ecuaciones lineales y simultáneas y se escribe los resultados en un archivo de salida, en el caso de Matlab y se despliegan los resultados en una caja lista en el caso de Visual C#.

Archivos fuente en MATLAB:

```
%SisEcuac1.m
% APERTURA DE ARCHIVOS:
ent = fopen('D:\AnalNumat\matriz.dat', 'r');
sal = fopen('D:\AnalNumat\rsisecuac1.dat', 'w');
% LECTURA DE LA DIMENSION DEL SISTEMA:
N = fscanf(ent, '%d', 1);
A = LeeMatriz(ent, N);
B = LeeVector(ent, N);
fprintf(sal, 'RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y SIMULTANEAS:\r\n');
fprintf(sal, 'MATRIZ DE COEFICIENTES:\r\n');
EscribeMatriz(sal, A, N);
fprintf(sal, 'VECTOR DE TERMINOS INDEPENDIENTES:\r\n');
EscribeVector(sal, B, N);
[det, X] = Gauss(A, B, N);
fprintf(sal, 'METODO DE GAUSS: VECTOR DE SOLUCIONES:\r\n');
EscribeVector(sal, X, N);
fprintf(sal, 'DETERMINANTE = %f\r\n', det);
[det, X] = GaussJordan(A, B, N);
fprintf(sal, 'METODO DE GAUSS - JORDAN: VECTOR DE SOLUCIONES:\r\n');
EscribeVector(sal, X, N);
fprintf(sal, 'DETERMINANTE = %f\r\n', det);
fprintf(sal, 'METODO DE GAUSS - SEIDEL: VECTOR DE SOLUCIONES:\r\n');
[X, estado, msg] = GaussSeidel(A, B, N);
if estado > 0
   EscribeVector(sal, X, N);
end;
fprintf(sal, msg);
fprintf(sal, '\r\nUSO DEL METODO MATLAB:\r\n');
Y = A \ B';
fprintf(sal, 'VECTOR DE SOLUCIONES:\r\n');
EscribeVector(sal, Y, N);
fclose(ent);
ent = fopen('D:\AnalNumat\matrizcr.dat', 'r');
N = fscanf(ent, '%d', 1);
NS = fscanf(ent, '%d', 1);
A = LeeMatriz(ent, N);
fprintf(sal, 'RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y SIMULTANEAS:\r\n');
fprintf(sal, 'METODO DE CROUT:\r\n');
fprintf(sal, 'MATRIZ DE COEFICIENTES:\r\n');
EscribeMatriz(sal, A, N);
for i = 1 : NS
   B = LeeVector(ent, N);
   fprintf(sal, 'VECTOR DE TERMINOS INDEPENDIENTES:\r\n');
   EscribeVector(sal, B, N);
   [det, A, B] = Crout(A, B, N, i);
   if i == 1
      fprintf(sal, 'DETERMINANTE = %f\r\n', det);
   end;
   fprintf(sal, 'VECTOR DE SOLUCIONES:\r\n');
   EscribeVector(sal, B, N);
end;
```

```
fclose('all');
% LeeMatriz.m
function A = LeeMatriz(f, m)
for i = 1 : m
  for j = 1 : m
    A(i, j) = fscanf(f, '%f', 1);
  end;
end;
return

% LeeVector.m
function A = LeeVector(f, m)
for i = 1 : m
  A(i) = fscanf(f, '%f', 1);
end;
return

% EscribeMatriz.m
function EscribeMatriz(f, A, m)
for i = 1 : m
  fprintf(f, 'FILA   %d:\r\n',i);
  for j = 1 : m
    fprintf(f, '%f  ',A(i, j));
    if (mod(j, 8) == 0) | (j == m)
      fprintf(f, '\r\n');
    end;
  end;
end;
return

% EscribeVector.m
function EscribeVector(f, A, m)
for i = 1 : m
  fprintf(f, '%f  ',A(i));
  if (mod(i, 8) == 0) | (i == m)
    fprintf(f, '\r\n');
  end;
end;
return

% Gauss.m
function [det, X] = Gauss(A, B, N)
for k = 1 : N - 1
  if A(k,k) == 0
    % buscar un A(k,k) > 0
    for i = k + 1: N
      if A(i,k) ~= 0 % intercambiar las filas i con la k:
        X = A(i,:); A(i,:) = A(k,:); A(k,:) = X;
        det = B(i); B(i) = B(k); B(k) = det;
        break;
      end;
    end;
    if A(k,k) == 0
      fprintf('Error!, no existe solución para este sistema');
      exit;
    end;
  end;
  % Eliminación gaussiana:
  for i = k + 1 : N
    det = -A(i,k)/A(k,k);
    for j = k : N
```

```
        A(i, j) = A(i, j) + det * A(k,j);
      end;
      B(i) = B(i) + det * B(k);
    end;
end;
% Retrosustitución:
det = 1;
for k = N : -1 : 1
   det = det * A(k,k);
   for j = k + 1 : N
      B(k) = B(k) - A(k, j) * B(j);
   end;
   B(k) = B(k) / A(k, k);
end;
X = B;

% gaussjordan.m
function [det, X] = GaussJordan(A, B, N)
det = 1;
for k = 1 : N
   if A(k,k) == 0
      % buscar un A(k,k) ~= 0
      for i = k + 1: N
         if A(i,k) ~= 0 % intercambiar las filas i con la k:
            X = A(i,:); A(i,:) = A(k,:); A(k,:) = X;
            det = B(i); B(i) = B(k); B(k) = det;
            break;
         end;
      end;
      if A(k,k) == 0
         fprintf('Error!, no existe solución para este sistema');
         exit;
      end;
   end;
   % Convertir la matriz de coeficientes a de identidad:
   fac = A(k,k);
   det = det * fac;
   for j = k : N
      A(k, j) = A(k, j)/fac;
   end;
   B(k) = B(k) / fac;
   for i = 1 : N
     if i ~= k
       for j = k + 1: N
         A(i, j) = A(i, j) - A(i, k) * A(k,j);
       end;
       B(i) = B(i) - A(i, k) * B(k);
     end;
   end;
end;
X = B;
% gaussseidel.m
function [X, ST, msg] = GaussSeidel(A, B, N)
niter = 100;
precision = 1e-7;
for i = 1 : N
   X(i) = 0;
   if A(i,i) == 0
      msg = 'Error!, no existe solución para este sistema\nIntercambiar filas\n';
      fprintf(msg);
      ST = -1;
```

```
    return;
  end;
end;
for iter = 1 : niter
  d = 0;
  for i = 1 : N
    s = 0;
    for j = 1 : N
      if i ~= j
        s = s + A(i, j) * X(j);
      end;
    end;
    t = (B(i) - s) / A(i, i);
    max = abs(t - X(i));
    if max > d
      d = max;
    end;
    X(i) = t;
  end;
  if d <= precision
    break;
  end;
end;
ST = 1;
if d > precision
  [msg, er] = sprintf('Atención!, no se alcanza la precisión requerida: %f\n', d);
  fprintf(msg);
  ST = -1;
else
  [msg, er] = sprintf('Solución en %d iteraciones\n', iter);
  fprintf(msg);
  end;
return;

% crout.m
function [det, A, B] = Crout(A, B, N, St)
det = 1;
if St == 1 % calcular las matrices L, U y el determinante:
  for i = 1 : N
    for j = 2 : N
      l = j - 1;
      if i < j l = i - 1; end;
      for k = 1 : l
        A(i, j) = A(i, j) - A(i, k) * A(k,j);
      end;
      if i < j A(i, j) = A(i, j) / A(i, i); end;
    end;
    det = det * A(i,i);
  end;
end;
% Cálculo del vector C:
for i = 1 : N
  for j = 1 : i - 1
    B(i) = B(i) - A(i, j) * B(j);
  end;
  B(i) = B(i) / A(i, i);
end;
% Cálculo del vector X:
for i = N - 1 : -1 : 1
  for j = i + 1 : N
    B(i) = B(i) - A(i, j) * B(j);
```

```
   end;
end;
```

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa, como se había indicado anteriormente, la entrada – salida, se realiza sobre archivos, cuyos contenidos son:

matriz.dat:

```
3
3
1
2
1
-4
-1
2
-1
4
1
2
3
```

matrizcr.dat:

```
3
2
3
1
2
1
-4
-1
2
-1
4
1
2
3
3
2
1
```

rsisecuac1.dat

```
RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y SIMULTANEAS:
MATRIZ DE COEFICIENTES:
FILA   1:
3.000000  1.000000  2.000000
FILA   2:
```

```
1.000000  -4.000000  -1.000000
FILA   3:
2.000000  -1.000000  4.000000
VECTOR DE TERMINOS INDEPENDIENTES:
1.000000  2.000000  3.000000
METODO DE GAUSS: VECTOR DE SOLUCIONES:
0.186047  -0.581395  0.511628
DETERMINANTE = -43.000000
METODO DE GAUSS - JORDAN: VECTOR DE SOLUCIONES:
0.186047  -0.581395  0.511628
DETERMINANTE = -43.000000
METODO DE GAUSS - SEIDEL: VECTOR DE SOLUCIONES:
0.186047  -0.581395  0.511628
Solución en 11 iteraciones

USO DEL METODO MATLAB:
VECTOR DE SOLUCIONES:
0.186047  -0.581395  0.511628
RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y SIMULTANEAS:
METODO DE CROUT:
MATRIZ DE COEFICIENTES:
FILA   1:
3.000000  1.000000  2.000000
FILA   2:
1.000000  -4.000000  -1.000000
FILA   3:
2.000000  -1.000000  4.000000
VECTOR DE TERMINOS INDEPENDIENTES:
1.000000  2.000000  3.000000
DETERMINANTE = -43.000000
VECTOR DE SOLUCIONES:
0.186047  -0.581395  0.511628
VECTOR DE TERMINOS INDEPENDIENTES:
3.000000  2.000000  1.000000
VECTOR DE SOLUCIONES:
1.302326  -0.069767  -0.418605
```

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 3.2
Como se puede apreciar, existen un cuadro de lista, el cual permitirá presentar los resultados; y cuatro botones, mismos que tienen la funcionalidad:

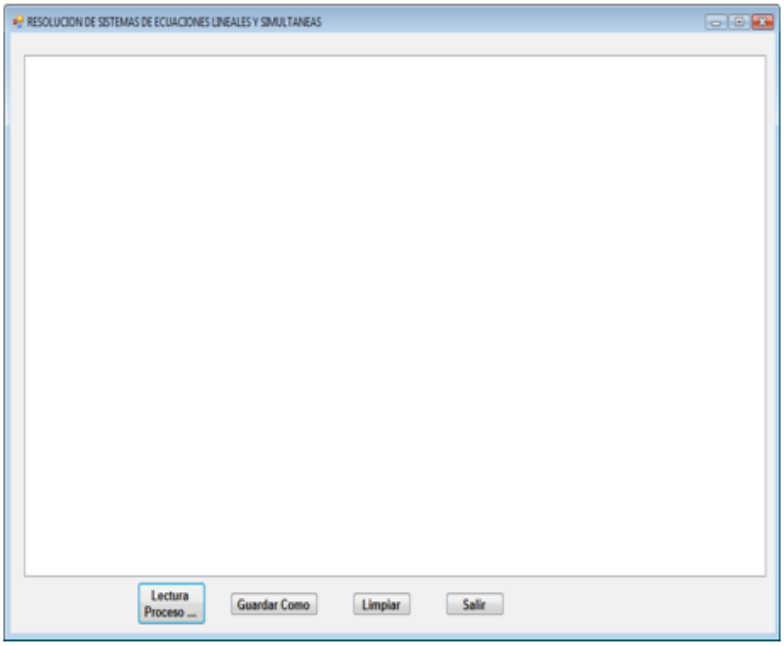


Figura 3.2 Ventana para resolución de sist. ec. lin y simult. en Visual C#

- Lectura Proceso…: abre una ventana para registrar el nombre del archivo el cual permite leer los datos del sistema de ecuaciones lineales y simultáneas desde el medio de almacenamiento permanente; ejecuta los métodos para calcular las soluciones y despliega los resultados.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Botón Lectura Proceso…:

```
private void b_lectura_Click(object sender, EventArgs e)
 {
   try
   {
    // Abrir el archivo de entrada a traves del cuadro de dialogo:
    CEntSalArchivos objes = new CEntSalArchivos("d:\\");
    objes.AbrirArchivoEnt();
    // Leer el número de ecuaciones o de incógnitas:
    CResolSistemaEcuaciones.n = Int16.Parse(objes.LeerLinea());
    if (CResolSistemaEcuaciones.n > 0)
    {
     double det;
     // Definir el arreglo de términos independientes:
     CResolSistemaEcuaciones.B = new double[CResolSistemaEcuaciones.n];
    // Definir la matriz de coeficientes:
    CResolSistemaEcuaciones.A = new double[CResolSistemaEcuaciones.n][];
    for (int i = 0; i < CResolSistemaEcuaciones.n; i++)
      CResolSistemaEcuaciones.A[i] = new double[CResolSistemaEcuaciones.n];
     /* Leer la matriz de coeficientes, el arreglo de términos independientes y
              * generar los resultados en el listbox: */
     // Prueba del método de Gauss:
     CLectEscrMatrices.n = CResolSistemaEcuaciones.n;
     CLectEscrMatrices.a = CResolSistemaEcuaciones.A;
     CLectEscrMatrices.b = CResolSistemaEcuaciones.B;
     CLectEscrMatrices.obent = objes;
     CLectEscrMatrices.lb = listBox1;
     CLectEscrMatrices.LeeMatriz();
     CLectEscrMatrices.LeeVector();
     listBox1.Items.Add("RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y
SIMULTANEAS POR EL METODO DE GAUSS");
     CLectEscrMatrices.EscribeMatriz(" DE COEFICIENTES:");
     CLectEscrMatrices.EscribeVector(" DE TERMINOS INDEPENDIENTES:");
     det = CResolSistemaEcuaciones.Gauss();
     CLectEscrMatrices.EscribeVector(" SOLUCION:");
     listBox1.Items.Add("DETERMINANTE = " + det.ToString());
     // Cerrar flujo de entrada:
     objes.CerrarLectura();
     // Prueba del método de Gauss - Jordan:
```

```
        // Abrir flujo de entrada:
        objes.AbrirArchivoEnt();
        CResolSistemaEcuaciones.n = Int16.Parse(objes.LeerLinea());
        CLectEscrMatrices.LeeMatriz();
        CLectEscrMatrices.LeeVector();
        listBox1.Items.Add("RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y
SIMULTANEAS POR EL METODO DE GAUSS - JORDAN");
        CLectEscrMatrices.EscribeMatriz(" DE COEFICIENTES:");
        CLectEscrMatrices.EscribeVector(" DE TERMINOS INDEPENDIENTES:");
        det = CResolSistemaEcuaciones.GaussJordan();
        CLectEscrMatrices.EscribeVector(" SOLUCION:");
        listBox1.Items.Add("DETERMINANTE = " + det.ToString());
        // Cerrar flujo de entrada:
        objes.CerrarLectura();
        // Prueba del método de Crout:
      // Abrir flujo de entrada:
      objes.AbrirArchivoEnt();
      CResolSistemaEcuaciones.n = Int16.Parse(objes.LeerLinea());
      CLectEscrMatrices.LeeMatriz();
      CLectEscrMatrices.LeeVector();
      listBox1.Items.Add("RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y
SIMULTANEAS POR EL METODO DE CROUT");
                  CLectEscrMatrices.EscribeMatriz(" DE COEFICIENTES:");
                  CLectEscrMatrices.EscribeVector(" DE TERMINOS INDEPENDIENTES:");
      det = CResolSistemaEcuaciones.Crout(true);
      CLectEscrMatrices.EscribeVector(" SOLUCION:");
      listBox1.Items.Add("DETERMINANTE = " + det.ToString());
      // Cerrar flujo de entrada:
      objes.CerrarLectura();
      // Prueba del método de Gauss - Seidel:
      // Inicializar a null el nombre del archivo de entrada:
      // objes.FijNullNArchEnt();
      // Abrir flujo de entrada:
      objes.AbrirArchivoEnt();
      CResolSistemaEcuaciones.n = Int16.Parse(objes.LeerLinea());
      CLectEscrMatrices.LeeMatriz();
      CLectEscrMatrices.LeeVector();
      listBox1.Items.Add("RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y
SIMULTANEAS POR EL METODO DE GAUSS - SEIDEL");
      CLectEscrMatrices.EscribeMatriz(" DE COEFICIENTES:");
      CLectEscrMatrices.EscribeVector(" DE TERMINOS INDEPENDIENTES:");
      CResolSistemaEcuaciones.maxIter = 100;
      CResolSistemaEcuaciones.precision = 1e-7;
      CResolSistemaEcuaciones.lb = listBox1;
      CResolSistemaEcuaciones.GaussSeidel();
      CLectEscrMatrices.EscribeVector(" SOLUCION:");
      // Cerrar flujo de entrada:
      objes.CerrarLectura();
      listBox1.Items.Add("");
      listBox1.Items.Add("");
      listBox1.Items.Add("");
    }
   }
   catch (Exception ex)
   {
```

```
    MessageBox.Show(ex.Message);
  }
 }
}
```

Botón Guardar Como …:

```
private void b_guardar_como_Click(object sender, EventArgs e)
    {
        CListBox.lb = listBox1;
        CListBox.GuardarComo("c:\\");
    }
```

Botón Limpiar:

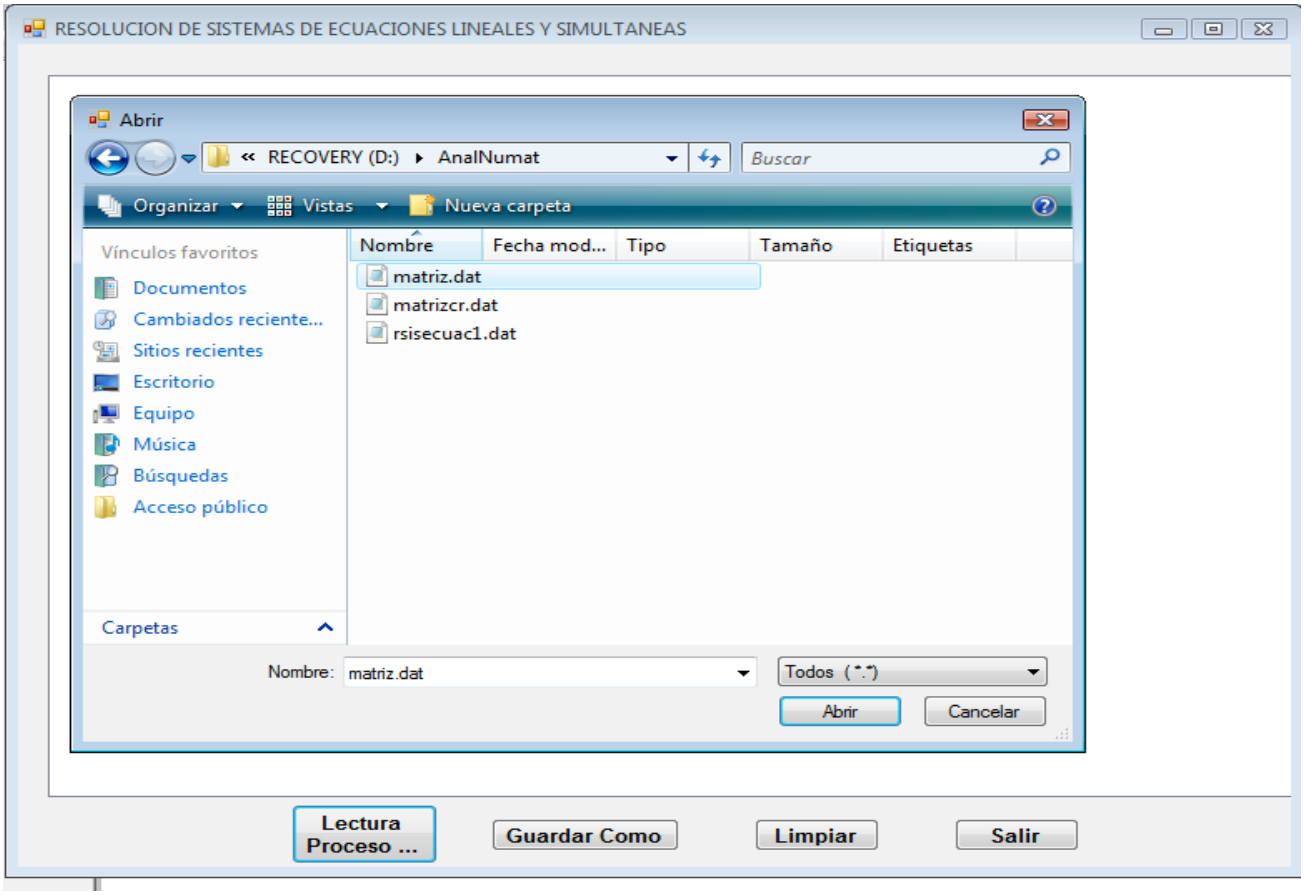
```
private void b_limpiar_Click(object sender, EventArgs e)
    {
        CListBox.lb = listBox1;
        CListBox.Limpiar();
    }
```

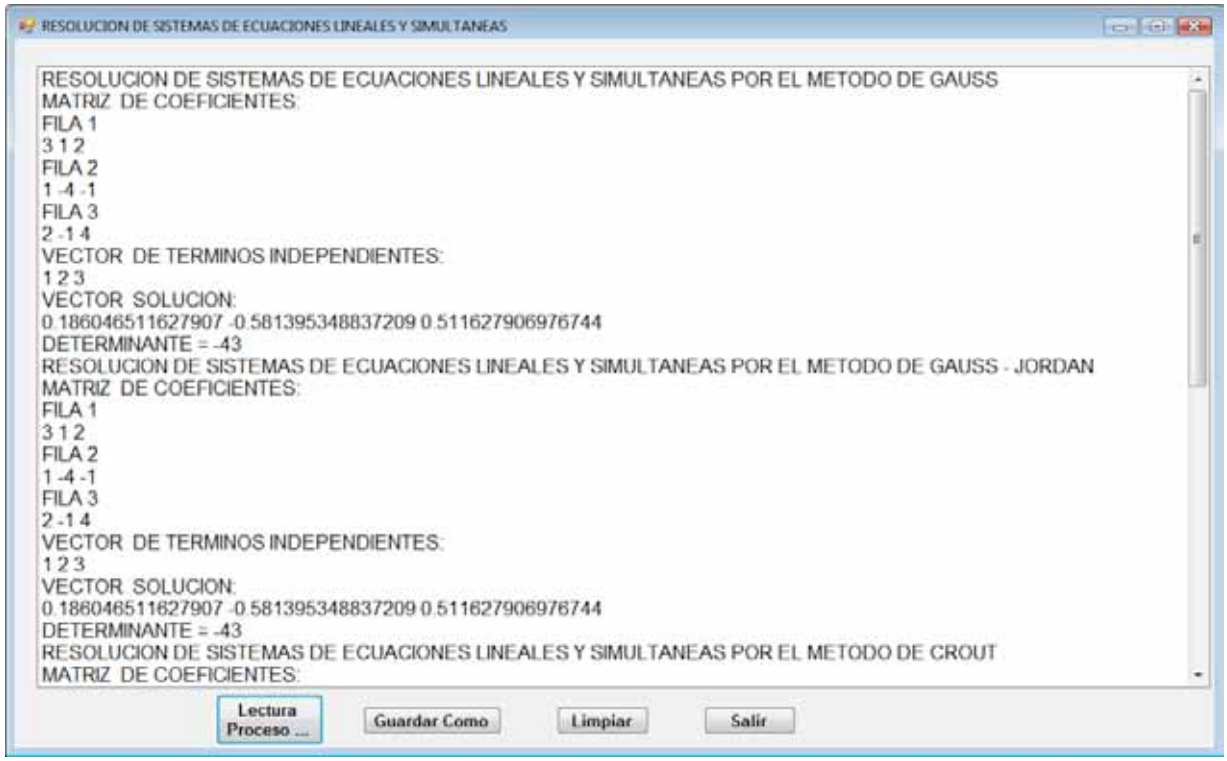
Botón Salir:

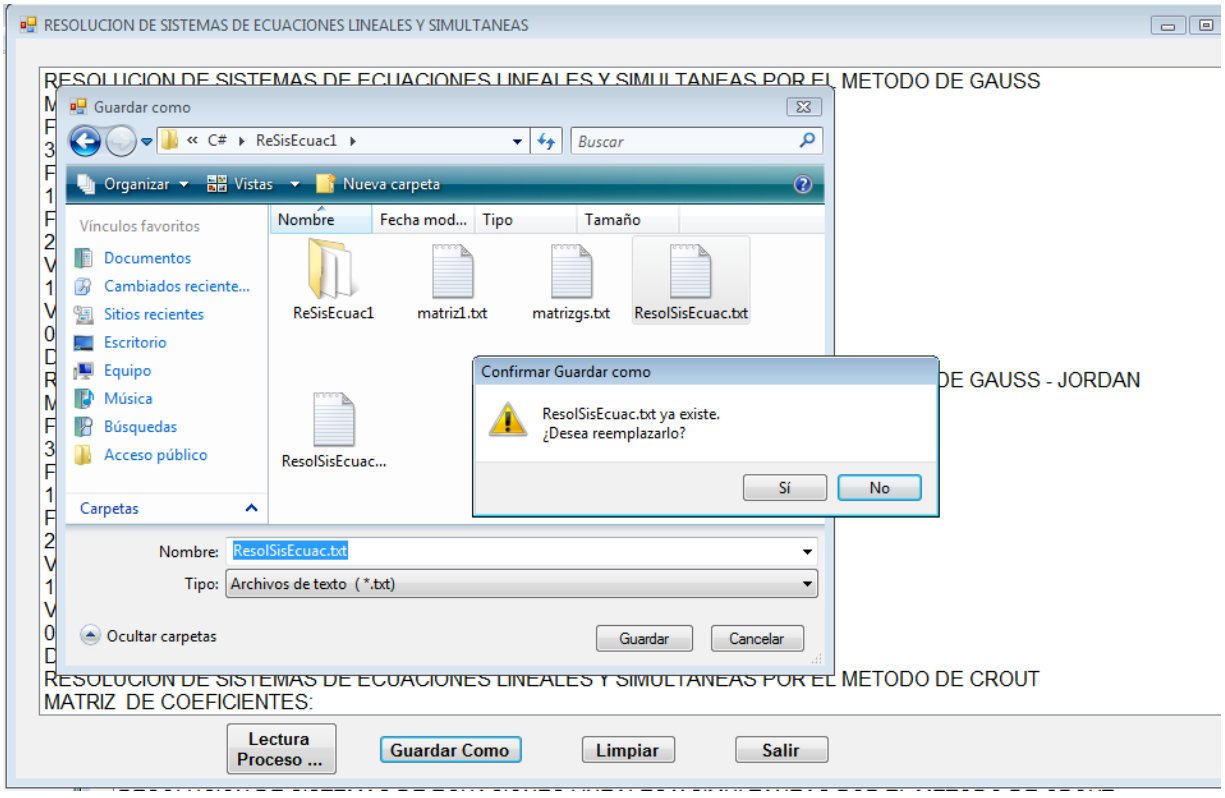
```
private void b_salir_Click(object sender, EventArgs e)
    {
        this.Close();
        this.Dispose();
    }
```

Las referencias a las clases CEntSalArchivos, CResolSistemaEcuaciones, CLectEscrMatrices y CListBox, provienen de la definición de las respectivas clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:




RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y SIMULTANEAS
RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y SIMULTANEAS POR EL METODO DE GAUSS
MATRIZ DE COEFICIENTES:
FILA 1
3 1 2
FILA 2
1 -4 -1
FILA 3
2 -1 4
VECTOR DE TERMINOS INDEPENDIENTES:
1 2 3
VECTOR SOLUCION:
0.186046511627907 -0.581395348837209 0.511627906976744
DETERMINANTE = -43
RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y SIMULTANEAS POR EL METODO DE GAUSS - JORDAN
MATRIZ DE COEFICIENTES:
FILA 1
3 1 2
FILA 2
1 -4 -1
FILA 3
2 -1 4
VECTOR DE TERMINOS INDEPENDIENTES:
1 2 3
VECTOR SOLUCION:
0.186046511627907 -0.581395348837209 0.511627906976744
DETERMINANTE = -43
RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y SIMULTANEAS POR EL METODO DE CROUT
MATRIZ DE COEFICIENTES:
Lectura Proceso ...
Guardar Como
Limpiar
Salir


RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y SIMULTANEAS
Guardar como
C# ▸ ReSisEcuac1 ▸
Buscar
Organizar
Vistas
Nueva carpeta
Vínculos favoritos
Documentos
Cambiados reciente...
Sitios recientes
Escritorio
Equipo
Música
Búsquedas
Acceso público
Carpetas
Nombre
Fecha mod...
Tipo
Tamaño
ReSisEcuac1
matriz1.txt
matrizgs.txt
ResolSisEcuac.txt
ResolSisEcuac...
Confirmar Guardar como
ResolSisEcuac.txt ya existe.
¿Desea reemplazarlo?
Sí
No
Nombre: ResolSisEcuac.txt
Tipo: Archivos de texto (*.txt)
Ocultar carpetas
Guardar
Cancelar
DE GAUSS - JORDAN
RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y SIMULTANEAS POR EL METODO DE CROUT
MATRIZ DE COEFICIENTES:
Lectura Proceso ...
Guardar Como
Limpiar
Salir

Finalmente el contenido del archivo ResolSisEcuac.txt, generado al presionar el botón Guardar Como…, es:

RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y SIMULTANEAS POR EL METODO DE GAUSS
MATRIZ DE COEFICIENTES:
FILA 1
3 1 2
FILA 2
1 -4 -1
FILA 3
2 -1 4
VECTOR DE TERMINOS INDEPENDIENTES:
1 2 3
VECTOR SOLUCION:
0.186046511627907 -0.581395348837209 0.511627906976744
DETERMINANTE = -43
RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y SIMULTANEAS POR EL METODO DE GAUSS - JORDAN
MATRIZ DE COEFICIENTES:
FILA 1
3 1 2
FILA 2
1 -4 -1
FILA 3
2 -1 4
VECTOR DE TERMINOS INDEPENDIENTES:
1 2 3
VECTOR SOLUCION:
0.186046511627907 -0.581395348837209 0.511627906976744
DETERMINANTE = -43
RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y SIMULTANEAS POR EL METODO DE CROUT
MATRIZ DE COEFICIENTES:
FILA 1
3 1 2
FILA 2
1 -4 -1
FILA 3
2 -1 4
VECTOR DE TERMINOS INDEPENDIENTES:
1 2 3
VECTOR SOLUCION:
0.186046511627907 -0.581395348837209 0.511627906976744
DETERMINANTE = -43
RESOLUCION DE SISTEMAS DE ECUACIONES LINEALES Y SIMULTANEAS POR EL METODO DE GAUSS - SEIDEL

MATRIZ DE COEFICIENTES:
FILA 1
3 1 2
FILA 2
1 -4 -1
FILA 3
2 -1 4
VECTOR DE TERMINOS INDEPENDIENTES:
1 2 3
VECTOR SOLUCION:
0.186046541943642 -0.58139532685865 0.511627897313517

MATRIZ INVERSA

La matriz inversa de una matriz Anxn se denota por A^{-1} también de nxn en donde el producto de A x A^{-1} = A^{-1} x A = I (Matriz Identidad).

Deducción de fórmulas:

$A \times A^{-1} = A^{-1} \times A = I$ (3.71)

Sea: $A^{-1} = X$ (3.72)

Entonces, para calcular la matriz inversa, se necesita resolver el sistema:

$AX = I$ (3.73)

La matriz X, se la puede descomponer en los vectores columna: $X_1, X_2, X_3, \ldots\ldots, X_N$ (3.74)

De la misma forma se puede proceder con la matriz I: $I_1, I_2, I_3, \ldots\ldots, I_N$ (3.75)

Sustituyendo (3.74) y (3.75) en (3.73):

$A (X_1, X_2, X_3, \ldots\ldots, X_N) = I_1, I_2, I_3, \ldots\ldots, I_N$ (3.76)

(3.76) Genera los siguientes sistemas de ecuaciones lineales y simultáneas:

$AX_1 = I_1$

$AX_2 = I_2$

$AX_3 = I_3$

...

$AX_N = I_N$

Los valores de los vectores columna I: $I_1, I_2, I_3, \ldots\ldots, I_N$, están dados por:

$$I_1 \qquad I_2 \quad \ldots \quad I_N$$

$$\begin{pmatrix} 1 \\ 0 \\ 0 \\ : \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 1 \\ 0 \\ : \\ 0 \end{pmatrix} \quad \begin{pmatrix} 0 \\ 0 \\ 0 \\ : \\ 1 \end{pmatrix}$$

Ejemplo 3.9.

Dada la matriz:

$$A=\begin{bmatrix} -4 & 1 & -1 \\ 2 & 5 & -2 \\ -1 & 3 & 8 \end{bmatrix} \qquad \text{calcular } A^{-1}$$

Solución:

Se aplica el método de Crout, por ser el más apropiado para resolver varios sistemas de ecuaciones a la vez.

Los resultados se muestran a continuación:

-4	1	-1
-4	-1/4	1/4
2	5	-2
2	11/2	-5/11
-1	3	8
-1	11/4	19/2

1	
-1/4	-46/209
0	
1/11	14/209
0	
-1/19	-1/19

0	
0	11/209
1	
2/11	33/209
0	
-1/19	-1/19

0	
0	-3/209
0	
0	10/209
1	
-1/19	2/19

Y los cálculos típicos se describen:

L_{22}= 5- 2(-1/4) = 11/2

L_{32} =3 – ¼ = 11/4

U_{23} = (-2-1/2)2/11= -5/11

L_{33} = 8(-1)(1/4) - 11/4(-5/11) = 19/2

C_2 = (0+1/2)2/11 = 1/11

C_3 = [0-(-1)(-1/4) - (11/4)(1/11)](2/19) = -1/19

X_2 = 1/11 - (-5/11)(-1/19) = 14/209

X_1 = -1/4-(-1/4)(14/209) - (1/4)(-1/19) = -46/209

C_2 = [1 - (2) *0](2/11) = 2/11

C_3 = [0 - (-1)(0) - (11/4)(2/11)](2/19) = -1/19

X_2 = 2/11 - (-5/11)(-1/19) = 33/209

$X_1 = 0 - (-1/4)(33/209) - (1/4)(-1/19) = 11/209$

$C_2 = [\ 0 - (2)(0)\ ](2/11) = 0$

$C_3 = [\ 1 - (-1)(0) - (11/4)(0)\ ](2/19) = 2/19$

$X_2 = 0 - (-5/11)(2/19) = 10/209$

$X_1 = - (-1/4)(10/209) - (1/4)(2/19) = -3/209$

Implementación de los algoritmos en MATLAB y VISUAL C#:

Para cada lenguaje, se elabora un solo programa, desde el cual se abren archivos para lectura de datos; se ejecuta cada uno de los métodos incluyendo el que calcula la matriz inversa y se escribe los resultados en un archivo de salida, en el caso de Matlab y se despliegan los resultados en una caja lista en el caso de Visual C#.

Archivos fuente en MATLAB:

```
%prbMatInv.m
% APERTURA DE ARCHIVOS:
ent = fopen('D:\AnalNumat\matrizinv.dat', 'r');
sal = fopen('D:\AnalNumat\rsmatrizinv.dat', 'w');
% LECTURA DE LA DIMENSION DEL SISTEMA:
N = fscanf(ent, '%d', 1);
A = LeeMatriz(ent, N);
fprintf(sal, 'CALCULO DE LA MATRIZ INVERSA:\r\n');
fprintf(sal, 'MATRIZ DE COEFICIENTES:\r\n');
EscribeMatriz(sal, A, N);
B = MatInvCr(A, N);
fprintf(sal, 'METODO DE CROUT: MATRIZ INVERSA:\r\n');
EscribeMatriz(sal, B, N);
fprintf(sal, '\r\nUSO DEL METODO MATLAB:\r\n');
B = inv(A);
EscribeMatriz(sal, B, N);
fclose('all');

% MatInvCr.m
% CALCULA LA MATRIZ INVERSA MEDIANTE LA RESOLUCION
%  DE SISTEMAS DE ECUACIONES LINEALES, POR EL METODO DE CROUT
function B = MatInvCr(A, N)
  for I = 1 : N
   for J = 1 : N
      B(I, J) = 0;
   end
   B(I, I) = 1;
  end
  for I = 1 : N
   [det, A, B(:, I)] = Crout(A, B(:, I), N, I);
  end
```

La definición de las funciones LeeMatriz y EscribeMatriz, ya se la realizó anteriormente.

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa, como se había indicado anteriormente, la entrada – salida, se realiza sobre archivos, cuyos contenidos son:

matrizinv.dat:
3
-4
1
-1
2
5
-2
-1
3
8

rsmatrizinv.dat
CALCULO DE LA MATRIZ INVERSA:
MATRIZ DE COEFICIENTES:
FILA 1:
-4.000000 1.000000 -1.000000
FILA 2:
2.000000 5.000000 -2.000000
FILA 3:
-1.000000 3.000000 8.000000
METODO DE CROUT: MATRIZ INVERSA:
FILA 1:
-0.220096 0.052632 -0.014354
FILA 2:
0.066986 0.157895 0.047847
FILA 3:
-0.052632 -0.052632 0.105263

USO DEL METODO MATLAB:
FILA 1:
-0.220096 0.052632 -0.014354
FILA 2:
0.066986 0.157895 0.047847
FILA 3:
-0.052632 -0.052632 0.105263

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 3.3 Como se puede apreciar, existen un cuadro de lista, el cual permitirá presentar los resultados; y cuatro botones, mismos que tienen la funcionalidad:

- Lectura Proceso…: abre una ventana para registrar el nombre del archivo el cual permite leer los datos de la matriz inversa desde el medio de almacenamiento permanente; ejecuta los métodos para calcular la matriz inversa y despliega los resultados.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

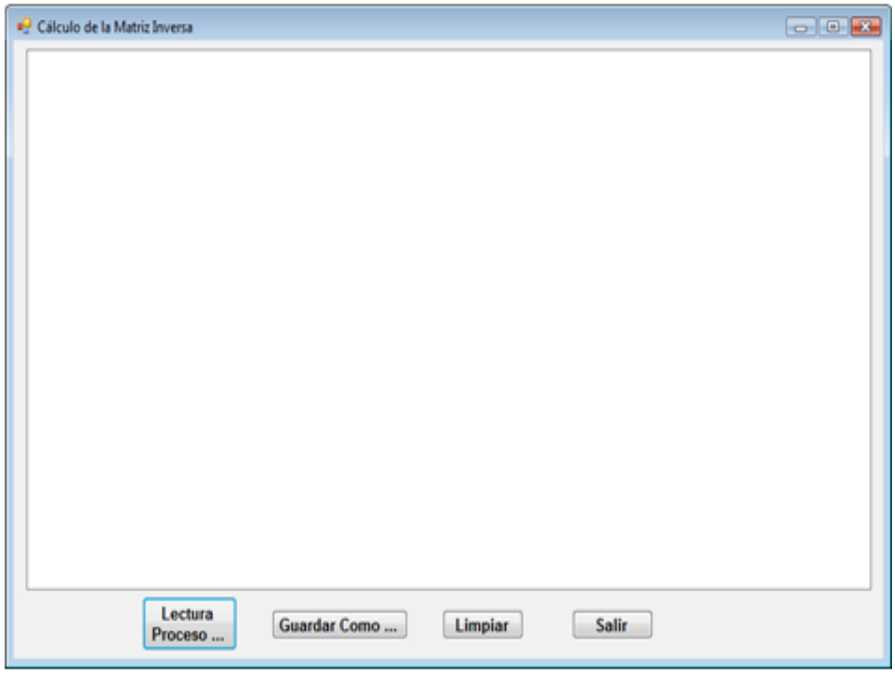


Figura 3.3 Ventana para calcular la matriz inversa en Visual C#

Se describe a continuación el código fuente:

Botón Lectura Proceso…:

```
private void b_lectura_Click(object sender, EventArgs e)
{
  try
  {
    // Abrir el archivo de entrada a traves del cuadro de dialogo:
    CEntSalArchivos objes = new CEntSalArchivos("d:\\");
    objes.AbrirArchivoEnt();
    // Leer el número de ecuaciones o de incógnitas:
    CResolSistemaEcuaciones.n = Int16.Parse(objes.LeerLinea());
```

```
 if (CResolSistemaEcuaciones.n > 0)
 {
    // Definir la matriz de coeficientes:

 CResolSistemaEcuaciones.A = new double[CResolSistemaEcuaciones.n][];
 for (int i = 0; i < CResolSistemaEcuaciones.n; i++)
   CResolSistemaEcuaciones.A[i] = new double[CResolSistemaEcuaciones.n];
 // Leer la matriz de coeficientes y generar los resultados en el listbox:
 // Cálculo de la matriz inversa por el método de Crout:
 CLectEscrMatrices.n = CResolSistemaEcuaciones.n;
 CLectEscrMatrices.a = CResolSistemaEcuaciones.A;
 CLectEscrMatrices.obent = objes;
 CLectEscrMatrices.lb = listBox1;
 CLectEscrMatrices.LeeMatriz();
 listBox1.Items.Add("CALCULO DE LA MATRIZ INVERSA POR EL METODO DE CROUT");
 CLectEscrMatrices.EscribeMatriz(" DE COEFICIENTES:");
 CResolSistemaEcuaciones.MatrizInversa();
 CLectEscrMatrices.EscribeMatriz(" INVERSA:");
 // Cerrar flujo de entrada:
 objes.CerrarLectura();
 listBox1.Items.Add("");
 listBox1.Items.Add("");
 listBox1.Items.Add("");
  }
 }
 catch (Exception ex)
 {
   MessageBox.Show(ex.Message);
 }
}
```

Botón Guardar Como …:

```
private void b_guardar_como_Click(object sender, EventArgs e)
    {
       CListBox.lb = listBox1;
       CListBox.GuardarComo("c:\\");
    }
```

Botón Limpiar:

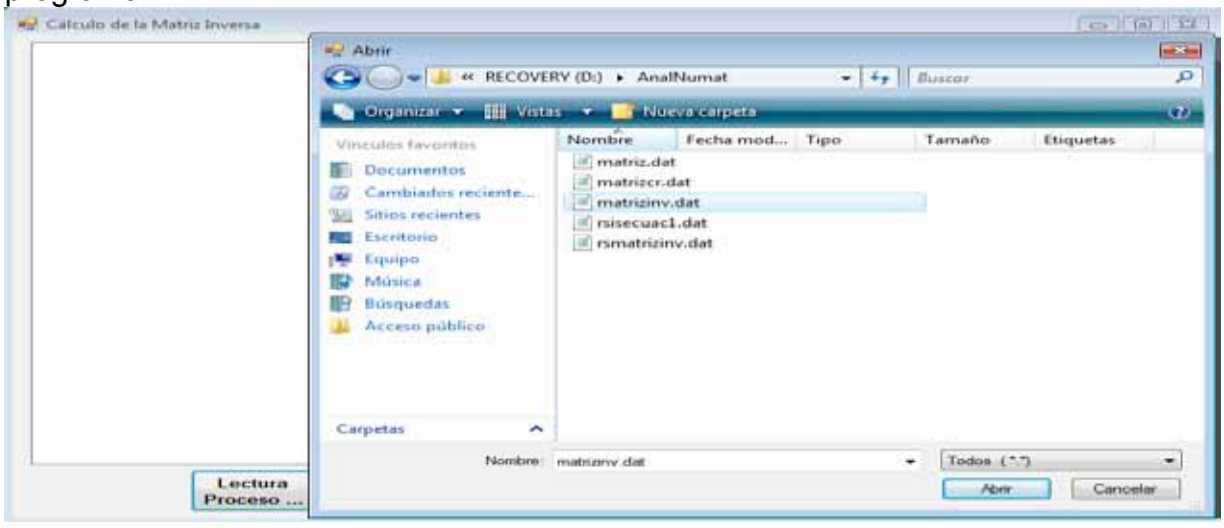
```
private void b_limpiar_Click(object sender, EventArgs e)
    {
       CListBox.lb = listBox1;
       CListBox.Limpiar();
    }
```

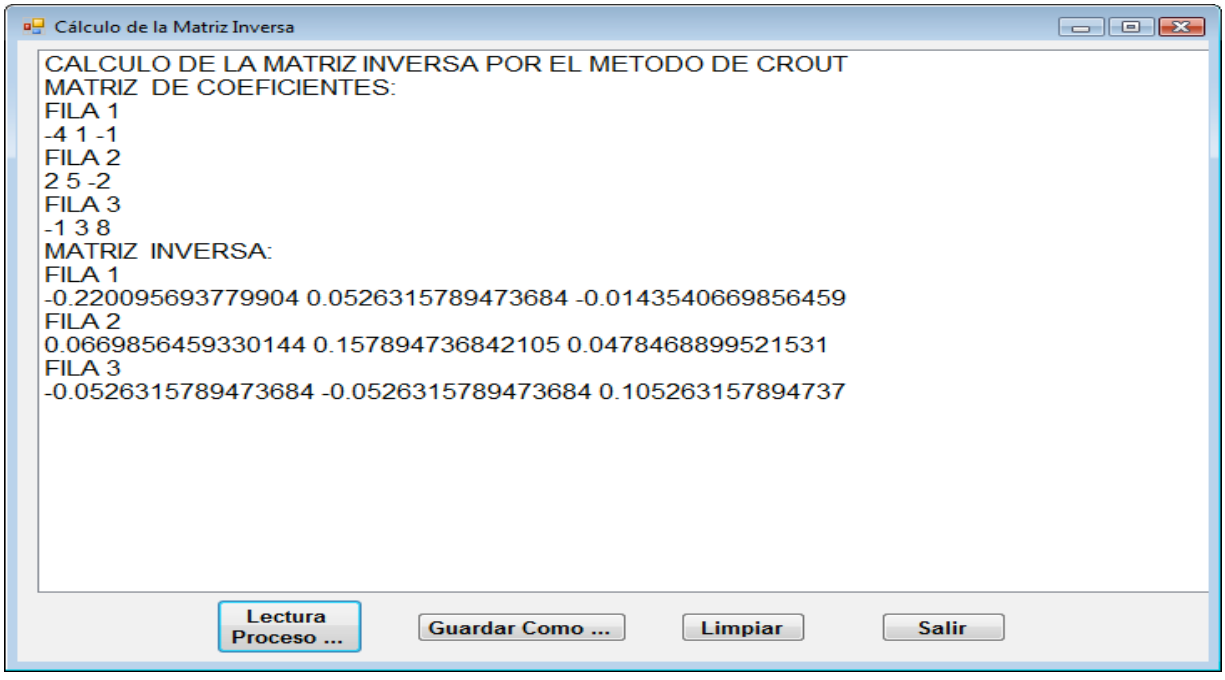
Botón Salir:

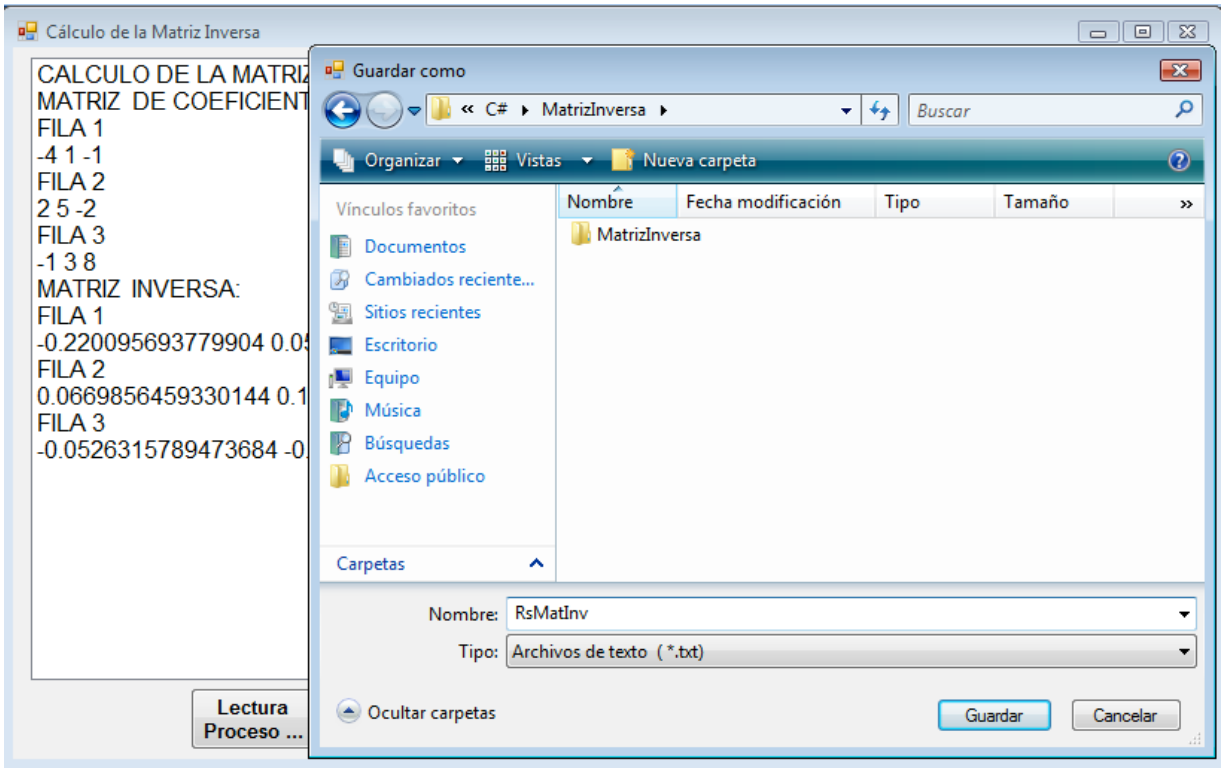
```
private void b_salir_Click(object sender, EventArgs e)
    {
        this.Close();
        this.Dispose();
    }
```

Las referencias a las clases CEntSalArchivos, CResolSistemaEcuaciones, CLectEscrMatrices y CListBox, provienen de la definición de las respectivas clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:






Cálculo de la Matriz Inversa
CALCULO DE LA MATRIZ
MATRIZ DE COEFICIENT
FILA 1
-4 1 -1
FILA 2
2 5 -2
FILA 3
-1 3 8
MATRIZ INVERSA:
FILA 1
-0.220095693779904 0.0
FILA 2
0.0669856459330144 0.1
FILA 3
-0.0526315789473684 -0.
Lectura Proceso ...
Guardar como
« C# ▸ MatrizInversa ▸
Buscar
Organizar
Vistas
Nueva carpeta
Vínculos favoritos
Documentos
Cambiados reciente...
Sitios recientes
Escritorio
Equipo
Música
Búsquedas
Acceso público
Carpetas
Nombre
Fecha modificación
Tipo
Tamaño
MatrizInversa
Nombre: RsMatInv
Tipo: Archivos de texto (*.txt)
Ocultar carpetas
Guardar
Cancelar

RESOLUCIÓN DE SISTEMAS SIMÉTRICOS DE ECUACIONES LINEALES Y SIMULTÁNEOS

Se tiene un sistema simétrico de ecuaciones lineales y simultáneas, cuando la matriz de coeficientes A, es simétrica, es decir:

AX = B

$a_{ij} = a_{ji}$

∀i = 1, 2, 3,……,N
∀j = 1, 2, 3,……,N

Con el fin de sistematizar el método para resolver este tipo de ecuaciones y optimizar tiempo de proceso y espacio de memoria, se realiza las siguientes simplificaciones:

$$u_{ij} = \quad ji / \quad ii \qquad (3.81)$$

Comprobación de la fórmula (3.81):

Para el caso de la solución del ejemplo 3.8 (obsérvese que es un sistema simétrico):

$U_{23} = L_{32}/L_{22} = (-9/4)/(/21/4) = -3/7$

Deducción de fórmulas:

Sustituyendo (3.81) en las fórmulas (3.37) y (3.57) :

$$ij = a_{ij} - \sum_{k=1}^{j-1} ik \quad jk / \quad kk \qquad (3.82)$$

$$x_N = c_N \quad (3.87)$$

Con el objeto de sistematizar la resolución de este tipo de sistemas. En los procesos

$$_{i1} = a_{i1} \quad (3.83)$$

$$ind(i,j) = (i^2 - i + 2j)/2 \quad (3.88)$$

$$c_i = (b_i - \sum_{j=1} {}_{ij}c_j)/ {}_{ii} \quad (3.84)$$

$$c_1 = b_1 / {}_{11} \quad (3.85)$$

$$x_i = c_i - (\sum_{j=i+1}^{N} {}_{ji}x_j)/ {}_{ii} \quad (3.86)$$

de lectura – escritura de la matriz de coeficientes y en el método de crout, se emplea la función:

La cual, transforma los índices matriciales (i, j) en un índice lineal ind, convirtiendo de esta manera la matriz de coeficientes en un vector. La figura 3.4 ilustra esta conversión.

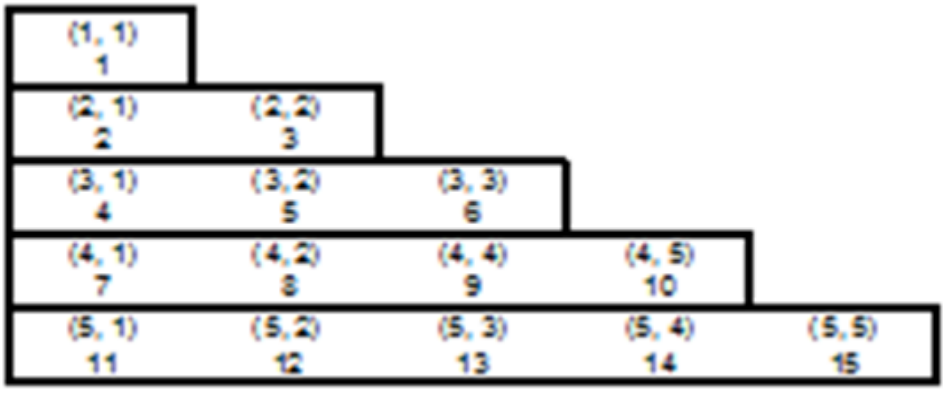


Figura 3.4 Conversión de índices matriciales en lineales en una Matriz Simétrica

Algoritmo para el método de Crout Simétrico:

1) Aplicando la fórmula 3.83 se determinan los valores de la primera columna de L.
2) Aplicando la fórmula 3.82 se calcula la siguiente columna de la matriz L.
3) Repetir el paso 2 hasta calcular todas las columnas de la matriz L.
4) Aplicando la fórmula 3.85, se calcula el primer elemento del vector C.
5) Aplicando la fórmula 3.84 se calcula el siguiente elemento del vector C.
6) Repetir el paso 5 hasta calcular todos los elementos del vector C.
7) Aplicando la fórmula 3.87 se determina el último elemento del vector X.
8) Aplicando la fórmula 3.86 se calcula el elemento anterior del vector X.
9) Repetir el paso 8 hasta calcular todos los elementos del vector X.

Ejemplo 3.10:

Resolver el sistema de ecuaciones lineales y simultáneas del ejemplo 3.8.

Solución:

Se plantea el esquema:

-4 -4			
1 1	5 21/4		
-1 -1	-2 -9/4	8 51/7	
2 2	-4 -7/2	3 1	-5 -110/17

8	
-2	-178/99
-2	
0	164/495
4	
14/51	112/495
-6	
58/465	58/165

Los cálculos típicos, se detallan a continuación:

$l_{43} = a_{43} - \sum l_{ij}\, l_{ik} / l_{kk}$

$= a_{43} - l_{41}\, l_{31} / l_{11} - l_{42}\, l_{32} / l_{22}$

$l_{33} = a_{33} - \sum l_{3j}\, l_{3k} / l_{kk}$

$= a_{43} - l_{31}^{2} / l_{11} - l_{32}^{2} / l_{22}$

$l_{22} = 5 - 1^2 / (-4) = 21/4$

$l_{32} = -2 - (-1)(1) /(-4) = -9/4$

$l_{42} = -4 - 2(1) /(4) = -7/2$

$l_{33} = 8 - (-1)^2/ (-4) - (-9/4)^2/ 21/4 = 51/7$

$l_{43} = 3 - (2)(-1) /(-4) - (-7/2)(-9/4)/21/4 = 1$

$l_{44} = -5-4/(-4)-(49/4)/(21/4) - 1 /(51/7) = -110/17$

c2 = (-2 - (1)(-2)) 2/21 = 0
c3 = (4 - (-1)(-2) - (-9/4)(0)) 7/51 = 14/51

x4 = c4 = 58/165

x3 = 14/51 – 7/51(58/165) = 112/495

x2 = 0 – (-3/7(112/495) + (-2/3)(58/165) = 164/495

x1 = -2 – (-1/4(164/495) + 1/4(112/495) + (-1/2)(58/165)) = -178/99

Cálculo del determinante:

det = (-4)(21/4)(51/7)(-110/17) = 990

Implementación de los algoritmos en MATLAB y VISUAL C#:

Para cada lenguaje, se elabora un solo programa, desde el cual se abren archivos para lectura de datos; se ejecuta cada uno de los métodos incluyendo el que resuelve el sistema simétrico de ecuaciones lineales y simultáneas y se escribe los resultados en un archivo de salida, en el caso de Matlab y se despliegan los resultados en una caja lista en el caso de Visual C#.

Archivos fuente en MATLAB:

```
% SisEcuac2.m
ent = fopen('D:\AnalNumat\matrizsm.dat', 'r');
sal = fopen('D:\AnalNumat\rsisecuac2.dat', 'w');
% LECTURA DEL NUMERO DE VECTORES A PROCESAR
NS = fscanf(ent, '%d', 1);
% LECTURA DE LA DIMENSION DEL SISTEMA:
N = fscanf(ent, '%d', 1);
%LECTURA DE LA MATRIZ DE COEFICIENTES
A = LeeMatSim(ent, N);
fprintf(sal, 'RESOLUCION DE SISTEMAS SIMETRICOS DE ECUACIONES LINEALES Y SIMULTANEAS:\r\n');
fprintf(sal, 'METODO DE CROUT SIMETRICO:\r\nMATRIZ DE COEFICIENTES:\r\n');
EscribeMatSim(sal, A, N);
for i = 1 : NS
  B = LeeVector(ent, N);
  fprintf(sal, 'VECTOR DE TERMINOS INDEPENDIENTES:\r\n');
  EscribeVector(sal, B, N);
  [det, A, B] = CroutSim(A, B, N, i);
  if i == 1
    fprintf(sal, 'DETERMINANTE = %f\r\n', det);
  end;
  fprintf(sal, 'VECTOR DE SOLUCIONES:\r\n');
  EscribeVector(sal, B, N);
end;
fclose('all');

% Ind.m
function a = Ind(i, j)
a = (i ^ 2 - i + 2 * j) / 2;
return

% LeeMatSim.m
function A = LeeMatSim(f, m)
for i = 1 : m
   for j = 1 : i
```

```
    A(Ind(i, j)) = fscanf(f, '%f', 1);
  end;
end;
return

% EscribeMatSim.m
function EscribeMatSim(f, A, m)
for i = 1 : m
  fprintf(f, 'FILA   %d:\r\n',i);
  for j = 1 : m
    if i > j
      fprintf(f, '%f  ',A(Ind(i, j)));
    else
      fprintf(f, '%f  ',A(Ind(j, i)));
    end;
    if (mod(j, 8) == 0) | (j == m)
      fprintf(f, '\r\n');
    end;
  end;
end;
return

% croutsim.m
function [det, A, B] = CroutSim(A, B, N, St)
det = 1;
if St == 1 % calcular las matrices L, U y el determinante:
  det = A(Ind(1, 1));
  for i = 2 : N
    for j = 2 : i
      for k = 1 : j - 1
        A(Ind(i, j)) = A(Ind(i, j)) - A(Ind(i, k)) * A(Ind(j, k)) / A(Ind(k, k));
      end;
    end;
    det = det * A(Ind(i,i));
  end;
end;
% Cálculo del vector C:
for i = 1 : N
  for j = 1 : i - 1
    B(i) = B(i) - A(Ind(i, j)) * B(j);
  end;
  B(i) = B(i) / A(Ind(i, i));
end;
% Cálculo del vector X:
for i = N - 1 : -1 : 1
  for j = i + 1 : N
    B(i) = B(i) - A(Ind(j, i)) * B(j) / A(Ind(i, i));
  end;
end;
```

La definición de las funciones LeeVector y EscribeVector, ya se la realizó anteriormente.

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa, como se había indicado anteriormente, la entrada – salida, se realiza sobre archivos, cuyos contenidos son:

matrizsm.dat:
2
4
-4
1
5
-1
-2
8
2
-4
3
-5
8
-2
4
-6
4
3
2
1

rsisecuac2.dat:
RESOLUCION DE SISTEMAS SIMETRICOS DE ECUACIONES LINEALES Y SIMULTANEAS:
METODO DE CROUT SIMETRICO:
MATRIZ DE COEFICIENTES:
FILA 1:
-4.000000 1.000000 -1.000000 2.000000
FILA 2:
1.000000 5.000000 -2.000000 -4.000000
FILA 3:
-1.000000 -2.000000 8.000000 3.000000
FILA 4:
2.000000 -4.000000 3.000000 -5.000000
VECTOR DE TERMINOS INDEPENDIENTES:
8.000000 -2.000000 4.000000 -6.000000
DETERMINANTE = 990.000000
VECTOR DE SOLUCIONES:
-1.797980 0.331313 0.226263 0.351515
VECTOR DE TERMINOS INDEPENDIENTES:
4.000000 3.000000 2.000000 1.000000
VECTOR DE SOLUCIONES:
-1.424242 0.424242 0.484848 -0.818182

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 3.5 Como se puede apreciar, existen un cuadro de lista, el cual permitirá presentar los resultados; y cuatro botones, mismos que tienen la funcionalidad:

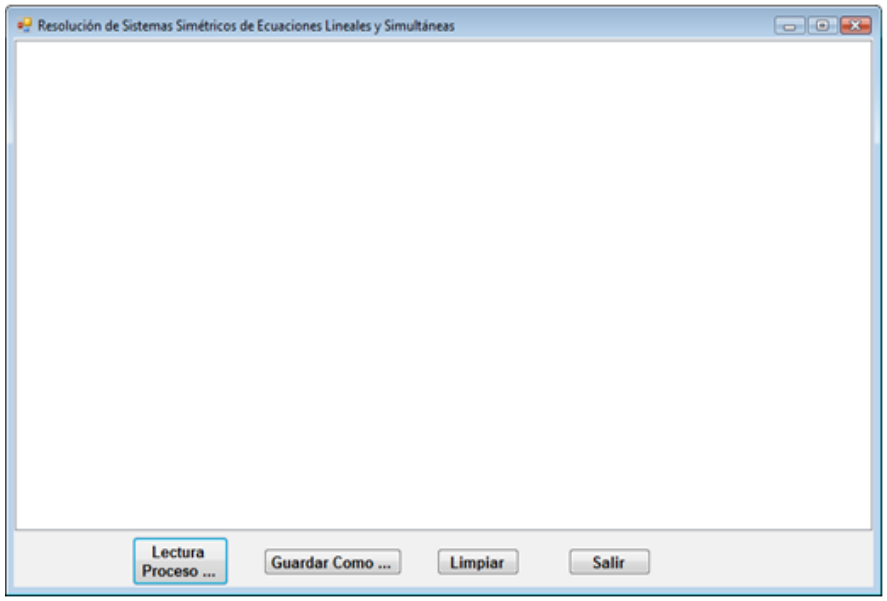


Figura 3.5 Ventana para resolver sist. simétricos de ecuaciones en Visual C#

- Lectura Proceso…: abre una ventana para registrar el nombre del archivo el cual permite leer los datos del sistema simétrico de ecuaciones lineales y simultáneas desde el medio de almacenamiento permanente; ejecuta los métodos para calcular la matriz inversa y despliega los resultados.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Botón Lectura Proceso…:

```
private void b_lectura_Click(object sender, EventArgs e)
{
  try
  {
    int i, ns;
    // Abrir el archivo de entrada a traves del cuadro de dialogo:
    CEntSalArchivos objes = new CEntSalArchivos("d:\\");
    objes.AbrirArchivoEnt();
    // Leer el número de procesamiento de soluciones:
    ns = Int16.Parse(objes.LeerLinea());
    // Leer el número de ecuaciones o de incógnitas:
    CResolSisSimetricosEcuaciones.n = Int16.Parse(objes.LeerLinea());
    if (CResolSisSimetricosEcuaciones.n > 0)
```

```
    {
      double det;
      int n = CResolSisSimetricosEcuaciones.n;
      // Definir el arreglo de términos independientes:
      CResolSisSimetricosEcuaciones.B = new double[n];
      // Definir la matriz de coeficientes:
      CResolSisSimetricosEcuaciones.A = new double[n * (n + 1) / 2];
      /* Leer la matriz de coeficientes, el arreglo de términos independientes y generar los resultados en
el listbox: */
      // Prueba del método de Crout:
      CLectEscrMatrices.n = n;
      CLectEscrMatrices.c = CResolSisSimetricosEcuaciones.A;
      CLectEscrMatrices.b = CResolSisSimetricosEcuaciones.B;
      CLectEscrMatrices.obent = objes;
      CLectEscrMatrices.lb = listBox1;
      CLectEscrMatrices.LeeMatrizSimetrica();
      listBox1.Items.Add("RESOLUCION DE SISTEMAS SIMETRICOS DE ECUACIONES LINEALES Y
SIMULTANEAS");
      listBox1.Items.Add("POR EL METODO DE CROUT");
      CLectEscrMatrices.EscribeMatrizSimetrica(" DE COEFICIENTES:");
      for (i = 0; i < ns; i++)
      {
        CLectEscrMatrices.LeeVector();
        CLectEscrMatrices.EscribeVector(" DE TERMINOS INDEPENDIENTES:");
        det = CResolSisSimetricosEcuaciones.CroutSimetrico(i == 0);
        CLectEscrMatrices.EscribeVector(" SOLUCION:");
        if (i == 0) listBox1.Items.Add("DETERMINANTE = " + det.ToString());
      }
      // Cerrar flujo de entrada:
      objes.CerrarLectura();
      listBox1.Items.Add("");
      listBox1.Items.Add("");
      listBox1.Items.Add("");
    }
   }
   catch (Exception ex)
   {
     MessageBox.Show(ex.Message);
   }
  }
```

Botón Guardar Como …:

```
  private void b_guardar_como_Click(object sender, EventArgs e)
      {
         CListBox.lb = listBox1;
         CListBox.GuardarComo("c:\\");
      }
```

Botón Limpiar:

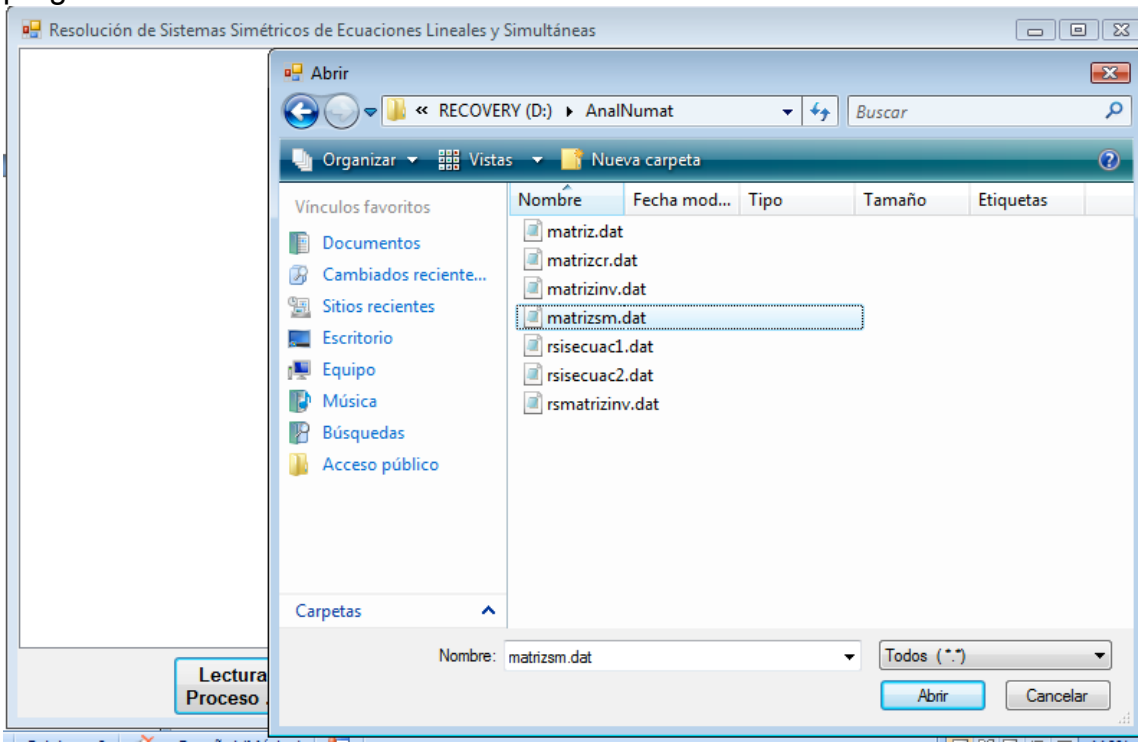
```
private void b_limpiar_Click(object sender, EventArgs e)
    {
        CListBox.lb = listBox1;
        CListBox.Limpiar();
    }
```

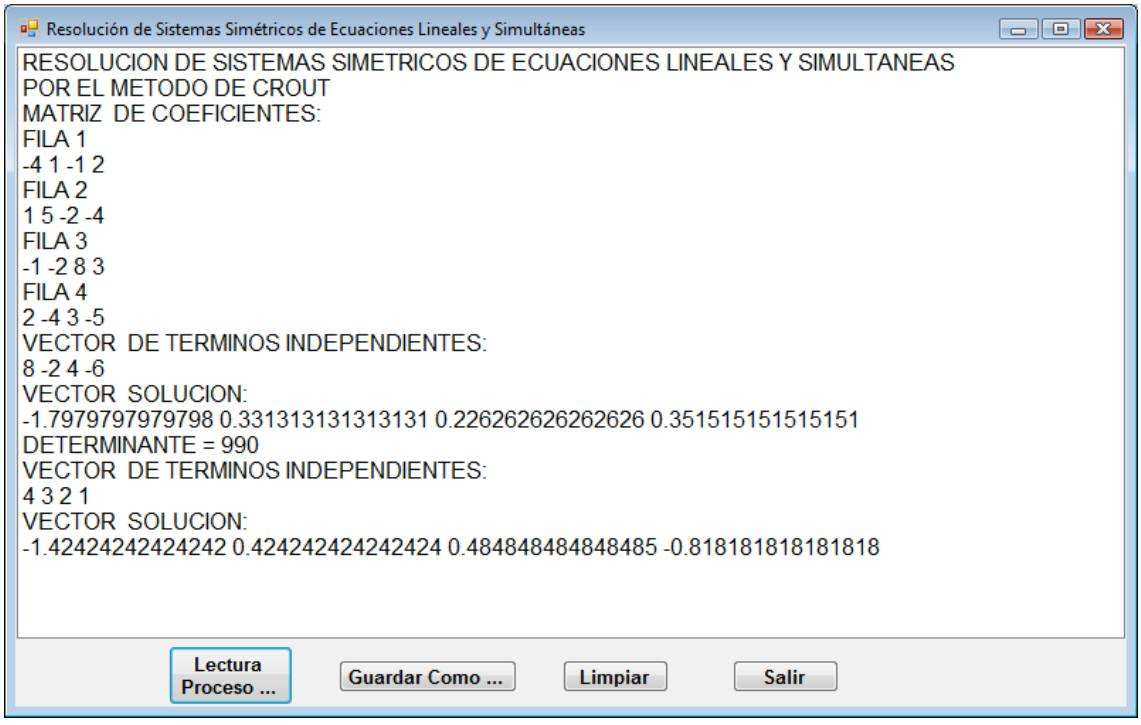
Botón Salir:

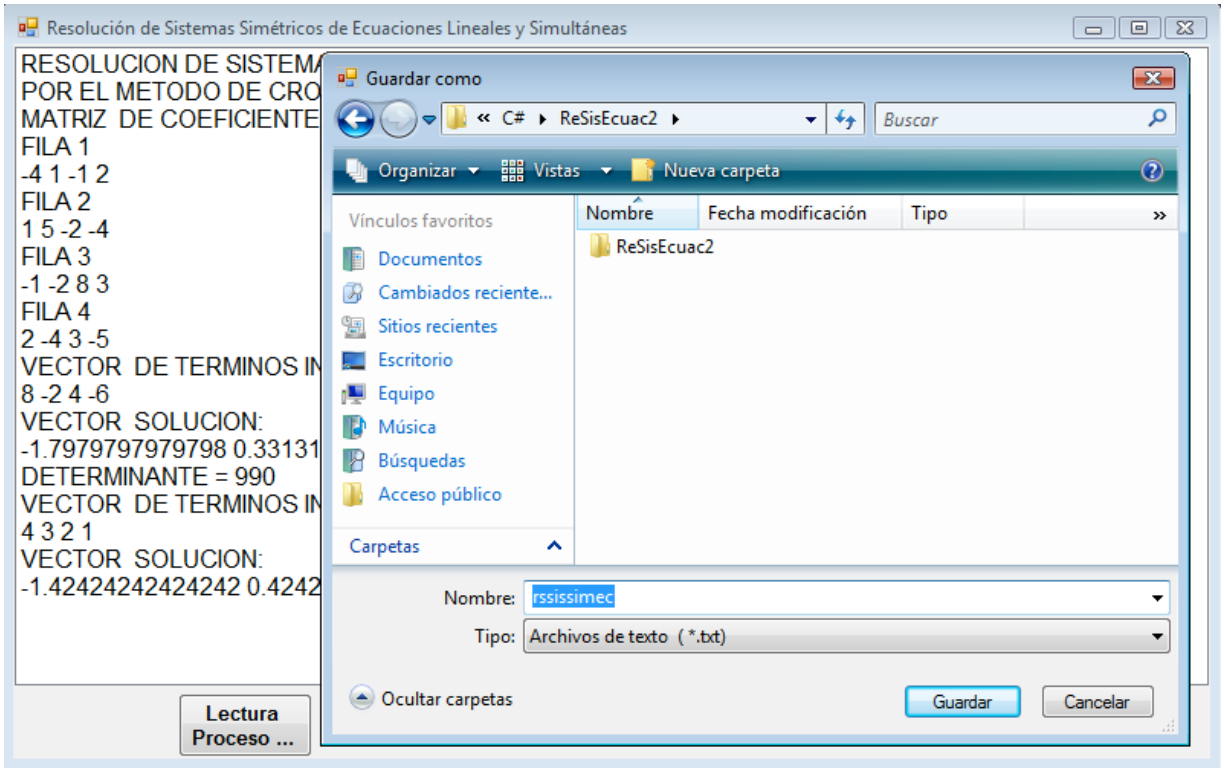
```
private void b_salir_Click(object sender, EventArgs e)
    {
        this.Close();
        this.Dispose();
    }
```

Las referencias a las clases CEntSalArchivos, CResolSisSimetricosEcuaciones, CLectEscrMatrices y CListBox, provienen de la definición de las respectivas clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:




Resolución de Sistemas Simétricos de Ecuaciones Lineales y Simultáneas
RESOLUCION DE SISTEMAS SIMETRICOS DE ECUACIONES LINEALES Y SIMULTANEAS
POR EL METODO DE CROUT
MATRIZ DE COEFICIENTES:
FILA 1
-4 1 -1 2
FILA 2
1 5 -2 -4
FILA 3
-1 -2 8 3
FILA 4
2 -4 3 -5
VECTOR DE TERMINOS INDEPENDIENTES:
8 -2 4 -6
VECTOR SOLUCION:
-1.7979797979798 0.331313131313131 0.226262626262626 0.351515151515151
DETERMINANTE = 990
VECTOR DE TERMINOS INDEPENDIENTES:
4 3 2 1
VECTOR SOLUCION:
-1.42424242424242 0.424242424242424 0.484848484848485 -0.818181818181818
Lectura Proceso ...
Guardar Como ...
Limpiar
Salir


Resolución de Sistemas Simétricos de Ecuaciones Lineales y Simultáneas
Guardar como
« C# ▸ ReSisEcuac2 ▸
Buscar
Organizar
Vistas
Nueva carpeta
Vínculos favoritos
Documentos
Cambiados reciente...
Sitios recientes
Escritorio
Equipo
Música
Búsquedas
Acceso público
Carpetas
Nombre
Fecha modificación
Tipo
ReSisEcuac2
Nombre: rssissimec
Tipo: Archivos de texto (*.txt)
Ocultar carpetas
Guardar
Cancelar
Lectura Proceso ...

RESOLUCIÓN DE SISTEMAS SIMÉTRICOS EN BANDA DE ECUACIONES LINEALES Y SIMULTÁNEOS

Un sistema simétrico en banda de ecuaciones lineales y simultáneas, es un caso particular de un sistema simétrico de ecuaciones lineales y simultáneas, en el que la matriz de coeficientes A, se encuentra formada por valores diferentes de cero en la franja de la diagonal principal y el resto de elementos igual a cero. De esta manera, para la resolución, se toma la mitad de la franja, cuyo ancho es M (ver figura 3.6).

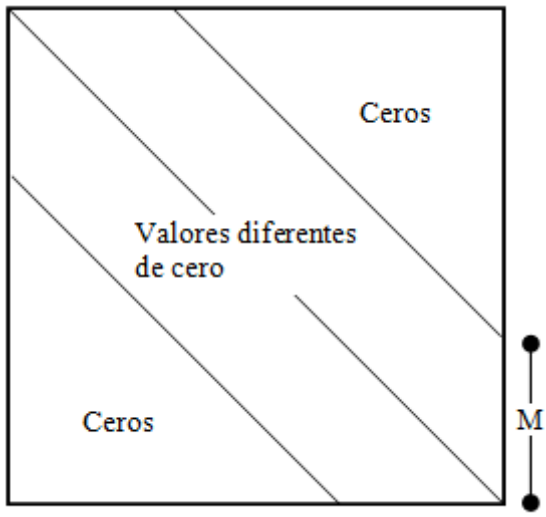


Figura 3.6 Matriz de Coeficientes

Las formulas para la resolución de este tipo de sistemas de ecuaciones, sigue siendo las que se emplean para los sistemas simétricos, sin considerar los ceros que se encuentran fuera de la franja.

Con el objeto de sistematizar la resolución de este tipo de sistemas. En los procesos de lectura – escritura de la matriz de coeficientes y en el método de crout, se emplea la función:

$$inb(i,j,m) = \begin{cases} (i^2 - i + 2j)/2; i \le m \\ (2im - 2i + 2j - m^2 + m)/2; i > m \end{cases} \qquad (3.91)$$

La cual, transforma los índices matriciales (i, j) en un índice lineal inb, convirtiendo de esta manera la matriz de coeficientes en un vector. Se debe tener en cuenta que i > 0 y j > 0. Para valores desde cero pueden producirse índices repetidos. La figura 3.7 ilustra esta conversión.

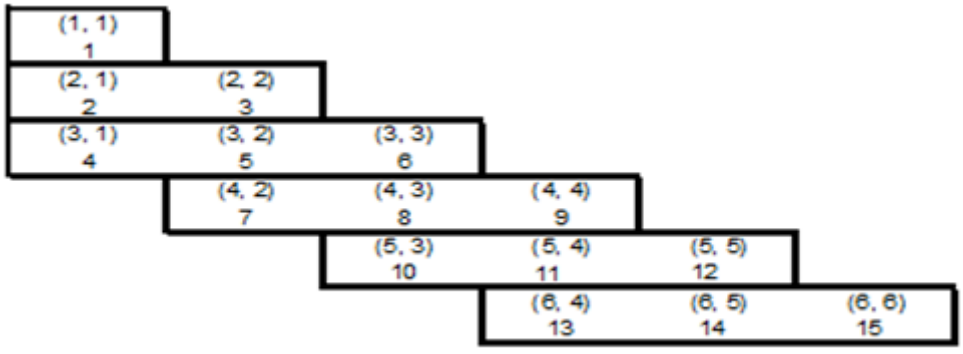


Figura 3.7 Conversión de índices matriciales en lineales en una Matriz Banda

Ejemplo 3.11:

Dado el sistema:

6X1 + 3X2 + 2X3 = 4
3X1 + 4X2 + X3 + 2X4 = 3
2X1 + X2 + 5X3 + 3X4 + 4X5 = 1
2X2 + 3X3 + 7X4 + 5X5 + 3X6 = -2
4X3 + 5X4 + 2X5 + 2X6 = 5
3X4 + 2X5 + 8X6 = 6

Resolverlo por el método de Crout y calcular el determinante de la matriz de coeficientes.

Solución:

Planteamiento matricial:

$$\begin{pmatrix} 6 & 3 & 2 & 0 & 0 & 0 \\ 3 & 4 & 1 & 2 & 0 & 0 \\ 2 & 1 & 5 & 3 & 4 & 0 \\ 0 & 2 & 3 & 7 & 5 & 3 \\ 0 & 0 & 4 & 5 & 2 & 2 \\ 0 & 0 & 0 & 3 & 2 & 8 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \end{pmatrix} = \begin{pmatrix} 4 \\ 3 \\ 1 \\ 2 \\ 5 \\ 6 \end{pmatrix}$$

Como se puede observar, es un sistema simétrico en banda, de 6 ecuaciones, con un ancho M = 3.

Se plantea el esquema:

6					
6					
3	4				
3	2 1/2				
2	1	5			
2	0	4 1/3			
	2	3	7		
	2	3	3 21/65		
		4	5	2	
		4	2 3/13	-3 41/216	
			3	2	8
			3	- 1/72	5 201/689

4	
2/3	-2302/3646
3	
2/5	3508/3646
1	
- 1/13	8936/3646
-2	
- 167/216	-2562/3646
5	
-2 141/689	-8063/3646
6	
5711/3646	5711/3646

Los cálculos típicos, se detallan a continuación:

L22 = 4 - 3^2/6 = 5/2
L32 = 1 - 3*2/6 = 0
L33 = 5 - 2^2/6 = 13/3
L43 = 3 - 0 = 3
L44 = 7 - 2/(5/2) - 3/(13/3) = 216/65
L54 = 5 - 3 * 4 /(13/3) = 29/13
L55 = 2 - 4^2/(13/3) - (29/13)^2 / (216/65) = -689/216

L65 = 2 - (29/13)*3/(216/65) = -1/72
L66 = 8 - 3^2 / (216/65) - (-1/72)^2 / (-689/216) = 3646/689
c1 = 4/6 = 2/3
c2 = (3 - 3*(2/3))/(5/2) = 2/5
c3 = (1 - 2*(2/3) - 0)/(13/3) = -1/13
c4 = (-2 - 2*(2/5) - 3*(-1/13))/(216/65) = -167/216
c5 = (5 - 4*(-1/13) - (29/13)*(-167/216))/(-689/216) = -1519/689
c6 = (6 - 3*(-167/216) - (-1/72)*(-1519/689))/(3646/689) = 5711/3646
x6 = c6 = 5711/3646
x5 = -1519/689 - (-1/72)*(5711/3646)/(-689/216) = -8063/3646
x4 = -167/216 - ((29/13)*(-8063/3646)+3*(5711/3646))/(216/65) = -2562/3646
x3 = -1/13 - (3*(-2562/3646) + 4*(-8063/3646))/(13/3) = 8936/3646
x2 = 2/5 - (0 + 2*(-2562/3646))/(5/2) = 3508/3646
x1 = 2/3 - (3*(3508/3646) + 2*(8936/3646))/6 = -2302/3646
Det = 6*(5/2)*(13/3)*(216/65)*(-689/216)*(3646/689) = -3646

Implementación de los algoritmos en MATLAB y VISUAL C#:

Para cada lenguaje, se elabora un solo programa, desde el cual se abren archivos para lectura de datos; se ejecuta cada uno de los métodos incluyendo el que resuelve el sistema simétrico en banda de ecuaciones lineales y simultáneas y se escribe los resultados en un archivo de salida, en el caso de Matlab y se despliegan los resultados en una caja lista en el caso de Visual C#.

Archivos fuente en MATLAB:

```
% SisEcuac3.m
ent = fopen('\\VBOXSVR\compartido\AnalisisNumerico\AnalNumat\matrizbd.dat', 'r');
sal = fopen('\\VBOXSVR\compartido\AnalisisNumerico\AnalNumat\rsisecuac3.dat', 'w');
% LECTURA DEL NUMERO DE VECTORES DE TI A PROCESAR
NS = fscanf(ent, '%d', 1);
% LECTURA DE LA DIMENSION DEL SISTEMA:
N = fscanf(ent, '%d', 1);
%LECTURA DEL ANCHO DE BANDA
M = fscanf(ent, '%d', 1);
%LECTURA DE LA MATRIZ DE COEFICIENTES
A = LeeMatBan(ent, N, M);
fprintf(sal, 'RESOLUCION DE SISTEMAS SIMETRICOS EN BANDA DE ECUACIONES LINEALES Y
SIMULTANEAS:\n');
fprintf(sal, 'MATRIZ DE COEFICIENTES:\n');
EscribeMatBan(sal, A, N, M);
for i = 1 : NS
  B = LeeVector(ent, N);
  fprintf(sal, 'VECTOR DE TERMINOS INDEPENDIENTES:\n');
  EscribeVector(sal, B, N);
  [det, A, B] = CroutBan(A, B, N, M, i);
  if i == 1
    fprintf(sal, 'DETERMINANTE = %f\n', det);
  end;
  fprintf(sal, 'VECTOR DE SOLUCIONES:\n');
```

```
  EscribeVector(sal, B, N);
end;
fclose('all');

% Inb.m
% TRANSFORMA LOS INDICES MATRICIALES I,J EN EL INDICE VECTORIAL INB,
%   PARA UNA MATRIZ BANDA INFERIOR  DE ANCHO M */
function a = Inb(i, j, m)
if (i > m)
   a = (m - m ^ 2 + 2 * i * m - 2 * i + 2 * j) / 2;
else
   a = (i ^ 2 - i + 2 * j) / 2;
end;
return;

% LeeMatBan.m
function A = LeeMatBan(f, n, m)
for i = 1 : n
   l = 1;
   if (i > m) l = i - m + 1; end;
   for j = l : i
      A(Inb(i, j, m)) = fscanf(f, '%f', 1);
   end;
end;
return;

% EscribeMatBan.m
function EscribeMatBan(f, A, n, m)
for i = 1 : n
   fprintf(f, 'FILA   %d:\r\n',i);
   for j = 1 : n
      fprintf(f, '%f  ',RecuBan(A, i, j, m));
      if (mod(j, 8) == 0) | (j == n)
         fprintf(f, '\r\n');
      end;
   end;
end;
return

% RecuBan.m
function v = RecuBan(A, i, j, m)
if (j <= i - m) | (j >= i + m)
   v = 0;
elseif (i < j)
   v = A(Inb(j, i, m));
else
   v = A(Inb(i, j, m));
end;
return;

% croutban.m
function [det, A, B] = CroutBan(A, B, N, m, St)
det = 1;
if St == 1 % calcular las matrices L, U y el determinante:
   det = A(Inb(1, 1, m));
   for i = 2 : N
      l = 2;
      if (i > m) l = i - m + 2; end;
      for j = l : i
         for k = l - 1 : j - 1
            A(Inb(i, j, m)) = A(Inb(i, j, m)) - A(Inb(i, k, m)) * A(Inb(j, k, m)) / A(Inb(k, k, m));
```

```
        end;
      end;
      det = det * A(Inb(i,i, m));
    end;
end;
% Cálculo del vector C:
for i = 1 : N
  l = 1;
  if (i > m) l = i - m + 1; end;
  for j = l : i - 1
    B(i) = B(i) - A(Inb(i, j, m)) * B(j);
  end;
  B(i) = B(i) / A(Inb(i, i, m));
end;
% Cálculo del vector X:
for i = N - 1 : -1 : 1
  l = m + i - 1;
  if (i > N - m) l = N; end;
  for j = i + 1 : l
    B(i) = B(i) - A(Inb(j, i, m)) * B(j) / A(Inb(i, i, m));
  end;
end;
```

La definición de las funciones LeeVector y EscribeVector, ya se la realizó anteriormente.

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa, como se había indicado anteriormente, la entrada – salida, se realiza sobre archivos, cuyos contenidos son:

matrizbd.dat:
2
6
3
6
3
4
2
1
5
2
3
7
4
5
2
3
2
8
4

3
1
-2
5
6
1
0
0
0
0
0

rsisecuac3.dat:
RESOLUCION DE SISTEMAS SIMETRICOS EN BANDA DE ECUACIONES LINEALES Y SIMULTANEAS:
MATRIZ DE COEFICIENTES:
FILA 1:
6.000000 3.000000 2.000000 0.000000 0.000000 0.000000
FILA 2:
3.000000 4.000000 1.000000 2.000000 0.000000 0.000000
FILA 3:
2.000000 1.000000 5.000000 3.000000 4.000000 0.000000
FILA 4:
0.000000 2.000000 3.000000 7.000000 5.000000 3.000000
FILA 5:
0.000000 0.000000 4.000000 5.000000 2.000000 2.000000
FILA 6:
0.000000 0.000000 0.000000 3.000000 2.000000 8.000000
VECTOR DE TERMINOS INDEPENDIENTES:
4.000000 3.000000 1.000000 -2.000000 5.000000 6.000000
DETERMINANTE = -3646.000000
VECTOR DE SOLUCIONES:
-0.631377 0.962150 2.450905 -0.702688 -2.211465 1.566374
VECTOR DE TERMINOS INDEPENDIENTES:
1.000000 0.000000 0.000000 0.000000 0.000000 0.000000
VECTOR DE SOLUCIONES:
0.469007 -0.409764 -0.292375 0.262205 0.036753 -0.107515

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 3.8

Como se puede apreciar, existen un cuadro de lista, el cual permitirá presentar los resultados; y cuatro botones, mismos que tienen la funcionalidad:

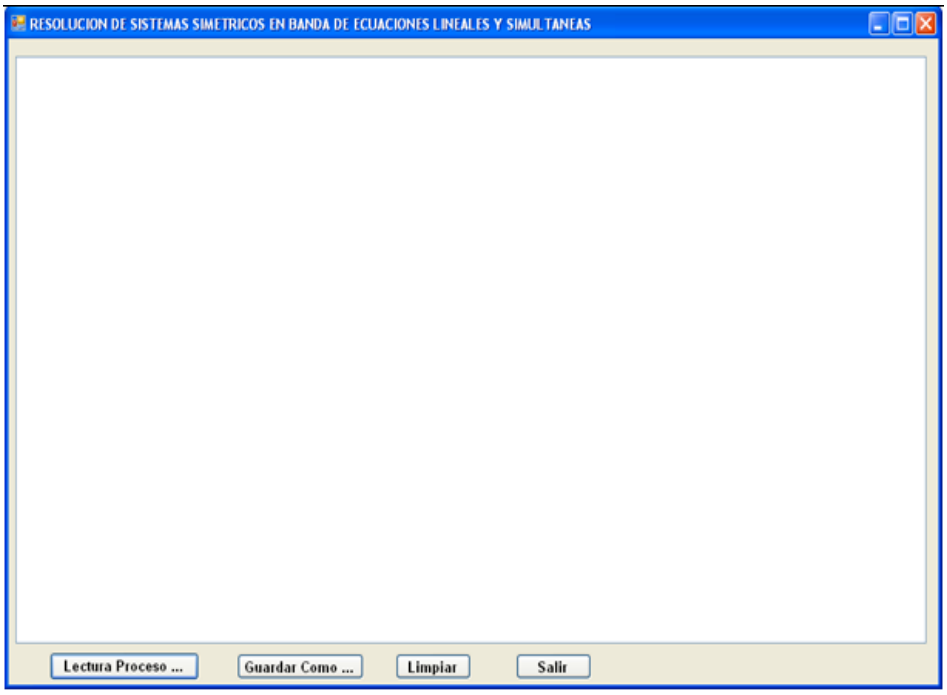


Figura 3.8 Ventana para resolver sistemas simétricos en banda de ecuaciones en Visual C#

- Lectura Proceso…: abre una ventana para registrar el nombre del archivo el cual permite leer los datos del sistema simétrico en banda de ecuaciones lineales y simultáneas desde el medio de almacenamiento permanente; ejecuta los métodos para calcular la matriz inversa y despliega los resultados.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Botón Lectura Proceso…:

```
private void b_lectura_Click(object sender, EventArgs e)
{
  try
  {
    int i, ns;
    // Abrir el archivo de entrada a traves del cuadro de dialogo:
    CEntSalArchivos objes = new CEntSalArchivos("c:\\");
    objes.AbrirArchivoEnt();
    // Leer el número de procesamiento de soluciones:
    ns = Int16.Parse(objes.LeerLinea());
    // Leer el número de ecuaciones o de incógnitas:
    CResolSisSimBanda.n = Int16.Parse(objes.LeerLinea());
    if (CResolSisSimBanda.n > 0)
    {
      double det;
      int n = CResolSisSimBanda.n;
```

```
      // Leer el ancho de banda:
      CResolSisSimBanda.m = Int16.Parse(objes.LeerLinea());
      int m = CResolSisSimBanda.m;
      // Definir el arreglo de términos independientes:
      CResolSisSimBanda.B = new double[n];
      // Definir la matriz de coeficientes:
      CResolSisSimBanda.A = new double[m * (m + 1) / 2 + m * (n - m) + 1];
      /* Leer la matriz de coeficientes, el arreglo de términos independientes y generar los resultados en
el listbox: */
      // Prueba del método de Crout:
      CLectEscrMatrices.n = n;
      CLectEscrMatrices.m = m;
      CLectEscrMatrices.c = CResolSisSimBanda.A;
      CLectEscrMatrices.b = CResolSisSimBanda.B;
      CLectEscrMatrices.obent = objes;
      CLectEscrMatrices.lb = listBox1;
      CLectEscrMatrices.LeeMatrizSimBanda();
      listBox1.Items.Add("RESOLUCION DE SISTEMAS SIMETRICOS EN BANDA DE ECUACIONES
LINEALES Y SIMULTANEAS");
      listBox1.Items.Add("POR EL METODO DE CROUT");
      CLectEscrMatrices.EscribeMatrizSimBanda(" DE COEFICIENTES:");
      for (i = 0; i < ns; i++)
      {
        CLectEscrMatrices.LeeVector();
        CLectEscrMatrices.EscribeVector(" DE TERMINOS INDEPENDIENTES:");
        det = CResolSisSimBanda.CroutSimBanda(i == 0);
        CLectEscrMatrices.EscribeVector(" SOLUCION:");
        if (i == 0) listBox1.Items.Add("DETERMINANTE = " + det.ToString());
      }
      // Cerrar flujo de entrada:
      objes.CerrarLectura();
      listBox1.Items.Add("");
      listBox1.Items.Add("");
      listBox1.Items.Add("");
    }
  }
  catch (Exception ex)
  {
    MessageBox.Show(ex.Message);
  }
}
```

Botón Guardar Como …:

```
  private void b_guardar_como_Click(object sender, EventArgs e)
      {
          CListBox.lb = listBox1;
          CListBox.GuardarComo("c:\\");
      }
```

Botón Limpiar:

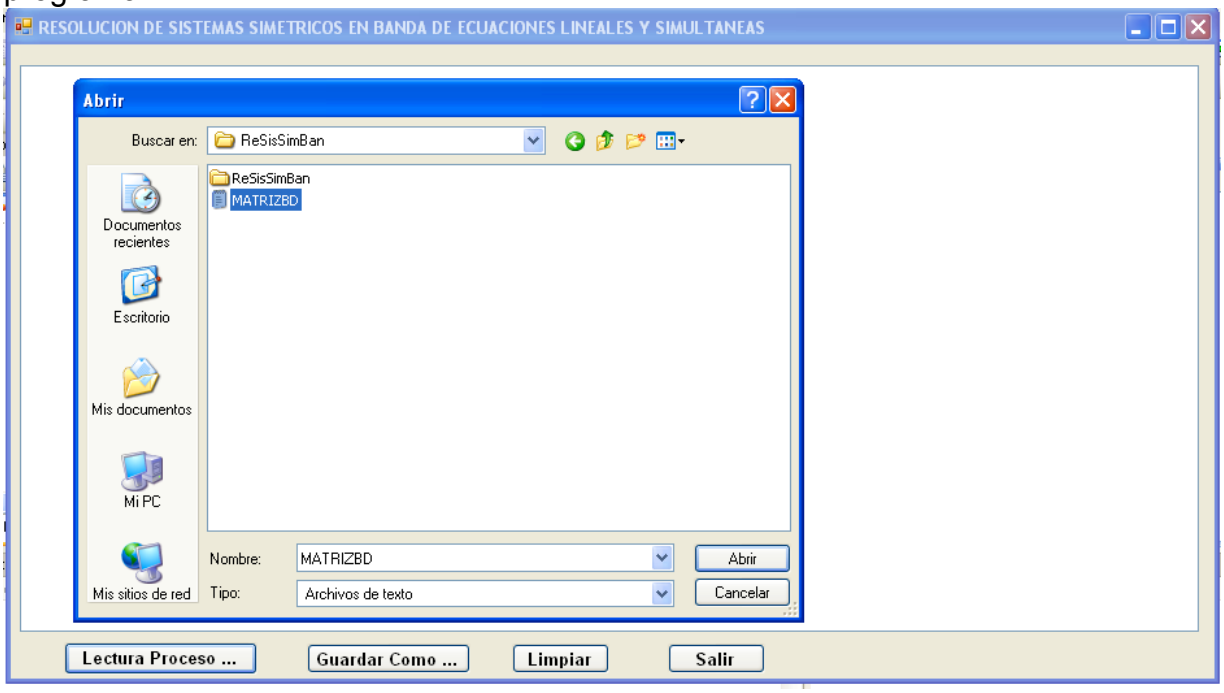
```
private void b_limpiar_Click(object sender, EventArgs e)
    {
        CListBox.lb = listBox1;
        CListBox.Limpiar();
    }
```

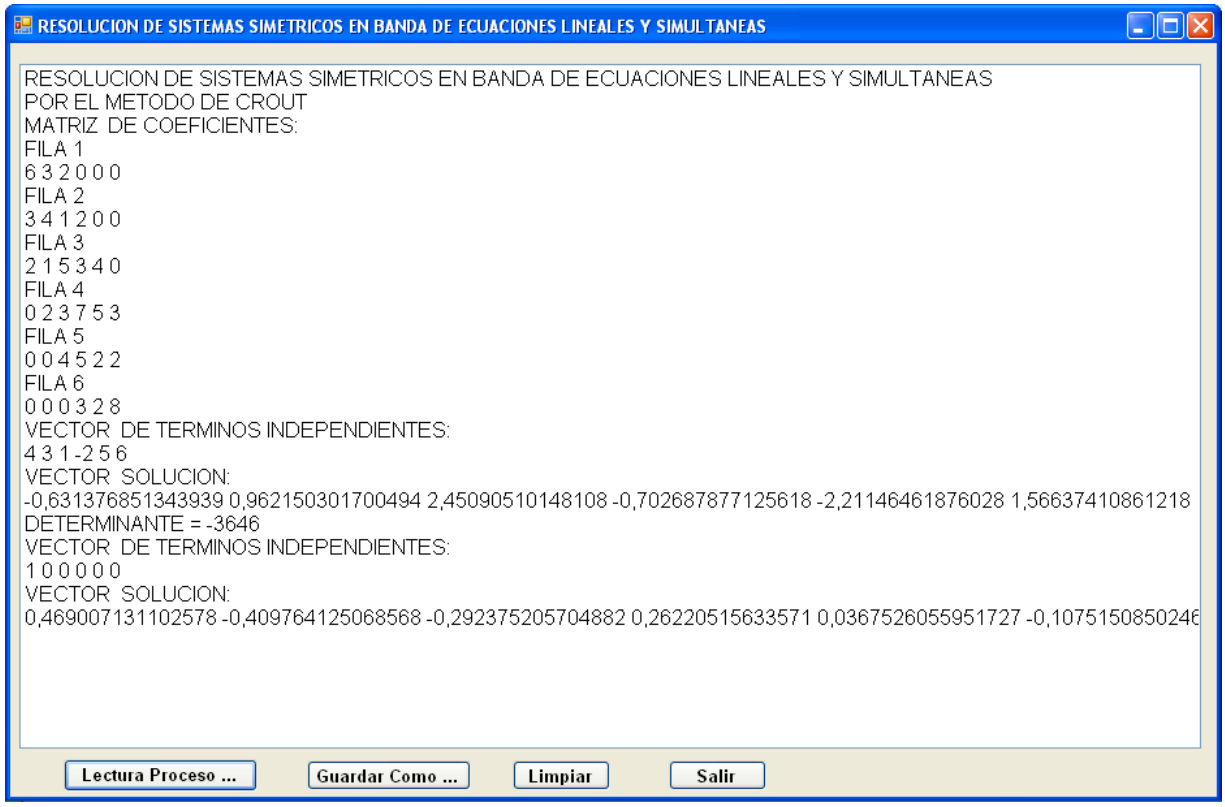
Botón Salir:

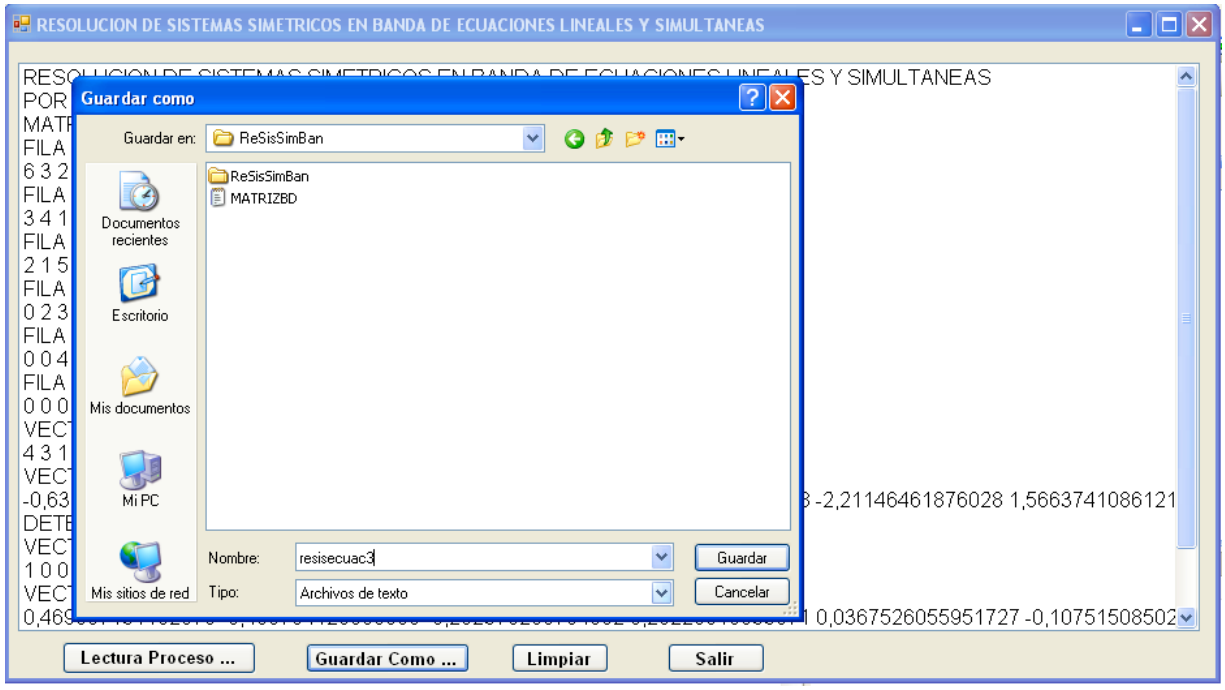
```
private void b_salir_Click(object sender, EventArgs e)
    {
        this.Close();
        this.Dispose();
    }
```

Las referencias a las clases CEntSalArchivos, CResolSisSimBanda, CLectEscrMatrices y CListBox, provienen de la definición de las respectivas clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:




RESOLUCION DE SISTEMAS SIMETRICOS EN BANDA DE ECUACIONES LINEALES Y SIMULTANEAS
RESOLUCION DE SISTEMAS SIMETRICOS EN BANDA DE ECUACIONES LINEALES Y SIMULTANEAS
POR EL METODO DE CROUT
MATRIZ DE COEFICIENTES:
FILA 1
6 3 2 0 0 0
FILA 2
3 4 1 2 0 0
FILA 3
2 1 5 3 4 0
FILA 4
0 2 3 7 5 3
FILA 5
0 0 4 5 2 2
FILA 6
0 0 0 3 2 8
VECTOR DE TERMINOS INDEPENDIENTES:
4 3 1 -2 5 6
VECTOR SOLUCION:
-0,631376851343939 0,962150301700494 2,45090510148108 -0,702687877125618 -2,21146461876028 1,56637410861218
DETERMINANTE = -3646
VECTOR DE TERMINOS INDEPENDIENTES:
1 0 0 0 0 0
VECTOR SOLUCION:
0,469007131102578 -0,409764125068568 -0,292375205704882 0,26220515633571 0,0367526055951727 -0,107515085024
Lectura Proceso ...
Guardar Como ...
Limpiar
Salir


RESOLUCION DE SISTEMAS SIMETRICOS EN BANDA DE ECUACIONES LINEALES Y SIMULTANEAS
Guardar como
Guardar en:
ReSisSimBan
ReSisSimBan
MATRIZBD
Documentos recientes
Escritorio
Mis documentos
Mi PC
Mis sitios de red
Nombre:
resisecuac3
Tipo:
Archivos de texto
Guardar
Cancelar
Lectura Proceso ...
Guardar Como ...
Limpiar
Salir

RESOLUCIÓN DE SISTEMAS SIMÉTRICOS SKY LINE DE ECUACIONES LINEALES Y SIMULTÁNEOS

Un sistema simétrico Skyline de ecuaciones lineales y simultáneas, es un caso particular de un sistema simétrico de ecuaciones lineales y simultáneas, en el que la matriz de coeficientes A, se encuentra formada por valores diferentes de cero en la franja de la diagonal principal y el resto de elementos igual a cero. De esta manera, para la resolución, se toma la mitad de la franja, cuyo ancho de banda es variable. Una alternativa para representar la matriz como un vector es la de mantener en un arreglo las direcciones de los elementos de la diagonal principal de la matriz (ver figura 3.9). Existen varias alternativas para definir los elementos del arreglo de direcciones: a) Contar manualmente y registrarlos a través de una entrada de datos; b) Registrar la matriz triangular inferior, incluyendo los ceros, de esta manera el proceso de lectura, genera automáticamente el vector de direcciones; etc. En este texto, se adopta la alternativa (b).

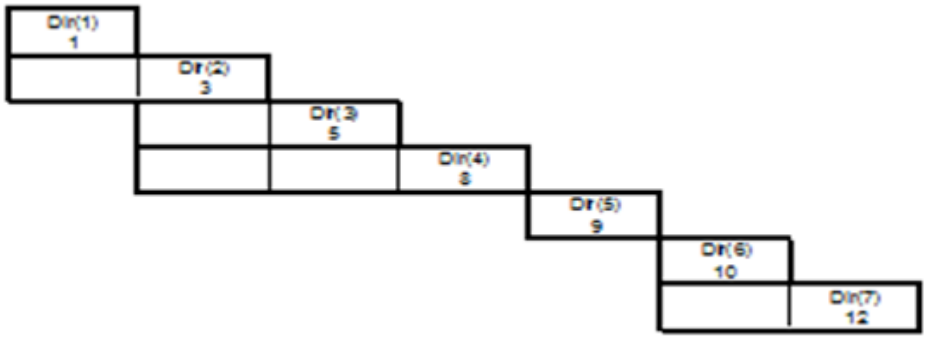


Figura 3.9 Direcciones de los elementos de la diagonal principal en una Matriz Skyline

Las formulas para la resolución de este tipo de sistemas de ecuaciones, sigue siendo las que se emplean para los sistemas simétricos, sin considerar los ceros que se encuentran fuera de la franja variable.

Con el objeto de sistematizar la resolución de este tipo de sistemas. En los procesos de lectura – escritura y recuperación de valores de la matriz de coeficientes y en el método de crout, se emplea la función:

$$insk(i, j, Dir) = Dir(i) + j - i \qquad (3.101)$$

La cual, transforma los índices matriciales (i, j) en un índice lineal insk, convirtiendo de esta manera la matriz de coeficientes en un vector. Se debe tener en cuenta que i > 0 y j > 0. Para valores desde cero pueden producirse índices repetidos.

Ejemplo 3.12:

Dado el sistema:

$$3X1 - 2X2 = -2$$
$$-2X1 - 3X2 + X3 + 2X4 = 1$$
$$X2 - 2X3 + 3X4 = 0$$
$$2X2 + 3X3 - X4 = 3$$
$$2X5 = 5$$
$$-2X6 + 3X7 = 2$$
$$3X6 + X7 = -1$$

Resolverlo por el método de Crout y calcular el determinante de la matriz de coeficientes.

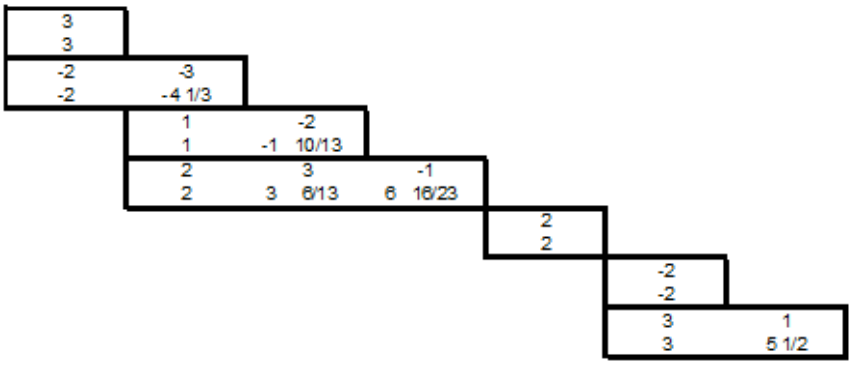
Solución:

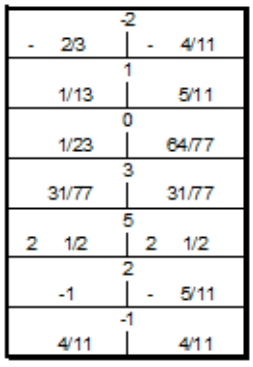
Planteamiento matricial:

$$\begin{pmatrix} 3 & -2 & 0 & 0 & 0 & 0 & 0 \\ -2 & -3 & 1 & 2 & 0 & 0 & 0 \\ 0 & 1 & -2 & 3 & 0 & 0 & 0 \\ 0 & 2 & 3 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 2 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -2 & 3 \\ 0 & 0 & 0 & 0 & 0 & 3 & 1 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} -2 \\ 1 \\ 0 \\ 3 \\ 5 \\ 2 \\ -1 \end{pmatrix}$$

Como se puede observar, es un sistema simétrico Skyline, de 7 ecuaciones, con un ancho de banda variable.

Se plantea el esquema:





Los cálculos típicos, se detallan a continuación:

L22 = -3 - (-2)^2 / 3 = -13/3
L33 = -2 - 1^2 / (-13/3) = -23/13
L43 = 3 - 2*1/(-13/3) = 45/13
L44 = -1 - 2^2 / (-13/3) - (45/13)^2 / (-23/13) = 154/23
L77 = 1 - 3^2 / (-2) = 11/2
c1 = -2/3
c2 = (1 - (-2)*(-2/3))/(-13/3) = 1/13
c3 = (0 - 1*(1/13))/(-23/13) = 1/23
c4 = (3 - 2*(1/13) - (45/13)*(1/23))/(154/23) = 31/77
c5 = 5/2
c6 = 2/(-2) = -1
c7 = (-1 - 3*(-1))/(11/2) = 4/11
x7 = c7 = 4/11
x6 = -1 - 3*(4/11)/(-2) = -5/11
x5 = c5 = 5/2
x4 = c4 = 31/77
x3 = 1/23 - (45/13)*(31/77)/(-23/13) = 64/77
x2 = 1/13 - (1*(64/77) + 2*(31/77))/(-13/3) = 5/11
x1 = -2/3 -(-2)*(5/11)/3 = -4/11
Det = 3*(-13/3)*(-23/13)*(154/23)*2*(-2)*(11/2) = -3388

Implementación de los algoritmos en MATLAB y VISUAL C#:

Para cada lenguaje, se elabora un solo programa, desde el cual se abren archivos para lectura de datos; se ejecuta cada uno de los métodos incluyendo el que resuelve el sistema simétrico Skyline de ecuaciones lineales y simultáneas y se escribe los resultados en un archivo de salida, en el caso de Matlab y se despliegan los resultados en una caja lista en el caso de Visual C#.

Archivos fuente en MATLAB:

```
% SisEcuac4.m
ent = fopen('\\VBOXSVR\compartido\AnalisisNumerico\AnalNumat\matrizsl.dat', 'r');
sal = fopen('\\VBOXSVR\compartido\AnalisisNumerico\AnalNumat\rsisecuac4.dat', 'w');
% LECTURA DEL NUMERO DE VECTORES DE TI A PROCESAR
NS = fscanf(ent, '%d', 1);
% LECTURA DE LA DIMENSION DEL SISTEMA:
N = fscanf(ent, '%d', 1);
%LECTURA DE LA MATRIZ DE COEFICIENTES
[A, Dir] = LeeMSkyLine(ent, N);
fprintf(sal, 'RESOLUCION DE SISTEMAS SIMETRICOS SKYLINE DE ECUACIONES LINEALES Y
SIMULTANEAS:\r\n');
fprintf(sal, 'MATRIZ DE COEFICIENTES:\r\n');
EscribeSkyLine(sal, A, N, Dir);
for i = 1 : NS
  B = LeeVector(ent, N);
  fprintf(sal, 'VECTOR DE TERMINOS INDEPENDIENTES:\r\n');
  EscribeVector(sal, B, N);
  [det, A, B] = CroutSkyLine(A, B, N, Dir, i);
```

```
  if i == 1
    fprintf(sal, 'DETERMINANTE = %f\r\n', det);
  end;
  fprintf(sal, 'VECTOR DE SOLUCIONES:\r\n');
  EscribeVector(sal, B, N);
end;
fclose('all');

% Insk.m
function a = Insk(i, j, Dr)
a = Dr(i) + j - i;
return;

% LeeMSkyLine.m
function [A, Dir] = LeeMSkyLine(f, n)
PrimerCeroDeFila = 0;
Cont = 0;
for i = 1 : n
  for j = 1 : i
    Aux = fscanf(f, '%f', 1);
    if (i == j)
      Cont = Cont + 1;
      A(Cont) = Aux;
      Dir(i)= Cont;
      PrimerCeroDeFila = 1;
    elseif (Aux ~= 0)
      Cont = Cont + 1;
      A(Cont) = Aux;
      PrimerCeroDeFila = 0;
    elseif (PrimerCeroDeFila == 0)
      Cont = Cont + 1;
      A(Cont) = Aux;
    end;
  end;
end;
return;

% EscribeSkyLine.m
function EscribeSkyLine(f, A, n, Dr)
for i = 1 : n
  fprintf(f, 'FILA   %d:\r\n',i);
  for j = 1 : i
    fprintf(f, '%f  ',RecuSkyLine(A, i, j, Dr));
    if (mod(j, 8) == 0) | (j == n)
      fprintf(f, '\r\n');
    end;
  end;
  for j = i + 1 : n
    fprintf(f, '%f  ',RecuSkyLine(A, j, i, Dr));
    if (mod(j, 8) == 0) | (j == n)
      fprintf(f, '\r\n');
    end;
  end;
end;
fprintf(f, '\r\nVECTOR DE DIRECCIONES\r\n');
for i = 1 : n
  fprintf(f, '%d  ', Dr(i));
end;
fprintf(f, '\r\n\r\n');
return
```

```
% RecuSkyLine.m
function Val = RecuSkyLine(X, I, J, Dir)
if (I + J == 2)| (J > I - Dir(I) + Dir(I - 1))
  Val = X(Insk(I, J, Dir));
else
  Val = 0;
end;
return;

% croutskyline.m
function [det, A, B] = CroutSkyLine(A, B, N, Dr, St)
det = 1;
if St == 1 % calcular las matrices L, U y el determinante:
  det = A(Insk(1, 1, Dr));
  for i = 2 : N
    lim = i - Dr(i) + Dr(i - 1) + 1;
    for j = lim + 1 : i
      for k = lim : j - 1
        A(Insk(i, j, Dr)) = A(Insk(i, j, Dr)) - A(Insk(i, k, Dr)) * RecuSkyLine(A, j, k, Dr) / A(Insk(k, k, Dr));
      end;
    end;
    det = det * A(Insk(i,i, Dr));
  end;
end;
% Cálculo del vector C:
for i = 1 : N
  d1 = 0;
  if i > 1 d1 = Dr(i - 1); end;
  for j = i - Dr(i) + d1 + 1 : i - 1
    B(i) = B(i) - A(Insk(i, j, Dr)) * B(j);
  end;
  B(i) = B(i) / A(Insk(i, i, Dr));
end;
% Cálculo del vector X:
for i = N - 1 : -1 : 1
  for j = i + 1 : N
    B(i) = B(i) - RecuSkyLine(A, j, i, Dr) * B(j) / A(Insk(i, i, Dr));
  end;
end;
```

La definición de las funciones LeeVector y EscribeVector, ya se la realizó anteriormente.

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa, como se había indicado anteriormente, la entrada – salida, se realiza sobre archivos, cuyos contenidos son:

matrizsl.dat:
2
7
3
-2
-3
0

1
-2
0
2
3
-1
0
0
0
0
2
0
0
0
0
0
-2
0
0
0
0
0
3
1
-2
1
0
3
5
2
-1
1
0
0
0
0
0
0

rsisecuac4.dat:
RESOLUCION DE SISTEMAS SIMETRICOS SKYLINE DE ECUACIONES LINEALES Y SIMULTANEAS:
MATRIZ DE COEFICIENTES:
FILA 1:
3.000000 -2.000000 0.000000 0.000000 0.000000 0.000000 0.000000
FILA 2:

-2.000000 -3.000000 1.000000 2.000000 0.000000 0.000000 0.000000
FILA 3:
0.000000 1.000000 -2.000000 3.000000 0.000000 0.000000 0.000000
FILA 4:
0.000000 2.000000 3.000000 -1.000000 0.000000 0.000000 0.000000
FILA 5:
0.000000 0.000000 0.000000 0.000000 2.000000 0.000000 0.000000
FILA 6:
0.000000 0.000000 0.000000 0.000000 0.000000 -2.000000 3.000000
FILA 7:
0.000000 0.000000 0.000000 0.000000 0.000000 3.000000 1.000000

VECTOR DE DIRECCIONES
1 3 5 8 9 10 12

VECTOR DE TERMINOS INDEPENDIENTES:
-2.000000 1.000000 0.000000 3.000000 5.000000 2.000000 -1.000000
DETERMINANTE = -3388.000000
VECTOR DE SOLUCIONES:
-0.363636 0.454545 0.831169 0.402597 2.500000 -0.454545 0.363636
VECTOR DE TERMINOS INDEPENDIENTES:
1.000000 0.000000 0.000000 0.000000 0.000000 0.000000 0.000000
VECTOR DE SOLUCIONES:
0.272727 -0.090909 0.090909 0.090909 0.000000 0.000000 0.000000

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 3.10 Como se puede apreciar, existen un cuadro de lista, el cual permitirá presentar los resultados; y cuatro botones,

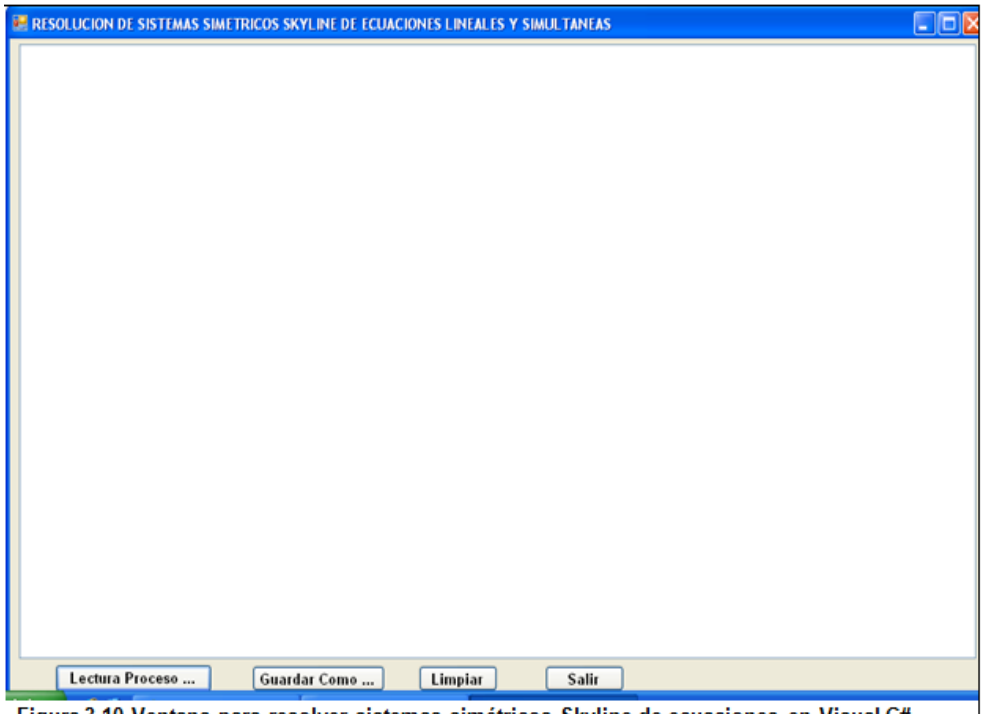


Figura 3.10 Ventana para resolver sistemas simétricos Skyline de ecuaciones en Visual C#

mismos que tienen la funcionalidad:

- Lectura Proceso…: abre una ventana para registrar el nombre del archivo el cual permite leer los datos del sistema simétrico en banda de ecuaciones lineales y simultáneas desde el medio de almacenamiento permanente; ejecuta los métodos para calcular la matriz inversa y despliega los resultados.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Botón Lectura Proceso…:

```
private void b_lectura_Click(object sender, EventArgs e)
{
    try
    {
        int i, ns;
       // Abrir el archivo de entrada a traves del cuadro de dialogo:
        CEntSalArchivos objes = new CEntSalArchivos("c:\\");
        objes.AbrirArchivoEnt();
        // Leer el número de procesamiento de soluciones:
        ns = Int16.Parse(objes.LeerLinea());
        // Leer el número de ecuaciones o de incógnitas:
        CResolSisSimSkyline.n = Int16.Parse(objes.LeerLinea());
        if (CResolSisSimSkyline.n > 0)
        {
            double det;
            int n = CResolSisSimSkyline.n;
           // Definir el arreglo de términos independientes:
           CResolSisSimSkyline.B = new double[n];
           // Definir la matriz de coeficientes:
           CResolSisSimSkyline.A = new double[n * n / 3];
           // Definir el vector de direcciones de la matriz de coeficientes:
           CLectEscrMatrices.Dir = new int[n + 1];
          /* Leer la matriz de coeficientes, el arreglo de términos independientes y
                * generar los resultados en el listbox: */
          // Prueba del método de Crout:
          CLectEscrMatrices.n = n;
          CLectEscrMatrices.c = CResolSisSimSkyline.A;
          CLectEscrMatrices.b = CResolSisSimSkyline.B;
          CLectEscrMatrices.obent = objes;
          CLectEscrMatrices.lb = listBox1;
          CLectEscrMatrices.LeeMatrizSimSkyline();
```

```
            listBox1.Items.Add("RESOLUCION DE SISTEMAS SIMETRICOS SKYLINE
DE ECUACIONES LINEALES Y SIMULTANEAS");
            listBox1.Items.Add("POR EL METODO DE CROUT");
             CLectEscrMatrices.EscribeMatrizSimSkyline(" DE COEFICIENTES:");
             CResolSisSimSkyline.Dir = CLectEscrMatrices.Dir;
             for (i = 0; i < ns; i++)
             {
                  CLectEscrMatrices.LeeVector();
                  CLectEscrMatrices.EscribeVector("              DE              TERMINOS
INDEPENDIENTES:");
                 det = CResolSisSimSkyline.CroutSimSkyline(i == 0);
                 CLectEscrMatrices.EscribeVector(" SOLUCION:");
                 if (i == 0) listBox1.Items.Add("DETERMINANTE = " + det.ToString());
             }
             // Cerrar flujo de entrada:
             objes.CerrarLectura();
              listBox1.Items.Add("");
              listBox1.Items.Add("");
              listBox1.Items.Add("");
        }
    }
    catch (Exception ex)
    {
         MessageBox.Show(ex.Message);
     }
}
```

Botón Guardar Como …:

```
  private void b_guardar_como_Click(object sender, EventArgs e)
      {
          CListBox.lb = listBox1;
          CListBox.GuardarComo("c:\\");
      }
```

Botón Limpiar:

```
  private void b_limpiar_Click(object sender, EventArgs e)
      {
          CListBox.lb = listBox1;
          CListBox.Limpiar();
      }
```

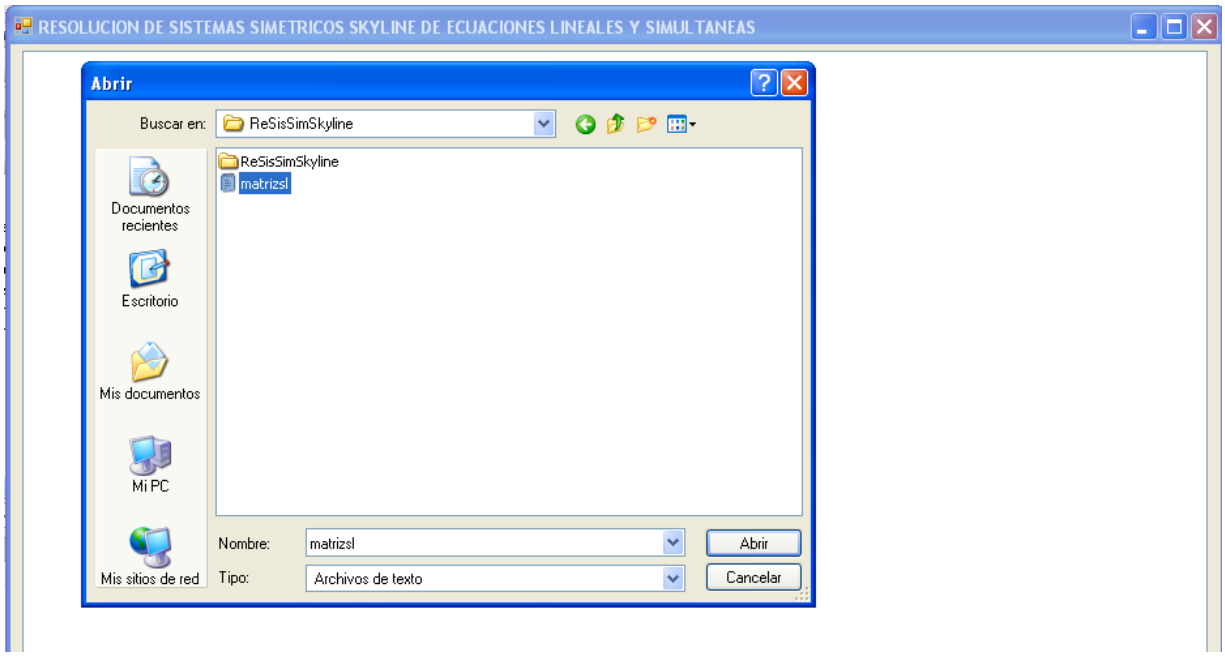
Botón Salir:

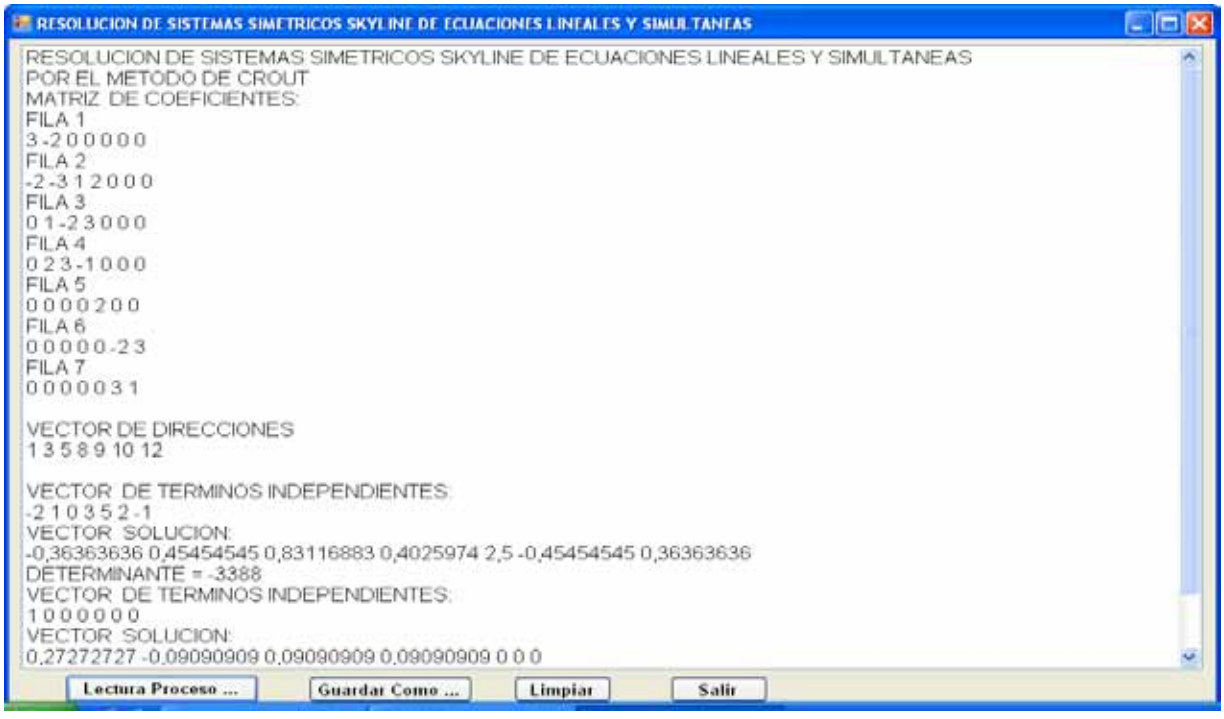
```
   private void b_salir_Click(object sender, EventArgs e)
      {
          this.Close();
          this.Dispose();
```

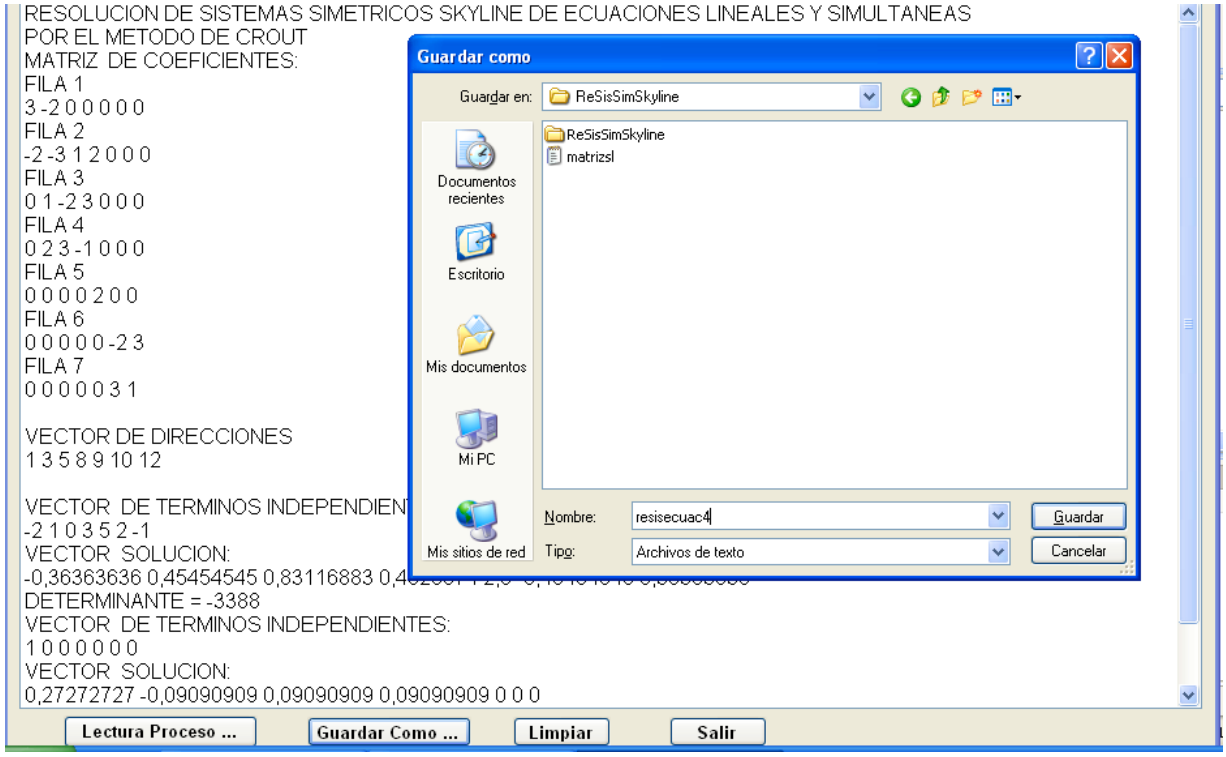
}

Las referencias a las clases CEntSalArchivos, CResolSisSimSkyline, CLectEscrMatrices y CListBox, provienen de la definición de las respectivas clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:






RESOLUCION DE SISTEMAS SIMETRICOS SKYLINE DE ECUACIONES LINEALES Y SIMULTANEAS
POR EL METODO DE CROUT
MATRIZ DE COEFICIENTES:
FILA 1
3 -2 0 0 0 0 0
FILA 2
-2 -3 1 2 0 0 0
FILA 3
0 1 -2 3 0 0 0
FILA 4
0 2 3 -1 0 0 0
FILA 5
0 0 0 0 2 0 0
FILA 6
0 0 0 0 0 -2 3
FILA 7
0 0 0 0 0 3 1
VECTOR DE DIRECCIONES
1 3 5 8 9 10 12
VECTOR DE TERMINOS INDEPENDIEN
-2 1 0 3 5 2 -1
VECTOR SOLUCION:
DETERMINANTE = -3388
VECTOR DE TERMINOS INDEPENDIENTES:
1 0 0 0 0 0 0
VECTOR SOLUCION:
0,27272727 -0,09090909 0,09090909 0,09090909 0 0 0
Lectura Proceso ...
Guardar Como ...
Limpiar
Salir
Guardar como
Guardar en:
ReSisSimSkyline
ReSisSimSkyline
matrizsl
Documentos recientes
Escritorio
Mis documentos
Mi PC
Mis sitios de red
Nombre:
resisecuac4
Tipo:
Archivos de texto
Guardar
Cancelar

EJERCICIOS DEL CAPITULO III

1.- Dado el sistema:

$-5x_1 + 2x_2 - 3x_3 - x_4 = -1$
$-3x_1 - 2x_2 - 5x_3 + 3x_4 = 2$
$2x_1 + 4x_2 - 2x_3 + 3x_4 = 1$
$-x_1 + 3x_2 + 3x_3 + 4x_4 = -3$

Resolverlo por el método de Gauss y calcular su determinante. Expresar las operaciones y resultados en números racionales.

2.- Dado el sistema del problema del numeral 1

Resolverlo por el método de Gauss - Jordan y calcular su determinante. Expresar las operaciones y resultados en números racionales.

3.- Dado el sistema del problema del numeral 1

Resolverlo por el método de Gauss – Seidel con una precisón de 1e-2.

4.- Dado el sistema del problema del numeral 1

Resolverlo por el método de Crout y calcular su determinante. Expresar las operaciones y resultados en números racionales.

5.- Ejecutar los programas en Matlab y Visual C#.NET para la resolución de sistemas de ecuaciones lineales y simultáneas, ingresando los datos del sistema del numeral 1.

6.- Dada la matriz de coeficientes del sistema del numeral 1. Calcular la matriz inversa por el método de Crout. Expresar las operaciones y resultados en números racionales.

7.- Ejecutar los programas en Matlab y Visual C#.NET para calcular la matriz inversa por el método de Crout, ingresando los datos del sistema del numeral 6.

8.- Dado el sistema del problema del numeral 1

Resolverlo por el método de Crout Simétrico y calcular su determinante. Expresar las operaciones y resultados en números racionales. **Sugerencia**: Intercambiar las filas 2 y 3 de ese sistema para poder obtener un sistema simétrico.

9.- Ejecutar los programas en Matlab y Visual C#.NET para la resolución de sistemas simétricos de ecuaciones lineales y simultáneas, ingresando los datos del sistema del numeral 8.

10.- Dado el sistema:

$$\begin{pmatrix} 4 & -3 & 0 & 0 & 0 & 0 & 0 \\ -3 & 2 & -1 & 0 & 0 & 0 & 0 \\ 0 & -1 & 3 & 4 & 0 & 0 & 0 \\ 0 & 0 & 4 & 5 & -2 & 0 & 0 \\ 0 & 0 & 0 & -2 & -4 & 2 & 0 \\ 0 & 0 & 0 & 0 & 2 & -3 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} 2 \\ -3 \\ -1 \\ 4 \\ 1 \\ -2 \\ 3 \end{pmatrix}$$

Resolverlo por el método de Crout - Banda y calcular su determinante. Expresar las operaciones y resultados en números racionales.

11.- Ejecutar los programas en Matlab y Visual C#.NET para la resolución de sistemas simétricos en banda de ecuaciones lineales y simultáneas, ingresando los datos del sistema del numeral 10.

12.- Dado el sistema:

$$\begin{pmatrix} 4 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & -2 & 1 & 2 & 0 & 0 & 0 \\ 0 & 1 & -3 & 4 & 0 & 0 & 0 \\ 0 & 2 & 4 & -1 & 3 & 0 & 0 \\ 0 & 0 & 0 & 3 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -2 & 3 \\ 0 & 0 & 0 & 0 & 0 & 3 & 2 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \end{pmatrix} = \begin{pmatrix} -1 \\ 2 \\ 3 \\ 1 \\ 4 \\ -2 \\ 1 \end{pmatrix}$$

Resolverlo por el método de Crout - Skyline y calcular su determinante. Expresar las operaciones y resultados en números racionales.

13.- Ejecutar los programas en Matlab y Visual C#.NET para la resolución de sistemas simétricos Skyline de ecuaciones lineales y simultáneas, ingresando los datos del sistema del numeral 12.

14.- Desarrollar un programa en Matlab y otro en Visual C# para implementar el método de Kramer para resolución de sistemas de ecuaciones lineales y simultáneas.

4

INTERPOLACIÓN

Generalidades

La interpolación permite generar nuevos puntos a partir de un conjuto finito de puntos conocidos.

Existen varios métodos para interpolar, los que se tratarán en este capítulo son:

- Interpolación Lineal.
- Interpolación de Lagrange.
- Interpolación de Newton.

INTERPOLACION LINEAL:

Consiste en generar el nuevo punto, trazando una secante (segmento AC), sobre la curva y = f(x) (ver figura 4.1). Para deducir la fórmula de interpolación, se establece la semejanza entre los triángulos ABC y ADE, de manera que:

$$\frac{\overline{DE}}{\overline{AD}} = \frac{\overline{BC}}{\overline{AB}} \therefore \frac{y - y1}{x - x1} = \frac{y2 - y1}{x2 - x1} (4.1)$$

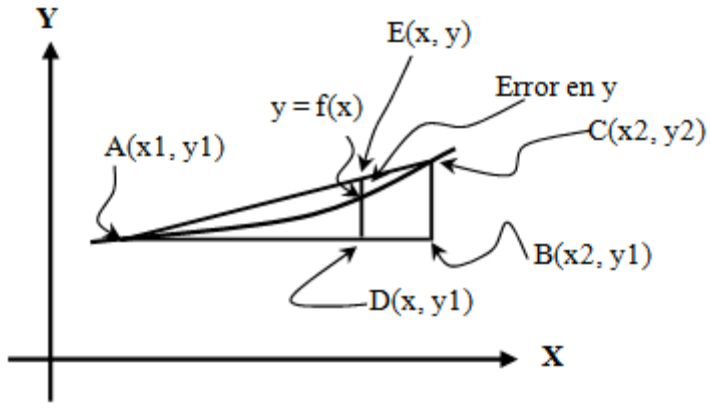


Figura 4.1 Interpolación Lineal

Despejando de (4.1), el valor de y:

$$y = \frac{y2 - y1}{x2 - x1}(x - x1) + y1.; (4.2)$$

La fórmula 4.2 permite calcular la ordenada *y*; como entradas se tiene la absisa *x* y los puntos A(x1, y1) y C(x2, y2); mismos que son los puntos de intersección de la curva real y = f(x) y la recta AC; x1 < x2. En la figura 4.1 puede apreciarse el error que se comete al calcular la ordenada de la curva real y de la recta.

Ejemplo 4.1

Dados los puntos A(1; -1) y C(3; 15) tomados de la curva real $y = 2x^2 – 3$; calcular la ordenada *y*; si x = 2.8 además del error absoluto y relativo.

Solución:
Aplicando la fórmula 4.2:

$$y = \frac{y2 - y1}{x2 - x1}(x - x1) + y1 = \frac{15 - (-1)}{3 - 1}(2.8 - 1)) + (-1) = 13.4$$

Y = f(2.8) = $2(2.8)^2$ - 3 = 12.68
EAy = |12.68 - 13.4| = 0.72; δy = 0.72 / 12.68 = 0.05678 = 5.68 %

Ejemplo 4.2
Desarrollar una aplicación Matlab y otra en Visual C#, que tenga una función para calcular la ordenada y por el método de interpolación lineal; dicha función debe recibir las coordenadas de dos puntos de la curva real y el valor de la abscisa *x*. la aplicación debe calcular también los errores absoluto y relativo. Considerar los datos del ejemplo 4.1.
Solución:

Programas en MATLAB y VISUAL C#:

Para cada lenguaje, se elabora un solo programa, desde el cual se ingresan las abscisas de los puntos de la curva real y la abscisa a interpolar. El programa debe desplegar las ordenadas interpolada y exacta y los errores absoluto y relativo.

Archivos fuente en MATLAB:

```
% PrbInterpolacionLineal.m
% Prueba interpolación lineal y calcula errores.
x(1) = input('Ingrese la abscisa inicial: ');
while 1
  x(2) = input('Ingrese la abscisa final: ');
  if x(2) > x(1) break; end;
end;
y(1) = fit(x(1));
y(2) = fit(x(2));
disp(sprintf('y1: %f', y(1)));
disp(sprintf('y2: %f', y(2)));
while 1
   xit = input(sprintf('Ingrese la abscisa a interpolar (salir < %f o > %f): ', x(1), x(2)));
   % Validación:
```

```
  if (xit < x(1) | xit > x(2)) break; end;
  % Cálculo de la ordenada interpolada:
  yit = InterpolacionLineal(x, y, xit);
  % Cálculo de Y:
  Y = fit(xit);
  % Calcular los errores:
  erAbs = abs(Y - yit);
  erRel = erAbs / abs(Y);
  % Desplegar resultados:
  disp(sprintf('Ordenada (exacta): %f', Y));
  disp(sprintf('Ordenada (interpolada linealmente): %f', yit));
  disp(sprintf('Error absoluto: %f', erAbs));
  disp(sprintf('Error relativo: %f\n', erRel));
end;
% InterpolacionLineal.m
function y = InterpolacionLineal(x, y, xit)
  y = (y(2) - y(1)) / (x(2) - x(1)) * (xit - x(1)) + y(1);
return

% fit.m
function y = fit(x)
y = 2 * x ^2 - 3;
return
```

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa:

```
Command Window

  To get started, select MATLAB Help or Demos from the Help menu.

Ingrese la abscisa inicial: 1
Ingrese la abscisa final: 3
y1: -1.000000
y2: 15.000000
Ingrese la abscisa a interpolar (salir < 1.000000 o > 3.000000): 2.8
Ordenada (exacta): 12.680000
Ordenada (interpolada linealmente): 13.400000
Error absoluto: 0.720000
Error relativo: 0.056782

Ingrese la abscisa a interpolar (salir < 1.000000 o > 3.000000): 2
Ordenada (exacta): 5.000000
Ordenada (interpolada linealmente): 7.000000
Error absoluto: 2.000000
Error relativo: 0.400000

Ingrese la abscisa a interpolar (salir < 1.000000 o > 3.000000): 0
>>
```

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 4.2.

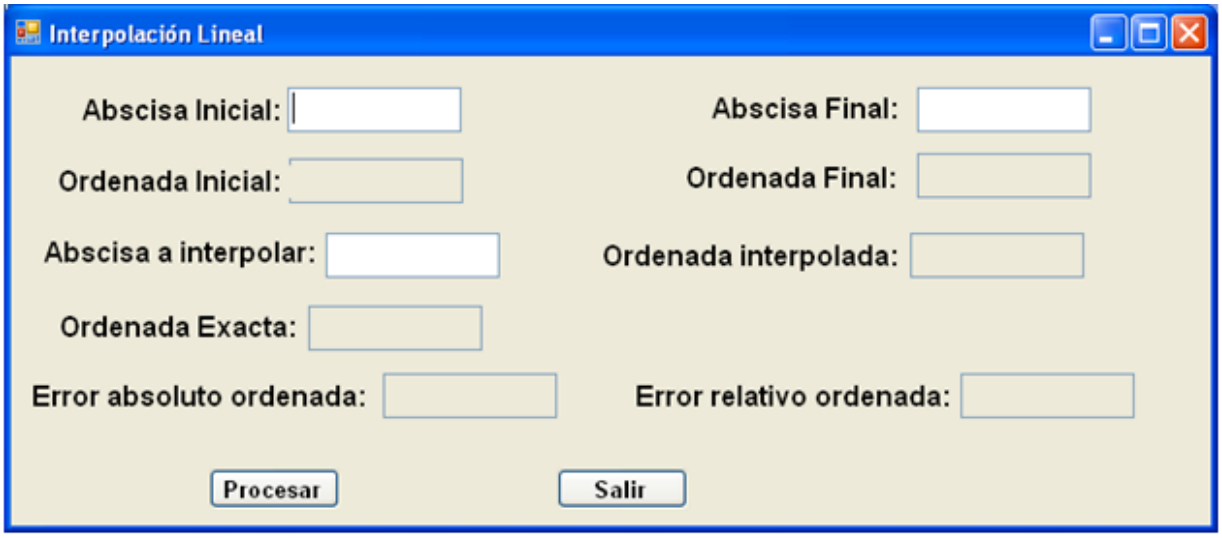


Figura 4.2 Ventana para Interpolación Lineal en Visual C#

Como se puede apreciar, existen nueve cuadros de edición, mismos que permitirán ingresar los datos; y presentar los resultados; y dos botones, mismos que tienen la funcionalidad:

- Procesar: Ejecuta el método para interpolar linealmente y desplegar los resultados. Y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Función Externa:

```
// Funcion a ser analizada:
static double fit(double x)
{
      return 2 * Math.Pow(x, 2) - 3;
}
```

Botón Procesar:

```
private void b_procesar_Click(object sender, EventArgs e)
{
  // Prueba interpolación lineal y calcula errores.
  // Declaracion de variables
  double x, y, Y, erAbs, erRel;
  try
  {
    // Definición del vector de abscisas:
```

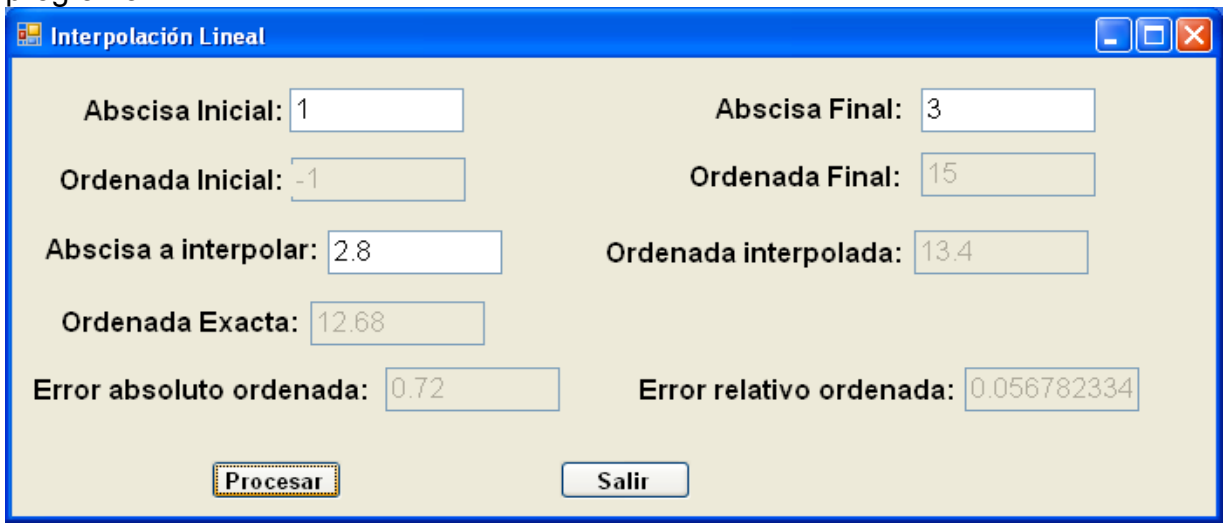
```
    CInterpolacion.vx = new double[2];
    // Transferencia de captura de datos a variables:
    CInterpolacion.vx[0] = double.Parse(txtAbscIni.Text);
    CInterpolacion.vx[1] = double.Parse(txtAbscFin.Text);
    // Validación:
    if (CInterpolacion.vx[1] < CInterpolacion.vx[0]) return;
      x = double.Parse(txtAbscInt.Text);
      // Validación:
      if (x < CInterpolacion.vx[0] || x > CInterpolacion.vx[1]) return;
      // Definición del vector de ordenadas:
      CInterpolacion.vy = new double[2];
      CInterpolacion.vy[0] = fit(CInterpolacion.vx[0]);
      CInterpolacion.vy[1] = fit(CInterpolacion.vx[1]);
      // Cálculo de la ordenada interpolada:
      y = CInterpolacion.InterpolacionLineal(x);
      // Cálculo de la ordenada exacta:
      Y = fit(x);
      // Calcular los errores:
      erAbs = Math.Abs(Y - y);
      erRel = erAbs / Math.Abs(Y);
      // Transferir variables a pantalla:
      txtOrdIni.Text = Math.Round(CInterpolacion.vy[0], 9).ToString();
      txtOrdFin.Text = Math.Round(CInterpolacion.vy[1], 9).ToString();
      txtOrdInt.Text = Math.Round(y, 9).ToString();
      txtOrdExacta.Text = Math.Round(Y, 9).ToString();
      txtErAbs.Text = Math.Round(erAbs, 9).ToString();
      txtErRel.Text = Math.Round(erRel, 9).ToString();
  }
  catch (Exception ex)
  {
     MessageBox.Show(ex.Message);
  }
}
```

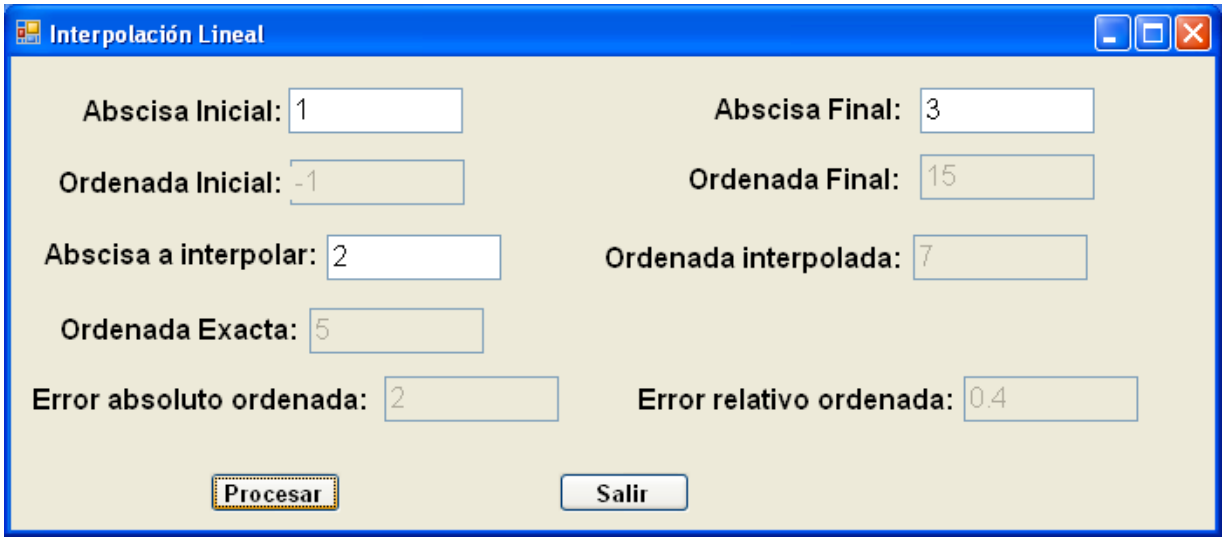
Botón Salir:

```
private void cb_salir_Click(object sender, EventArgs e)
{
    this.Close();
    this.Dispose();
}
```

Las referencias a la clase CInterpolacion, provienen de la definición de la respectiva clase incorporada en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:





INTERPOLACION DE LAGRANGE:

Consiste en generar el nuevo punto, a partir de un grupo de puntos $(x_0; y_0)$, $(x_1; y_1)$, $(x_2; y_2)$, … $(x_n; y_n)$ para lo cual se emplea la fórmula de los polinomios de Lagrange:

$$L(x,n) = \sum_{j=0}^{n} p_j(x) y_j ;4.3$$

Donde L(x, n) es el polinomio polinomio de Lagrange de grado $p_j = \prod_{i=0, i\neq j}^{n} \frac{x - x_i}{x_j - x_i} ;4.4$ interpolador; p_j es el n; n es el grado del

polinomio de interpolación.

$$p_j(x_k) = \begin{cases} 1; j = k \\ 0; j \neq k \end{cases} ;4.5$$

La fórmula 4.5 garantiza que exista un solo polinomio interpolador y que todos los puntos dados pasen por la curva real y = f(x).

Ejemplo 4.3.
Obtener el polinomio de intepolación del problema 4.1 aplicando las fórmulas de interpolación mediante los polinomios de Lagrange, dados los puntos:

a) A(1, -1) y C(3, 15)
b) A(1, -1), B(2, 5) y C(3, 15)

Solución:

a) Dado que se tienen dos puntos, n = 1

$$L(x,1) = \sum_{j=0}^{1} p_j(x) y_j = p_0(x)y_0 + p_1(x)y_1 = \frac{x - x_1}{x_0 - x_1} y_0 + \frac{x - x_0}{x_1 - x_0} y_1$$

$$= \frac{x-3}{-2}(-1) + \frac{x-1}{2}(15) = 8x - 9$$

b) Dado que se tienen tres puntos; n = 2.

$$L(x,2) = \sum_{j=0}^{2} p_j(x) y_j = p_0(x)y_0 + p_1(x)y_1 + p_2(x)y_2$$

$$= \frac{x - x_1}{x_0 - x_1} \cdot \frac{x - x_2}{x_0 - x_2} y_0 + \frac{x - x_0}{x_1 - x_0} \cdot \frac{x - x_2}{x_1 - x_2} y_1 + \frac{x - x_0}{x_2 - x_0} \cdot \frac{x - x_1}{x_2 - x_1} y_2$$

$$= \frac{x-2}{-1} \cdot \frac{x-3}{-2}(-1) + \frac{x-1}{1} \cdot \frac{x-3}{-1}(5) + \frac{x-1}{2} \cdot \frac{x-2}{1}(15) = 2x^2 - 3$$

Como se puede observar: para el ejercicio (a), el polinomio interpolador corresponde al que se puede generar por interpolación lineal. Para el ejercicio (b), el polinomio interpolador corresponde a la función del ejercicio 4.1.

Ejemplo 4.4
Calcular L(π/2, 4) para y = f(x) = e^xcos(2x), en el dominio -2π ≤ x ≤ 2π. Tomar cinco abscisas equidistantes. Calcular además el error absoluto y el relativo. Usando la interpolación mediante los polinomios de Lagrange.

Solución
El proceso se desarrolla en la siguiente tabla:

x = 1.57079633

Punto	X	y = f(X)	Fracciones 0	Fracciones1	Fracciones2	Fracciones 3	Fracciones4
0	-6.2831853	0.0018674	-1.5	2.5	1.25	0.83333333	0.625
1	-3.1415926	0.0432139	-0.25	-0.5	1.5	0.75	0.5
2	0	1	0.16666667	0.25	0.5	0.5	0.25
3	3.1415927	23.140693	0.375	0.5	0.75	1.5	-0.5
4	6.2831853	535.49166					
		$p_J(x)$ =	0.0234375	-0.15625	0.703125	0.46875	-0.0390625
		$y_j p_J(x)$ =	4.3768E-05	-0.0067522	0.703125	10.8471997	-20.917643

L(π/2) = -9.3740273 y exacta = -4.8104774

Error Abso= 4.5635492 Error relativo = 0.94866866 = 94.87 %

$Fracciones_i$, para i = 0, 1, 2, 3, 4; corresponden a las fracciones cuyos productos generan los $p_j(x)$. La suma de los $y_j p_j(x)$ produce L(π/2, 4).

Ejemplo 4.5
Desarrollar una aplicación Matlab y otra en Visual C#, que tenga una función para calcular la ordenada y por el método de interpolación mediante los polinomios de Lagrange; dicha función debe recibir las coordenadas de dos puntos (extremos) de la curva real; el valor de la abscisa *x* y el número de puntos; la aplicación debe calcular también los puntos intermedios y los errores absoluto y relativo. Considerar los datos del ejemplo 4.1.
Solución:

Programas en MATLAB y VISUAL C#:

Para cada lenguaje, se elabora un solo programa, desde el cual se ingresan las abscisas de los puntos extremos de la curva real y la abscisa a interpolar y el número de puntos. El programa debe calcular los puntos intermedios y desplegar las ordenadas interpolada y exacta y los errores absoluto y relativo.

Archivos fuente en MATLAB:

```
% PrbInterpolacionLagrange.m
% Prueba interpolación de Lagrange y calcula errores.
x(1) = input('Ingrese la abscisa inicial: ');
while 1
  xn = input('Ingrese la abscisa final: ');
  if xn > x(1) break; end;
end;
y(1) = fit(x(1));
yn = fit(xn);
disp(sprintf('Ordenada inicial: %f', y(1)));
disp(sprintf('Ordenada final: %f', yn));
while 1
   n = input('Ingrese el número de puntos (Salir < 2): ');
   if n < 2 break; end;
   xit = input(sprintf('Ingrese la abscisa a interpolar (salir < %f o > %f): ', x(1), xn));
   % Validación:
   if (xit < x(1) | xit > xn) break; end;
   % Cálculo del resto de puntos:
   if n > 2
      h = (xn - x(1)) / (n - 1);
      for i = 2 : n - 1
         x(i) = x(i-1) + h;
         y(i) = fit(x(i));
      end
   end;
   x(n) = xn;
   y(n) = yn;
   % Cálculo de la ordenada interpolada:
   yit = InterpolacionLagrange(x, y, xit, n);
   % Cálculo de Y:
   Y = fit(xit);
   % Calcular los errores:
   erAbs = abs(Y - yit);
   erRel = erAbs / abs(Y);
   % Desplegar resultados:
   disp(sprintf('Ordenada (exacta): %f', Y));
   disp(sprintf('Ordenada (interpolada con polinomios de Lagrange): %f', yit));
   disp(sprintf('Error absoluto: %f', erAbs));
   disp(sprintf('Error relativo: %f\n', erRel));
end;

% InterpolacionLagrange.m
function y = InterpolacionLagrange(vx, vy, x, n)
  y = 0;
```

```
  for j = 1 : n
    L = 1;
    for i = 1 : n
        if i ~= j
            L = L * (x - vx(i)) / (vx(j) - vx(i));
        end;
    end;
    y = y + vy(j) * L;
  end;
return

% fit.m
function y = fit(x)
y = 2 * x ^2 - 3;
return
```

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa:

```
Command Window

  To get started, select MATLAB Help or Demos from the Help menu.

Ingrese la abscisa inicial: 1
Ingrese la abscisa final: 3
Ordenada inicial: -1.000000
Ordenada final: 15.000000
Ingrese el número de puntos (Salir < 2): 2
Ingrese la abscisa a interpolar (salir < 1.000000 o > 3.000000): 2.8
Ordenada (exacta): 12.680000
Ordenada (interpolada con polinomios de Lagrange): 13.400000
Error absoluto: 0.720000
Error relativo: 0.056782

Ingrese el número de puntos (Salir < 2): 3
Ingrese la abscisa a interpolar (salir < 1.000000 o > 3.000000): 2.8
Ordenada (exacta): 12.680000
Ordenada (interpolada con polinomios de Lagrange): 12.680000
Error absoluto: 0.000000
Error relativo: 0.000000

Ingrese el número de puntos (Salir < 2): 1
>>
```

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 4.3

Figura 4.3 Ventana para Interpolación mediante Polinomios de Lagrange en Visual C#

Como se puede apreciar, existen diez cuadros de edición, mismos que permitirán ingresar los datos; y presentar los resultados; y dos botones, mismos que tienen la funcionalidad:

- Procesar: Ejecuta el método para interpolar mediante Polinomios de Lagrange y desplegar los resultados. Y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Función Externa:

```
// Funcion a ser analizada:
static double fit(double x)
{
      return 2 * Math.Pow(x, 2) - 3;
}
```

Evento que carga el formulario:

```
private void Form1_Load(object sender, EventArgs e)
{
     CInterpolacion.AsignaMemoria(false, 50);
}
```

Botón Procesar:

```
private void b_procesar_Click(object sender, EventArgs e)
{
    // Prueba interpolación mediante polinomios de Lagrange y calcula errores.
    // Declaracion de variables
    double x, y, Y, h, erAbs, erRel;
     int n;  // Número de puntos.
     try
     {
         // Transferencia de captura de datos a variables:
         n = int.Parse(txtNPuntos.Text);
         // Validación:
         if (n < 2) return;
         // Transferencia de captura de datos a variables:
         CInterpolacion.vx[0] = double.Parse(txtAbscIni.Text);
         CInterpolacion.vx[n - 1] = double.Parse(txtAbscFin.Text);
         // Validación:
         if (CInterpolacion.vx[n - 1] < CInterpolacion.vx[0]) return;
         x = double.Parse(txtAbscInt.Text);
         // Validación:
         if (x < CInterpolacion.vx[0] || x > CInterpolacion.vx[n - 1]) return;
         // Definición del vector de ordenadas:
         CInterpolacion.vy[0] = fit(CInterpolacion.vx[0]);
         CInterpolacion.vy[n - 1] = fit(CInterpolacion.vx[n - 1]);
         // Cálculo del resto de puntos:
         if (n > 2)
         {
             h = (CInterpolacion.vx[n - 1] - CInterpolacion.vx[0]) / (n - 1);
             for (int i = 1; i < n - 1; i++)
             {
                 CInterpolacion.vx[i] = CInterpolacion.vx[i - 1] + h;
                 CInterpolacion.vy[i] = fit(CInterpolacion.vx[i]);
              }
         }
         // Cálculo de la ordenada interpolada:
         y = CInterpolacion.InterpolacionLagrange(x, n);
         // Cálculo de la ordenada exacta:
         Y = fit(x);
         // Calcular los errores:
         erAbs = Math.Abs(Y - y);
         erRel = erAbs / Math.Abs(Y);
         // Transferir variables a pantalla:
         txtOrdIni.Text = Math.Round(CInterpolacion.vy[0], 9).ToString();
         txtOrdFin.Text = Math.Round(CInterpolacion.vy[n - 1], 9).ToString();
         txtOrdInt.Text = Math.Round(y, 9).ToString();
         txtOrdExacta.Text = Math.Round(Y, 9).ToString();
         txtErAbs.Text = Math.Round(erAbs, 9).ToString();
```

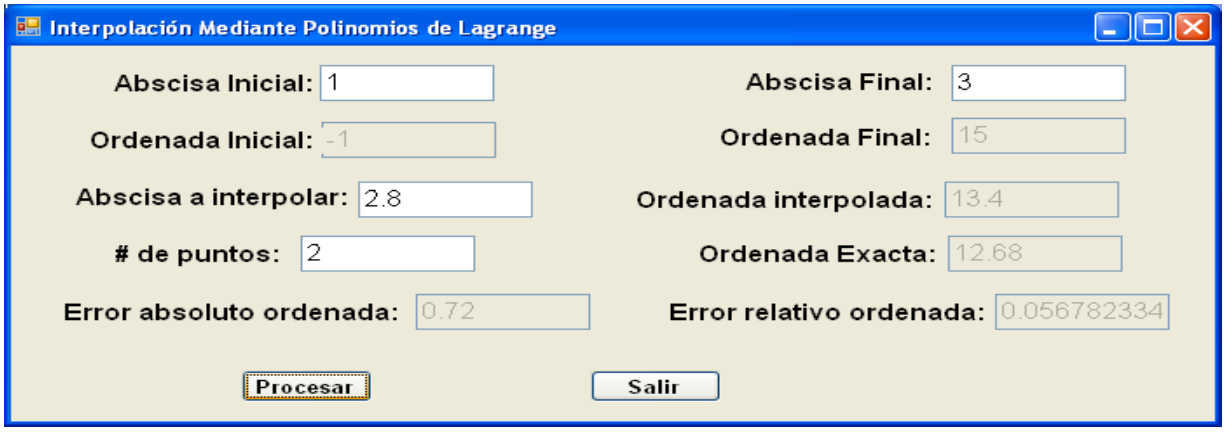
```
            txtErRel.Text = Math.Round(erRel, 9).ToString();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
}
```

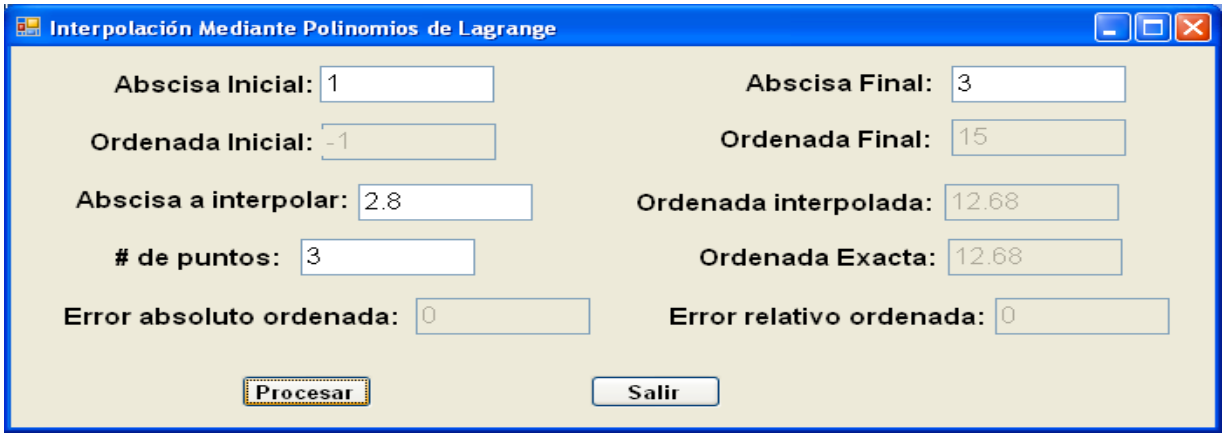
Botón Salir:

```
private void cb_salir_Click(object sender, EventArgs e)
{
    this.Close();
    this.Dispose();
}
```

Las referencias a la clase CInterpolacion, provienen de la definición de la respectiva clase incorporada en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:





INTERPOLACION DE NEWTON:

Consiste en generar el nuevo punto, a partir de un grupo de puntos (x_0; y_0), (x_1; y_1), (x_2; y_2), … (x_n; y_n) para lo cual se emplea las fórmulas de las diferencias divididas definidas por:
Conocida una función real f(x), definida sobre las abscisas x_0, x_1, x_2, ..., x_n no necesariamente equidistantes; entonces:

$$f(x_j, x_{j+1}) = \frac{f(x_{j+1}) - f(x_j)}{x_{j+1} - x_j}; 4.6$$

$$f(x_j, x_{j+1}, x_{j+2}) = \frac{f(x_{j+1}, x_{j+2}) - f(x_j, x_{j+1})}{x_{j+2} - x_j}; 4.7$$

...

$$f(x_0, x_1, x_2, \ldots x_i, \ldots x_n) = \frac{f(x_1, x_2, x_3, \ldots x_i, \ldots x_n) - f(x_0, x_1, x_2, \ldots x_i, \ldots x_{n-1})}{x_n - x_0}; 4.8$$

$$\forall j = 0,1,2,3,\ldots, n-1$$

Las fórmulas 4.6, 4.7 y 4.8 corresponden a las diferencias finitas divididas de primer, segundo y n-esimo grado, respectivamente.
Para obtener el polinomio de interpolación de Newton, se emplea la fórmula:

$$P(x) = f(x_0) + f(x_0, x_1)(x - x_0) + f(x_0, x_1, x_2)(x - x_0)(x - x_1) + \ldots$$
$$+ f(x_0, x_1, x_2, \ldots x_i, \ldots x_n)(x - x_0)(x - x_1)\ldots(x - x_{n-1}); 4.9$$

Ejemplo 4.6.
Obtener el polinomio de intepolación del problema 4.1 aplicando las fórmulas de interpolación de Newton mediante diferencias divididas, dados los puntos:
a) A(1, -1) y C(3, 15)
b) A(1, -1), B(2, 5) y C(3, 15)

Solución:
a) El cálculo de la primera diferencia dividida se desarrolla en la siguiente tabla:

Punto	X	y = f(x) = $2x^2 - 3$	IDD
0	1	-1	8
1	3	15	

De la tabla anterior: $f(x_0) = -1$; $f(x_0, x_1) = 8$; con estos valores, se aplica 4.9:
$P(x) = -1 + 8(x - 1) = 8x - 9$

b) El cálculo de las diferencias divididas se desarrolla en la siguiente tabla:

Punto	X	$y = f(x) = 2x^2 - 3$	IDD	IIDD
0	1	-1	6	2
1	2	5	10	
2	3	15		

De la tabla anterior: $f(x_0) = -1$; $f(x_0, x_1) = 6$ y $f(x_0, x_1, x_2) = 2$; con estos valores, se aplica 4.9:
$P(x) = -1 + 6(x - 1) + 2(x - 1)(x - 2) = -1 + 6x - 6 + 2x^2 - 6x + 4 = 2x^2 - 3$.

Como se puede observar, el método de Newton, ha producido los mismos resultados que el de Lagrange.

Ejemplo 4.7
Calcular el polinomio de interpolación de cuarto grado P(x, 4) y P(π/2, 4) para $y = f(x) = e^x\cos(2x)$, en el dominio $-2\pi \le x \le 2\pi$. Tomar cinco abscisas equidistantes. Mediante interpolación por diferencias divididas de Newton.
Solución
El cálculo de las diferencias divididas, se desarrolla en la siguiente tabla:

x =1.570796327

Punto	X	y = f(x)	IDD	IIDD	IIIDD	IVDD
0	-6.28318531	0.00186744	0.013160992			
1	-3.14159265	0.04321392	0.304554469	0.04637671	0.10894819	
2	0	1	7.047601352	1.07318924	2.52113665	0.191955859
3	3.14159265	23.1406926	163.0863767	24.8343424		
4	6.28318531	535.491656				

Por tanto $f(x_0) = 0.00186744$; $f(x_0, x_1) = 0.013160992$; $f(x_0, x_1, x_2) = 0.04637671$; $f(x_0, x_1, x_2, x_3) = 0.108948193$; y $f(x_0, x_1, x_2, x_3, x_4) = 0.191955859$.
Entonces el polinomio de interpolación es:
$P(x, 4) = 0.00187+0.01316(x + 2\pi)+0.04638(x + 2\pi)(x + \pi)+0.10895(x + 2\pi)(x + \pi)x + 0.19196(x + 2\pi)(x + \pi)x(x - \pi)$
Sustituyendo valores y operando $P(\pi/2, 4) = -9.374026529$.

Como se puede observar los resultados son bastante parecidos en los métodos de Lagrange y Newton. IDD, IIDD, … corresponden a la primera y segunda diferencia dividida, etc..

Ejemplo 4.8
Desarrollar una aplicación Matlab y otra en Visual C#, que tenga una función para calcular la ordenada y por el método de interpolación de Newton, mediante diferencias divididas; dicha función debe recibir las coordenadas de dos puntos (extremos) de la curva real; el valor de la abscisa *x* y el número de puntos; la aplicación debe calcular también los puntos intermedios y los errores absoluto y relativo. Considerar los datos del ejemplo 4.1.
Solución:

Programas en MATLAB y VISUAL C#:

Para cada lenguaje, se elabora un solo programa, desde el cual se ingresan las abscisas de los puntos extremos de la curva real y la abscisa a interpolar y el número de puntos. El programa debe calcular los puntos intermedios y desplegar las ordenadas interpolada y exacta y los errores absoluto y relativo.

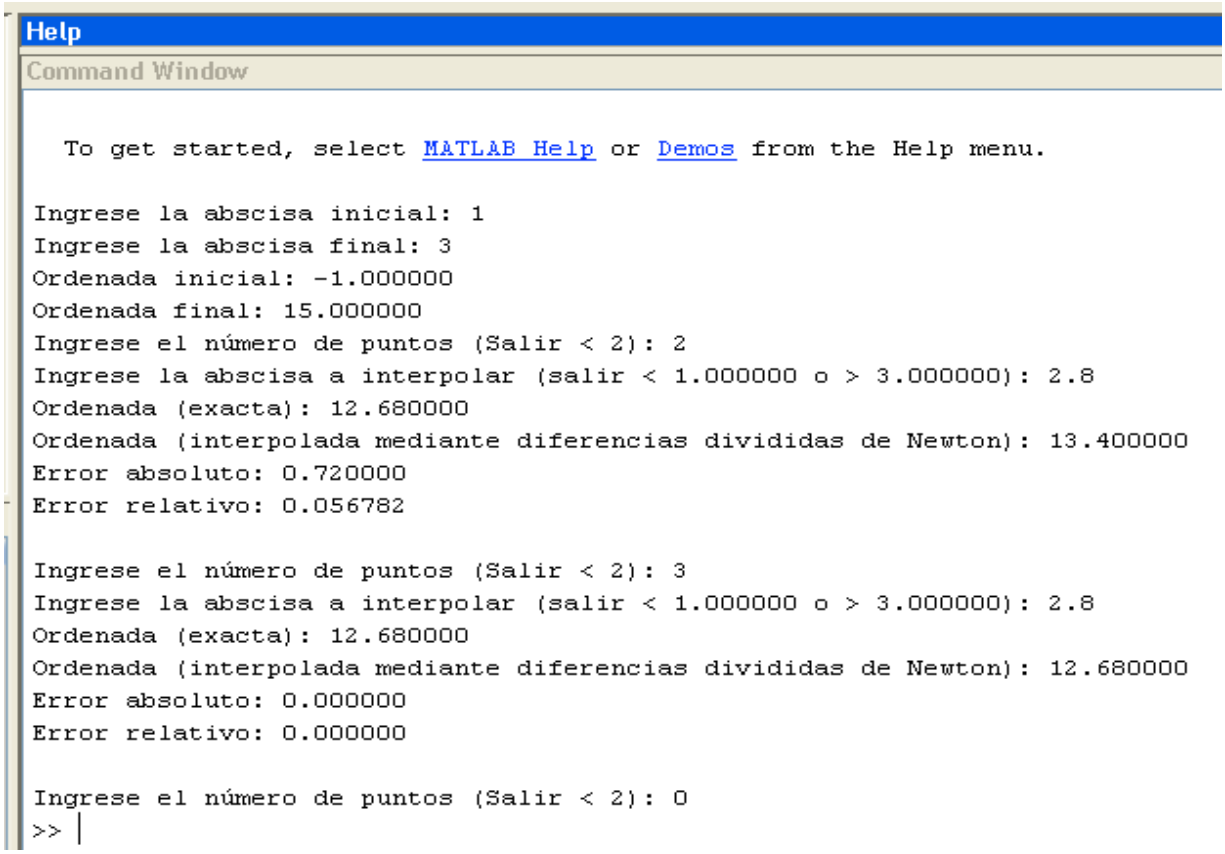
Archivos fuente en MATLAB:

```
% PrbInterpolacionNewton.m
% Prueba interpolación de Newton mediante diferencias divididas y calcula errores.
x(1) = input('Ingrese la abscisa inicial: ');
while 1
  xn = input('Ingrese la abscisa final: ');
  if xn > x(1) break; end;
end;
y(1) = fit(x(1));
yn = fit(xn);
disp(sprintf('Ordenada inicial: %f', y(1)));
disp(sprintf('Ordenada final: %f', yn));
while 1
   n = input('Ingrese el número de puntos (Salir < 2): ');
   if n < 2 break; end;
   xit = input(sprintf('Ingrese la abscisa a interpolar (salir < %f o > %f): ', x(1), xn));
   % Validación:
   if (xit < x(1) | xit > xn) break; end;
   % Cálculo del resto de puntos:
   if n > 2
      h = (xn - x(1)) / (n - 1);
      for i = 2 : n - 1
         x(i) = x(i-1) + h;
         y(i) = fit(x(i));
      end
   end;
```

```
  x(n) = xn;
  y(n) = yn;
  % Cálculo de la ordenada interpolada:
  yit = InterpolacionNewton(x, y, xit, n);
  % Cálculo de Y:
  Y = fit(xit);
  % Calcular los errores:
  erAbs = abs(Y - yit);
  erRel = erAbs / abs(Y);
  % Desplegar resultados:
  disp(sprintf('Ordenada (exacta): %f', Y));
  disp(sprintf('Ordenada (interpolada mediante diferencias divididas de Newton): %f',
yit));
  disp(sprintf('Error absoluto: %f', erAbs));
  disp(sprintf('Error relativo: %f\n', erRel));
end;

% InterpolacionNewton.m
% Interpolación de Newton mediante diferencias divididas.
function [y, vdd] = InterpolacionNewton(vx, vy, x, n)
 vdd = vy;
 y = vy(1);
 p = 1;
 for j = 1 : n - 1
   k = 1;
   p =  p * (x - vx(j));
   for i = 1 : n - j
     vdd(i) = (vdd(i + 1) - vdd(i)) / (vx(k + j) - vx(k));
     k = k + 1;
   end;
   y = y + vdd(1) * p;
 end;
return

% fit.m
function y = fit(x)
y = 2 * x ^2 - 3;
return
```

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa:



Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 4.4

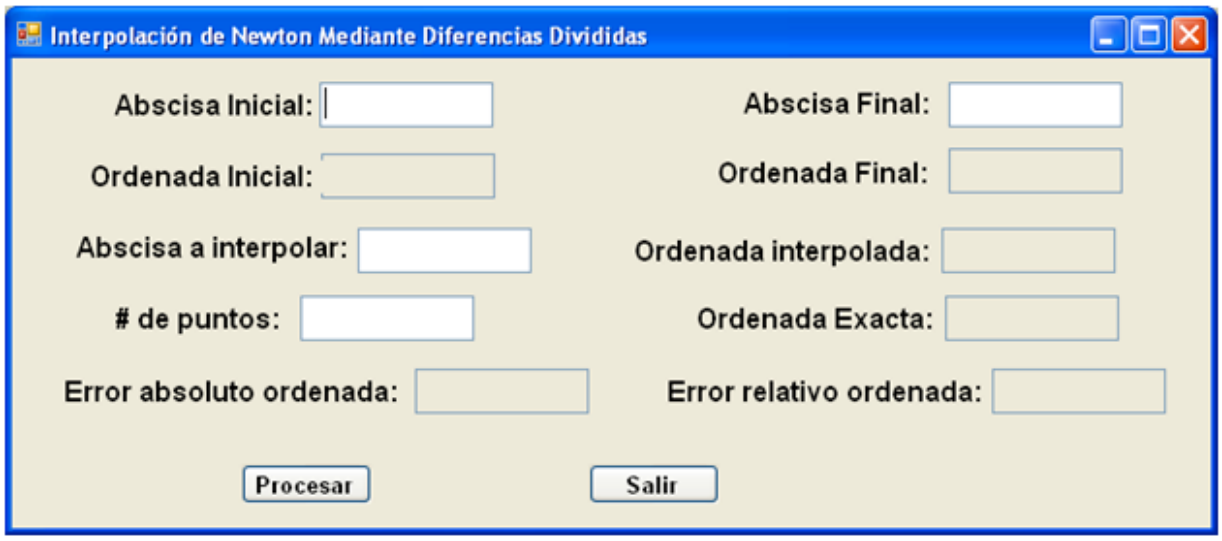


Figura 4.4 Ventana para Interpolación de Newton mediante Diferencias Divididas en Visual C#

Como se puede apreciar, existen diez cuadros de edición, mismos que permitirán ingresar los datos; y presentar los resultados; y dos botones, mismos que tienen la funcionalidad:

- Procesar: Ejecuta el método para interpolar mediante Diferencias Divididas de Newton y desplegar los resultados. Y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Función Externa:

```
// Funcion a ser analizada:
static double fit(double x)
{
      return 2 * Math.Pow(x, 2) - 3;
}
```

Evento que carga el formulario:

```
private void Form1_Load(object sender, EventArgs e)
{
     CInterpolacion.AsignaMemoria(true, 50);
}
```

Botón Procesar:

```
private void b_procesar_Click(object sender, EventArgs e)
{
```

```
    // Prueba interpolación de Newton mediante DIferencias Divididas y calcula errores.
    // Declaracion de variables
    double x, y, Y, h, erAbs, erRel;
    int n;  // Número de puntos.
    try
    {
        // Transferencia de captura de datos a variables:
        n = int.Parse(txtNPuntos.Text);
        // Validación:
        if (n < 2) return;
        // Transferencia de captura de datos a variables:
        CInterpolacion.vx[0] = double.Parse(txtAbscIni.Text);
        CInterpolacion.vx[n - 1] = double.Parse(txtAbscFin.Text);
       // Validación:
       if (CInterpolacion.vx[n - 1] < CInterpolacion.vx[0]) return;
       x = double.Parse(txtAbscInt.Text);
       // Validación:
       if (x < CInterpolacion.vx[0] || x > CInterpolacion.vx[n - 1]) return;
       // Definición del vector de ordenadas:
       CInterpolacion.vy[0] = fit(CInterpolacion.vx[0]);
       CInterpolacion.vy[n - 1] = fit(CInterpolacion.vx[n - 1]);
       // Cálculo del resto de puntos:
       if (n > 2)
       {
            h = (CInterpolacion.vx[n - 1] - CInterpolacion.vx[0]) / (n - 1);
           for (int i = 1; i < n - 1; i++)
           {
               CInterpolacion.vx[i] = CInterpolacion.vx[i - 1] + h;
               CInterpolacion.vy[i] = fit(CInterpolacion.vx[i]);
            }
       }
       // Cálculo de la ordenada interpolada:
       y = CInterpolacion.InterpolacionNewton(x, n);
       // Cálculo de la ordenada exacta:
       Y = fit(x);
       // Calcular los errores:
       erAbs = Math.Abs(Y - y);
       erRel = erAbs / Math.Abs(Y);
       // Transferir variables a pantalla:
       txtOrdIni.Text = Math.Round(CInterpolacion.vy[0], 9).ToString();
       txtOrdFin.Text = Math.Round(CInterpolacion.vy[n - 1], 9).ToString();
      txtOrdInt.Text = Math.Round(y, 9).ToString();
      txtOrdExacta.Text = Math.Round(Y, 9).ToString();
      txtErAbs.Text = Math.Round(erAbs, 9).ToString();
      txtErRel.Text = Math.Round(erRel, 9).ToString();
```

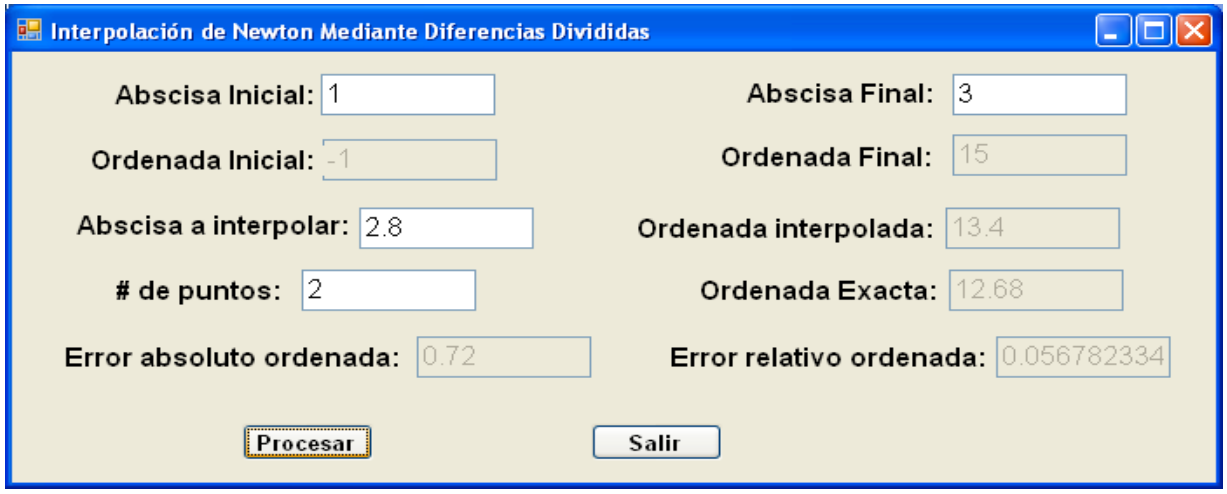
```
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
```

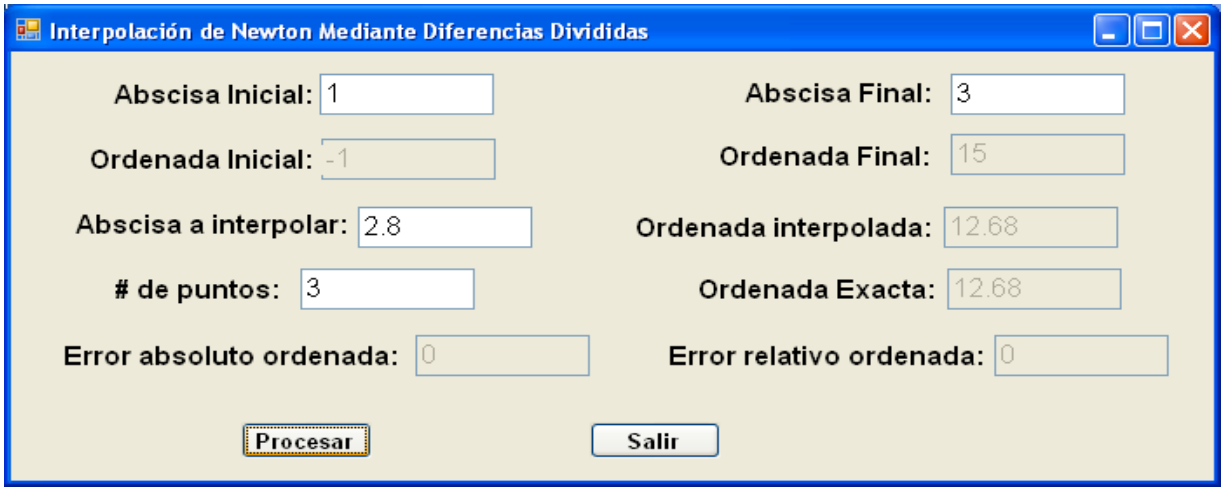
Botón Salir:

```
private void cb_salir_Click(object sender, EventArgs e)
{
    this.Close();
    this.Dispose();
}
```

Las referencias a la clase CInterpolacion, provienen de la definición de la respectiva clase incorporada en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:




Interpolación de Newton Mediante Diferencias Divididas
Abscisa Inicial: 1
Abscisa Final: 3
Ordenada Inicial: -1
Ordenada Final: 15
Abscisa a interpolar: 2.8
Ordenada interpolada: 12.68
de puntos: 3
Ordenada Exacta: 12.68
Error absoluto ordenada: 0
Error relativo ordenada: 0
Procesar
Salir

EJERCICIOS DEL CAPITULO IV

1.-Dada la función: f(x) = 2cos(3x) - 5e $^{-x}$ * sin(x), en el dominio, - π <= x <= π usando cinco abscisas equidistantes, calcular el polinomio de interpolación y L(π/4, 4), así como los errores absoluto y relativo; a través de interpolación por los polinomios de Lagrange.

2.- Ejecutar los programas en Matlab y Visual C# para interpolación mediante los polinomios de Lagrange, incorporando la función del numeral 1.

3.-Dadas las condiciones de la función del problema del numeral 1, calcular el polinomio de interpolación y P(π/4, 4), así como los errores absoluto y relativo; a través de interpolación por diferencias divididas de Newton. Analizar los resultados.

4.- Ejecutar los programas en Matlab y Visual C# para interpolación mediante diferencias divididas de Newton, incorporando la función del numeral 1.

5.- Conocidos los puntos (-2; -35), (1; 1), (2; 29), (3; 125), interpolar para x = 1.5, usando el método de Aitken.

6.- Implementar en Matlab y Visual C#, el proceso de interpolación por el método de Aitken.

5

AJUSTE DE CURVAS

En este capítulo, se analizarán los siguientes temas:

- Generalidades.
- Deducción del mecanismo para ajustar una muestra de puntos a una función polinómica a través del método de mínimos cuadrados.
- Implementación del método anteriormente descrito en Matlab y Visual C#.NET.
- Ajuste de curvas por interpolación mediante Polinomios de Lagrange.
- Implementación del método anteriormente descrito en Matlab y Visual C#.NET.
- Ajuste de curvas por interpolación mediante Diferencias Divididas de Newton.
- Implementación del método anteriormente descrito en Matlab y Visual C#.NET.

AJUSTE DE UNA MUESTRA A UNA FUNCION POLINOMICA USANDO EL METODO DE LOS MINIMOS CUADRADOS.

Generalidades.

Introducción:

En el presente capitulo, se presenta un modelo estadístico, mismo que permitirá ajustar una muestra obtenida de la medición de un experimento del mundo real (en medicina, biología, informática, física, química, administración, mercadeo, economía, entre otras disciplinas); a una función polinómica de la forma:

$$y = f(x, n) = A_0 + A_1x + A_2x^2 + \ldots + A_{n-1}x^{n-1} + A_nx^n \qquad (5.1)$$

Donde: **y** es la variable dependiente; **x** es la variable independiente; **n** es el grado de la función polinómica; y $\mathbf{A_0}$, $\mathbf{A_1}$, $\mathbf{A_2,\ldots, A_{n-1}, A_n}$ son los coeficientes de la función polinómica; dicho modelo estadístico se denomina de los mínimos cuadrados y previo a su descripción, se realizan las siguientes definiciones:

Muestra estadística:

(o muestra aleatoria o simplemente muestra) es un subconjunto de casos o individuos de una población estadística. Se obtienen con la intención de inferir propiedades de la totalidad de la población, para lo cual deben ser representativas de la misma. Para cumplir esta característica la inclusión de sujetos en la muestra debe seguir una técnica de muestreo. Por otra parte, en ocasiones, el muestreo puede ser más exacto que el estudio de toda la población porque el manejo de un menor

número de datos provoca también menos errores en su manipulación. En cualquier caso, el conjunto de individuos de la muestra son los sujetos realmente estudiados. El número de sujetos que componen la muestra suele ser inferior que el de la población, pero suficiente para que la estimación de los parámetros determinados tenga un nivel de confianza adecuado. Para que el tamaño de la muestra sea idóneo es preciso recurrir a su cálculo. La muestra se representa a través de un conjunto de puntos en el plano cartesiano, dando lugar a un diagrama de dispersión.

Diagrama de dispersión:

Es la representación gráfica del grado de relación entre dos variables cualitativas (variable independiente y variable dependiente).

Ejemplo 5.1:

Dada la siguiente muestra de ventas (en miles de USD) de un producto en los doce meses del año:

Mes	1	2	3	4	5	6	7	8	9	10	11	12
Venta	85	93	61	73	110	93	89	54	91	60	79	87.5

La variable independiente, se encuentra representada por el mes y la dependiente por la venta; el diagrama de dispersión se lo representa por la figura 5.1. Sobre este diagrama, se genera una función polinómica, misma que tiende a pasar por todos los puntos, lo cual produce un grado del polinomio que se aproxima al número de puntos de la muestra. En este capítulo, se desarrollará un modelo que permita considerar como entradas el tamaño de la muestra (p), los puntos que representa la muestra y el grado del polinomio (n); como resultado se obtendrá los coeficientes del polinomio y la representación del mismo en el diagrama de dispersión.

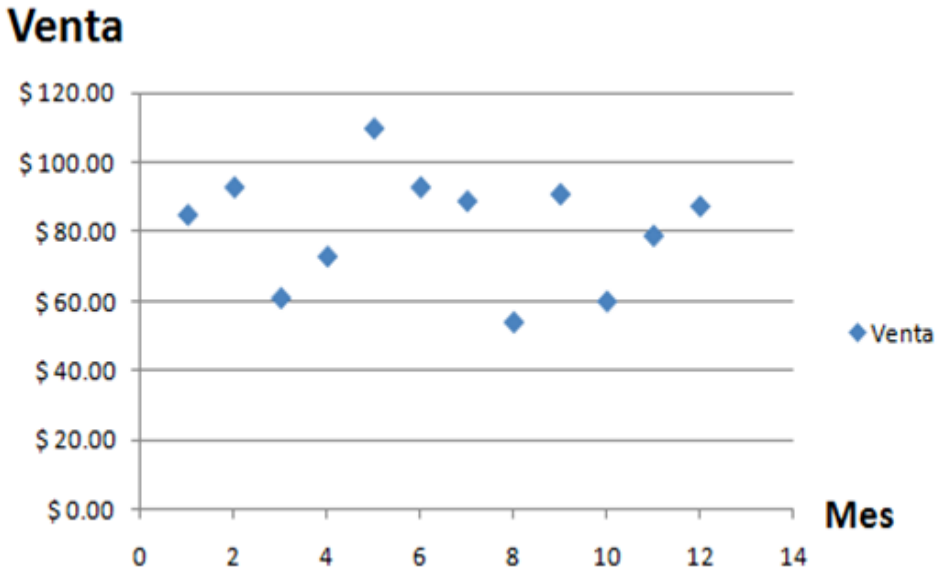


Figura 5.1 Diagrama de dispersión ventas por mes

Deducción del mecanismo para ajustar una muestra de puntos a una función polinómica a través del método de mínimos cuadrados.

Se tiene una muestra de p puntos y se desea ajustar a una función polinómica de la forma (5.1), donde n ≤ p. (5.2)
Sean y = ordenada dada por (5.1); ý = ordenada medida (muestra),
Entonces error = ý – y (5.3) ; error total = ∑ (ý – y) (5.4) ;
Para minimizar errores por el método de mínimos cuadrados se usa:

$$S = \sum_{i=1}^{p} (\acute{y}_i - y)^2 \qquad (5.5)$$

Sustituyendo (5.1) en (5.5):

$$S = \sum_{i=1}^{p} (\acute{y}_i - A_0 - A_1 x - A_2 x^2 - \ldots - A_{n-1} x^{n-1} - A_n x^n)^2 \qquad (5.6)$$

$$\partial S/\partial A_0 = 0;\ \partial S/\partial A_1 = 0;\ \ldots\ \partial S/\partial A_{n-1} = 0;\ \partial S/\partial A_n = 0; \qquad (5.7)$$

Aplicando (5.7) en (5.6), simplificando y ordenando:

$$\partial S/\partial A_0 = 2\sum_{i=1}^{p} (\acute{y}_i - A_0 - A_1 x - A_2 x^2 - \ldots - A_{n-1} x^{n-1} - A_n x^n) = 0$$

$$pA_0 + A_1 \sum_{i=1}^{p} x_i + A_2 \sum_{i=1}^{p} x_i^2 + \ldots + A_{n-1} \sum_{i=1}^{p} x_i^{n-1} + A_n \sum_{i=1}^{p} x_i^n = \sum_{i=1}^{p} \acute{y}_i$$

…

$$\partial S/\partial A_n = 2\sum_{i=1}^{p}(\acute{y}_i - A_0 - A_1x - A_2x^2 - \ldots - A_{n-1}x^{n-1} - A_nx^n)x^n = 0$$

$$\partial S/\partial A_1 = 2\sum_{i=1}^{p}(\acute{y}_i - A_0 - A_1x - A_2x^2 - \ldots - A_{n-1}x^{n-1} - A_nx^n)x = 0$$

$$A_0\sum_{i=1}^{p}x_i + A_1\sum_{i=1}^{p}x_i^2 + A_2\sum_{i=1}^{p}x_i^3 + \ldots + A_{n-1}\sum_{i=1}^{p}x_i^n + A_n\sum_{i=1}^{p}x_i^{n+1} = \sum_{i=1}^{p}x_i\acute{y}_i$$

$$\partial S/\partial A_2 = 2\sum_{i=1}^{p}(\acute{y}_i - A_0 - A_1x - A_2x^2 - \ldots - A_{n-1}x^{n-1} - A_nx^n)x^2 = 0$$

$$A_0\sum_{i=1}^{p}x_i^2 + A_1\sum_{i=1}^{p}x_i^3 + A_2\sum_{i=1}^{p}x_i^4 + \ldots + A_{n-1}\sum_{i=1}^{p}x_i^{n+1} + A_n\sum_{i=1}^{p}x_i^{n+2} = \sum_{i=1}^{p}x_i^2\acute{y}_i$$

$$A_0\sum_{i=1}^{p} x_i^{\,n} + A_1\sum_{i=1}^{p} x_i^{\,n+1} + A_2\sum_{i=1}^{p} x_i^{\,n+2} + \ldots + A_{n-1}\sum_{i=1}^{p} x_i^{\,2n-1} + A_n\sum_{i=1}^{p} x_i^{\,2n} = \sum_{i=1}^{p} x_i^{\,n}\acute{y}_i$$

$$\begin{pmatrix} p & \sum x_i & \sum x_i^2 & \ldots & \sum x_i^n \\ \sum x_i & \sum x_i^2 & \sum x_i^3 & \ldots & \sum x_i^{n+1} \\ \sum x_i^2 & \sum x_i^3 & \sum x_i^4 & \ldots & \sum x_i^{n+2} \\ & & \ldots & & \\ \sum x_i^n & \sum x_i^{n+1} & \sum x_i^{n+2} & \ldots & \sum x_i^{2n} \end{pmatrix} \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ \ldots \\ A_n \end{pmatrix} = \begin{pmatrix} \sum \acute{y}_i \\ \sum x_i\acute{y}_i \\ \sum x_i^2\acute{y}_i \\ \ldots \\ \sum x_i^n\acute{y}_i \end{pmatrix} \quad (5.8)$$

Como se puede observar, se genera un sistema de n + 1 ecuaciones por n+1 incógnitas, planteándolo matricialmente

Algoritmo para ajustar una muestra a un polinomio:

1) Calcular las constantes de graficación.
2) Graficar los ejes de coordenadas y los puntos de la muestra.
3) Aplicando la fórmula 5.8 calcular la matriz de coeficientes y el vector de términos independientes.
4) Resolver el sistema de ecuaciones aplicando el método crout simétrico.
5) Graficar el polinomio una vez conocidos sus coeficientes.
6) Repetir desde el paso (3) para cada grado del polinomio.

Ejemplo 5.2:

Dada la muestra del ejemplo 5.1, ajustarla a un polinomio de grado 5.

Solución:

p = tamaño de la muestra = 12; n = grado del polinomio = 5.

Se debe ajustar a un polinomio de la forma:

$y = A_0 + A_1x + A_2x^2 + A_3x^3 + A_4x^4 + A_5x^5$

Aplicando 8, se debe plantear el siguiente sistema:

$$\begin{pmatrix} 12 & \sum_{i=1}^{12} x_i & \sum_{i=1}^{12} x_i^2 & \sum_{i=1}^{12} x_i^3 & \sum_{i=1}^{12} x_i^4 & \sum_{i=1}^{12} x_i^5 \\ \sum_{i=1}^{12} x_i & \sum_{i=1}^{12} x_i^2 & \sum_{i=1}^{12} x_i^3 & \sum_{i=1}^{12} x_i^4 & \sum_{i=1}^{12} x_i^5 & \sum_{i=1}^{12} x_i^6 \\ \sum_{i=1}^{12} x_i^2 & \sum_{i=1}^{12} x_i^3 & \sum_{i=1}^{12} x_i^4 & \sum_{i=1}^{12} x_i^5 & \sum_{i=1}^{12} x_i^6 & \sum_{i=1}^{12} x_i^7 \\ \sum_{i=1}^{12} x_i^3 & \sum_{i=1}^{12} x_i^4 & \sum_{i=1}^{12} x_i^5 & \sum_{i=1}^{12} x_i^6 & \sum_{i=1}^{12} x_i^7 & \sum_{i=1}^{12} x_i^8 \\ \sum_{i=1}^{12} x_i^4 & \sum_{i=1}^{12} x_i^5 & \sum_{i=1}^{12} x_i^6 & \sum_{i=1}^{12} x_i^7 & \sum_{i=1}^{12} x_i^8 & \sum_{i=1}^{12} x_i^9 \\ \sum_{i=1}^{12} x_i^5 & \sum_{i=1}^{12} x_i^6 & \sum_{i=1}^{12} x_i^7 & \sum_{i=1}^{12} x_i^8 & \sum_{i=1}^{12} x_i^9 & \sum_{i=1}^{12} x_i^{10} \end{pmatrix} \begin{pmatrix} A_0 \\ A_1 \\ A_2 \\ A_3 \\ A_4 \\ A_5 \end{pmatrix} = \begin{pmatrix} \sum_{i=1}^{12} \acute{y}_i \\ \sum_{i=1}^{12} x_i \acute{y}_i \\ \sum_{i=1}^{12} x_i^2 \acute{y}_i \\ \sum_{i=1}^{12} x_i^3 \acute{y}_i \\ \sum_{i=1}^{p} x_i^4 \acute{y}_i \\ \sum_{i=1}^{12} x_i^5 \acute{y}_i \end{pmatrix}$$

Desarrollando los cálculos:

	Mes (x)	Venta(y)	x^2	x^3	x^4	x^5	x^6	x^7	x^8	x^9	x^10
	1	$ 85.00	1	1	1	1	1	1	1	1	1
	2	$ 93.00	4	8	16	32	64	128	256	512	1024
	3	$ 61.00	9	27	81	243	729	2187	6561	19683	59049
	4	$ 73.00	16	64	256	1024	4096	16384	65536	262144	1048576
	5	$ 110.00	25	125	625	3125	15625	78125	390625	1953125	9765625
	6	$ 93.00	36	216	1296	7776	46656	279936	1679616	10077696	60466176
	7	$ 89.00	49	343	2401	16807	117649	823543	5764801	40353607	282475249
	8	$ 54.00	64	512	4096	32768	262144	2097152	16777216	134217728	1073741824
	9	$ 91.00	81	729	6561	59049	531441	4782969	43046721	387420489	3486784401
	10	$ 60.00	100	1000	10000	100000	1000000	10000000	100000000	1000000000	1E+10
	11	$ 79.00	121	1331	14641	161051	1771561	19487171	214358881	2357947691	2.5937E+10
	12	$ 87.50	144	1728	20736	248832	2985984	35831808	429981696	5159780352	6.1917E+10
∑	78	975.5	650	6084	60710	630708	6735950	73399404	812071910	9092033028	1.0277E+11

	y*x	$y*x^2$	$y*x^3$	$y*x^4$	$y*x^5$
	85	85	85	85	85
	186	372	744	1488	2976
	183	549	1647	4941	14823
	292	1168	4672	18688	74752
	550	2750	13750	68750	343750
	558	3348	20088	120528	723168
	623	4361	30527	213689	1495823
	432	3456	27648	221184	1769472
	819	7371	66339	597051	5373459
	600	6000	60000	600000	6000000
	869	9559	105149	1156639	12723029
	1050	12600	151200	1814400	21772800
∑	6247	51619	481849	4817443	50294137

Como se puede apreciar, se tiene un sistema simétrico de 6 ecuaciones por 6 incógnitas, resolviéndolo por el método de Crout:

12 12					
78 78	650 143				
650 650	6084 1859	60710 1.3347e+003			
6084 6084	60710 21164	630708 26026	6735950 1.1583e+004		
60710 60710	630708 236093	6735950 3.7828e+005	73399404 3.0116e+005	812071910 9.4135e+004	
630708 630708	6735950 2636348	73399404 4963530	812071910 5.3346e+006	9092033028 3.0594e+006	1.02769E+11 7.0720e+005

975.5	
81.2917	147.8863636
6247	
-0.6556	-87.47589896
51619	
-0.0014	35.36357851
481849	
0.1020	-5.840924088
4817443	
0.0724	0.41567899
50294137	
-0.010562783	-0.010562783

Por tanto el polinomio es:

$$y = A_0 + A_1x + A_2x^2 + A_3x^3 + A_4x^4 + A_5x^5$$

$$y = 147.8863636 - 87.47589896\,x + 35.36357851\,x^2 - 5.840924088\,x^3 + 0.41567899\,x^4 - -0.010562783\,x^5$$

Implementación de los algoritmos en MATLAB y VISUAL C#:

Para cada lenguaje, se elabora un solo programa, desde el cual se abren archivos para lectura de datos; se ejecuta cada uno de los métodos incluyendo los que generan la matriz de coeficientes y el vector de términos independientes; se resuelve el sistema, se grafica la familia de polinomios y se escribe los resultados en un archivo de salida, en el caso de Matlab y se despliegan los resultados en una caja lista en el caso de Visual C#.

Archivos fuente en MATLAB:

```
% ajuste.m
% ESTE PROGRAMA EFECTUA EL AJUSTE DE CURVAS A FUNCIONES POLINOMICAS
%  DE LA FORMA F(X)=A[0]+A[1]*X+...+A[N -1]*X**(N-1)+A[N]*X**N
%  POR EL METODO DE LOS MINIMOS CUADRADOS; P ES EL NUMERO DE PUNTOS
%  CUYAS COORDENADAS SON (Xi,Yi); N ES EL GRADO DEL POLINOMIO Y A ES
%    VECTOR DE COEFICIENTES DEL POLINOMIO AJUSTADO.
%    PARA ESTE PROGRAMA. ADEMAS REALIZA LA GRAFICACION.
global A N;
% APERTURA DE ARCHIVOS:
ent = fopen('D:\AnalNumat\puntos.dat', 'r');
sal = fopen('D:\AnalNumat\ajuste.dat', 'w');
% LECTURA DEL NUMERO DE PUNTOS DE LA MUESTRA:
P = fscanf(ent, '%d', 1);
% CALCULO INICIAL DE N Y LECTURA DE LOS PUNTOS DE LA MUESTRA:
N = P - 1;
X = LeeVector(ent, P);
Y = LeeVector(ent, P);
fclose(ent);
% PROCESO:
fprintf(sal,'AJUSTE DE POLINOMIOS POR EL METODO DE LOS MINIMOS CUADRADOS\r\n');
fprintf(sal,'VECTOR DE ABSCISAS:\r\n');
EscribeVector(sal,X,P);
fprintf(sal,'\r\n');
fprintf(sal,'VECTOR DE ORDENADAS:\r\n');
EscribeVector(sal,Y,P);
fprintf(sal,'\r\n');
colr = ['r', 'g', 'y', 'b', 'c', 'm'];
pos = 0;
while 1
  fprintf(sal,sprintf('COEFICIENTES DEL POLINOMIO DE GRADO N = %d DE LA FORMA:\r\n', N));
  fprintf(sal,'F(X)=A[0]+A[1]*X+...+A[N-1]*X**(N-1)+A[N]*X**N\r\n');
  fprintf(sal,'DADOS EN EL ORDEN A[0],A[1],...,A[N-1],A[N]\r\n');
  [Z, A] = GeneraMatrizYVector(X, Y, N+1, P);
```

```
  [det, Z, A] = CroutSim(Z, A, N + 1, 1);
  EscribeVector(sal, A, N+1);
  fprintf(sal,'\r\n\r\n');
  % Invertir el orden del vector A, para calcular puntos:
  for i = 1 : (N + 1) / 2
     det = A(i);
     A(i) = A(N - i + 2);
     A(N - i + 2) = det;
  end;
  % Grafica la muestra:
  plot(X, Y, '*');
  hold on;
  % Generación de los puntos de la función:
  xc(1) = X(1);
  yc(1) = HORNER(xc(1));
  ni = 100;
  delta = (X(P) - X(1)) / ni;
  for i = 2 : ni + 1
    xc(i) = xc(i - 1) + delta;
    yc(i) = HORNER(xc(i));
  end;
  % Grafica el polinomio:
  plot(xc, yc, colr(mod(pos, 6) + 1));
  pos = pos + 1;
  while 1
    N = input(sprintf('INGRESE EL GRADO DEL POLINOMIO < %d (SALIR <= 0): ', P));
    if N < P
      break;
    else
       disp(sprintf('EL VALOR N = %d, DEBE SER MENOR QUE %d', N, P));
    end;
  end;
  if N <= 0
    break;
  end;
end;
fclose(sal);


%GeneraMatrizYVector.m
% FUNCION QUE RECIBE UN NUMERO P DE PUNTOS Y SUS COORDENADAS
%  (Xi,Yi)), Y EL GRADO DE AJUSTE N, A UN POLINOMIO; REGRESA LA MATRIZ
%  A DE N FILAS POR N COLUMNAS Y EL VECTOR B DE N ELEMENTOS; QUE CONFORMAN
%  EL SISTEMA DE ECUACIONES SIMULTANEAS LINEALES PARA AJUSTAR UNA CURVA A
%  UN POLINOMIO DE GRADO N.
function [Z, B] =  GeneraMatrizYVector(X, Y, N, P)
 K = 0;
 for I = 1 : N
   Z(Ind(I, I)) = 0;
   B(I) = 0;
   for J = 1 : P
     Z(Ind(I, I)) = Z(Ind(I, I)) + X(J) ^ K;
     B(I) = B(I) + Y(J) * X(J) ^ (I-1);
   end;
   for  L=I+1 : N
     Z(Ind(L,I)) = 0;
     for J = 1 : P
       Z(Ind(L,I)) = Z(Ind(L,I)) + X(J) ^ (K+L-I);
     end;
   end;
   K = K + 2;
 end;
```

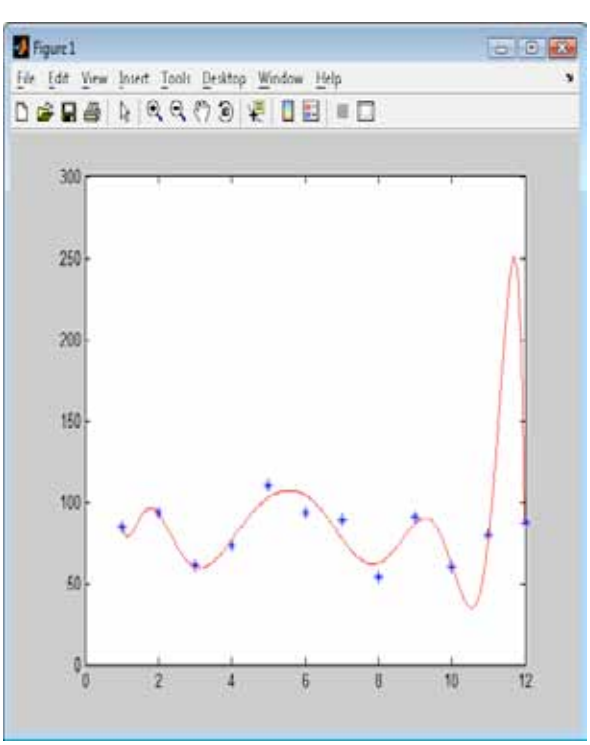
La definición de las funciones LeeVector, EscribeVector, CroutSim y HORNER, ya se la realizó anteriormente.

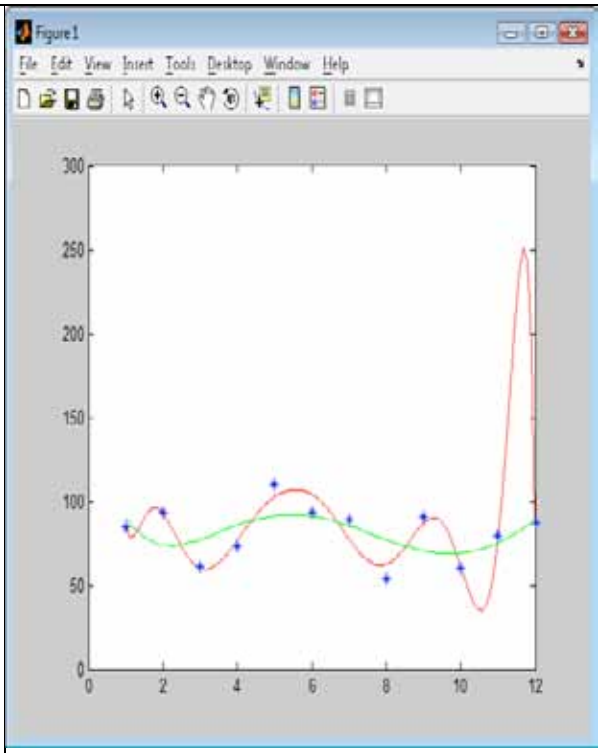
Resultados obtenidos:

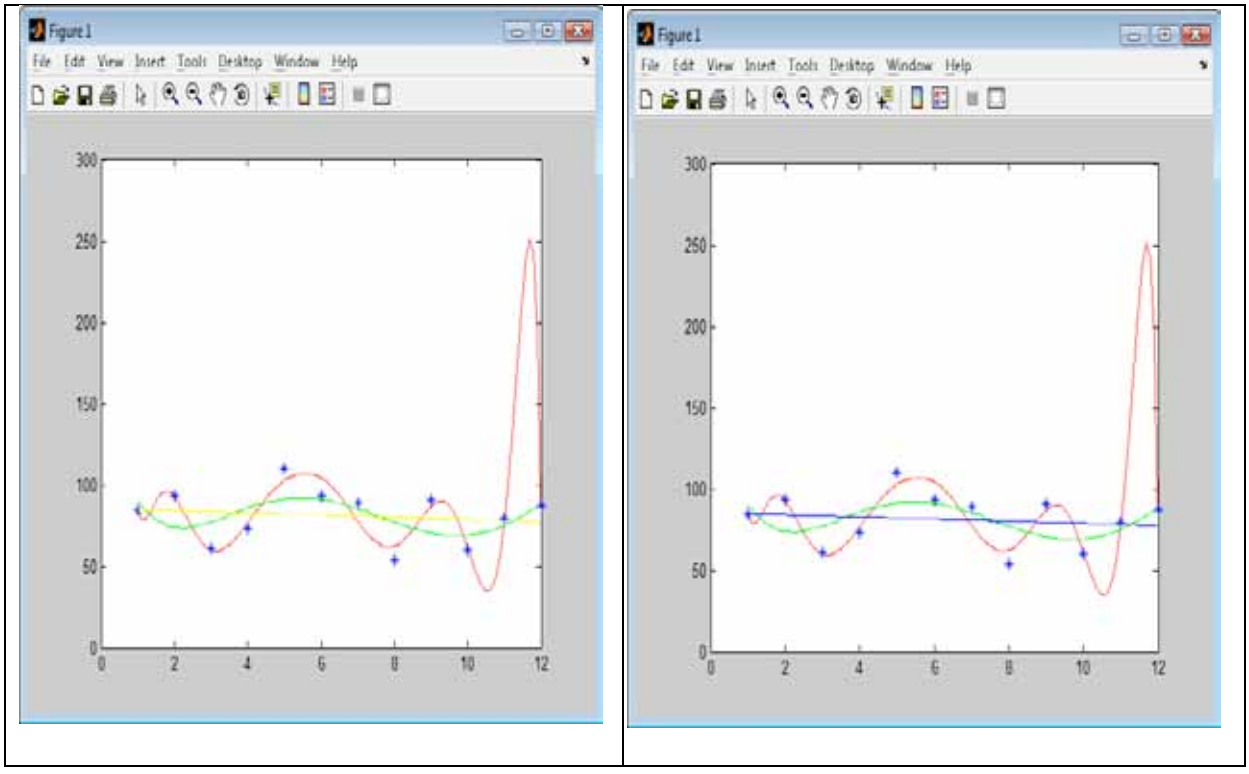
Se presenta a continuación los resultados que produce la ejecución de este programa, como se había indicado anteriormente, la entrada – salida, se realiza sobre archivos, cuyos contenidos son:

puntos.dat:
12
1
2
3
4
5
6
7
8
9
10
11
12
85
93
61
73
110
93
89
54
91
60
79
87.5

```
Command Window

  To get started, select MATLAB Help or Demos from the Help menu.

Warning: Function call CroutSim invokes inexact match C:\AnalisisNumer

> In ajuste at 34
Warning: Function call HORNER invokes inexact match C:\AnalisisNumeric

> In ajuste at 48
INGRESE EL GRADO DEL POLINOMIO < 12 (SALIR <= 0): 5
INGRESE EL GRADO DEL POLINOMIO < 12 (SALIR <= 0): 2
INGRESE EL GRADO DEL POLINOMIO < 12 (SALIR <= 0): 14
EL VALOR N = 14, DEBE SER MENOR QUE 12
INGRESE EL GRADO DEL POLINOMIO < 12 (SALIR <= 0): 1
INGRESE EL GRADO DEL POLINOMIO < 12 (SALIR <= 0): 0
>>
```







ajuste.dat:

AJUSTE DE POLINOMIOS POR EL METODO DE LOS MINIMOS CUADRADOS
VECTOR DE ABSCISAS:
1.000000 2.000000 3.000000 4.000000 5.000000 6.000000 7.000000 8.000000
9.000000 10.000000 11.000000 12.000000

VECTOR DE ORDENADAS:
85.000000 93.000000 61.000000 73.000000 110.000000 93.000000 89.000000
54.000000
91.000000 60.000000 79.000000 87.500000

COEFICIENTES DEL POLINOMIO DE GRADO N = 11 DE LA FORMA:
F(X)=A[0]+A[1]*X+...+A[N-1]*X**(N-1)+A[N]*X**N
DADOS EN EL ORDEN A[0],A[1],...,A[N-1],A[N]
2349.158228 -7404.022421 9968.630885 -7339.915015 3323.094358 -
984.274611 196.774301 -26.766037
2.441228 -0.142745 0.004830 -0.000072

COEFICIENTES DEL POLINOMIO DE GRADO N = 5 DE LA FORMA:
F(X)=A[0]+A[1]*X+...+A[N-1]*X**(N-1)+A[N]*X**N
DADOS EN EL ORDEN A[0],A[1],...,A[N-1],A[N]

147.886364 -87.475899 35.363579 -5.840924 0.415679 -0.010563

COEFICIENTES DEL POLINOMIO DE GRADO N = 2 DE LA FORMA:
F(X)=A[0]+A[1]*X+...+A[N-1]*X**(N-1)+A[N]*X**N
DADOS EN EL ORDEN A[0],A[1],...,A[N-1],A[N]
85.511364 -0.637737 -0.001374

COEFICIENTES DEL POLINOMIO DE GRADO N = 1 DE LA FORMA:
F(X)=A[0]+A[1]*X+...+A[N-1]*X**(N-1)+A[N]*X**N
DADOS EN EL ORDEN A[0],A[1],...,A[N-1],A[N]
85.553030 -0.655594

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 5.2 Como se puede apreciar, existen un cuadro de texto, mismo que permite ingresar el grado del polinomio; un cuadro de imagen, el cual desplegará los gráficos; un cuadro de lista, el cual permitirá presentar los resultados; y seis botones, mismos que tienen la funcionalidad:

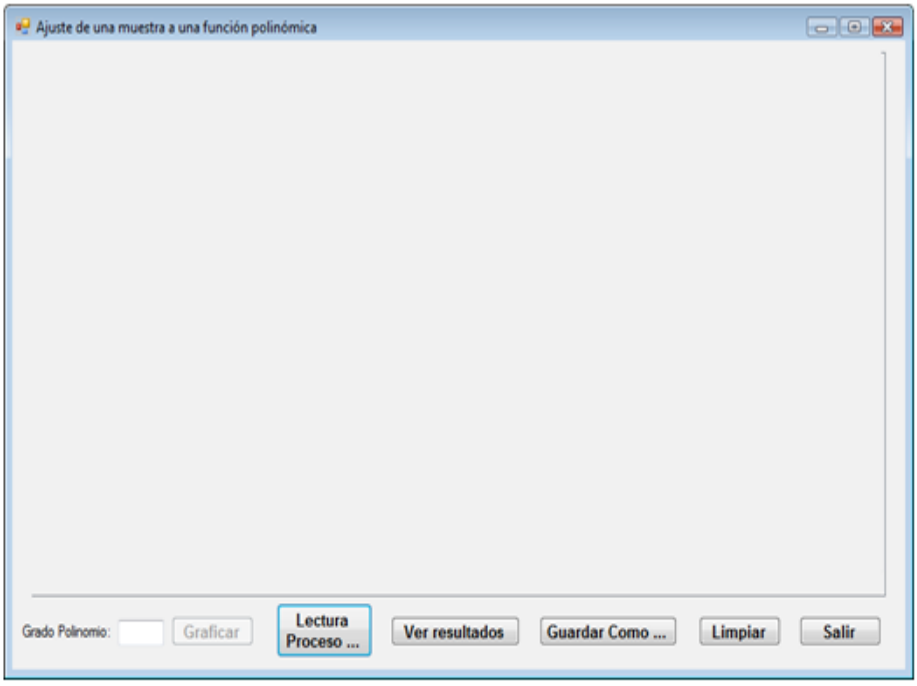
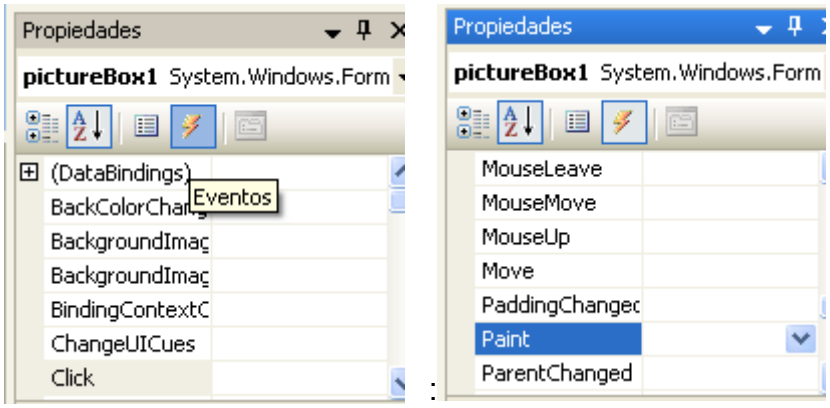


Figura 5.2 Ventana para ajustar una muestra a una función polinómica en Visual C#

- Graficar: valida el grado del polinomio, genera la matriz de coeficientes y el vector de términos independientes, resuelve el sistema y grafica el polinomio.
- Lectura Proceso…: abre una ventana para registrar el nombre del archivo el cual permite leer los datos de la muestra desde el medio de almacenamiento permanente; ejecuta los métodos para generar la matriz de coeficientes y el vector de términos independientes para el grado del polinomio igual al tamaño de

la muestra menos uno, resuelve el sistema y grafica el polinomio. y despliega los resultados.
- Ver resultados: visualiza los resultados en el cuadro de texto y oculta el cuadro de imagen y viceversa.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

Para que funcione el proceso de graficación, debe activarse el evento paint, en este caso debe accederse a las propiedades del pictureBox del formulario, realizar los pasos que se detalla gráficamente a continuación:



Luego de hacer doble clic sobre el evento Paint, se genera el siguiente código fuente:

```
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{

}
```

Se describe a continuación el código fuente:

Variables y funciones miembro de la clase Form1:

```
// Datos Miembro:
Random r = new Random(DateTime.Now.Millisecond);
private Bitmap offScreenBitMap = null;
double EH, EV, MAXX, MAXY, MINX, MINY, X0, X1, Y0, Y1;
double MAXPX, MAXPY, CX, CY;
int n;

// Funciones miembro:
public Form1()
{
   InitializeComponent();
}
```

```
private void MaxMin (double [] v, int n, ref double max, ref double min)
{
   max = v[0];
   min = v[0];
   for (int i = 1; i < n; i++)
   {
      if (max < v[i]) max = v[i];
      if (min > v[i]) min = v[i];
   }
}

private void calculaGraficaPolinomio()
{
   int i;
   // Desplegar resultados del polinomio:
   listBox1.Items.Add("COEFICIENTES DEL POLINOMIO DE GRADO N = " + n.ToString() + " DE LA
FORMA");
   listBox1.Items.Add("F(X)=A[0]+A[1]*X+...+A[N-1]*X**(N-1)+A[N]*X**N");
   listBox1.Items.Add("DADOS EN EL ORDEN A[0],A[1],...,A[N-1],A[N]");
   // Calcular el polinomio de grado n:
   CGenMatrizYVector.n = ++n;
   CGenMatrizYVector.GeneraMatrizYVector();
   CResolSisSimetricosEcuaciones.n = n;
   CResolSisSimetricosEcuaciones.A = CGenMatrizYVector.T;
   CResolSisSimetricosEcuaciones.B = CGenMatrizYVector.A;
   CResolSisSimetricosEcuaciones.CroutSimetrico(true);
   CLectEscrMatrices.n = n;
   CLectEscrMatrices.b = CGenMatrizYVector.A;
   CLectEscrMatrices.EscribeVector(" POLINOMIO:");
   listBox1.Items.Add("");
   /* DIBUJAR EL POLINOMIO */
   // Invertir el orden del vector A, para calcular puntos:
   double aux;
   for (i = 0; i < n / 2; i++)
   {
      aux = CGenMatrizYVector.A[i];
      CGenMatrizYVector.A[i] = CGenMatrizYVector.A[n - i - 1];
      CGenMatrizYVector.A[n - i - 1] = aux;
   }
   CHorner.a = CGenMatrizYVector.A;
   CHorner.n = n - 1;
   X1 = X0;
   int red = r.Next(255);
   int green = r.Next(255);
   int blue = r.Next(255);
   Graphics g = Graphics.FromImage(offScreenBitMap);
   Pen p = new Pen(Color.FromArgb(red, green, blue), 2);
   for (i = 0; i < MAXPX; i++)
   {
      g.DrawEllipse(p, (float)(Math.Round((X1 - X0) * EH) + CX), (float)(2 * MAXPY - Math.Round(EV *
(CHorner.horner(X1) - Y0)) + CY), 2, 2);
      X1 = X1 + 1 / EH;
   }
   pictureBox1.Invalidate();
   p.Dispose();
```

```
    g.Dispose();
}
```

Evento Paint del objeto pictureBox1:

```
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
    if (offScreenBitMap == null)
    {
        return; // Nothing to do
    }
    else
    {
        e.Graphics.DrawImage(offScreenBitMap, new Point(0, 0));
    }
}
```

Botón Graficar:

```
private void b_Graficar_Click(object sender, EventArgs e)
{
    try
    {
        n = Int32.Parse(txtGra.Text);
        if (n < 1 || n >= CGenMatrizYVector.p) {
            MessageBox.Show("Error: el grado " + n.ToString() + " debe estar entre 1 y " +
(CGenMatrizYVector.p - 1).ToString());
            return;
        }
        calculaGraficaPolinomio();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Botón Lectura Proceso…:

```
private void b_lectura_Click(object sender, EventArgs e)
{
    try
    {
        // Abrir el archivo de entrada a traves del cuadro de dialogo:
        CEntSalArchivos objes = new CEntSalArchivos("d:\\");
```

```
        objes.AbrirArchivoEnt();
        // Leer el tamaño de la muestra:
        CGenMatrizYVector.p = Int16.Parse(objes.LeerLinea());
        if (CGenMatrizYVector.p > 0)
        {
          // Definir el arreglo de abscisas de la muestra:
          CGenMatrizYVector.X = new double[CGenMatrizYVector.p];
          CLectEscrMatrices.b = CGenMatrizYVector.X;
          CLectEscrMatrices.n = CGenMatrizYVector.p;
          CLectEscrMatrices.obent = objes;
          CLectEscrMatrices.LeeVector();
          // Definir el arreglo de ordenadas de la muestra:
          CGenMatrizYVector.Y = new double[CGenMatrizYVector.p];
          CLectEscrMatrices.b = CGenMatrizYVector.Y;
          CLectEscrMatrices.LeeVector();
          n = CGenMatrizYVector.p;
          CGenMatrizYVector.A = new double[n];
          CGenMatrizYVector.T = new double[n * (n + 1) / 2];
          n--;
          CLectEscrMatrices.lb = listBox1;
          listBox1.Items.Add("AJUSTE DE POLINOMIOS POR EL METODO DE LOS MINIMOS
CUADRADOS");
          CLectEscrMatrices.b = CGenMatrizYVector.X;
          CLectEscrMatrices.EscribeVector(" DE ABSCISAS:");
          listBox1.Items.Add("");
          CLectEscrMatrices.b = CGenMatrizYVector.Y;
          CLectEscrMatrices.EscribeVector(" DE ORDENADAS:");
          listBox1.Items.Add("");

          // Calcular las constantes de graficación:
          int i;
          if (offScreenBitMap == null)
          {
            offScreenBitMap = new Bitmap(pictureBox1.Width, pictureBox1.Height);
          }
          Graphics g = Graphics.FromImage(offScreenBitMap);
          Pen p = new Pen(Color.FromArgb(0, 0, 0), 2);
          MAXPX = Math.Round(0.75 * pictureBox1.Width, 0);
          MAXPY = Math.Round(0.40 * pictureBox1.Height, 0);
          CX = Math.Round(0.11 * pictureBox1.Width, 0);
          CY = Math.Round(0.08 * pictureBox1.Height, 0);
          /* CALCULA EL MAXIMO Y MINIMO DE LAS COORDENADAS DE LOS PUNTOS */
          MaxMin(CGenMatrizYVector.X, CGenMatrizYVector.p, ref MAXX, ref MINX);
          MaxMin(CGenMatrizYVector.Y, CGenMatrizYVector.p, ref MAXY, ref MINY);
          /* CALCULA ESCALAS Y PUNTOS INICIALES DE REFERENCIA */
          EH = MAXPX / (MAXX - MINX);
          EV = 2 * MAXPY / (MAXY - MINY);
          X0 = 0;
          Y0 = 0;
          X1 = MAXX;
          Y1 = MAXY;
          if (MAXX < 0)
          {
            EH = MAXPX / Math.Abs(MINX);
            X0 = MINX;
```

```
            X1 = 0;
          }
          else if (MINX >= 0)
            EH = MAXPX / MAXX;
          else
          {
            X0 = MINX;
            X1 = MAXX;
          }
          if (MAXY < 0)
          {
            EV = 2 * MAXPY / Math.Abs(MINY);
            Y0 = MINY;
            Y1 = 0;
          }
          else if (MINY >= 0)
            EV = 2 * MAXPY / MAXY;
          else
          {
            Y0 = MINY;
            Y1 = MAXY;
          }
          /* DIBUJAR LOS EJES DEL SISTEMA DE REFERENCIA */
          g.DrawLine(p, (float)CX, (float)(Math.Round(2 * MAXPY + Y0 * EV) + CY),
              (float)(Math.Round((X1 - X0) * EH) + Math.Round(1.25 * CX)),                (float)(Math.Round(2
* MAXPY + Y0 * EV) + CY));
          Font df = new Font("Arial", 14);
          SolidBrush db = new SolidBrush(Color.Black);
          g.DrawString("X", df, db, (float)(Math.Round((X1 - X0) * EH) + Math.Round(1.4375 * CX)),
                (float)(Math.Round(2 * MAXPY + Y0 * EV) + CY - 3));
              g.DrawLine(p, (float)(Math.Round(-X0 * EH) + CX), (float)(Math.Round(2 * MAXPY) + CY),
                 (float)(Math.Round(-X0 * EH) + CX),
                 (float)(Math.Round(2 * MAXPY - (Y1 - Y0) * EV) + Math.Round(0.5 * CY)));
              g.DrawString("Y", df, db, (float)(Math.Round(-X0 * EH) + Math.Round(0.9375 * CX)),
                (float)(Math.Round(2 * MAXPY - (Y1 - Y0) * EV)));
          /* DIBUJAR LOS PUNTOS */
          p.Dispose();
          int red = r.Next(255);
          int green = r.Next(255);
          int blue = r.Next(255);
          p = new Pen(Color.FromArgb(red, green, blue), 2);
          for (i = 0; i < CGenMatrizYVector.p; i++)
            g.DrawEllipse(p, (float)(Math.Round(EH * (CGenMatrizYVector.X[i] - X0)) + CX),
                  (float)(Math.Round(2 * MAXPY - EV * (CGenMatrizYVector.Y[i] - Y0)) + CY), 6, 6);
          p.Dispose();
          g.Dispose();
          calculaGraficaPolinomio();
          b_Graficar.Enabled = true;
        }
      }
      catch (Exception ex)
      {
        MessageBox.Show(ex.Message);
      }
    }
```

Botón Ver resultados:

```
private void b_ver_Click(object sender, EventArgs e)
{
  if (b_ver.Text == "Ver resultados")
  {
    b_ver.Text = "Ver gráficos";
    pictureBox1.Hide();
    listBox1.Show();
  }
  else
  {
    b_ver.Text = "Ver resultados";
    pictureBox1.Show();
    listBox1.Hide();
  }
}
```

Botón Guardar Como …:

```
private void b_guardar_como_Click(object sender, EventArgs e)
    {
      CListBox.lb = listBox1;
      CListBox.GuardarComo("c:\\");
    }
```

Botón Limpiar:

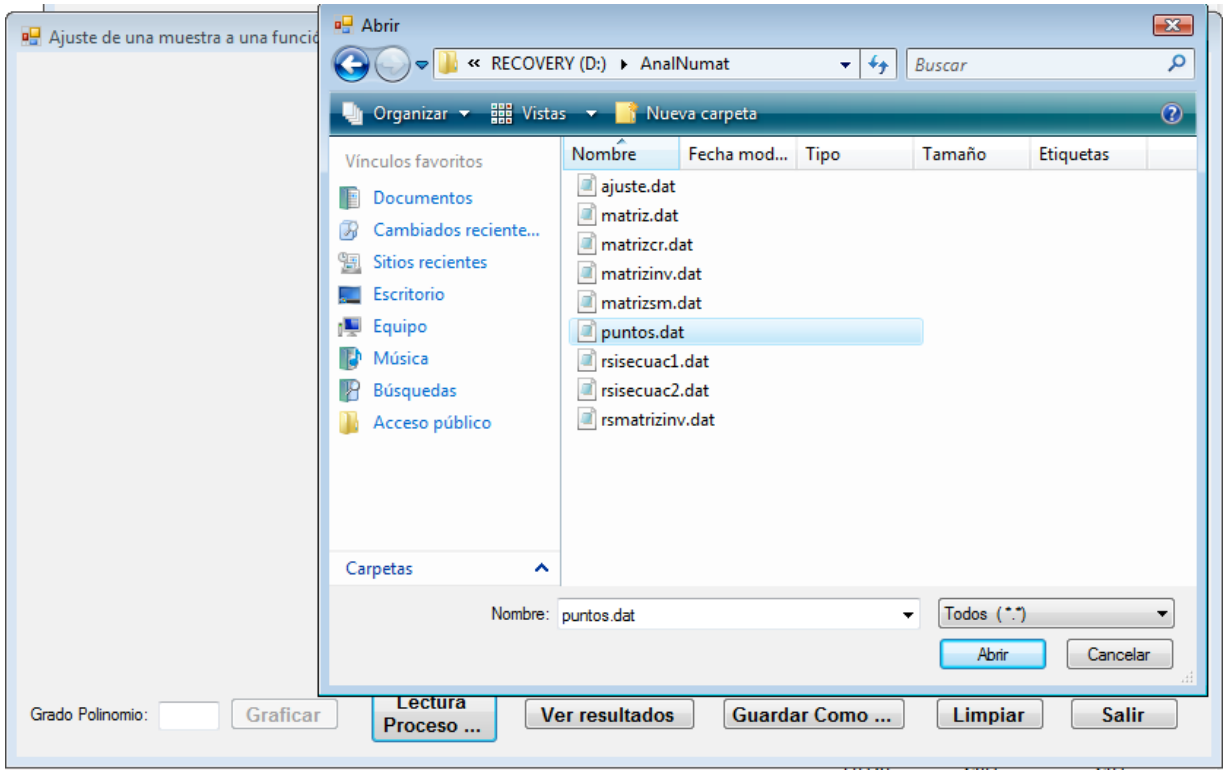
```
private void b_limpiar_Click(object sender, EventArgs e)
    {
      CListBox.lb = listBox1;
      CListBox.Limpiar();
    }
```

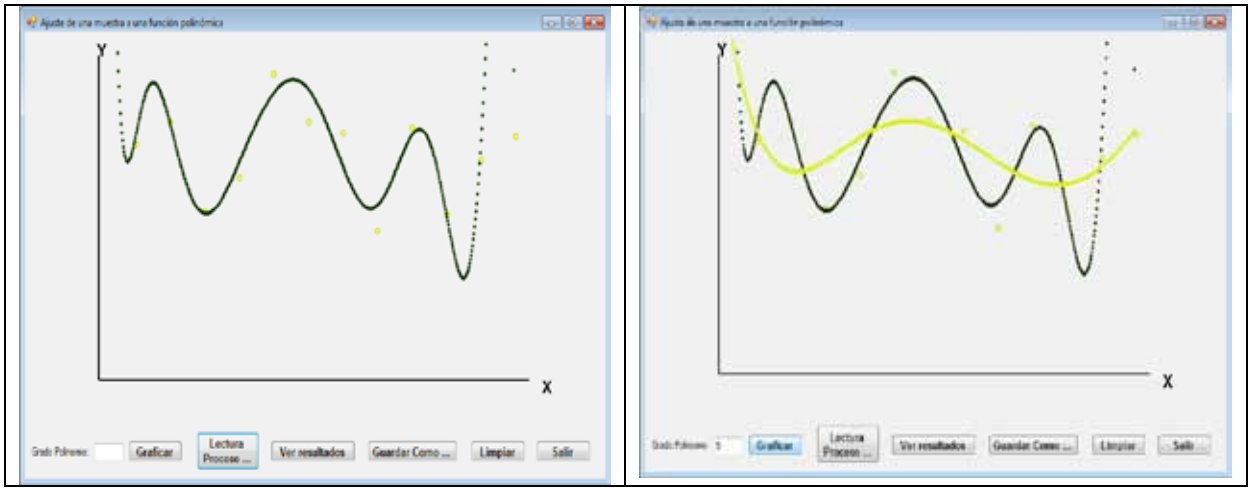
Botón Salir:

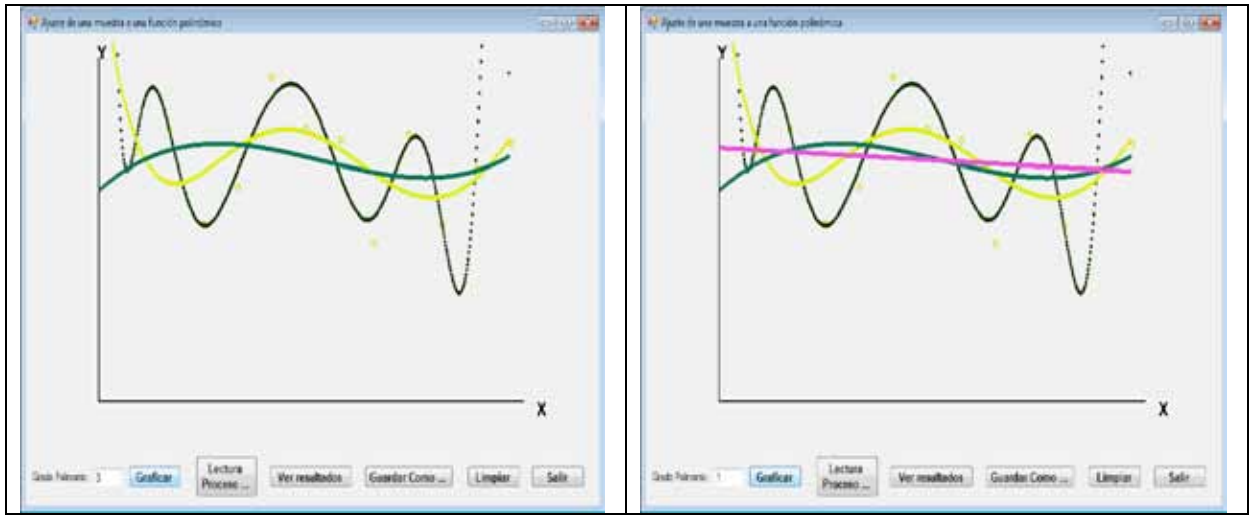
```
private void b_salir_Click(object sender, EventArgs e)
    {
      this.Close();
      this.Dispose();
    }
```

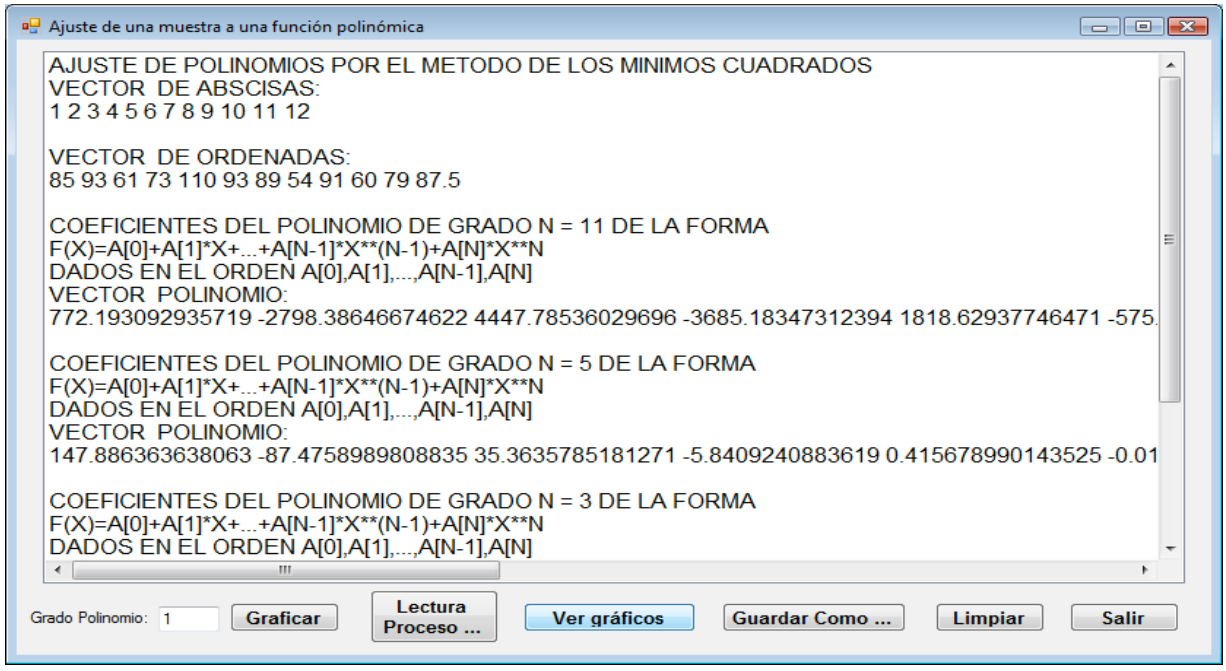
Las referencias a las clases CGenMatrizYVector, CEntSalArchivos, CResolSisSimetricosEcuaciones, CLectEscrMatrices, CHorner y CListBox, provienen de la definición de las respectivas clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

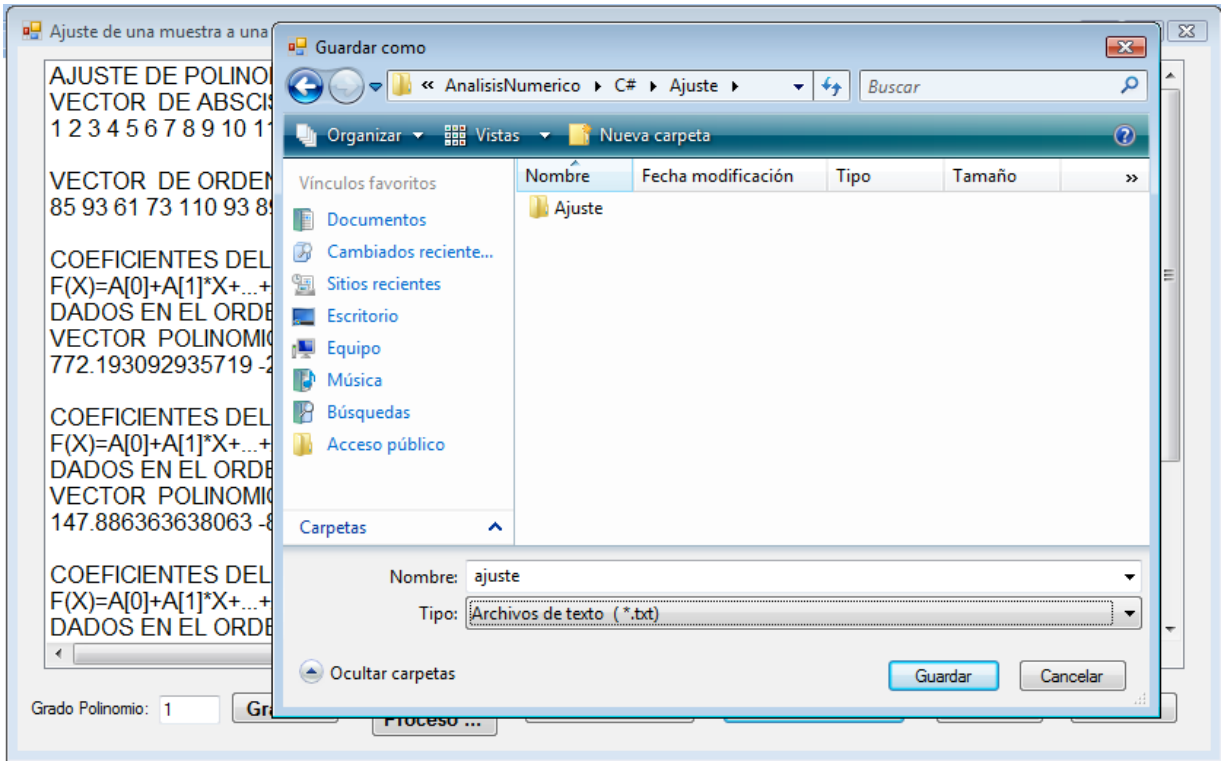
Se presenta a continuación los resultados que produce la ejecución de este programa:


Ajuste de una muestra a una funci
Abrir
RECOVERY (D:) ▸ AnalNumat
Buscar
Organizar
Vistas
Nueva carpeta
Vínculos favoritos
Documentos
Cambiados reciente...
Sitios recientes
Escritorio
Equipo
Música
Búsquedas
Acceso público
Nombre
Fecha mod...
Tipo
Tamaño
Etiquetas
ajuste.dat
matriz.dat
matrizcr.dat
matrizinv.dat
matrizsm.dat
puntos.dat
rsisecuac1.dat
rsisecuac2.dat
rsmatrizinv.dat
Carpetas
Nombre: puntos.dat
Todos (*.*)
Abrir
Cancelar
Grado Polinomio:
Graficar
Lectura Proceso ...
Ver resultados
Guardar Como ...
Limpiar
Salir


Y
X
Graficar
Lectura Proceso ...
Ver resultados
Guardar Como ...
Limpiar
Salir
Y
X


Y
X
Graficar
Lectura Proceso ...
Ver resultados
Guardar Como ...
Limpiar
Salir


Ajuste de una muestra a una función polinómica
AJUSTE DE POLINOMIOS POR EL METODO DE LOS MINIMOS CUADRADOS
VECTOR DE ABSCISAS:
1 2 3 4 5 6 7 8 9 10 11 12
VECTOR DE ORDENADAS:
85 93 61 73 110 93 89 54 91 60 79 87.5
COEFICIENTES DEL POLINOMIO DE GRADO N = 11 DE LA FORMA
F(X)=A[0]+A[1]*X+...+A[N-1]*X**(N-1)+A[N]*X**N
DADOS EN EL ORDEN A[0],A[1],...,A[N-1],A[N]
VECTOR POLINOMIO:
772.193092935719 -2798.38646674622 4447.78536029696 -3685.18347312394 1818.62937746471 -575.
COEFICIENTES DEL POLINOMIO DE GRADO N = 5 DE LA FORMA
F(X)=A[0]+A[1]*X+...+A[N-1]*X**(N-1)+A[N]*X**N
DADOS EN EL ORDEN A[0],A[1],...,A[N-1],A[N]
VECTOR POLINOMIO:
147.886363638063 -87.4758989808835 35.3635785181271 -5.8409240883619 0.415678990143525 -0.01
COEFICIENTES DEL POLINOMIO DE GRADO N = 3 DE LA FORMA
F(X)=A[0]+A[1]*X+...+A[N-1]*X**(N-1)+A[N]*X**N
DADOS EN EL ORDEN A[0],A[1],...,A[N-1],A[N]
Grado Polinomio:
1
Graficar
Lectura Proceso ...
Ver gráficos
Guardar Como ...
Limpiar
Salir



Ajuste.txt:

AJUSTE DE POLINOMIOS POR EL METODO DE LOS MINIMOS CUADRADOS
VECTOR DE ABSCISAS:
1 2 3 4 5 6 7 8 9 10 11 12

VECTOR DE ORDENADAS:
85 93 61 73 110 93 89 54 91 60 79 87.5

COEFICIENTES DEL POLINOMIO DE GRADO N = 11 DE LA FORMA
F(X)=A[0]+A[1]*X+...+A[N-1]*X**(N-1)+A[N]*X**N
DADOS EN EL ORDEN A[0],A[1],...,A[N-1],A[N]
VECTOR POLINOMIO:
772.193092935719 -2798.38646674622 4447.78536029696 -3685.18347312394 1818.62937746471 -575.887139274014 121.605801964184 -17.3267057107536 1.64470659239433 -0.0995500372432104 0.00347066808153631 -5.29594954157183E-05

COEFICIENTES DEL POLINOMIO DE GRADO N = 5 DE LA FORMA
F(X)=A[0]+A[1]*X+...+A[N-1]*X**(N-1)+A[N]*X**N
DADOS EN EL ORDEN A[0],A[1],...,A[N-1],A[N]
VECTOR POLINOMIO:
147.886363638063 -87.4758989808835 35.3635785181271 -5.8409240883619 0.415678990143525 -0.0105627828059102

```
COEFICIENTES DEL POLINOMIO DE GRADO N = 3 DE LA FORMA
F(X)=A[0]+A[1]*X+...+A[N-1]*X**(N-1)+A[N]*X**N
DADOS EN EL ORDEN A[0],A[1],...,A[N-1],A[N]
VECTOR  POLINOMIO:
71.5909090909072 10.1212953712967 -1.99000999001023 0.101981351981364

COEFICIENTES DEL POLINOMIO DE GRADO N = 1 DE LA FORMA
F(X)=A[0]+A[1]*X+...+A[N-1]*X**(N-1)+A[N]*X**N
DADOS EN EL ORDEN A[0],A[1],...,A[N-1],A[N]
VECTOR  POLINOMIO:
85.5530303030303 -0.655594405594406
```

AJUSTE DE UNA CURVA POR INTERPOLACION MEDIANTE LOS POLINOMIOS DE LAGRANGE

Introducción:

En esta sección se presentan dos procesos para ajustar curvas por interpolación mediante Polinomios de Lagrange: uno para funciones discretas, es decir: la muestra se encuentra conformada por un conjunto de puntos y se desconoce como se encuentra definida la función.
En el segundo se tiene una función continua, definida en un dominio dado por los extremos de las abscisas una funcíon fit, la cual recibe como entrada la abscisa *x*; y devuelve como resultado la ordenada *y*. Para este caso los puntos se generan, dividiendo el intervalo horizontal, en abscisas equidistantes. Para cualquiera de los dos procesos, la curva ajustada, se la obtiene al dividir el intervalo, en segmentos pequeños para las a*b*scisas y llamar a la función para interpolar mediante Polinomios de Lagrange (ver capítulo IV) a fin de obtener la ordenada. Para efectos de implementación, se ha tomado para la función discreta, los puntos de la muestra cuya curva es ajustada por el método de los mínimos cuadrados; y para la función continua a: $y = f(x) = e^{x}\cos(2x)$.

Implementación de los algoritmos en MATLAB y VISUAL C#:

Para Matlab, se elabora un programa, desde el cual se ingresan un indicador de si se va a procesar una función discreta (1) o una función continua (0); en el caso de que el indicador sea 1, la muestra de puntos, se definen en el código fuente del programa

y se grafican, seguidamente, se generan y grafican con línea entrecortada los puntos cuyas ordenadas se calculan al llamar a la función que interpola mediante los Polinomios de Lagrange. En el caso de que el indicador sea 0, se debe ingresar el número de puntos y el intervalo en el que se debe graficar la función real, luego se divide en base a los datos ingresados en abscisas equidistantes y se calculan las ordenadas, empleando la función real y = f(x); se grafican estos puntos; seguidamente, se generan y grafican con línea entrecortada los puntos cuyas ordenadas se calculan al llamar a la función que interpola mediante los Polinomios de Lagrange y con línea continua los puntos cuyas ordenadas se calculan con la fórmula y = f(x). El proceso para Visual C#, se describe más adelante.

Archivos fuente en MATLAB:

```
% AjusLag.m
% FUNCION PRINCIPAL
% Realiza el ajuste de curvas usando la interpolación de Lagrange
clc; % Limpia pantalla
while 1
  clear; % Libera espacio de variables
  disp('Ajuste de curvas por interpolación mediante Polinomios de Lagrange');
  fdis = input('Ingresar 1 = función discreta, 0 = función continua, otro valor salir : ');
  if (fdis ~= 0 & fdis ~= 1) break; end;
  if fdis == 1
     vx = [1 2 3 4 5 6 7 8 9 10 11 12];
     vy = [85 93 61 73 110 93 89 54 91 60 79 87.5];
     n = length(vx);
  else
    while 1
       n = input('Ingresar el número de puntos (> 1): ');
       if n > 1 break; end;
    end;
    vx(1) = input('Ingresar el límite inicial: ');
    while 1
      lf = input(sprintf('Ingresar el límite final (> %f): ', vx(1)));
       if lf > vx(1) break; end;
    end;
    delta = (lf - vx(1)) / (n - 1);
    for i = 2 : n - 1
       vx(i) = vx(i - 1) + delta;
       vy(i) = fit(vx(i));
    end;
    vx(n) = lf;
    vy(1) = fit(vx(1));
    vy(n) = fit(vx(n));
  end
  colr = ['m', 'r', 'g', 'y', 'b', 'c'];
  pos = 0;
  % Grafica la muestra:
  plot(vx, vy, '*');
  hold on; % retiene gráfico
  % Generación de los puntos por interpolación:
  xc(1) = vx(1);
  yc(1) = InterpolacionLagrange(vx, vy, xc(1), n);
  ni = 100;
  delta = (vx(n) - vx(1)) / ni;
  for i = 2 : ni + 1
    xc(i) = xc(i - 1) + delta;
    yc(i) = InterpolacionLagrange(vx, vy, xc(i), n);
```

```
  end;
  % Grafica el polinomio de interpolación:
  plot(xc, yc, 'm:');
  pos = pos + 1;
  if fdis ~= 1
    % Generación de los puntos de la curva real:
    xr(1) = vx(1);
    yr(1) =  fit(xr(1));
    delta = (vx(n) - vx(1)) / ni;
    for i = 2 : ni + 1
      xr(i) = xr(i - 1) + delta;
      yr(i) =  fit(xr(i));
    end;
    % Grafica la función real:
    plot(xr, yr, colr(mod(pos, 6) + 1));
    pos = pos + 1;
  end;
  xlabel('Ajuste de la curva mediante interpolación por Polinomios de Lagrange');
  hold off;
end;
```

```
% fit.m
function y = fit(x)
y = exp(x) * cos(2 * x);
return
```

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa.,

```
Command Window
Ajuste de curvas por interpolación mediante Polinomios de Lagrange
Ingresar 1 = función discreta, 0 = función continua, otro valor salir : 1
Ajuste de curvas por interpolación mediante Polinomios de Lagrange
Ingresar 1 = función discreta, 0 = función continua, otro valor salir : 0
Ingresar el número de puntos (> 1): 12
Ingresar el límite inicial: -6.28
Ingresar el límite final (> -6.280000): 6.28
Ajuste de curvas por interpolación mediante Polinomios de Lagrange
Ingresar 1 = función discreta, 0 = función continua, otro valor salir : 0
Ingresar el número de puntos (> 1): 5
Ingresar el límite inicial: -6.28
Ingresar el límite final (> -6.280000): 6.28
Ajuste de curvas por interpolación mediante Polinomios de Lagrange
Ingresar 1 = función discreta, 0 = función continua, otro valor salir : -1
>>
```

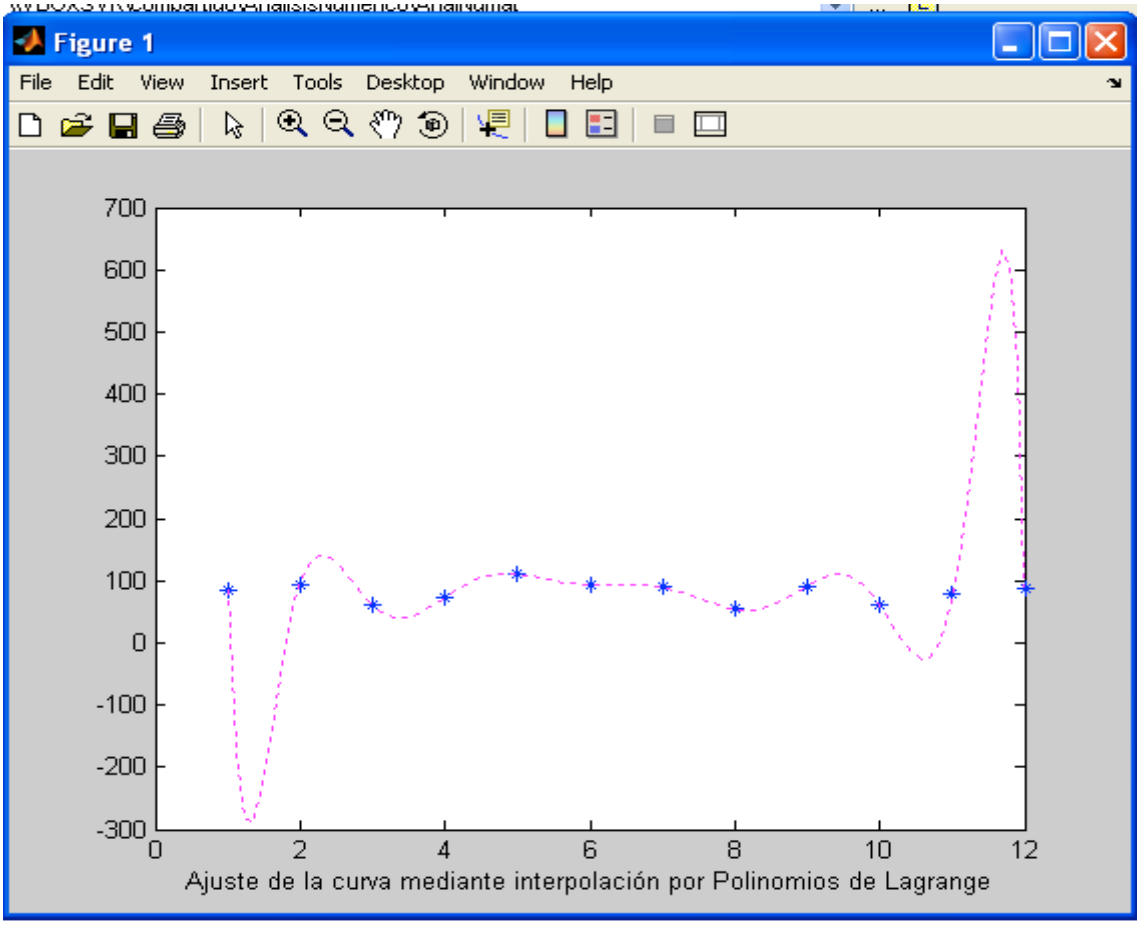


Figure 1
File Edit View Insert Tools Desktop Window Help
700
600
500
400
300
200
100
0
-100
-200
-300
0 2 4 6 8 10 12
Ajuste de la curva mediante interpolación por Polinomios de Lagrange

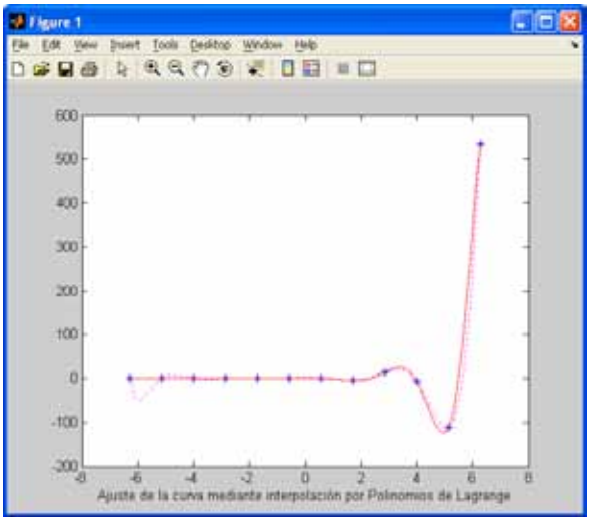


Figure 1
Ajuste de la curva mediante interpolación por Polinomios de Lagrange

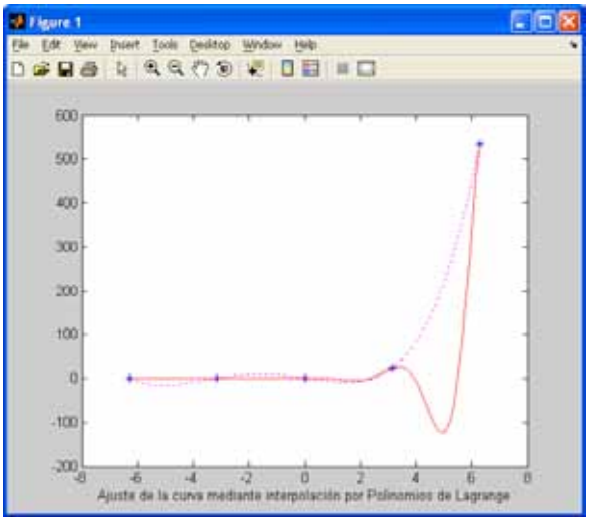


Figure 1
Ajuste de la curva mediante interpolación por Polinomios de Lagrange

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 5.3 Como se puede apreciar, existen dos botones de opción para seleccionar si la función es continua o discreta; tres cuadros de texto, mismos que permite ingresar el número de puntos y el intervalo en el que se grafica la curva; un cuadro de imagen, el cual desplegará los gráficos; y dos botones, mismos que tienen la funcionalidad:

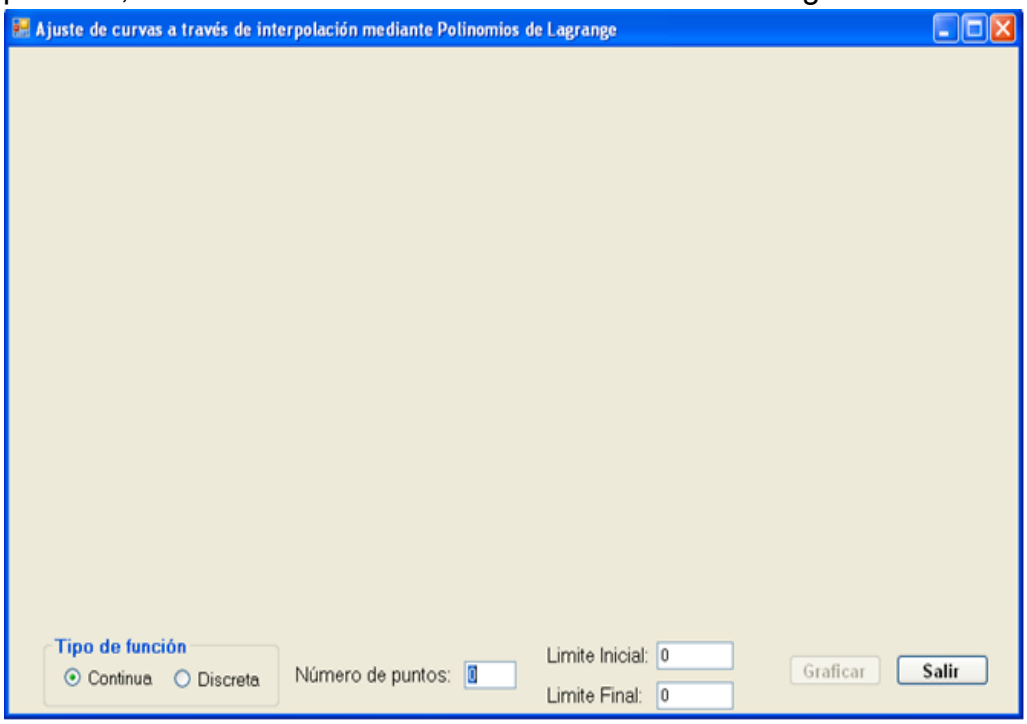


Figura 5.3 Ventana para ajustar curvas por Interpolación a través de Polinomios de Lagrange en Visual C#

- Graficar: en el caso de seleccionar la opción Continua, valida el número de puntos y el intervalo, luego se divide en base a los datos ingresados en abscisas equidistantes y se calculan las ordenadas, empleando la función real y = f(x). En el caso de seleccionar la opción Discreta, la muestra de puntos, se definen en el código fuente del programa; para cualquiera de los dos casos, se grafican, los puntos concírculos visibles; seguidamente, se generan y grafican con línea delgada los puntos cuyas ordenadas se calculan al llamar a la función que interpola mediante los Polinomios de Lagrange; a continuación, si la función es continua, se generan y grafican con línea gruesa los puntos cuyas ordenadas se calculan al llamar a la función y = f(x).
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Variables y funciones miembro de la clase Form1:

```
// Datos miembro:
private static double[] vdx = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
private static double[] vdy = { 85, 93, 61, 73, 110, 93, 89, 54, 91, 60, 79, 87.5 };
private static int tf = 30;
Random r = new Random(DateTime.Now.Millisecond);
private Bitmap offScreenBitMap = null;
```

```
double EH, EV, MAXX, MAXY, MINX, MINY, X0, X1, Y0, Y1;
double MAXPX, MAXPY, CX, CY;
int n;

// Funciones miembro:
public Form1()
{
    InitializeComponent();
}

private void MaxMin (double [] v, int n, ref double max, ref double min)
{
   max = v[0];
   min = v[0];
   for (int i = 1; i < n; i++)
   {
      if (max < v[i]) max = v[i];
      if (min > v[i]) min = v[i];
   }
}

// Funcion a ser analizada:
static double fit(double x)
{
   return Math.Exp(x) * Math.Cos(2 * x);
}

private void calculaGraficaPolinomio(bool real)
{
   int i;
   X1 = X0;
   int red = r.Next(255);
   int green = r.Next(255);
   int blue = r.Next(255);
   Graphics g = Graphics.FromImage(offScreenBitMap);
   Pen p = new Pen(Color.FromArgb(red, green, blue), 2);
   if (real)
      for (i = 0; i < MAXPX; i++)
      {
         g.DrawEllipse(p, (float)(Math.Round((X1 - X0) * EH) + CX), (float)(2 * MAXPY - Math.Round(EV *
(fit(X1) - Y0)) + CY), 3, 3);
         X1 = X1 + 1 / EH;
      }
   else
      for (i = 0; i < MAXPX; i++)
      {
          g.DrawEllipse(p, (float)(Math.Round((X1 - X0) * EH) + CX), (float)(2 * MAXPY - Math.Round(EV
* (CInterpolacion.InterpolacionLagrange(X1, n) - Y0)) + CY), 1, 1);
          X1 = X1 + 1 / EH;
      }
   pictureBox1.Invalidate();
   p.Dispose();
   g.Dispose();
}
```

Evento Paint del objeto pictureBox1:

```
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
  if (offScreenBitMap == null)
  {
    return; // Nothing to do
  }
  else
  {
    e.Graphics.DrawImage(offScreenBitMap, new Point(0, 0));
  }
}
```

Evento Load del objeto Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
  CInterpolacion.AsignaMemoria(false, tf);
}
```

Evento TextChanged del objeto txtNPuntos:

```
private void txtNPuntos_TextChanged(object sender, EventArgs e)
{
   b_Graficar.Enabled = false;
   try
   {
     int np = Int32.Parse(txtNPuntos.Text);
     if (np > 1)
       b_Graficar.Enabled = true;
   }
   catch (Exception ex)
   {
     MessageBox.Show(ex.Message);
   }
}
```

Opción Continua:

```
private void rbContinua_CheckedChanged(object sender, EventArgs e)
{
   label1.Show();
   txtNPuntos.Show();
   txtLIni.Show();
   txtLFin.Show();
   b_Graficar.Enabled = false;
   try
   {
```

```
        int np = Int32.Parse(txtNPuntos.Text);
        if (np > 1)
            b_Graficar.Enabled = true;
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

Opción Discreta:

```
private void rbDiscreta_CheckedChanged(object sender, EventArgs e)
{
    label1.Hide();
    txtNPuntos.Hide();
    txtLIni.Hide();
    txtLFin.Hide();
    b_Graficar.Enabled = true;
}
```

Botón Graficar:

```
private void b_Graficar_Click(object sender, EventArgs e)
{
   offScreenBitMap = null;
   try
   {
      if (rbDiscreta.Checked) {
         n = vdx.Length;
         for (int i = 0; i < n; i++)
         {
            CInterpolacion.vx[i] = vdx[i];
            CInterpolacion.vy[i] = vdy[i];
         }
      }
      else
      {
         double h, li, lf;
         n = Int32.Parse(txtNPuntos.Text);
         li = double.Parse(txtLIni.Text);
         lf = double.Parse(txtLFin.Text);
         if (li >= lf) return;
         h = (lf - li) / (n - 1);
         CInterpolacion.vx[0] = li;
         CInterpolacion.vy[0] = fit(CInterpolacion.vx[0]);
         for (int i = 1; i < n; i++)
         {
            CInterpolacion.vx[i] = CInterpolacion.vx[i - 1] + h;
            CInterpolacion.vy[i] = fit(CInterpolacion.vx[i]);
         }
      }
      // Calcular las constantes de graficación:
      if (offScreenBitMap == null)
```

```
{
    offScreenBitMap = new Bitmap(pictureBox1.Width, pictureBox1.Height);
}
Graphics g = Graphics.FromImage(offScreenBitMap);
Pen p = new Pen(Color.FromArgb(0, 0, 0), 2);
MAXPX = Math.Round(0.75 * pictureBox1.Width, 0);
MAXPY = Math.Round(0.40 * pictureBox1.Height, 0);
CX = Math.Round(0.11 * pictureBox1.Width, 0);
CY = Math.Round(0.08 * pictureBox1.Height, 0);
/* CALCULA EL MAXIMO Y MINIMO DE LAS COORDENADAS DE LOS PUNTOS */
MaxMin(CInterpolacion.vx, n, ref MAXX, ref MINX);
MaxMin(CInterpolacion.vy, n, ref MAXY, ref MINY);
/* CALCULA ESCALAS Y PUNTOS INICIALES DE REFERENCIA */
EH = MAXPX / (MAXX - MINX);
EV = 2 * MAXPY / (MAXY - MINY);
X0 = 0;
Y0 = 0;
X1 = MAXX;
Y1 = MAXY;
if (MAXX < 0)
{
    EH = MAXPX / Math.Abs(MINX);
    X0 = MINX;
    X1 = 0;
}
else if (MINX >= 0)
   EH = MAXPX / MAXX;
else
{
   X0 = MINX;
   X1 = MAXX;
}
if (MAXY < 0)
{
    EV = 2 * MAXPY / Math.Abs(MINY);
    Y0 = MINY;
    Y1 = 0;
}
else if (MINY >= 0)
   EV = 2 * MAXPY / MAXY;
else
{
   Y0 = MINY;
   Y1 = MAXY;
}
/* DIBUJAR LOS EJES DEL SISTEMA DE REFERENCIA */
g.DrawLine(p,(float)CX,(float)(Math.Round(2*MAXPY+Y0*EV) + CY),
    (float)(Math.Round((X1 - X0) * EH) + Math.Round(1.25 * CX)),
    (float)(Math.Round(2 * MAXPY + Y0 * EV) + CY));
Font df = new Font("Arial", 14);
SolidBrush db = new SolidBrush(Color.Black);
g.DrawString("X", df, db, (float)(Math.Round((X1 - X0) * EH) +
    Math.Round(1.4375 * CX)),
   (float)(Math.Round(2 * MAXPY + Y0 * EV) + CY - 3));
g.DrawLine(p, (float)(Math.Round(-X0 * EH) + CX),
```

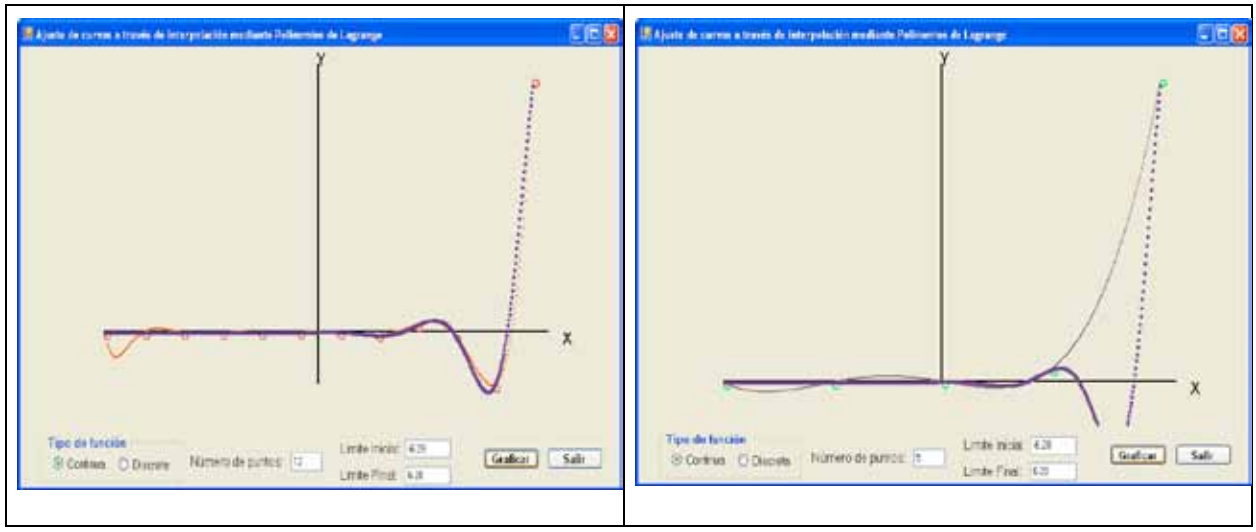
```
            (float)(Math.Round(2 * MAXPY) + CY),
            (float)(Math.Round(-X0 * EH) + CX),
            (float)(Math.Round(2*MAXPY-(Y1-Y0) * EV) + Math.Round(0.5 * CY)));
        g.DrawString("Y", df, db, (float)(Math.Round(-X0 * EH) +
            Math.Round(0.9375 * CX)),
            (float)(Math.Round(2 * MAXPY - (Y1 - Y0) * EV)));
        /* DIBUJAR LOS PUNTOS */
        p.Dispose();
        int red = r.Next(255);
        int green = r.Next(255);
        int blue = r.Next(255);
        p = new Pen(Color.FromArgb(red, green, blue), 2);
        for (int i = 0; i < n; i++)
          g.DrawEllipse(p, (float)(Math.Round(EH * (CInterpolacion.vx[i] -
              X0)) + CX),
              (float)(Math.Round(2 * MAXPY - EV * (CInterpolacion.vy[i] - Y0))
              + CY), 8, 8);
        p.Dispose();
        g.Dispose();
        calculaGraficaPolinomio(false);
        if (rbContinua.Checked)
            calculaGraficaPolinomio(true);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

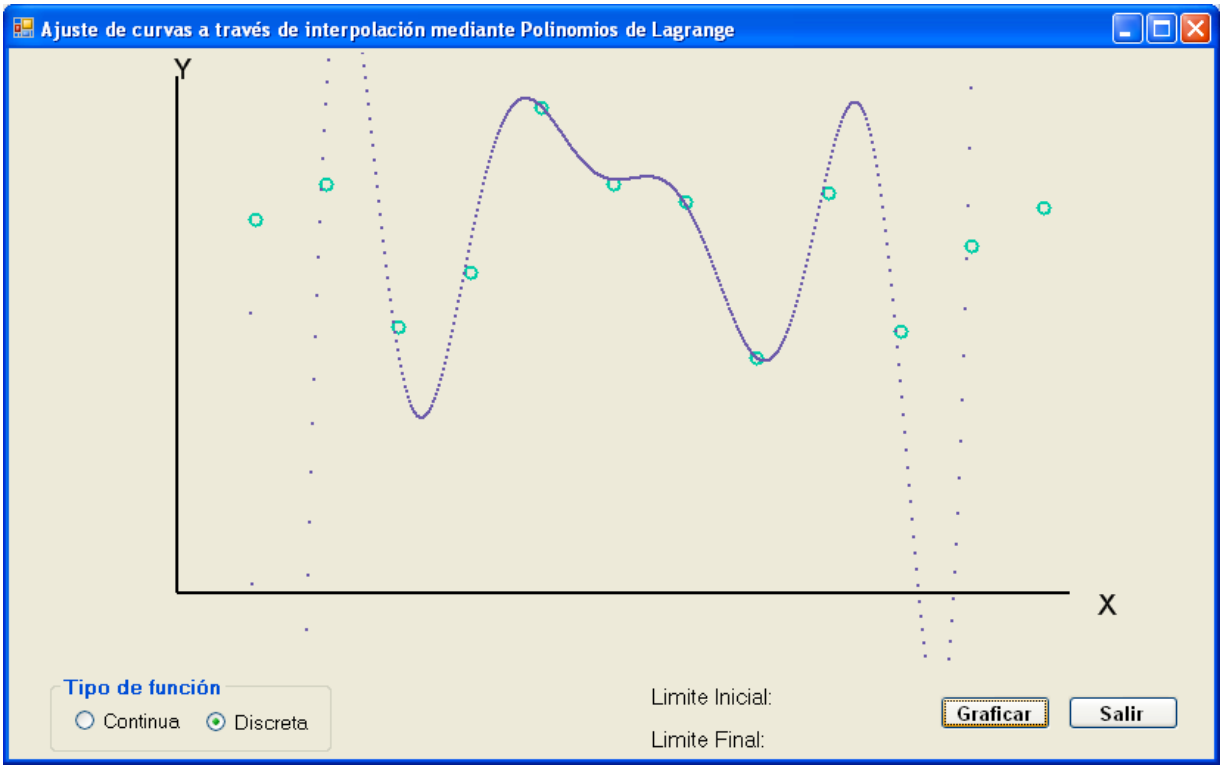
Botón Salir:

```
private void b_salir_Click(object sender, EventArgs e)
{
    this.Close();
    this.Dispose();
}
```

Las referencias a la clase CInterpolación, proviene de la definición de la respectiva clase incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:




Ajuste de curvas a través de interpolación mediante Polinomios de Lagrange
Y
X
Tipo de función
Continua
Discreta
Limite Inicial:
Limite Final:
Graficar
Salir

AJUSTE DE UNA CURVA POR INTERPOLACION MEDIANTE DIFERENCIAS DIVIDIDAS DE NEWTON

Introducción:

En esta sección se presentan dos procesos para ajustar curvas por interpolación mediante Diferencias Divididas de Newton: uno para funciones discretas, es decir: la muestra se encuentra conformada por un conjunto de puntos y se desconoce como se encuentra definida la función.
En el segundo se tiene una función continua, definida en un dominio dado por los extremos de las abscisas una funcíon f(x) , la cual recibe como entrada la abscisa *x*; y devuelve como resultado la ordenada *y*. Para este caso los puntos se generan, dividiendo el intervalo horizontal, en abscisas equidistantes. Para cualquiera de los dos procesos, la curva ajustada, se la obtiene al dividir el intervalo, en segmentos pequeños para las a*b*scisas y llamar a la función para interpolar mediante Diferencias Divididas de Newton (ver capítulo IV) a fin de obtener la ordenada. Para efectos de implementación, se ha tomado para la función discreta, los puntos de la muestra cuya curva es ajustada por el método de los mínimos cuadrados; y para la función continua a: $y = f(x) = e^{x}\cos(2x)$.

Implementación de los algoritmos en MATLAB y VISUAL C#:

Para Matlab, se elabora un programa, desde el cual se ingresan un indicador de si se va a procesar una función discreta (1) o una función continua (0); en el caso de que el indicador sea 1, la muestra de puntos, se definen en el código fuente del programa y se grafican, seguidamente, se generan y grafican con línea entrecortada los puntos cuyas ordenadas se calculan al llamar a la función que interpola mediante las Diferencias Divididas de Newton. En el caso de que el indicador sea 0, se debe ingresar el número de puntos y el intervalo en el que se debe graficar la función real, luego se divide en base a los datos ingresados en abscisas equidistantes y se calculan las ordenadas, empleando la función real y = f(x); se grafican estos puntos; seguidamente, se generan y grafican con línea entrecortada los puntos cuyas ordenadas se calculan al llamar a la función que interpola mediante las Diferencias Divididas de Newton y con línea continua los puntos cuyas ordenadas se calculan con la fórmula y = f(x). El proceso para Visual C#, se describe más adelante.

Archivos fuente en MATLAB:

```
% AjusNewton.m
% FUNCION PRINCIPAL
% Realiza el ajuste de curvas usando la interpolación de Newton
clc; % Limpia pantalla
while 1
```

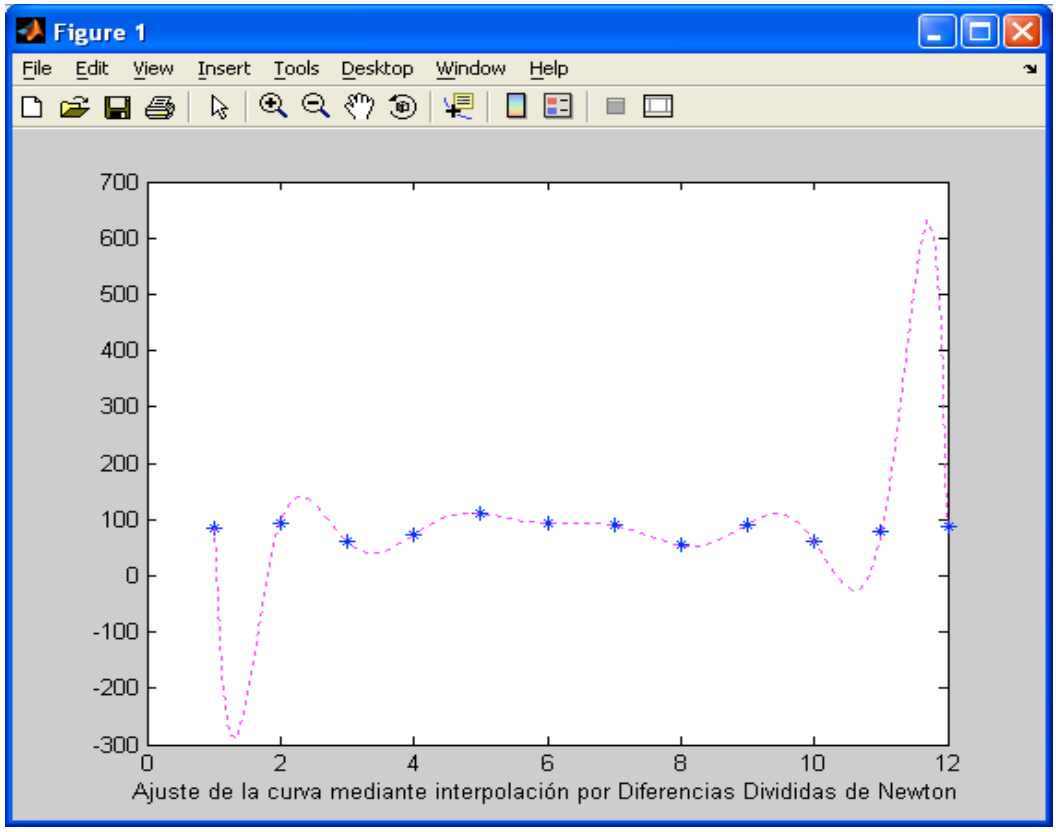
```
clear; % Libera espacio de variables
disp('Ajuste de curvas por interpolación mediante diferencias divididas de Newton');
fdis = input('Ingresar 1 = función discreta, 0 = función continua, otro valor salir : ');
if (fdis ~= 0 & fdis ~= 1) break; end;
if fdis == 1
   vx = [1 2 3 4 5 6 7 8 9 10 11 12];
   vy = [85 93 61 73 110 93 89 54 91 60 79 87.5];
   n = length(vx);
else
  while 1
     n = input('Ingresar el número de puntos (> 1): ');
     if n > 1 break; end;
  end;
  vx(1) = input('Ingresar el límite inicial: ');
  while 1
    lf = input(sprintf('Ingresar el límite final (> %f): ', vx(1)));
     if lf > vx(1) break; end;
  end;
  delta = (lf - vx(1)) / (n - 1);
  for i = 2 : n - 1
     vx(i) = vx(i - 1) + delta;
     vy(i) = fit(vx(i));
  end;
  vx(n) = lf;
  vy(1) = fit(vx(1));
  vy(n) = fit(vx(n));
end
colr = ['m', 'r', 'g', 'y', 'b', 'c'];
pos = 0;
% Grafica la muestra:
plot(vx, vy, '*');
hold on; % retiene gráfico
% Generación de los puntos por interpolación:
xc(1) = vx(1);
yc(1) = InterpolacionNewton(vx, vy, xc(1), n);
ni = 100;
delta = (vx(n) - vx(1)) / ni;
for i = 2 : ni + 1
  xc(i) = xc(i - 1) + delta;
  yc(i) = InterpolacionNewton(vx, vy, xc(i), n);
end;
% Grafica el polinomio de interpolación:
plot(xc, yc, 'm:');
pos = pos + 1;
if fdis ~= 1
  % Generación de los puntos de la curva real:
  xr(1) = vx(1);
  yr(1) =  fit(xr(1));
  delta = (vx(n) - vx(1)) / ni;
  for i = 2 : ni + 1
     xr(i) = xr(i - 1) + delta;
     yr(i) =  fit(xr(i));
  end;
  % Grafica la función real:
  plot(xr, yr, colr(mod(pos, 6) + 1));
  pos = pos + 1;
end;
xlabel('Ajuste de la curva mediante interpolación por Diferencias Divididas de Newton');
hold off;
end;
```

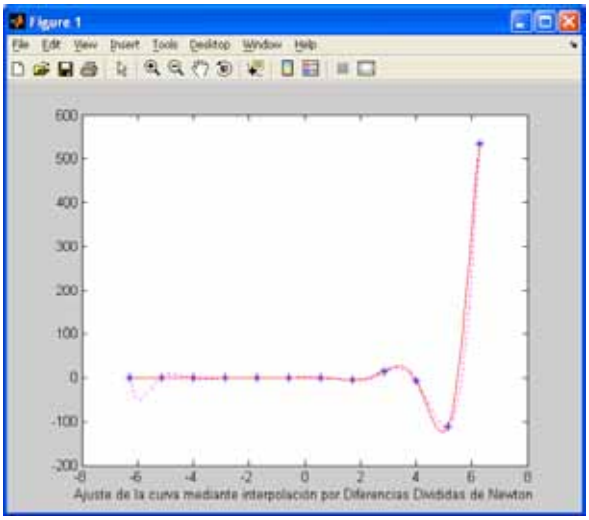
```
% fit.m
function y = fit(x)
y = exp(x) * cos(2 * x);
return
```

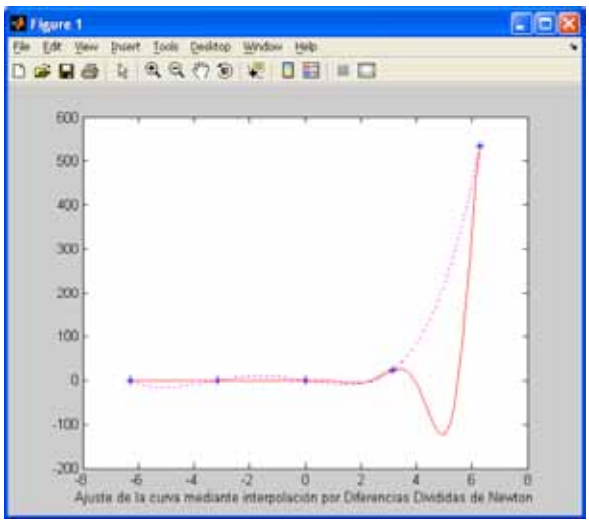
Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa.,

```
Command Window
Ajuste de curvas por interpolación mediante diferencias divididas de Newton
Ingresar 1 = función discreta, 0 = función continua, otro valor salir : 1
Ajuste de curvas por interpolación mediante diferencias divididas de Newton
Ingresar 1 = función discreta, 0 = función continua, otro valor salir : 0
Ingresar el número de puntos (> 1): 12
Ingresar el límite inicial: -6.28
Ingresar el límite final (> -6.280000): 6.28
Ajuste de curvas por interpolación mediante diferencias divididas de Newton
Ingresar 1 = función discreta, 0 = función continua, otro valor salir : 0
Ingresar el número de puntos (> 1): 5
Ingresar el límite inicial: -6.28
Ingresar el límite final (> -6.280000): 6.28
Ajuste de curvas por interpolación mediante diferencias divididas de Newton
Ingresar 1 = función discreta, 0 = función continua, otro valor salir : -1
>>
```







Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 5.4 Como se puede apreciar, existen dos botones de opción para seleccionar si la función es continua o discreta; tres cuadros de texto, mismos que permite ingresar el número de puntos y el intervalo en el que se grafica la curva; un cuadro de imagen, el cual desplegará los gráficos; y dos botones, mismos que tienen la funcionalidad:

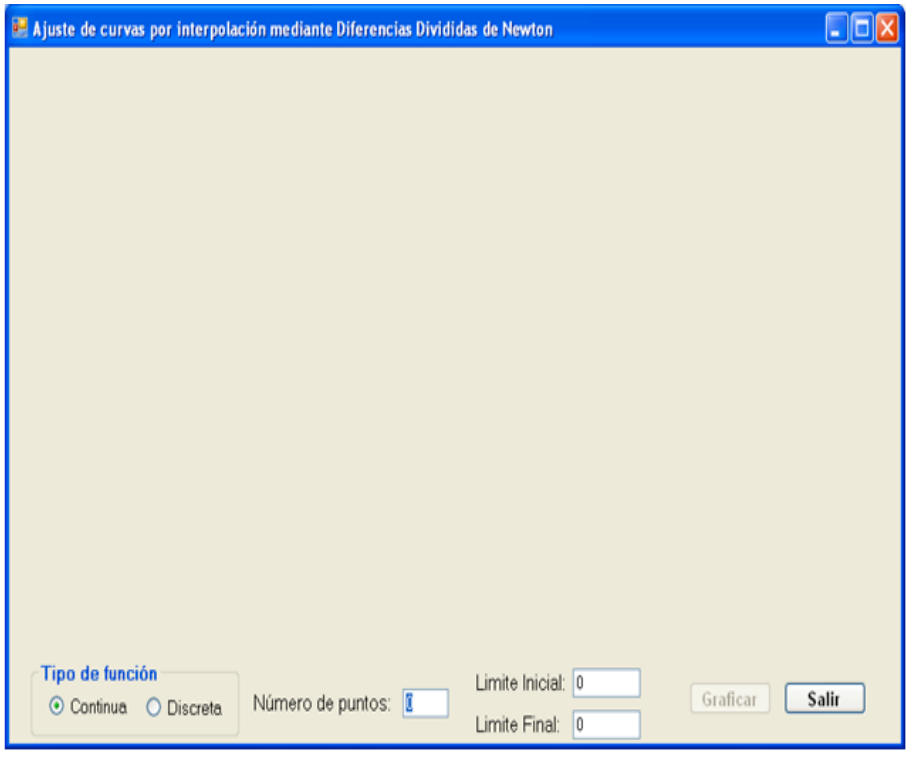


Figura 5.4 Ventana para ajustar curvas por Interpolación a través de Diferencias Divididas de Newton

- Graficar: en el caso de seleccionar la opción Continua, valida el número de puntos y el intervalo, luego se divide en base a los datos ingresados en abscisas equidistantes y se calculan las ordenadas, empleando la función real y = f(x). En el caso de seleccionar la opción Discreta, la muestra de puntos, se definen en el código fuente del programa; para cualquiera de los dos casos, se grafican, los puntos concírculos visibles; seguidamente, se generan y grafican con línea delgada los puntos cuyas ordenadas se calculan al llamar a la función que interpola mediante las Diferencias Divididas de Newton; a continuación, si la función es continua, se generan y grafican con línea gruesa los puntos cuyas ordenadas se calculan al llamar a la función y = f(x).
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Variables y funciones miembro de la clase Form1:

```
// Datos miembro:
private static double[] vdx = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12 };
private static double[] vdy = { 85, 93, 61, 73, 110, 93, 89, 54, 91, 60, 79, 87.5 };
private static int tf = 30;
Random r = new Random(DateTime.Now.Millisecond);
private Bitmap offScreenBitMap = null;
double EH, EV, MAXX, MAXY, MINX, MINY, X0, X1, Y0, Y1;
double MAXPX, MAXPY, CX, CY;
int n;

// Funciones miembro:
public Form1()
{
    InitializeComponent();
}

private void MaxMin (double [] v, int n, ref double max, ref double min)
{
   max = v[0];
   min = v[0];
   for (int i = 1; i < n; i++)
   {
      if (max < v[i]) max = v[i];
      if (min > v[i]) min = v[i];
   }
}

// Funcion a ser analizada:
static double fit(double x)
{
   return Math.Exp(x) * Math.Cos(2 * x);
}

private void calculaGraficaPolinomio(bool real)
{
   int i;
   X1 = X0;
```

```
  int red = r.Next(255);
  int green = r.Next(255);
  int blue = r.Next(255);
  Graphics g = Graphics.FromImage(offScreenBitMap);
  Pen p = new Pen(Color.FromArgb(red, green, blue), 2);
  if (real)
    for (i = 0; i < MAXPX; i++)
    {
      g.DrawEllipse(p, (float)(Math.Round((X1 - X0) * EH) + CX), (float)(2 * MAXPY - Math.Round(EV *
(fit(X1) - Y0)) + CY), 3, 3);
      X1 = X1 + 1 / EH;
    }
  else
    for (i = 0; i < MAXPX; i++)
    {
      g.DrawEllipse(p, (float)(Math.Round((X1 - X0) * EH) + CX), (float)(2 * MAXPY - Math.Round(EV
* (CInterpolacion.InterpolacionNewton(X1, n) - Y0)) + CY), 1, 1);
      X1 = X1 + 1 / EH;
    }
  pictureBox1.Invalidate();
  p.Dispose();
  g.Dispose();
}
```

Evento Paint del objeto pictureBox1:

```
private void pictureBox1_Paint(object sender, PaintEventArgs e)
{
  if (offScreenBitMap == null)
  {
    return; // Nothing to do
  }
  else
  {
    e.Graphics.DrawImage(offScreenBitMap, new Point(0, 0));
  }
}
```

Evento Load del objeto Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
  CInterpolacion.AsignaMemoria(true, tf);
}
```

Evento TextChanged del objeto txtNPuntos:

```
private void txtNPuntos_TextChanged(object sender, EventArgs e)
{
```

```
    b_Graficar.Enabled = false;
    try
    {
      int np = Int32.Parse(txtNPuntos.Text);
      if (np > 1)
        b_Graficar.Enabled = true;
    }
    catch (Exception ex)
    {
      MessageBox.Show(ex.Message);
    }
}
```

Opción Continua:

```
private void rbContinua_CheckedChanged(object sender, EventArgs e)
{
    label1.Show();
    txtNPuntos.Show();
    txtLIni.Show();
    txtLFin.Show();
    b_Graficar.Enabled = false;
    try
    {
      int np = Int32.Parse(txtNPuntos.Text);
      if (np > 1)
        b_Graficar.Enabled = true;
    }
    catch (Exception ex)
    {
      MessageBox.Show(ex.Message);
    }
}
```

Opción Discreta:

```
private void rbDiscreta_CheckedChanged(object sender, EventArgs e)
{
    label1.Hide();
    txtNPuntos.Hide();
    txtLIni.Hide();
    txtLFin.Hide();
    b_Graficar.Enabled = true;
}
```

Botón Graficar:

```
private void b_Graficar_Click(object sender, EventArgs e)
{
   offScreenBitMap = null;
   try
   {
      if (rbDiscreta.Checked) {
        n = vdx.Length;
```

```
    for (int i = 0; i < n; i++)
    {
        CInterpolacion.vx[i] = vdx[i];
        CInterpolacion.vy[i] = vdy[i];
    }
}
else
{
    double h, li, lf;
    n = Int32.Parse(txtNPuntos.Text);
    li = double.Parse(txtLIni.Text);
    lf = double.Parse(txtLFin.Text);
    if (li >= lf) return;
    h = (lf - li) / (n - 1);
    CInterpolacion.vx[0] = li;
    CInterpolacion.vy[0] = fit(CInterpolacion.vx[0]);
    for (int i = 1; i < n; i++)
    {
        CInterpolacion.vx[i] = CInterpolacion.vx[i - 1] + h;
        CInterpolacion.vy[i] = fit(CInterpolacion.vx[i]);
    }
}
// Calcular las constantes de graficación:
if (offScreenBitMap == null)
{
    offScreenBitMap = new Bitmap(pictureBox1.Width, pictureBox1.Height);
}
Graphics g = Graphics.FromImage(offScreenBitMap);
Pen p = new Pen(Color.FromArgb(0, 0, 0), 2);
MAXPX = Math.Round(0.75 * pictureBox1.Width, 0);
MAXPY = Math.Round(0.40 * pictureBox1.Height, 0);
CX = Math.Round(0.11 * pictureBox1.Width, 0);
CY = Math.Round(0.08 * pictureBox1.Height, 0);
/* CALCULA EL MAXIMO Y MINIMO DE LAS COORDENADAS DE LOS PUNTOS */
MaxMin(CInterpolacion.vx, n, ref MAXX, ref MINX);
MaxMin(CInterpolacion.vy, n, ref MAXY, ref MINY);
/* CALCULA ESCALAS Y PUNTOS INICIALES DE REFERENCIA */
EH = MAXPX / (MAXX - MINX);
EV = 2 * MAXPY / (MAXY - MINY);
X0 = 0;
Y0 = 0;
X1 = MAXX;
Y1 = MAXY;
if (MAXX < 0)
{
    EH = MAXPX / Math.Abs(MINX);
    X0 = MINX;
    X1 = 0;
}
else if (MINX >= 0)
    EH = MAXPX / MAXX;
else
{
    X0 = MINX;
    X1 = MAXX;
```

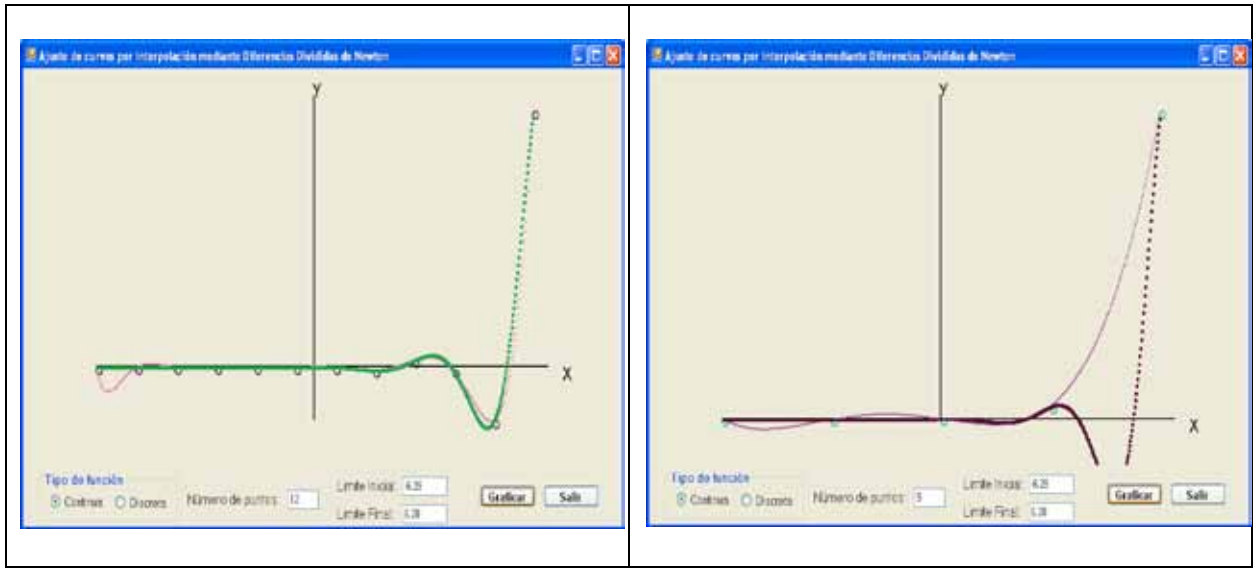
```
        }
        if (MAXY < 0)
        {
            EV = 2 * MAXPY / Math.Abs(MINY);
            Y0 = MINY;
            Y1 = 0;
        }
        else if (MINY >= 0)
           EV = 2 * MAXPY / MAXY;
        else
        {
           Y0 = MINY;
           Y1 = MAXY;
        }
        /* DIBUJAR LOS EJES DEL SISTEMA DE REFERENCIA */
        g.DrawLine(p,(float)CX,(float)(Math.Round(2*MAXPY+Y0*EV) + CY),
            (float)(Math.Round((X1 - X0) * EH) + Math.Round(1.25 * CX)),
            (float)(Math.Round(2 * MAXPY + Y0 * EV) + CY));
        Font df = new Font("Arial", 14);
        SolidBrush db = new SolidBrush(Color.Black);
        g.DrawString("X", df, db, (float)(Math.Round((X1 - X0) * EH) +
            Math.Round(1.4375 * CX)),
           (float)(Math.Round(2 * MAXPY + Y0 * EV) + CY - 3));
        g.DrawLine(p, (float)(Math.Round(-X0 * EH) + CX),
           (float)(Math.Round(2 * MAXPY) + CY),
           (float)(Math.Round(-X0 * EH) + CX),
           (float)(Math.Round(2*MAXPY-(Y1-Y0) * EV) + Math.Round(0.5 * CY)));
        g.DrawString("Y", df, db, (float)(Math.Round(-X0 * EH) +
           Math.Round(0.9375 * CX)),
           (float)(Math.Round(2 * MAXPY - (Y1 - Y0) * EV)));
        /* DIBUJAR LOS PUNTOS */
        p.Dispose();
        int red = r.Next(255);
        int green = r.Next(255);
        int blue = r.Next(255);
        p = new Pen(Color.FromArgb(red, green, blue), 2);
        for (int i = 0; i < n; i++)
          g.DrawEllipse(p, (float)(Math.Round(EH * (CInterpolacion.vx[i] -
             X0)) + CX),
             (float)(Math.Round(2 * MAXPY - EV * (CInterpolacion.vy[i] - Y0))
             + CY), 8, 8);
        p.Dispose();
        g.Dispose();
        calculaGraficaPolinomio(false);
        if (rbContinua.Checked)
           calculaGraficaPolinomio(true);
     }
     catch (Exception ex)
     {
        MessageBox.Show(ex.Message);
     }
  }
```

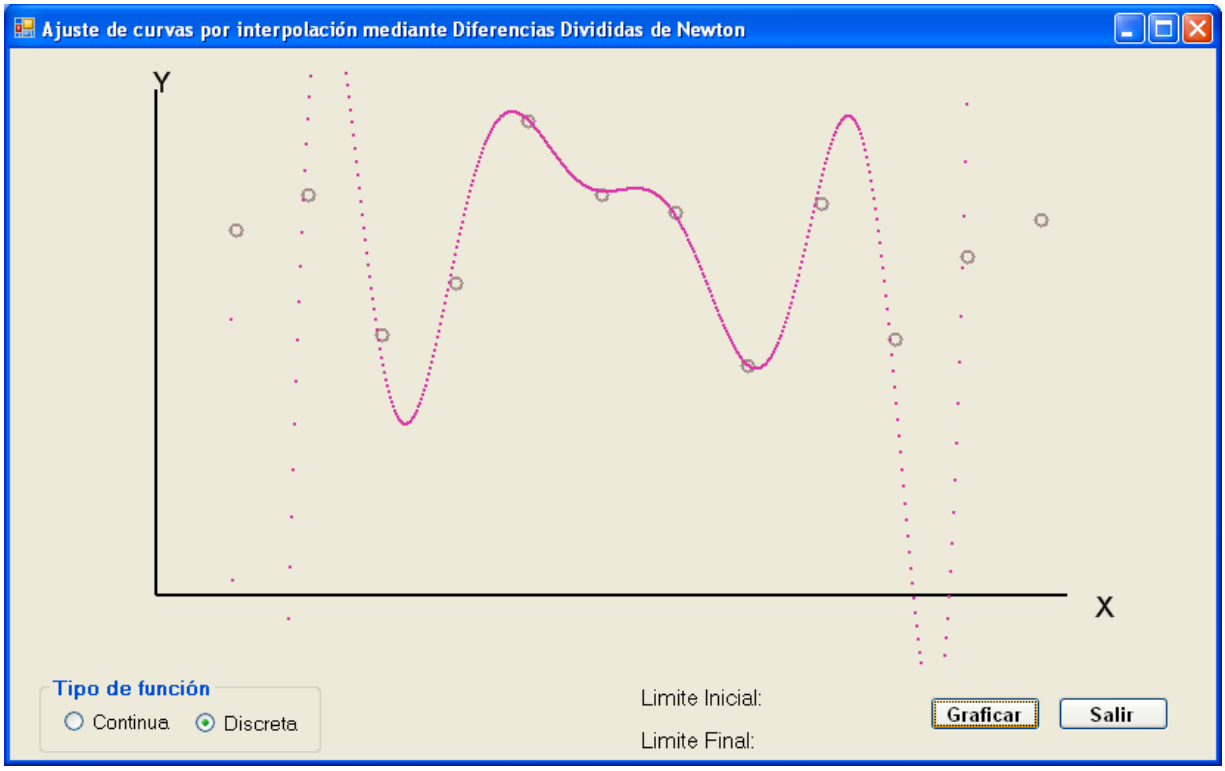
Botón Salir:

```
private void b_salir_Click(object sender, EventArgs e)
{
   this.Close();
   this.Dispose();
}
```

Las referencias a la clase CInterpolación, proviene de la definición de la respectiva clase incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:





Como puede observarse, los resultados obtenidos en el ajuste de curvas por interpolación mediante Polinomios de Lagrange y Diferencias Divididas de Newton, son bastante parecidos. El ajuste a la curva real, tiene mayor precisión, cuando la muestra dispone de mayor cantidad de puntos. Los puntos de la muestra y los de la curva real, forman puntos de intersección.

EJERCICIOS DEL CAPITULO V

1.- Ajustar la muestra del ejemplo 5.2 a un polinomio de sexto grado.

2.- Ejecutar el programa en Matlab y en Visual C# para ajuste de curvas por interpolación mediante Polinomios de Lagrange usando la función: f(x) = 2cos(3x) - $5e^{-x}$ * sin(x), en el dominio, - π <= x <= π.

3.- Ejecutar los programas en Matlab y Visual C# para ajuste de curvas por interpolación mediante Diferencias Divididas de Newton, incorporando la función del numeral 1.

4.- Implementar los programas en Matlab y Visual C# para ajuste de curvas por interpolación por el método de Aitken, incorporando la función del numeral 1. Analizar los resultados para los tres problemas.

6

DIFERENCIACIÓN (DERIVACIÓN) NÚMERICA

Generalidades

Este capítulo se centra en el cálculo de una aproximación de la derivada de una función contínua o discreta en un punto dado.

Se revisarán los siguientes temas:

- Calculo aproximado de derivadas de primer orden.
- Calculo aproximado de derivadas de segundo orden y orden superior.
- Calculo de derivadas de cualquier orden en polinomios aplicando la regla de derivación.
- Errores en la diferenciación numérica.

.

Calculo aproximado de derivadas de primer orden.

Se parte de la definición:

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \approx \frac{f(x+h) - f(x)}{h}; (6.1)$$

Cuando f(x) es continua en un dominio li ≤ x ≤ lf , resulta fácil calcular f'(x), si se conoce el valor de x y un h bastante pequeño.

Ejemplo 6.1:
Dada $f(x) = e^x$. Calcular f'(2) aplicando la fórmula 6.1. Hacer variar h para divisores de 10. Para cada caso calcular el error absoluto y el error relativo y seleccionar el valor más cercano a la derivada.

Solución:
El proceso se describe en la tabla siguiente:

k	h_k	$f_k = f(1+h_k)$	f(x)	f'(x)	Error Absoluto	Error Relativo
1	0.1	8.166169913	7.3890561	7.771138136	0.38208204	5.1709%
2	0.01	7.463317347	7.3890561	7.426124839	0.03706874	0.5017%
3	0.001	7.396448851	7.3890561	7.392751859	0.00369576	0.0500%
4	0.0001	7.389795041	7.3890561	7.389425564	0.00036947	0.0050%
5	0.00001	7.38912999	7.3890561	7.389093044	3.6945E-05	0.0005%
6	0.000001	7.389063488	7.3890561	7.389059794	3.6953E-06	0.0001%
7	0.0000001	7.389056838	7.3890561	7.389056451	3.5222E-07	0.0000%
8	0.00000001	7.389056173	7.3890561	7.389056034	6.5229E-08	0.0000%
9	0.000000001	7.389056106	7.3890561	7.389056478	3.7886E-07	0.0000%
10	1E-10	7.3890561	7.3890561	7.389058254	2.1552E-06	0.0000%

Seleccionando para k = 8, $f'(x) = e^x \rightarrow f'(2) = e^2 \approx 7.389056034$.

Ejemplo 6.2
Dada $f(x) = e^x\cos(2x)$. Calcular f'(2) aplicando la fórmula 6.1. Hacer variar h para divisores de 10. Para cada caso calcular el error absoluto y el error relativo y seleccionar el valor más cercano a la derivada.
Solución:
El proceso se describe en la tabla siguiente:

k	h_k	$f_k = f(x+h_k)$	f(x)	f'(x)	Error Absoluto	Error Relativo
1	0.1	-4.003553169	-4.82980938	8.262562147	1.90825934	30.0310%
2	0.01	-4.764416524	-4.82980938	6.539285912	0.18498311	2.9111%
3	0.001	-4.823436645	-4.82980938	6.372738617	0.01843581	0.2901%
4	0.0001	-4.829173769	-4.82980938	6.356145757	0.00184295	0.0290%
5	0.00001	-4.829745838	-4.82980938	6.354487093	0.00018429	0.0029%
6	0.000001	-4.829803029	-4.82980938	6.354321234	1.843E-05	0.0003%
7	0.0000001	-4.829808748	-4.82980938	6.354304638	1.834E-06	0.0000%
8	0.00000001	-4.82980932	-4.82980938	6.35430295	1.4647E-07	0.0000%
9	0.000000001	-4.829809377	-4.82980938	6.354302862	5.7654E-08	0.0000%
10	1E-10	-4.829809383	-4.82980938	6.35430375	9.4583E-07	0.0000%

Seleccionando para k = 9, $f'(x) = e^x (\cos(2x) - 2\text{sen}(2x)) \rightarrow f'(2) \approx 6.354302862$.

Como puede observarse en los dos ejemplos anteriores, no siempre se obtiene la derivada más aproximada cuando h es más pequeño.

Ejemplo 6.3
Implementar un programa en Matlab para definir la función $f(x) = 4 / (2 - e^{x\wedge 2})$, generar su derivada simbólicamente, ingresar el valor en el que se desea evaluar la derivada de primer orden., el cual debe estar en el dominio $-5 \leq x \leq 5$, luego de lo

cual debe calcularse la derivada aplicando la fórmula 6.1; calcular además el error absoluto y el error relativo Asumir un valor de h = 1e-8.

Solución:

Programa en Matlab:

```
% DerivadaIOrden.m
clc; % Limpia pantalla
clear; % Libera espacio de variables
syms x % Define como simbolo a x
f = 4/(2 - exp(x^2)) % Define la función f(x)
derivada = diff(f) % Genera simbólicamente f'(x)
h = 1e-8;
disp('Cálculo de la derivada de primer orden en un punto dado');
while 1
  x = input('Ingresar x (salir < -5 o > 5): ');
  if (abs(x) > 5) break; end;
  fpe = eval(derivada);
  fx = eval(f);
  x1 = x;
  x = x + h;
  fxh = eval(f);
  fp = (fxh - fx) / h;
  era = abs(fpe - fp);
  err = era / abs(fpe) * 100;
  disp(sprintf('fp(%f) = %f, fpexacta(%f) = %f, Error absoluto = %f, Error relativo = %f%%\n', x1, fp, x1, fpe, era, err));
end;
```

Los resultados, se presentan a continuación:

```
Command Window

f =

4/(2-exp(x^2))



derivada =

8/(2-exp(x^2))^2*x*exp(x^2)


Cálculo de la derivada de primer orden en un punto dado
Ingresar x (salir < -5 o > 5): 1
fp(1.000000) = 42.149718, fpexacta(1.000000) = 42.149722, Error absoluto = 0.000003, Error relativo = 0.000008%

Ingresar x (salir < -5 o > 5): 5
fp(5.000000) = 0.000000, fpexacta(5.000000) = 0.000000, Error absoluto = 0.000000, Error relativo = 0.000004%

Ingresar x (salir < -5 o > 5): -1.5
fp(-1.500000) = -2.030686, fpexacta(-1.500000) = -2.030686, Error absoluto = 0.000000, Error relativo = 0.000001%

Ingresar x (salir < -5 o > 5): 6
>>
```

Calculo aproximado de derivadas de primer orden usando Interpolación por Diferencias Divididas de Newton.

Aplicando la fórmula 6.1, se puede reemplazar f(x) por la función que permite interpolar mediante diferencias divididas de Newton.

Ejemplo 6.4
Tomar los datos del ejemplo 5.4, pag. 192 del libro "Numerical Methods in Engineering with MATLAB" de JAAN KIUSALAAS y aplicando la fórmula 6.1 y la función para interpolar mediante diferencias divididas de Newton, calcular f´(x), para x = 2. Asumir un valor de h = 1e-8.

Solución:
La función f(x), se encuentra dada por la siguiente tabla, misma que ha sido tomada del libro "Numerical Methods in Engineering with MATLAB" de JAAN KIUSALAAS:

x	1.5	1.9	2.1	2.4	2.6	3.1
$f(x)$	1.0628	1.3961	1.5432	1.7349	1.8423	2.0397

Utilizando la función de interpolación de Newton:
vx = [1.5 1.9 2.1 2.4 2.6 3.1]
vy = [1.0628 1.3961 1.5432 1.7349 1.8423 2.0397]
n = 6
f(2) = InterpolacionNewton(vx, vy, n, 2) = 1.4714802183

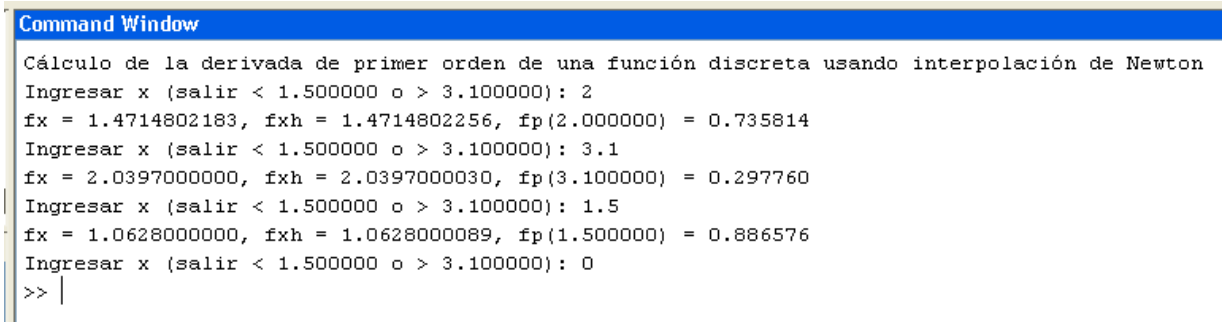
f(2 + 1e-8) = InterpolacionNewton(vx, vy, n, 2 + 1e-8) = 1.4714802256
Aplicando la fórmula 6.1:
f'(2) = (f(2 + 1e-8) - f(2)) / 1e-8 = (1.4714802256 - 1.4714802183) / 1e-8 = 0.735814
El resultado obtenido en el indicado libro es f´(2) = 0.7355.

Ejemplo 6.5
Implementar en Matlab y Visual C#, un programa para procesar el ejemplo 6.4.

Solución:
Programa en Matlab:

```
% DerIlOrdenInterNewton.m
clc; % Limpia pantalla
clear; % Libera espacio de variables
% Datos tomados del libro "Numerical Methods in Engineering with MATLAB"
% JAAN KIUSALAAS Pag. 192 Example 5.4
vx = [1.5 1.9 2.1 2.4 2.6 3.1];
vy = [1.0628 1.3961 1.5432 1.7349 1.8423 2.0397];
n = length(vx);
h = 1e-8;
disp('Cálculo de la derivada de primer orden de una función discreta usando
interpolación de Newton');
while 1
  x = input(sprintf('Ingresar x (salir < %f o > %f): ', vx(1), vx(6)));
  if (x < vx(1) | x > vx(6)) break; end;
  fx = InterpolacionNewton(vx, vy, x, n);
  fxh = InterpolacionNewton(vx, vy, x + h, n);
  fp = (fxh - fx) / h;
  disp(sprintf('fx = %12.10f, fxh = %12.10f, fp(%f) = %f', fx, fxh, x, fp));
end;
```

La función InterpolacionNewton se encuentra descrita en el Capítulo IV. Los resultados, se presentan a continuación:



Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 6.1

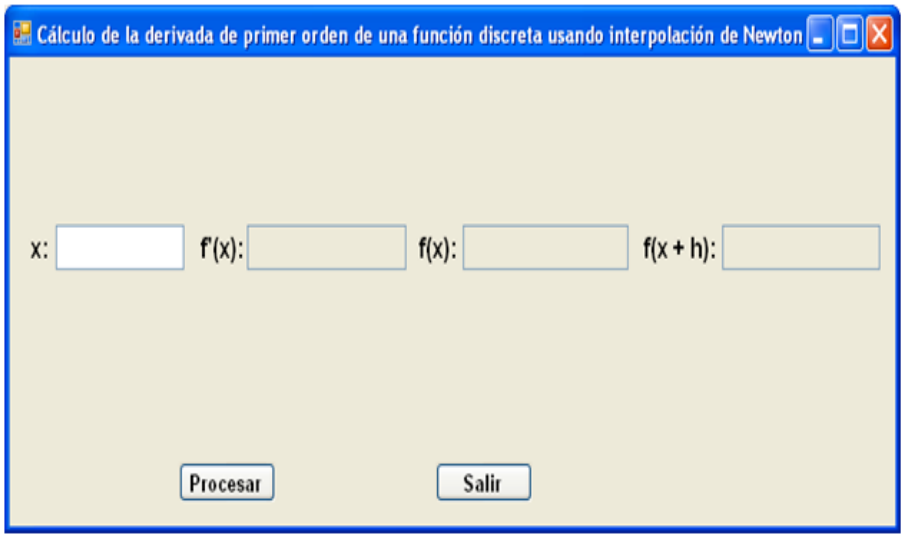


Figura 6.1 Ventana para calcular la derivada I orden por Interpolación de Newton en Visual C#

Como se puede apreciar, existen cuatro cuadros de texto, mismos que permite ingresar la abscisa x en la que se va a calcular la derivada y los tres restantes para desplegar los resultados; y dos botones, mismos que tienen la funcionalidad:

- Procesar:. Valida el valor ingresado de x; calcula la derivada de primer orden aplicando la fórmula 6.1 y la función para interpolar mediante diferencias divididas de Newton.
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Variables y funciones miembro de la clase Form1:

```
// Datos miembro:
// Datos tomados del libro "Numerical Methods in Engineering with MATLAB"
// JAAN KIUSALAAS Pag. 192 Example 5.4
private static double[] vdx = { 1.5, 1.9, 2.1, 2.4, 2.6, 3.1 };
private static double[] vdy = { 1.0628, 1.3961, 1.5432, 1.7349, 1.8423, 2.0397 };
int n = 6;
double h = 1e-8;

// Funciones miembro:
```

Evento Load del objeto Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
   n = vdx.Length;
   CInterpolacion.AsignaMemoria(true, 20);
   for (int i = 0; i < n; i ++) {
      CInterpolacion.vx[i] = vdx[i];
      CInterpolacion.vy[i] = vdy[i];
   }
}
```

Botón Salir:

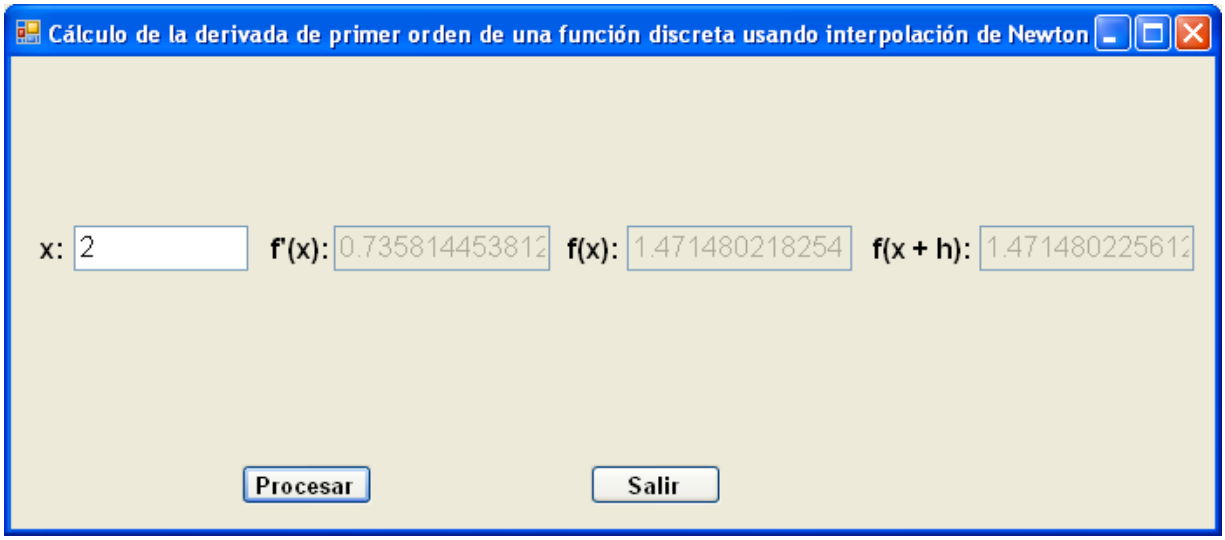
```
private void b_salir_Click(object sender, EventArgs e)
{
   this.Close();
   this.Dispose();
}
```

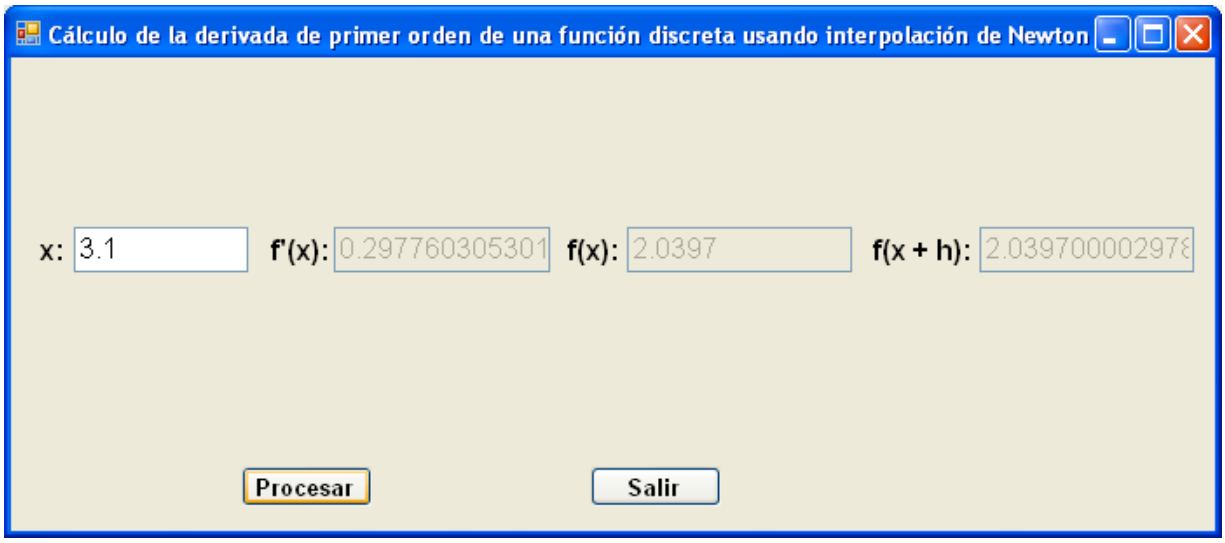
Botón Procesar:

```
private void b_procesar_Click(object sender, EventArgs e)
{
   double x, fx, fxh, fp;
   try
   {
      x = double.Parse(txtX.Text);
      txtFp.Text = "";
      txtFx.Text = "";
      txtFxh.Text = "";
      if (x < CInterpolacion.vx[0] || x > CInterpolacion.vx[5]) return;
      fx = CInterpolacion.InterpolacionNewton(x, n);
      fxh = CInterpolacion.InterpolacionNewton(x + h, n);
      fp = (fxh - fx) / h;
      txtFp.Text = Math.Round(fp, 12).ToString();
      txtFx.Text = Math.Round(fx, 12).ToString();
      txtFxh.Text = Math.Round(fxh, 12).ToString();
  }
  catch (Exception ex)
  {
      MessageBox.Show(ex.Message);
  }
}
```

Las referencias a la clase CInterpolación, proviene de la definición de la respectiva clase incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:





Calculo aproximado de derivadas de segundo orden y orden superior:

Para calcular derivadas superiores al primer orden, se utilizará la Serie de Taylor y sus
variaciones.

La Serie de Taylor, se encuentra dada por:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2!}f''(x) + \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(4)}(x) + ...(6.2)$$

Sustituyendo h por -h en 6.2, se obtiene:

$$f(x-h) = f(x) - hf'(x) + \frac{h^2}{2!}f''(x) - \frac{h^3}{3!}f'''(x) + \frac{h^4}{4!}f^{(4)}(x) - ...(6.3)$$

Sustituyendo h por 2h en 6.2, se obtiene:

$$f(x+2h) = f(x) + 2hf'(x) + \frac{4h^2}{2!}f''(x) + \frac{8h^3}{3!}f'''(x) + \frac{16h^4}{4!}f^{(4)}(x) + ...(6.4)$$

Sustituyendo h por -2h en 6.2, se obtiene:

$$f(x-2h) = f(x) - 2hf'(x) + \frac{4h^2}{2!}f''(x) - \frac{8h^3}{3!}f'''(x) + \frac{16h^4}{4!}f^{(4)}(x) - ...(6.5)$$

Realizando 6.2 + 6.3, se obtiene:

$$f(x+h) + f(x-h) = 2(f(x) + \frac{h^2}{2!}f''(x) + \frac{h^4}{4!}f^{(4)}(x) + ...);(6.6)$$

Realizando 6.2 - 6.3, se obtiene:

$$f(x+h) - f(x-h) = 2(hf'(x) + \frac{h^3}{3!}f'''(x) + \frac{h^5}{5!}f^{(5)}(x) + ...);(6.7)$$

Realizando 6.4 + 6.5, se obtiene:

$$f(x+2h) + f(x-2h) = 2(f(x) + \frac{4h^2}{2!}f''(x) + \frac{16h^4}{4!}f^{(4)}(x) + ...);(6.8)$$

Realizando 6.4 - 6.5, se obtiene:

$$f(x+2h) - f(x-2h) = 2(2hf'(x) + \frac{8h^3}{3!}f'''(x) + \frac{32h^5}{5!}f^{(5)}(x) + ...),(6.9)$$

Aproximación de diferencias centradas:

Despejando f´(x) de 6.7, se obtiene:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} - \frac{h^2}{3!}f'''(x) - \frac{h^4}{5!}f^{(5)}(x) - ...;(6.10)$$

Para un valor pequeño de h, se puede descartar desde el tercer término de 6.10 y sustituyendo el segundo por $O(h^2)$:

$$f'(x) = \frac{f(x+h) - f(x-h)}{2h} + O(h^2);(6.11)$$

Despejando f´´(x) de 6.6, se obtiene:

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} - \frac{h^2}{12}f^{(4)}(x) - \frac{h^4}{360}f^{(6)}(x) - ...;(6.12)$$

Para un valor pequeño de h, se puede descartar desde el tercer término de 6.12 y sustituyendo el segundo por $O(h^2)$:

$$f''(x) = \frac{f(x+h) - 2f(x) + f(x-h)}{h^2} + O(h^2);(6.13)$$

Las derivadas superiores a

segundo orden, se las puede obtener, efectundo operaciones entre las fórmulas de 6.2 a 6.9 resultando ser:

$$f'''(x) = \frac{f(x+2h) - 2f(x+h) + 2f(x-h) - f(x-2h)}{2h^3} + O(h^2); (6.14)$$

$$f^{(4)}(x) = \frac{f(x+2h) - 4f(x+h) + 6f(x) + 4f(x-h) - f(x-2h)}{h^4} + O(h^2); (6.15)$$

$O(h^2)$ es el orden de h^2 y representa el error que se puede cometer al calcular la derivada. Para calcular f'(x), f''(x), f'''(x), etc, con las fórmulas para aproximación de diferencias centradas, se necesitan dos, tres, cuatro, etc puntos, respectivamente.

Ejemplo 6.6
Dada la función y = f(x) = e^xcos(2x), calcular f''(x), f'''(x) y $f^{(4)}$(x) usando aproximación de diferencias centradas, asumir x = 2, h = 0.1

Solución:

El proceso se encuentra desarrollado en la siguiente tabla:

	x + 2h	x + h	x	x - h	x - 2h	
	2.2	2.1	2	1.9	1.8	
k	f(x + 2h)	f(x + h)	f(x)	f(x - h)	f(x - 2h)	$f^{(k)}$(x)
2		-4.00355317	-4.82980938	-5.28832663		36.7738969
3	-2.7736833	-4.00355317		-5.28832663	-5.42507228	40.921029
4	-2.7736833	-4.00355317	-4.82980938	-5.28832663	-5.42507228	-100.926889

Ejemplo 6.7
Implementar un programa en Matlab para resolver el problema del ejemplo 6.6, generarando las derivadas simbólicamente, ingresar el valor en el que se desean evaluar las derivadas de orden superior, las cuales deben estar en el dominio $-5 \leq x \leq 5$, luego de lo cual deben calcularse las derivadas aplicando las fórmulas 6.13, 6.14 y 6.15; calcular además los errores absoluto y relativo.

Solución:

Programa en Matlab:

```
% DerivadaOrdenSup.m
% Calculo de derivadas de orden superior usando
%  aproximación de diferencias centradas.
clc; % Limpia pantalla
clear; % Libera espacio de variables
syms x % Define como simbolo a x
f = exp(x)*cos(2*x) % Define la función f(x)
```

```
derivada2 = diff(f,2) % Genera simbólicamente f''(x)
derivada3 = diff(f,3) % Genera simbólicamente f'''(x)
derivada4 = diff(f,4) % Genera simbólicamente f(4)(x)
h = 0.1;
disp('Cálculo de la derivada de orden superior en un punto dado');
disp(' usando aproximación de diferencias centradas.');
while 1
  x = input('Ingresar x (salir < -5 o > 5): ');
  if (abs(x) > 5) break; end;
  fpe2 = eval(derivada2);
  fpe3 = eval(derivada3);
  fpe4 = eval(derivada4);
  fx = eval(f);
  x1 = x;
  x = x + h;
  fxh = eval(f);
  x = x + h;
  fx2h = eval(f);
  x = x - 3*h;
  fxmh = eval(f);
  x = x - h;
  fxm2h = eval(f);
  fp2 = (fxh - 2*fx + fxmh) / (h^2);
  fp3 = (fx2h - 2*fxh + 2*fxmh - fxm2h) / (2*h^3);
  fp4 = (fx2h - 4*fxh + 6*fx - 4*fxmh + fxm2h) / (h^4);
  era2 = abs(fpe2 - fp2);
  err2 = era2 / abs(fpe2) * 100;
  era3 = abs(fpe3 - fp3);
  err3 = era3 / abs(fpe3) * 100;
  era4 = abs(fpe4 - fp4);
  err4 = era4 / abs(fpe4) * 100;
  disp(sprintf('fp2(%f) = %f, fp2exacta(%f) = %f, Error absoluto = %f, Error relativo =
%f%%', x1, fp2, x1, fpe2, era2, err2));
  disp(sprintf('fp3(%f) = %f, fp3exacta(%f) = %f, Error absoluto = %f, Error relativo =
%f%%', x1, fp3, x1, fpe3, era3, err3));
  disp(sprintf('fp4(%f) = %f, fp4exacta(%f) = %f, Error absoluto = %f, Error relativo =
%f%%', x1, fp4, x1, fpe4, era4, err4));
end;
```

Los resultados, se presentan a continuación:

```
Command Window

f =

exp(x)*cos(2*x)



derivada2 =

-3*exp(x)*cos(2*x)-4*exp(x)*sin(2*x)



derivada3 =

-11*exp(x)*cos(2*x)+2*exp(x)*sin(2*x)



derivada4 =

-7*exp(x)*cos(2*x)+24*exp(x)*sin(2*x)

Cálculo de la derivada de orden superior en un punto dado
 usando aproximación de diferencias centradas.
Ingresar x (salir < -5 o > 5): 2
fp2(2.000000) = 36.773897, fp2exacta(2.000000) = 36.857653, Error absoluto = 0.083756, Error relativo = 0.227241%
fp3(2.000000) = 40.921029, fp3exacta(2.000000) = 41.943791, Error absoluto = 1.022762, Error relativo = 2.438411%
fp4(2.000000) = -100.926089, fp4exacta(2.000000) = -100.400681, Error absoluto = 0.526209, Error relativo = 0.524109%
Ingresar x (salir < -5 o > 5): 1
fp2(1.000000) = -6.437327, fp2exacta(1.000000) = -6.493294, Error absoluto = 0.055966, Error relativo = 0.861908%
fp3(1.000000) = 17.503772, fp3exacta(1.000000) = 17.386702, Error absoluto = 0.117070, Error relativo = 0.673331%
fp4(1.000000) = 66.837737, fp4exacta(1.000000) = 67.239871, Error absoluto = 0.402134, Error relativo = 0.598058%
Ingresar x (salir < -5 o > 5): -3
fp2(-3.000000) = -0.199057, fp2exacta(-3.000000) = -0.199057, Error absoluto = 0.000001, Error relativo = 0.000377%
fp3(-3.000000) = -0.491007, fp3exacta(-3.000000) = -0.490022, Error absoluto = 0.006215, Error relativo = 1.247980%
fp4(-3.000000) = 0.007507, fp4exacta(-3.000000) = -0.000758, Error absoluto = 0.008264, Error relativo = 1090.623819%
Ingresar x (salir < -5 o > 5): -8
>>
```

Aproximación de diferencias adelantadas:

Realizando 6.4 - 4(6.2) y despejando f'(x), se obtiene:

$$f'(x) = \frac{-f(x+2h)+4f(x+h)-3f(x)}{2h} + \frac{h^2}{3}f''''(x) + \frac{h^3}{4}f^{(4)}(x) + ...;(6.16)$$

Para un valor pequeño de h, se puede descartar desde el tercer término de 6.16 y sustituyendo el segundo por $O(h^2)$:

$$f'(x) = \frac{-f(x+2h)+4f(x+h)-3f(x)}{2h} + O(h^2);(6.17)$$

Las derivadas superiores a f'(x), se las obtiene al efectuar operaciones entre f(x) , f(x + h) ,
f(x + 2h) , f(x + 3h) , f(x + 4h). etc. y se las describe a continuación:

$$f''(x) = \frac{-f(x+3h)+4f(x+2h)-5f(x+h)+2f(x)}{h^2} + O(h^2); (6.18)$$

$$f'''(x) = \frac{-3f(x+4h)+14f(x+3h)-24f(x+2h)+18f(x+h)-5f(x)}{2h^3} + O(h^2); (6.19)$$

$$f^{(4)}(x) = \frac{-2f(x+5h)+11f(x+4h)-24f(x+3h)+26f(x+2h)-14f(x+h)+3f(x)}{h^4} + O(h^2); (6.2$$

Ejemplo 6.8
Resolver el problema del ejemplo 6.6 usando aproximación de diferencias adelantadas.

Solución:

El proceso se encuentra desarrollado en la siguiente tabla:

	x + 5h	x + 4h	x + 3h	x + 2h	x + h	x	
	2.5	2.4	2.3	2.2	2.1	2	
k	f(x + 5h)	f(x + 4h)	f(x + 3h)	f(x + 2h)	f(x + h)	f(x)	$f^{(k)}(x)$
2			-1.11862977	-2.7736833	-4.00355317	-4.82980938	38.2043641
3		0.96451673	-1.11862977	-2.7736833	-4.00355317	-4.82980938	49.5610895
4	3.45571286	0.96451673	-1.11862977	-2.7736833	-4.00355317	-4.82980938	-100.769251

Ejemplo 6.9
Implementar un programa en Matlab para resolver el problema del ejemplo 6.8, generarando las derivadas simbólicamente, ingresar el valor en el que se desean evaluar las derivadas de orden superior, las cuales deben estar en el dominio $-5 \leq x \leq 5$, luego de lo cual deben calcularse las derivadas aplicando las fórmulas 6.18, 6.19 y 6.20; calcular además los errores absoluto y relativo.

Solución:

Programa en Matlab:

```
% DerivadaOrdenSupDifAdelan.m
% Calculo de derivadas de orden superior usando
%  aproximación de diferencias adelantadas.
clc; % Limpia pantalla
clear; % Libera espacio de variables
syms x % Define como simbolo a x
f = exp(x)*cos(2*x) % Define la función f(x)
derivada2 = diff(f,2) % Genera simbólicamente f''(x)
derivada3 = diff(f,3) % Genera simbólicamente f'''(x)
```

```
derivada4 = diff(f,4) % Genera simbólicamente f(4)(x)
h = 0.1;
disp('Cálculo de la derivada de orden superior en un punto dado');
disp(' usando aproximación de diferencias adelantadas.');
while 1
  x = input('Ingresar x (salir < -5 o > 5): ');
  if (abs(x) > 5) break; end;
  fpe2 = eval(derivada2);
  fpe3 = eval(derivada3);
  fpe4 = eval(derivada4);
  fx = eval(f);
  x1 = x;
  x = x + h;
  fxh = eval(f);
  x = x + h;
  fx2h = eval(f);
  x = x + h;
  fx3h = eval(f);
  x = x + h;
  fx4h = eval(f);
  x = x + h;
  fx5h = eval(f);
  fp2 = (-fx3h + 4*fx2h - 5*fxh + 2*fx) / (h^2);
  fp3 = (-3*fx4h + 14*fx3h - 24*fx2h + 18*fxh - 5*fx) / (2*h^3);
  fp4 = (-2*fx5h + 11*fx4h - 24*fx3h + 26*fx2h - 14*fxh + 3*fx) / (h^4);
  era2 = abs(fpe2 - fp2);
  err2 = era2 / abs(fpe2) * 100;
  era3 = abs(fpe3 - fp3);
  err3 = era3 / abs(fpe3) * 100;
  era4 = abs(fpe4 - fp4);
  err4 = era4 / abs(fpe4) * 100;
  disp(sprintf('fp2(%f) = %f, fp2exacta(%f) = %f, Error absoluto = %f, Error relativo =
%f%%', x1, fp2, x1, fpe2, era2, err2));
  disp(sprintf('fp3(%f) = %f, fp3exacta(%f) = %f, Error absoluto = %f, Error relativo =
%f%%', x1, fp3, x1, fpe3, era3, err3));
  disp(sprintf('fp4(%f) = %f, fp4exacta(%f) = %f, Error absoluto = %f, Error relativo =
%f%%', x1, fp4, x1, fpe4, era4, err4));
end;
```

Los resultados, se presentan a continuación:

```
Cálculo de la derivada de orden superior en un punto dado
Ingresar x (salir < -5 o > 5): 2
fp2(2.000000) = 36.773897, fp2exacta(2.000000) = 36.857653, Error absoluto = 0.083756, Error relativo = 0.227241%
fp3(2.000000) = 40.921029, fp3exacta(2.000000) = 41.943791, Error absoluto = 1.022762, Error relativo = 2.438411%
fp4(2.000000) = -100.926889, fp4exacta(2.000000) = -100.400681, Error absoluto = 0.526209, Error relativo = 0.524109%
Ingresar x (salir < -5 o > 5): 1
fp2(1.000000) = -6.437327, fp2exacta(1.000000) = -6.493294, Error absoluto = 0.055966, Error relativo = 0.861908%
fp3(1.000000) = 17.503772, fp3exacta(1.000000) = 17.386702, Error absoluto = 0.117070, Error relativo = 0.673331%
fp4(1.000000) = 66.837737, fp4exacta(1.000000) = 67.239871, Error absoluto = 0.402134, Error relativo = 0.598058%
Ingresar x (salir < -5 o > 5): -3
fp2(-3.000000) = -0.199057, fp2exacta(-3.000000) = -0.199057, Error absoluto = 0.000001, Error relativo = 0.000377%
fp3(-3.000000) = -0.491807, fp3exacta(-3.000000) = -0.498022, Error absoluto = 0.006215, Error relativo = 1.247980%
fp4(-3.000000) = 0.007507, fp4exacta(-3.000000) = -0.000758, Error absoluto = 0.008264, Error relativo = 1090.623819%
Ingresar x (salir < -5 o > 5): -8
>>
```

Aproximación de diferencias retrasadas:

Realizando 6.5 - 4(6.3) y despejando f'(x), se obtiene:

$$f'(x)=\frac{f(x-2h)-4f(x-h)+3f(x)}{2h}-\frac{h^2}{3}f''''(x)-\frac{h^3}{4}f^{(4)}(x)-...;(6.21)$$

Para un valor pequeño de h, se puede descartar desde el tercer término de 6.16 y sustituyendo el segundo por $O(h^2)$:

$$f'(x)=\frac{f(x-2h)-4f(x-h)+3f(x)}{2h}+O(h^2);(6.22)$$

Las derivadas superiores a f'(x), se las obtiene al efectuar operaciones entre f(x) , f(x - h) ,
f(x - 2h) , f(x - 3h) , f(x - 4h). etc. y se las describe a continuación:

$$f''(x)=\frac{-f(x-3h)+4f(x-2h)-5f(x-h)+2f(x)}{h^2}+O(h^2);(6.23)$$

$$f'''(x)=\frac{3f(x-4h)-14f(x-3h)+24f(x-2h)-18f(x-h)+5f(x)}{2h^3}+O(h^2);(6.24)$$

$$f^{(4)}(x)=\frac{-2f(x-5h)+11f(x-4h)-24f(x-3h)+26f(x-2h)-14f(x-h)+3f(x)}{h^4}+O(h^2);(6.2$$

Las diferencias adelantadas y retrasadas permiten calcular f'(x), f''(x), f'''(x), etc. teniendo como entradas tres, cuatro, cinco, etc. puntos, respectivamente.

Ejemplo 6.10
Resolver el problema del ejemplo 6.6 usando aproximación de diferencias retrasadas.

Solución:

El proceso se encuentra desarrollado en la siguiente tabla:

	x - 5h	x - 4h	x - 3h	x - 2h	x - h	x	
	1.5	1.6	1.7	1.8	1.9	2	
k	f(x - 5h)	f(x - 4h)	f(x - 3h)	f(x - 2h)	f(x - h)	f(x)	$f^{(k)}(x)$
2			-5.29220244	-5.42507228	-5.28832663	-4.82980938	37.3927705
3		-4.94458639	-5.29220244	-5.42507228	-5.28832663	-4.82980938	48.0863708
4	-4.43683855	-4.94458639	-5.29220244	-5.42507228	-5.28832663	-4.82980938	-86.4916698

Ejemplo 6.11
Implementar un programa en Matlab para resolver el problema del ejemplo 6.10, generarando las derivadas simbólicamente, ingresar el valor en el que se desean evaluar las derivadas de orden superior, las cuales deben estar en el dominio $-5 \leq x \leq 5$, luego de lo cual deben calcularse las derivadas aplicando las fórmulas 6.23, 6.24 y 6.25; calcular además los errores absoluto y relativo.

Solución:

Programa en Matlab:

```
% DerivadaOrdenSupDifRetras.m
% Calculo de derivadas de orden superior usando
%  aproximación de diferencias retrasadas.
clc; % Limpia pantalla
clear; % Libera espacio de variables
syms x % Define como simbolo a x
f = exp(x)*cos(2*x) % Define la función f(x)
derivada2 = diff(f,2) % Genera simbólicamente f''(x)
derivada3 = diff(f,3) % Genera simbólicamente f'''(x)
derivada4 = diff(f,4) % Genera simbólicamente f(4)(x)
h = 0.1;
disp('Cálculo de la derivada de orden superior en un punto dado');
disp(' usando aproximación de diferencias retrasadas.');
while 1
  x = input('Ingresar x (salir < -5 o > 5): ');
  if (abs(x) > 5) break; end;
  fpe2 = eval(derivada2);
  fpe3 = eval(derivada3);
  fpe4 = eval(derivada4);
  fx = eval(f);
  x1 = x;
  x = x - h;
  fxmh = eval(f);
  x = x - h;
  fxm2h = eval(f);
  x = x - h;
  fxm3h = eval(f);
```

```
  x = x - h;
  fxm4h = eval(f);
  x = x - h;
  fxm5h = eval(f);
  fp2 = (-fxm3h + 4*fxm2h - 5*fxmh + 2*fx) / (h^2);
  fp3 = (3*fxm4h - 14*fxm3h + 24*fxm2h - 18*fxmh + 5*fx) / (2*h^3);
  fp4 = (-2*fxm5h + 11*fxm4h - 24*fxm3h + 26*fxm2h - 14*fxmh + 3*fx) / (h^4);
  era2 = abs(fpe2 - fp2);
  err2 = era2 / abs(fpe2) * 100;
  era3 = abs(fpe3 - fp3);
  err3 = era3 / abs(fpe3) * 100;
  era4 = abs(fpe4 - fp4);
  err4 = era4 / abs(fpe4) * 100;
  disp(sprintf('fp2(%f) = %f, fp2exacta(%f) = %f, Error absoluto = %f, Error relativo =
%f%%', x1, fp2, x1, fpe2, era2, err2));
  disp(sprintf('fp3(%f) = %f, fp3exacta(%f) = %f, Error absoluto = %f, Error relativo =
%f%%', x1, fp3, x1, fpe3, era3, err3));
  disp(sprintf('fp4(%f) = %f, fp4exacta(%f) = %f, Error absoluto = %f, Error relativo =
%f%%', x1, fp4, x1, fpe4, era4, err4));
end;
```

Los resultados, se presentan a continuación:

```
Cálculo de la derivada de orden superior en un punto dado
 usando aproximación de diferencias retrasadas.
Ingresar x (salir < -5 o > 5): 2
fp2(2.000000) = 37.392771, fp2exacta(2.000000) = 36.857653, Error absoluto = 0.535118, Error relativo = 1.451850%
fp3(2.000000) = 48.086371, fp3exacta(2.000000) = 41.943791, Error absoluto = 6.142580, Error relativo = 14.644789%
fp4(2.000000) = -86.491670, fp4exacta(2.000000) = -100.400681, Error absoluto = 13.909011, Error relativo = 13.853502%
Ingresar x (salir < -5 o > 5): 1
fp2(1.000000) = -7.048459, fp2exacta(1.000000) = -6.493294, Error absoluto = 0.555166, Error relativo = 8.549836%
fp3(1.000000) = 16.100824, fp3exacta(1.000000) = 17.386702, Error absoluto = 1.285878, Error relativo = 7.395756%
fp4(1.000000) = 70.702996, fp4exacta(1.000000) = 67.239871, Error absoluto = 3.463125, Error relativo = 5.150404%
Ingresar x (salir < -5 o > 5): -3
fp2(-3.000000) = -0.196897, fp2exacta(-3.000000) = -0.199057, Error absoluto = 0.002160, Error relativo = 1.085345%
fp3(-3.000000) = -0.520964, fp3exacta(-3.000000) = -0.490022, Error absoluto = 0.030942, Error relativo = 6.312942%
fp4(-3.000000) = -0.140677, fp4exacta(-3.000000) = -0.000758, Error absoluto = 0.139919, Error relativo = 18464.743055%
Ingresar x (salir < -5 o > 5): -8
>>
```

Calculo de derivadas de cualquier orden en polinomios aplicando la regla de derivación.

En este apartado, se desarrolla un procedimiemto para calcular derivadas de cualquier orden para polinomios de la forma:

$f(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \ldots + a_{n-1}x + a_n$ (6.26)

Donde: $a_0, a_1, a_2, \ldots, a_{n-1}, a_n$ son valores reales, correspondientes a los coeficientes del polinomio; $n > 0$ es un valor entero que corresponde al grado del polinomio.

Ejemplo 6.12

Dado el polinomio $f(x) = 5x^4 – 3x^2 + 2x -20$; calcular $f''(3)$ y $f^{(4)}(3)$.

Solución:

Aplicando sucesivamente dos veces la regla de derivación y sustituyendo x = 3:
$f'(x) = 20x^3 – 6x + 2$; $f''(x) = 60x^2 – 6$; para x = 3:
$f''(3) = 60(3)^2 – 6 = 534$.
Aplicando sobre $f''(x)$ sucesivamente dos veces la regla de derivación y sustituyendo x = 3:
$f'''(x) = 120x$; $f^{(4)}(x) = 120$; $f^{(4)}(3) = 120$.

Ejemplo 6.13

Implementar en Matlab y Visual C# un programa para ingresar el grado y los coeficientes del polinomio, el orden de la derivada y el valor a evaluar y generar simbólicamente el polinomio ingresado y la derivada resultante, calcular además el valor de la derivada.

Solución:

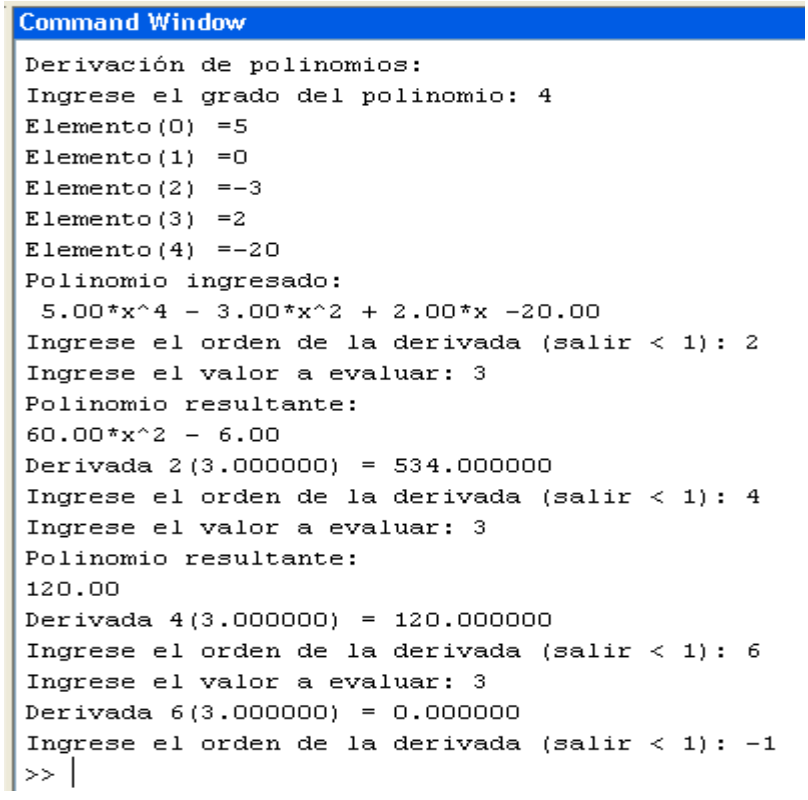
Programa en Matlab:

```
% PDerivNumPol.m
% Diferenciación numérica de polinomios.
% FUNCION PRINCIPAL:
clc; % Limpia pantalla
clear; % Libera espacio de variables
global A N
disp('Derivación de polinomios: ');
[N, A] = LeePolinomio;
disp('Polinomio ingresado: ');
EscribePolinomio(A, N + 1);
while 1
   od = input('Ingrese el orden de la derivada (salir < 1): ');
   if (od < 1)
      break
   end;
   x = input('Ingrese el valor a evaluar: ');
   fprintf(sprintf('Derivada %d(%f) = %f\n', od, x, DerivNumPol(od, x)));
end;

% EscribePolinomio.m
function EscribePolinomio(A, m)
gr = m - 1;
```

```
ep = [];
for i = 1 : m
   if (A(i) ~= 0)
     if (i == 1)
       if (gr > 0)
         ep = [ep, sprintf('%5.2f*', A(i))];
       else
         ep = [ep, sprintf('%5.2f', A(i))];
       end;
     else
       if (A(i) < 0)
          ep = [ep, ' -'];
       else
          ep = [ep, ' +'];
       end;
       if (abs(A(i)) ~= 1)
         if (gr > 0)
           ep = [ep, sprintf('%5.2f*', abs(A(i)))];
         else
           ep = [ep, sprintf('%5.2f', abs(A(i)))];
         end;
       end;
     end;
     if (gr > 0)
       if (gr == 1)
         ep = [ep, 'x'];
       else
         ep = [ep, sprintf('x^%d', gr)];
       end;
     end;
   end;
   gr = gr - 1;
end;
disp(ep);
return

% DerivNumPol.m
% Diferenciación numérica de polinomios.
function d = DerivNumPol(nd, x)
global A N;
if (nd > N)
   d = 0;
   return;
end;
% Obtener una copia de A:
for i = 1 : N + 1
```

```
  B(i) = A(i);
end;
% Aplicar la regla de derivación:
gd = N;
for i = 1 : nd
  gd = gd - 1;
  for j = 1 : gd + 1
    B(j) = (gd - j + 2) * B(j);
  end;
end;
disp('Polinomio resultante: ');
EscribePolinomio(B, gd + 1);
% Evaluar para x al polinomio resultante:
d = EvaluaPolinomio(B, gd, x);
return

% EvaluaPolinomio.m
function b = EvaluaPolinomio(A, N, x)
b = A(1);
for i = 1 : N
  b = A(i + 1) + b * x;
end
return
```

La definición de la función LeePolinomio, se definió en un capítulo anterior. Los resultados de la ejecución de este programa, se presenta en la siguiente pantalla:



Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 6.2

Figura 6.2 Ventana para calcular derivadas de Polinomios en Visual C#

Como se puede apreciar, existen dos cuadros de texto, mismos que permiten ingresar el orden de la derivada y la abscisa x en la que se va a calcular la derivada; un cuadro de lista, el cual permitirá presentar los resultados; y cuatro botones, mismos que tienen la funcionalidad:

- Lectura Proceso…: Valida las entradas; abre por una sola vez, una ventana para registrar el nombre del archivo el cual permite leer los datos del polinomio desde el medio de almacenamiento permanente; ejecuta los métodos para calcular la derivada y despliega los resultados.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Variables y funciones miembro de la clase Form1:

```
// Datos miembro:
static bool pv = true;

// Funciones miembro:
```

Evento Load del objeto Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
    COprPolinomios.d = new double[20];
}
```

Botón Salir:

```
private void b_salir_Click(object sender, EventArgs e)
{
  this.Close();
  this.Dispose();
}
```

Botón Lectura Proceso:

```
private void b_lectura_Click(object sender, EventArgs e)
{
   double x, rd;
   int ordenDeriv;
   // Transferir valores de los texbox a memoria:
   ordenDeriv = int.Parse(txtOrdD.Text);
   x = double.Parse(txtX.Text);
   if (ordenDeriv < 1) return;
   if (pv)
     try
     {
       // Abrir el archivo de entrada a traves del cuadro de dialogo:
       CEntSalArchivos objes = new CEntSalArchivos("c:\\");
       objes.AbrirArchivoEnt();
       // Leer el grado del polinomio:
       CLectEscrMatrices.n = Int16.Parse(objes.LeerLinea());
       if (CLectEscrMatrices.n > 0)
       {
         // Definir y leer el arreglo de coeficientes de la funcion:
         // Leer el arreglo de coeficientes y generar los resultados en el listbox:
         CLectEscrMatrices.n ++;
         CLectEscrMatrices.b = new double[CLectEscrMatrices.n];
         CLectEscrMatrices.obent = objes;
         CLectEscrMatrices.lb = listBox1;
         CLectEscrMatrices.LeeVector();
         // Cerrar flujo de entrada:
         objes.CerrarLectura();
         pv = false;
       }
     }
     catch (Exception ex)
     {
        MessageBox.Show(ex.Message);
     }
   listBox1.Items.Add("CALCULO DE LA DERIVADA GRADO " + ordenDeriv.ToString()
    + ", PARA x = " + x.ToString());
   COprPolinomios.n = CLectEscrMatrices.n - 1;
   COprPolinomios.a = CLectEscrMatrices.b;
   COprPolinomios.lb = listBox1;
   COprPolinomios.Copiar(COprPolinomios.n);
   COprPolinomios.EscribePolinomio("Polinomio ingresado: ");
   rd = COprPolinomios.DerivaPolinomio(ordenDeriv, x);
   String res;
   switch (ordenDeriv) {
```

```
            case 1: res = "f'"; break;
            case 2: res = "f''"; break;
            case 3: res = "f'''"; break;
            default:
               res = "f(" + ordenDeriv.ToString() + ")"; break;
        }
        res += "(" + x.ToString() + ") = " + rd.ToString();
        listBox1.Items.Add(res);
        listBox1.Items.Add("");
    }
```

Botón Guardar Como:

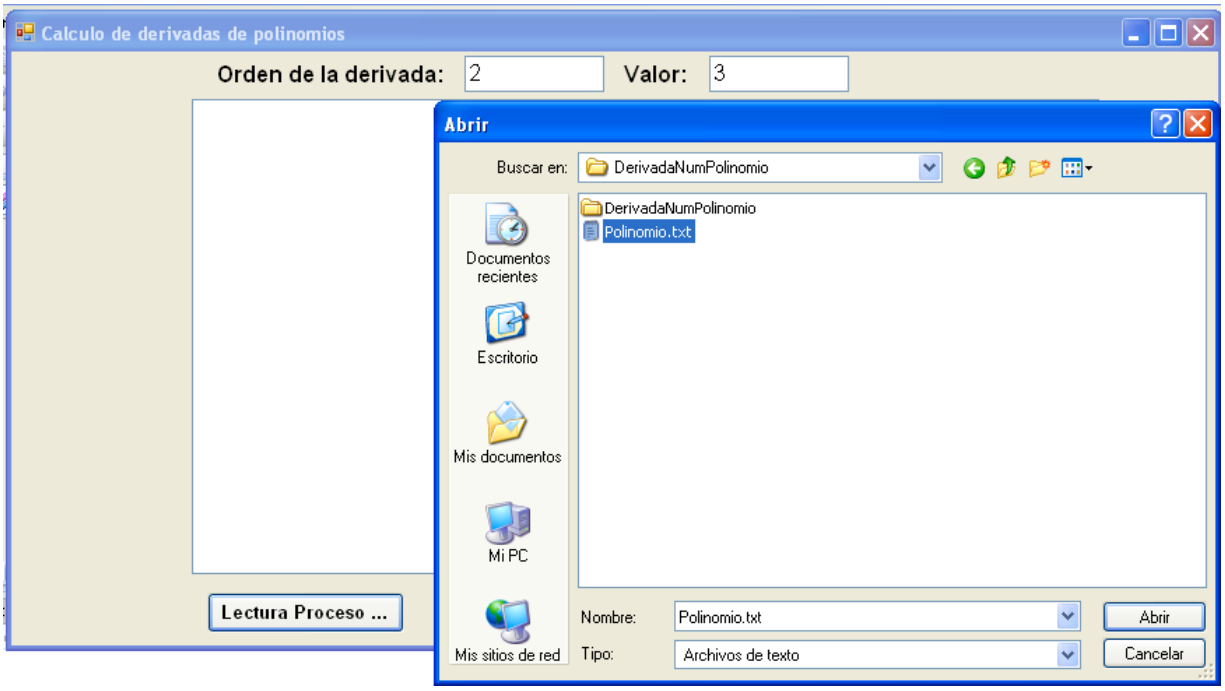
```
private void b_guardar_como_Click(object sender, EventArgs e)
{
   CListBox.lb = listBox1;
   CListBox.GuardarComo("c:\\");
}
```

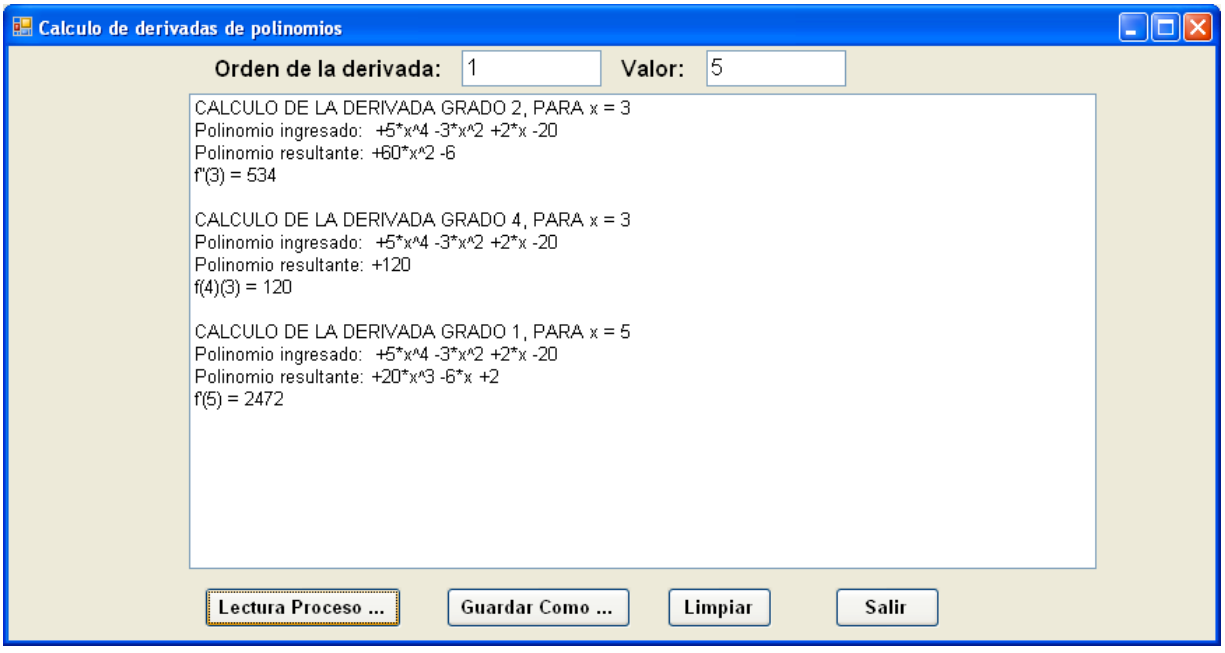
Botón Limpiar:

```
private void b_limpiar_Click(object sender, EventArgs e)
{
   CListBox.lb = listBox1;
   CListBox.Limpiar();
}
```

Las referencias a las clases CEntSalArchivos, CLectEscrMatrices, COprPolinomios, CListBox provienen de la definición de las respectivas clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:





Errores en la diferenciación numérica:

Tabulando los resultados obtenidos para aproximación de diferencias centradas, adelantadas y retrasadas:

Errores en la diferenciación numérica para $f(x) = e^x\cos(2x)$, $x = 2$, $h = 0.1$

Aproximación	k	$f^{(k)}(2)$	$f^{(k)}$Exacta(2)	Err. Abs.	Err. Rel.
Centrada	2	36.773897	36.857653	0.083756	0.23%
	3	40.921029	41.943791	1.022762	2.44%
	4	-100.926889	-100.400681	0.526209	0.52%
Adelantada	2	38.204364	36.857653	1.346712	3.65%
	3	49.561089	41.943791	7.617298	18.16%
	4	-100.769251	-100.400681	0.36857	0.37%
Retrasada	2	37.392771	36.857653	0.535118	1.45%
	3	48.086371	41.943791	6.14258	14.64%
	4	-86.49167	-100.400681	13.909011	13.85%

Conforme a los resultados obtenidos, se puede concluir que para la f(x) indicada, se comete menos errores en f''(2) y f'''(2) para la aproximación de diferencias centrada y en $f^{(4)}(2)$, los errores son menores para la aproximación de diferencias adelantada. Para valores de h más pequeños (por ejemplo, h = 0.01), se obtienen resultados de la siguiente tabla:

Errores en la diferenciación numérica para $f(x) = e^x\cos(2x)$, $x = 2$, $h = 0.01$

Aproximación	k	$f^{(k)}(2)$	$f^{(k)}$Exacta(2)	Err. Abs.	Err. Rel.
Centrada	2	36.856816	36.857653	0.000837	0.00%
	3	41.933528	41.943791	0.010263	0.02%
	4	-100.405996	-100.400681	0.005315	0.01%
Adelantada	2	36.867269	36.857653	0.009616	0.03%
	3	42.016399	41.943791	0.072608	0.17%
	4	-100.317592	-100.400681	0.083089	0.08%
Retrasada	2	36.866448	36.857653	0.008795	0.02%
	3	42.014805	41.943791	0.071014	0.17%
	4	-100.303448	-100.400681	0.097233	0.10%

Como se puede observar, son resultados, bastante interesantes, en todo caso, para calcular la derivada mediante aproximación de diferencias, se debe realizar un proceso iterativo, haciendo variar h, hasta lograr la convergencia.

EJERCICIOS DEL CAPITULO VI

1.- Dada la función: f(x) = 2cos(3x) - 5e $^{-x}$ * sin(x), calcular f'(3) aplicando la fórmula 6.1 y la regla de derivación. Para qué valor de k ($1 \leq k \leq 10$, k es un número entero; $h = 10^{-K}$) , se obtiene una mejor aproximación de f'(3) ?.

2.- Dada la siguiente muestra de ventas (en miles de USD) de un producto en los doce meses del año:

Mes	1	2	3	4	5	6	7	8	9	10	11	12
Venta	85	93	61	73	110	93	89	54	91	60	79	87.5

Calcular f'(x), aplicando la fórmula 6.1 e interpolación a través de Polinomios de Lagrange.

3.- Obtener f''(3), f'''(3) y $f^{(4)}(3)$ y sus correspondientes errores absoluto y relativo mediante aproximación de diferencias centradas, para h = 0.01 a partit de la función del problema 1.

4.- Obtener f''(3), f'''(3) y $f^{(4)}(3)$ y sus correspondientes errores absoluto y relativo mediante aproximación de diferencias adelantadas, para h = 0.01 a partit de la función del problema 1.

5.- Obtener f''(3), f'''(3) y $f^{(4)}(3)$ y sus correspondientes errores absoluto y relativo mediante aproximación de diferencias retrasadas, para h = 0.01 a partit de la función del problema 1.

6.- Implementar los programas en Matlab y Visual C# para calcular f''(x), f'''(x) y $f^{(4)}(x)$ mediante aproximación de diferencias centradas, mediante un proceso iterativo, haciendo variar a $h = 10^{-K}$ para todo $1 \leq k \leq 10$, k es un número entero; tomar como entradas el argumento x, y la función del problema 1. Especificar para que valor de h, se tiene la mejor aproximación.

INTEGRACIÓN NUMÉRICA

Generalidades

La integración numérica consiste en determinar el valor aproximado de la integral definida:

$$F(x) = \int_a^b f(x)dx; (7.1)$$

.donde f(x) es una función integrable en el dominio a ≤ x ≤ b, F(x) es la primitiva o antiderivada; aplicando un método numérico. Tiene la ventaja de calcular integrales que implican un proceso analíticamente complejo o imposible de resolver.

Definición: Dada la función continua f(x) definida en el dominio a ≤ x ≤ b, entonces la fórmula 7.1 se define como el area bajo (sobre) la curva f(x), comprendida entre las ordenadas que pasan por las abscisas a, b y el eje X como lo muestra la figura 7.1.

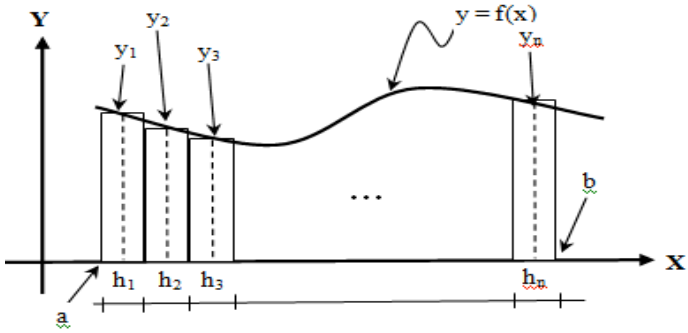


Figura 7.1 Definición de la integral definida

Para poder calcular el area de esa región se la divide en un grupo de rectángulos de base hi y altura yi. Para conseguir una mayor precisión, la base debe ser lo suficientemente pequeña, de manera que:

$$\int_a^b f(x)dx = \lim_{h_i \to 0} \sum_{i=1}^{n} h_i y_i = \lim_{n \to \infty} \sum_{i=1}^{n} h_i y_i; (7.2)$$

h_i se le denomina también “Longitud de paso”; n es el número de areas pequeñas.

En este capítulo Se revisarán los siguientes temas:

- Calculo de integrales, usando las Series de Taylor.
- Calculo aproximado de integrales usando las reglas rectangular, trapezoidal y de Simpson.
- Calculo aproximado de integrales usando otras fórmulas de Newton - Cotes.
- Calculo aproximado de integrales usando interpolación.
- Calculo de integrales múltiples definidas en polinomios aplicando la regla de integración.
- Errores en el calculo de integrales definidas.

Calculo de integrales, usando las Series de Taylor

Las series de Taylor, se aplican en la integración cuando la expresión a integrar contiene funciones que se pueden descomponer en dichas series.

Ejemplo 7.1

Calcular usando los seis primeros términos de la serie de Taylor.

$$F(x) = \int_0^1 sen(x^2)dx$$

Solución:

$$F(x) = \int_0^1 sen(x^2)dx = \int_0^1 \sum_{i=1}^{n} (-1)^{i+1} \frac{(x^2)^{2i-1}}{(2i-1)!}dx = \sum_{i=1}^{n} \frac{(-1)^{i+1}}{(2i-1)!}\int_0^1 x^{4i-2}dx = \sum_{i=1}^{n} \frac{(-1)^{i+1}}{(2i-1)!(4i-1)}$$

$$= \frac{1}{3} - \frac{1}{42} + \frac{1}{1320} - \frac{1}{75600} + \frac{1}{6894720} - \frac{1}{918086400} = 0.3102683$$

Ejemplo 7.2

Implementar un programa en Matlab, que tenga como entradas la función f(x) a integrarse, el intervalo y el número de términos de la serie de Taylor y que calcule el valor de la integral, además el error absoluto y relativo entre el valor que produce Matlab y el calculado. Desarrollar para f(x) = sen(x^2), repetir el proceso ingresando valores en el intervalo [-5, 5].

Solución:

Conocido el intervalo [a, b], la expresión a implementarse es:

$$F(x) = \int_0^1 sen(x^2)dx = \sum_{i=1}^{n} \frac{(-1)^{i+1}}{(2i-1)!(4i-1)}(b^{4i-1} - a^{4i-1})$$

Programa en Matlab:

```
% IntegralSerieTaylor.m
% Calculo de integrales usando series de Taylor, para f(x) = sin(x^2).
clc; % Limpia pantalla
clear; % Libera espacio de variables
disp('Calculo de integrales usando series de Taylor');
while 1
  % Ingreso de valores:
  a = input('Ingresar el valor inicial (salir < -5 o > 5): ');
  if (abs(a) > 5) break; end;
  b = input(sprintf('Ingresar el valor final (salir < -5 o > 5 o < %f): ', a));
  if (abs(b) > 5 | b < a) break; end;
  n = input('Ingresar el número de términos (salir < 1 o > 10): ');
```

```
  if (n < 1 | n > 10) break; end;
  % Cálculo de la integral usando series de Taylor:
  signo = 1;
  vintegra = 0;
  for i = 1 : n
     fac = 1;
     for j = 2 : 2*i - 1
        fac = fac * j;
     end;
     vintegra = vintegra + signo * (b^(4*i - 1) - a^(4*i - 1)) / fac / (4*i - 1);
     signo = -signo;
  end;
  % Cálculo de la integral usando Matlab:
  integrando = @fsin; % Se hacemos manipulable la función f(x) = sin(x^2).
  intMat = quad(integrando, a, b); % Calcula la integral
  % Calculo de los errores absoluto y relativo.
  era = abs(intMat - vintegra);
  err = era / abs(intMat) * 100;
  % Despliegue de resultados:
  disp(sprintf('Integral = %f, Integral Matlab = %f, Error absoluto = %f, Error relativo =
%f%%\n', vintegra, intMat, era, err));
end;

%fsin.m
function y = fsin(x)
y = sin(x.^2);
```

```
Command Window
Calculo de integrales usando series de Taylor
Ingresar el valor inicial (salir < -5 o > 5): 0
Ingresar el valor final (salir < -5 o > 5 o < 0.000000): 1
Ingresar el número de términos (salir < 1 o > 10): 6
Integral = 0.310268, Integral Matlab = 0.310268, Error absoluto = 0.000000, Error relativo = 0.000006%

Ingresar el valor inicial (salir < -5 o > 5): 0
Ingresar el valor final (salir < -5 o > 5 o < 0.000000): 1
Ingresar el número de términos (salir < 1 o > 10): 3
Integral = 0.310281, Integral Matlab = 0.310268, Error absoluto = 0.000013, Error relativo = 0.004211%

Ingresar el valor inicial (salir < -5 o > 5): -2
Ingresar el valor final (salir < -5 o > 5 o < -2.000000): 2
Ingresar el número de términos (salir < 1 o > 10): 8
Integral = 1.609548, Integral Matlab = 1.609553, Error absoluto = 0.000005, Error relativo = 0.000334%

Ingresar el valor inicial (salir < -5 o > 5): 7
>>
```

Calculo aproximado de integrales usando las reglas rectangular, trapezoidal y de Simpson.

A estas reglas se las conoce también como reglas de Newton - Cotes. Para aplicar estas reglas, se divide el area bajo (sobre) la curva en areas lo suficientemente pequeñas, cuya longitud de paso es h = (b – a) / n; n es el número de areas (elementos); a continuación se calculan $x_0 = a$,
$x_1 = x_0 + h$, …, $x_i = x_{i-1} + h$, …, $x_{n-1} = x_{n-2} + h$.
Para calcular el area total, se aplica:

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} f(x)dx; (7.3)$$

Regla rectangular:

Tomando un rectángulo de la figura 7.1 y sustituyendo yi = f(x) en 7.3:

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} y_i dx = \sum_{i=0}^{n-1} y_i \int_{x_i}^{x_{i+1}} dx = \sum_{i=0}^{n-1} y_i (x_{i+1} - x_i) = \sum_{i=0}^{n-1} y_i h = h \sum_{i=0}^{n-1} y_i$$

$$= h(y_0 + y_1 + y_2 + ... + y_{n-1}); (7.4)$$

Cabe destacar que y_i sale de la integral, debido a que es una constante para un rectángulo.

Regla trapezoidal:

Se utilizan trapecios en lugar de rectángulos. Para utilizar esta regla se debe deducir la ecuación de la recta que se indica en la figura 7.2, en este caso,
$y = f(x) = [(y_{i+1} – y_i) / h] x + y_i$ (7.5)

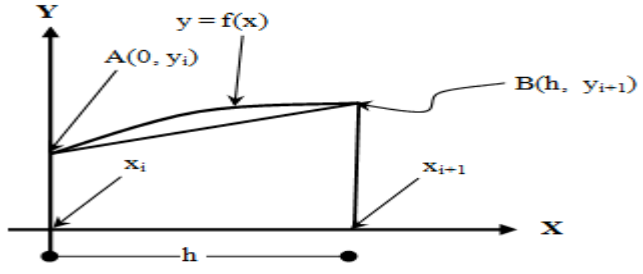


Figura 7.2 Regla trapezoidal

Sustituyendo 7.5 en 7.3:

$$\int_a^b f(x)dx = \sum_{i=0}^{n-1} \int_{x_i}^{x_{i+1}} \{[(y_{i+1} - y_i)/h]x + y_i\}dx = \sum_{i=0}^{n-1} \{[(y_{i+1} - y_i)/h]\frac{x^2}{2} + y_i x\}\Big|_{x_i}^{x_{i+1}} = \sum_{i=0}^{n-1} \frac{h}{2}(y_{i+1} + y_i)$$

$$= h(\frac{y_0 + y_n}{2} + y_1 + y_2 + ... + y_{n-1}); (7.6)$$

Regla de Simpson:

Usando dos trapecios adyacentes, se tiene lo que se indica en la figura 7.3, para tres puntos, debe plantearse la siguiente ecuación:
$y = Ax^2 + Bx + C$ (7.7).
Reemplazando los valores de los puntos de los puntos en (7.7), se obtiene el siguiente sistema de tres ecuaciones por tres incógnitas:

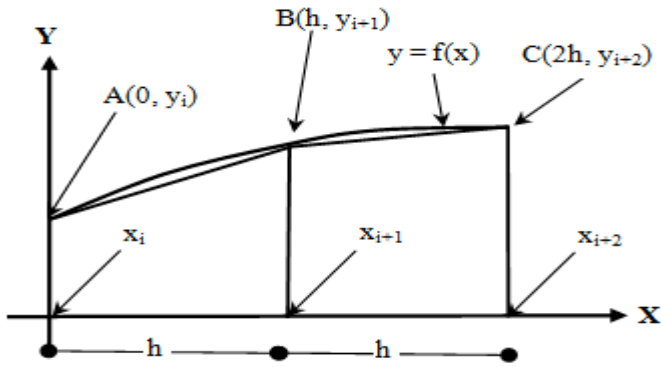


Figura 7.3 Regla de Simpson

$$\begin{cases} y_i = C;(7.8) \\ y_{i+1} = Ah^2 + Bh + C;(7.9) \\ y_{i+2} = 4Ah^2 + 2Bh + C;(7.10) \end{cases}$$

Resolviendo el sistema:

$$\begin{cases} A = \dfrac{y_{i+2} - 2y_{i+1} + y_i}{2h^2};(7.11) \\ B = \dfrac{-y_{i+2} + 4y_{i+1} - 3y_i}{2h^2};(7.12) \\ C = y_i;(7.13) \end{cases}$$

Sustituyendo 7.7 en 7.3:

$$\int_a^b f(x)dx = \sum_{i=0}^{n/2} \int_{x_i}^{x_{i+2}} (Ax^2 + Bx + C)dx = \sum_{i=0}^{n/2} (A\frac{x^3}{3} + B\frac{x^2}{2} + Cx)\Big|_{x_i}^{x_{i+2}} = \sum_{i=0}^{n/2} (8A\frac{h^3}{3} + 2Bh^2 + 2Ch);(7.14)$$

Sustituyendo 7.11, 7.12 y 7.13 en 7.14 y simplificando:

$$\int_a^b f(x)dx = \sum_{i=0}^{n/2} h(\frac{1}{3}y_{i+2} + \frac{4}{3}y_{i+1} + \frac{1}{3}y_i) = \frac{h}{3}(y_0 + 4y_1 + 2y_2 + 4y_3 + ... + 2y_{n-2} + 4y_{n-1} + y_n);(7.15)$$

Ejemplo 7.3

$F(x) = \int_0^1 sen(x^2)dx$ Calcular usando las reglas del rectángulo, trapecio y Simpson para seis

elementos.

Solución:
$y = f(x) = sen(x^2)$; a = 0; b = 1; h = (1 - 0) / 6 = 1/6

Para el cálculo de los puntos (x_i, y_i), se implementa la siguiente tabla:

i	0	1	2	3	4	5	6
x_i	0	1/6	1/3	1/2	2/3	5/6	1
$f(x_i)$	0	0.02777421	0.11088263	0.24740396	0.42995636	0.63995864	0.84147098

Regla de los rectángulos:
Aplicando la fórmula (7.4):
F(x) = (1/6) (0 + 0.027774206 + 0.110882629 + 0.247403959 + 0.429956364
+ 0.639958644) = 0.242662634

Regla de los trapecios:
Aplicando la fórmula (7.6):
F(x) = (1/6)(0 / 2 + 0.027774206 + 0.110882629 + 0.247403959 + 0.429956364
+ 0.639958644 + 0.841470985 / 2) = 0.312785216

Regla de Simpson:
Aplicando la fórmula (7.15):
F(x) = (1/3) (1/6)(0 + 4*0.027774206 + 2*0.110882629 + 4*0.247403959 +2*0.429956364
+ 4*0.639958644 + 0.841470985) = 0.310205345
Ejemplo 7.4

Implementar un programa en Matlab, que tenga como entradas la función f(x) a integrarse, el intervalo y el número de elementos que forman F(x) y que calcule el valor de la integral aplicando las reglas de los rectángulos, trapecios y de Simpson, además el error absoluto y relativo entre el valor que produce Matlab y el calculado. Desarrollar para f(x) = sen(x^2), repetir el proceso ingresando valores en el intervalo [-5, 5] y el número de elementos en el intervalo [3, 1000].

Solución:

Programa en Matlab:

```
% IntegralRgRecTrapSimpson.m
% Calculo de integrales usando las reglas del rectángulo, trapecio
% y de Simpson, para f(x) = sin(x^2).
clc; % Limpia pantalla
clear; % Libera espacio de variables
syms x;
f = sin(x^2);
disp(sprintf('Calculo de integrales usando las reglas del rectángulo, trapecio y de Simpson\n'));
while 1
  % Ingreso de valores:
  a = input('Ingresar el valor inicial (salir < -5 o > 5): ');
  if (abs(a) > 5) break; end;
```

```
  b = input(sprintf('Ingresar el valor final (salir < -5 o > 5 o < %f): ', a));
  if (abs(b) > 5 | b < a) break; end;
  n = input('Ingresar el número de elementos (salir < 3 o > 1000): ');
  if (n < 3 | n > 1000) break; end;
  % Cálculo de la longitud de paso:
  h = (b - a) / n;
  % Cálculo de la integral usando la regla del rectángulo:
  vinterec = 0;
  x = a;
  for i = 1 : n
     vinterec = vinterec + eval(f);
     x = x + h;
  end;
  vinterec = h * vinterec;
  % Cálculo de la integral usando la regla del trapecio:
  x = a;
  vintetra = eval(f) / 2;
  x = x + h;
  for i = 1 : n - 1
     vintetra = vintetra + eval(f);
     x = x + h;
  end;
  vintetra = h * (vintetra + eval(f) / 2);
  % Cálculo de la integral usando la regla de Simpson:
  x = a;
  vinteSim = eval(f);
  x = x + h;
  for i = 1 : n - 1
     fac = 2 * (mod(i, 2) + 1);
     vinteSim = vinteSim + fac * eval(f);
     x = x + h;
  end;
  vinteSim = h * (vinteSim + eval(f)) / 3;
  % Cálculo de la integral usando Matlab:
  integrando = @fsin; % Se hacemos manipulable la función f(x) = sin(x^2).
  intMat = quad(integrando, a, b); % Calcula la integral
  % Calculo de los errores absoluto y relativo y despliegue de resultados:
  era = abs(intMat - vinterec);
  err = era / abs(intMat) * 100;
  disp(sprintf('Integral rectangulos = %f, Integral Matlab = %f, Error absoluto = %f,
Error relativo = %f%%', vinterec, intMat, era, err));
  era = abs(intMat - vintetra);
  err = era / abs(intMat) * 100;
  disp(sprintf('Integral trapecios = %f, Integral Matlab = %f, Error absoluto = %f, Error
relativo = %f%%', vintetra, intMat, era, err));
  era = abs(intMat - vinteSim);
```

```
  err = era / abs(intMat) * 100;
  disp(sprintf('Integral Simpson = %f, Integral Matlab = %f, Error absoluto = %f, Error relativo = %f%%\n', vinteSim, intMat, era, err));
end;
```

Los resultados se presentan en la siguiente pantalla:

```
Command Window
Calculo de integrales usando las reglas del rectángulo, trapecio y de Simpson

Ingresar el valor inicial (salir < -5 o > 5): 0
Ingresar el valor final (salir < -5 o > 5 o < 0.000000): 1
Ingresar el número de elementos (salir < 3 o > 1000): 6
Integral rectangulos = 0.242663, Integral Matlab = 0.310268, Error absoluto = 0.067606, Error relativo = 21.789426%
Integral trapecios = 0.312785, Integral Matlab = 0.310268, Error absoluto = 0.002517, Error relativo = 0.811200%
Integral Simpson = 0.310205, Integral Matlab = 0.310268, Error absoluto = 0.000063, Error relativo = 0.020297%

Ingresar el valor inicial (salir < -5 o > 5): 0
Ingresar el valor final (salir < -5 o > 5 o < 0.000000): 1
Ingresar el número de elementos (salir < 3 o > 1000): 1000
Integral rectangulos = 0.309848, Integral Matlab = 0.310268, Error absoluto = 0.000421, Error relativo = 0.135581%
Integral trapecios = 0.310268, Integral Matlab = 0.310268, Error absoluto = 0.000000, Error relativo = 0.000023%
Integral Simpson = 0.310268, Integral Matlab = 0.310268, Error absoluto = 0.000000, Error relativo = 0.000006%

Ingresar el valor inicial (salir < -5 o > 5): -2
Ingresar el valor final (salir < -5 o > 5 o < -2.000000): 2
Ingresar el número de elementos (salir < 3 o > 1000): 500
Integral rectangulos = 1.609525, Integral Matlab = 1.609553, Error absoluto = 0.000028, Error relativo = 0.001738%
Integral trapecios = 1.609525, Integral Matlab = 1.609553, Error absoluto = 0.000028, Error relativo = 0.001738%
Integral Simpson = 1.609553, Integral Matlab = 1.609553, Error absoluto = 0.000000, Error relativo = 0.000005%

Ingresar el valor inicial (salir < -5 o > 5): -7
>>
```

Calculo aproximado de integrales usando otras fórmulas de Newton – Cotes.

Otra alternativa para deducir las fórmulas de Newton – Cotes consiste en utilizar las fórmulas de los Polinomios de Lagrange (ver fórmulas 4.3 y 4.4), en este caso se evalúa la integral para L(x, n), por tanto:

$$F(x)=\int_a^b f(x)dx=\int_a^b L(x,n)dx=\sum_{j=0}^{n} y_j\int_a^b p_j(x)dx;(7.16)$$

Ejemplo 7.5

Deducir la regla de Simpson, usando la fórmula 7.16.

Solución:
De acuerdo a la figura 7.3, para la deducción, se requieren tres puntos, entonces:

$$p_0(x)=\frac{x-x_1}{x_0-x_1}.\frac{x-x_2}{x_0-x_2}=\frac{x-h}{(-h)}.\frac{x-2h}{(-2h)}=\frac{1}{2h^2}(x^2-3hx+2h^2)$$

$$\therefore\int_a^b p_0(x)dx=\int_0^{2h} p_0(x)dx=\frac{1}{2h^2}\int_0^{2h}(x^2-3hx+2h^2)dx=\frac{1}{2h^2}\left(\frac{x^3}{3}-\frac{3hx^2}{2}+2h^2x\right)\Big|_0^{2h}$$

$$=\frac{1}{2h^2}\left(\frac{8h^3}{3}-6h^3+4h^3\right)=\frac{1}{2h^2}\left(\frac{2h^3}{3}\right)=\frac{h}{3};(7.17)$$

$$p_1(x)=\frac{x-x_0}{x_1-x_0}.\frac{x-x_2}{x_1-x_2}=\frac{x}{h}.\frac{x-2h}{(-h)}=-\frac{1}{h^2}(x^2-2hx)$$

$$\therefore\int_a^b p_1(x)dx=\int_0^{2h}p_1(x)dx=-\frac{1}{h^2}\int_0^{2h}(x^2-2hx)dx=-\frac{1}{h^2}(\frac{x^3}{3}-hx^2)\Big|_0^{2h}$$

$$=-\frac{1}{h^2}(\frac{8h^3}{3}-4h^3)=-\frac{1}{h^2}(-\frac{4h^3}{3})=\frac{4h}{3};(7.18)$$

$$p_2(x)=\frac{x-x_0}{x_2-x_0}.\frac{x-x_1}{x_2-x_1}=\frac{x}{2h}.\frac{x-h}{(h)}=\frac{1}{2h^2}(x^2-hx)$$

$$\therefore\int_a^b p_2(x)dx=\int_0^{2h}p_2(x)dx=\frac{1}{2h^2}\int_0^{2h}(x^2-hx)dx=\frac{1}{2h^2}(\frac{x^3}{3}-\frac{hx^2}{2})\Big|_0^{2h}$$

$$=\frac{1}{2h^2}(\frac{8h^3}{3}-2h^3)=\frac{1}{2h^2}(\frac{2h^3}{3})=\frac{h}{3};(7.19)$$

Sustituyendo (7.17), (7.18) y (7.19) en 7.16:

$$F(x)=\sum_{j=0}^{n/2}y_j\int_a^b p_j(x)dx=\sum_{j=0}^{n/2}(\frac{h}{3}y_j+\frac{4h}{3}y_{j+1}+\frac{h}{3}y_{j+2})$$

$$=\frac{h}{3}(y_0+4y_1+2y_2+4y_3+2y_4+...+2y_{n-2}+4y_{n-1}+y_n);(7.20)$$

Ejemplo 7.6

Deducir la siguiente regla de Newton – Cotes, con respecto a la de Simpson, usando la fórmula 7.16.

Solución:

Para la deducción, se requieren de cuatro puntos, entonces:

$$p_0(x)=\frac{x-x_1}{x_0-x_1}.\frac{x-x_2}{x_0-x_2}.\frac{x-x_3}{x_0-x_3}=\frac{x-h}{(-h)}.\frac{x-2h}{(-2h)}.\frac{x-3h}{(-3h)}$$

$$=-\frac{1}{6h^3}(x^3-6hx^2+11h^2x-6h^3)$$

$$\therefore\int_a^b p_0(x)dx=\int_0^{3h}p_0(x)dx=-\frac{1}{6h^3}\int_0^{3h}(x^3-6hx^2+11h^2x-6h^3)dx$$

$$=-\frac{1}{6h^3}(\frac{x^4}{4}-2hx^3+\frac{11h^2x^2}{2}-6h^3x)\Big|_0^{3h}=-\frac{1}{6h^3}(\frac{81h^4}{4}-54h^4+\frac{99h^4}{2}-18h^4)$$

$$=-\frac{1}{6h^3}(-\frac{9h^4}{4})=\frac{3h}{8};(7.21)$$

$$p_1(x)=\frac{x-x_0}{x_1-x_0}.\frac{x-x_2}{x_1-x_2}.\frac{x-x_3}{x_1-x_3}=\frac{x}{h}.\frac{x-2h}{(-h)}.\frac{x-3h}{(-2h)}=\frac{1}{2h^3}(x^3-5hx^2+6h^2x)$$

$$\therefore\int_a^b p_1(x)dx=\int_0^{3h}p_1(x)dx=\frac{1}{2h^3}\int_0^{3h}(x^3-5hx^2+6h^2x)dx=\frac{1}{2h^3}(\frac{x^4}{4}-\frac{5hx^3}{3}+3h^2x^2)\Big|_0^{3h}$$

$$=\frac{1}{2h^3}(\frac{81h^4}{4}-45h^4+27h^4)=\frac{1}{2h^3}(\frac{9h^4}{4})=\frac{9h}{8};(7.22)$$

$$p_2(x)=\frac{x-x_0}{x_2-x_0}.\frac{x-x_1}{x_2-x_1}.\frac{x-x_3}{x_2-x_3}=\frac{x}{2h}.\frac{x-h}{(h)}.\frac{x-3h}{(-h)}=-\frac{1}{2h^3}(x^3-4hx^2+3h^2x)$$

$$\therefore\int_a^b p_2(x)dx=\int_0^{3h}p_2(x)dx=-\frac{1}{2h^3}\int_0^{3h}(x^3-4hx^2+3h^2x)dx=-\frac{1}{2h^3}(\frac{x^4}{4}-\frac{4hx^3}{3}+\frac{3h^2x^2}{2})\Big|_0^{3h}$$

$$=-\frac{1}{2h^3}(\frac{81h^4}{4}-36h^4+\frac{27h^4}{2})=-\frac{1}{2h^3}(-\frac{9h^4}{4})=\frac{9h}{8};(7.23)$$

$$p_3(x)=\frac{x-x_0}{x_3-x_0}.\frac{x-x_1}{x_3-x_1}.\frac{x-x_2}{x_3-x_2}=\frac{x}{3h}.\frac{x-h}{(2h)}.\frac{x-2h}{(h)}=\frac{1}{6h^3}(x^3-3hx^2+2h^2x)$$

$$\therefore\int_a^b p_3(x)dx=\int_0^{3h}p_3(x)dx=\frac{1}{6h^3}\int_0^{3h}(x^3-3hx^2+2h^2x)dx=\frac{1}{6h^3}(\frac{x^4}{4}-hx^3+h^2x^2)\Big|_0^{3h}$$

$$=\frac{1}{6h^3}(\frac{81h^4}{4}-27h^4+9h^4)=\frac{1}{6h^3}(\frac{9h^4}{4})=\frac{3h}{8};(7.24)$$

Sustituyendo de 7.21 a 7.24 en 7.16:

$$F(x)=\sum_{j=0}^{n/3}y_j\int_a^b p_j(x)dx=\sum_{j=0}^{n/3}(\frac{3h}{8}y_j+\frac{9h}{8}y_{j+1}+\frac{9h}{8}y_{j+2}+\frac{3h}{8}y_{j+2})$$

$$=\frac{3h}{8}(y_0+3y_1+3y_2+2y_3+3y_4+...+3y_{n-2}+3y_{n-1}+y_n);(7.25)$$

A esta se le conoce como “tres octavos”.

La siguiente regla, se la deduce como la de “tres octavos” y requiere de cinco puntos.

$$F(x)=\frac{2h}{45}(7y_0+32y_1+12y_2+32y_3+14y_4+32y_5+12y_6+...+12y_{n-2}+32y_{n-1}+7y_n);(7.26)$$

Ejemplo 7.7

$F(x)=\int_0^1 sen(x^2)dx$ Calcular usando las reglas de tres octavos y dos cuarentaycincoavos para cuatro y cinco puntos respectivamente.

Solución:

$y = f(x) = sen(x^2)$; a = 0; b = 1; $h_4 = (1 - 0) / 3 = 1/3$; $h_5 = (1 - 0) / 4 = 1 / 4$
Para el cálculo de los puntos (x_i, y_i), se implementa las siguientes tablas:

i	0	1	2	3
x_i	0	1/3	2/3	1
$f(x_i)$	0	0.11088263	0.42995636	0.84147098

i	0	1	2	3	4
x_i	0	1/4	1/2	3/4	1
$f(x_i)$	0	0.06245932	0.24740396	0.53330267	0.84147098

Regla de tres octavos:
Aplicando la fórmula (7.25):
F(x) = (3/8)(1/3)(0 + 3*0.110882629 + 3*0.429956364 + 0.841470985) = 0.307998495

Regla de dos cuarentaycincoavos:
Aplicando la fórmula (7.26):
F(x) = (2/45)(1/4)(7*0 + 32* 0.062459318 + 12* 0.247403959 + 32* 0.533302674
+ 7* 0.841470985) = 0.310261424

Ejemplo 7.8

Implementar un programa en Matlab, que tenga como entradas la función f(x) a integrarse, el intervalo y el número de elementos que forman F(x) y que calcule el valor de la integral aplicando las reglas de tres octavos y de dos cuarentaycincoavos, además el error absoluto y relativo entre el valor que produce Matlab y el calculado. Desarrollar para $f(x) = sen(x^2)$, repetir el proceso ingresando valores en el intervalo [-5, 5] y el número de elementos en el intervalo [3, 1000].

Solución:

Programa en Matlab:

```
% IntegralNewtonCotes.m
% Calculo de integrales usando las reglas de Newton-Cotes
% Tres octavos y dos cuarentaycincoavos.
clc; % Limpia pantalla
clear; % Libera espacio de variables
syms x;
f = sin(x^2);
to = [3, 3, 2]; % coeficientes regla tres octavos
dc = [32, 12, 32, 14]; % coeficientes regla dos cuarentaycincoavos
disp(sprintf('Calculo de integrales usando las reglas de tres octavos y dos cuarentaycincoavos\n'));
while 1
```

```
% Ingreso de valores:
a = input('Ingresar el valor inicial (salir < -5 o > 5): ');
if (abs(a) > 5) break; end;
b = input(sprintf('Ingresar el valor final (salir < -5 o > 5 o < %f): ', a));
if (abs(b) > 5 | b < a) break; end;
n = input('Ingresar el número de elementos (salir < 3 o > 1000): ');
if (n < 3 | n > 1000) break; end;
% Cálculo de la longitud de paso:
h = (b - a) / n;
% Cálculo de la integral usando la regla de tres octavos:
x = a;
vinteTroc = eval(f);
x = x + h;
for i = 1 : n - 1
    vinteTroc = vinteTroc + to(mod(i - 1, 3) + 1) * eval(f);
    x = x + h;
end;
vinteTroc = 3 * h * (vinteTroc + eval(f)) / 8;
% Cálculo de la integral usando la regla de dos cuarentaycincoavos:
x = a;
vinteDocu = 7 * eval(f);
x = x + h;
for i = 1 : n - 1
    vinteDocu = vinteDocu + dc(mod(i - 1, 4) + 1) * eval(f);
    x = x + h;
end;
vinteDocu = 2 * h * (vinteDocu + 7 * eval(f)) / 45;
% Cálculo de la integral usando Matlab:
integrando = @fsin; % Se hacemos manipulable la función f(x) = sin(x^2).
intMat = quad(integrando, a, b); % Calcula la integral
% Calculo de los errores absoluto y relativo y despliegue de resultados:
era = abs(intMat - vinteTroc);
err = era / abs(intMat) * 100;
disp(sprintf('Integral Tres octavos = %f, Integral Matlab = %f, Error absoluto = %f,
Error relativo = %f%%', vinteTroc, intMat, era, err));
era = abs(intMat - vinteDocu);
err = era / abs(intMat) * 100;
disp(sprintf('Integral Dos cuarentaycincoavos = %f, Integral Matlab = %f, Error
absoluto = %f, Error relativo = %f%%\n', vinteDocu, intMat, era, err));
end;
```

Los resultados se presentan en la siguiente pantalla:

```
Command Window
Calculo de integrales usando las reglas de tres octavos y dos cuarentaycincoavos

Ingresar el valor inicial (salir < -5 o > 5): 0
Ingresar el valor final (salir < -5 o > 5 o < 0.000000): 1
Ingresar el número de elementos (salir < 3 o > 1000): 3
Integral Tres octavos = 0.307990, Integral Matlab = 0.310268, Error absoluto = 0.002270, Error relativo = 0.731568%
Integral Dos cuarentaycincoavos = 0.216267, Integral Matlab = 0.310268, Error absoluto = 0.094001, Error relativo = 30.296008%

Ingresar el valor inicial (salir < -5 o > 5): 0
Ingresar el valor final (salir < -5 o > 5 o < 0.000000): 1
Ingresar el número de elementos (salir < 3 o > 1000): 4
Integral Tres octavos = 0.266031, Integral Matlab = 0.310268, Error absoluto = 0.044237, Error relativo = 14.257697%
Integral Dos cuarentaycincoavos = 0.310261, Integral Matlab = 0.310268, Error absoluto = 0.000007, Error relativo = 0.002223%

Ingresar el valor inicial (salir < -5 o > 5): 0
Ingresar el valor final (salir < -5 o > 5 o < 0.000000): 1
Ingresar el número de elementos (salir < 3 o > 1000): 12
Integral Tres octavos = 0.310260, Integral Matlab = 0.310268, Error absoluto = 0.000009, Error relativo = 0.002834%
Integral Dos cuarentaycincoavos = 0.310268, Integral Matlab = 0.310268, Error absoluto = 0.000000, Error relativo = 0.000012%

Ingresar el valor inicial (salir < -5 o > 5): -2
Ingresar el valor final (salir < -5 o > 5 o < -2.000000): 2
Ingresar el número de elementos (salir < 3 o > 1000): 12
Integral Tres octavos = 1.638037, Integral Matlab = 1.609553, Error absoluto = 0.028484, Error relativo = 1.769704%
Integral Dos cuarentaycincoavos = 1.609716, Integral Matlab = 1.609553, Error absoluto = 0.000162, Error relativo = 0.010094%

Ingresar el valor inicial (salir < -5 o > 5): -8
>>
```

Ejemplo 7.9

Implementar un programa en Visual C#, que tenga como entradas la función f(x) a integrarse, el intervalo y el número de elementos que forman F(x) y que calcule el valor de la integral aplicando todas las reglas vistas de Newton - Cotes, como alternativa calcular la integral mediante series de Taylor. Desarrollar para f(x) = sen(x^2), validar ingresando valores para el número de elementos en el intervalo [3, 1000].

Solución:

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 7.4
Como se puede apreciar, existen dos botones de opción para seleccionar si la integración es usando las series de Taylor o las reglas de Newton - Cotes; tres cuadros de texto,

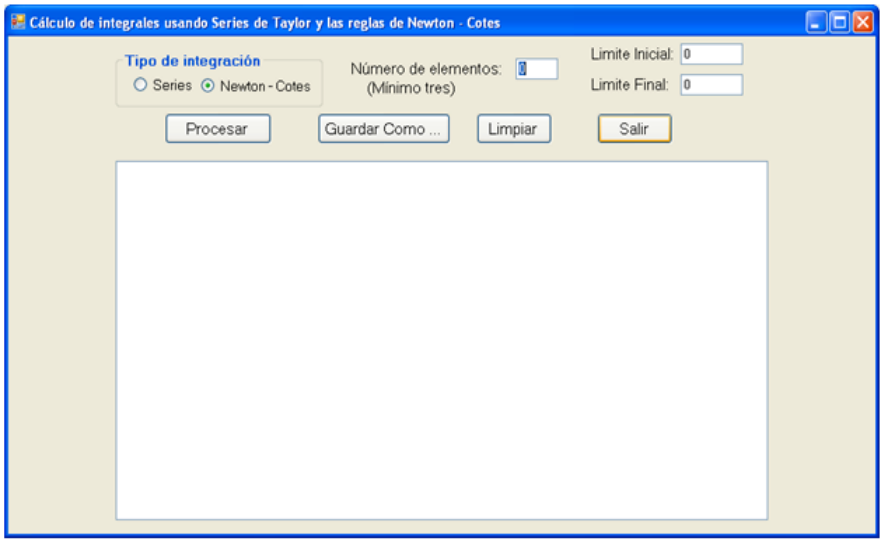


Figura 7.4 Ventana para calcular integrales usando Series de Taylor y Reglas N – C en Visual C#

mismos que permite ingresar el número de términos o de elementos y el intervalo en el que se calcula la integral ; un cuadro lista para desplegar los resultados; y cuatro botones, mismos que tienen la funcionalidad:

- Procesar: Ejecuta los métodos para calcular la integral mediante la serie de Taylor o las reglas de Newton – Cotes, respectivamente, de acuerdo a la opción que se haya escogido y despliega los resultados.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

```
// Funciones miembro:

// Función a ser analizada:
static double f(double x)
{
  return Math.Sin(x * x);
}
```

Opción Series:

```
private void rbSeries_CheckedChanged(object sender, EventArgs e)
{
  lvar.Text = "Número de términos:";
}
```

Opción Newton - Cotes:

```
private void rbNC_CheckedChanged(object sender, EventArgs e)
{
  lvar.Text = "Número de elementos:";
}
```

Botón Procesar:

```
private void b_procesar_Click(object sender, EventArgs e)
{
  // Calcula integrales por interpolación de Newton de una función continua.
  // Declaracion de variables
  double vinteg, limi, limf;
  int n;  // Número de elementos.
  try
  {
    // Transferencia de captura de datos a variables:
    n = int.Parse(txtNun.Text);
    // Validación:
    if (n < 3 || n > 1000) return;
```

```
      limi = double.Parse(txtLIni.Text);
      limf = double.Parse(txtLFin.Text);
      // Validación:
      if (limf < limi) return;
      CIntegracion.n = n;
      CIntegracion.li = limi;
      CIntegracion.lf = limf;
      if (rbNC.Checked)
      {
        CIntegracion.punf = f;
        vinteg = CIntegracion.RRectangulos();
        listBox1.Items.Add("Integral Rectangulos = " + Math.Round(vinteg, 9).ToString()
          + ", para " + CIntegracion.n.ToString() + " elementos");
        vinteg = CIntegracion.RTrapecios();
        listBox1.Items.Add("Integral Trapecios = " + Math.Round(vinteg, 9).ToString()
          + ", para " + CIntegracion.n.ToString() + " elementos");
        vinteg = CIntegracion.RSimpson();
        listBox1.Items.Add("Integral Simpson = " + Math.Round(vinteg, 9).ToString()
          + ", para " + CIntegracion.nc.ToString() + " elementos");
        vinteg = CIntegracion.RTresOctavos();
        listBox1.Items.Add("Integral Tres octavos = " + Math.Round(vinteg, 9).ToString()
          + ", para " + CIntegracion.nc.ToString() + " elementos");
        vinteg = CIntegracion.RDosCuarentaycincoavos();
        listBox1.Items.Add("Integral Dos cuarentaycincoavos = " + Math.Round(vinteg, 9).ToString()
          + ", para " + CIntegracion.nc.ToString() + " elementos");
        listBox1.Items.Add(" ");
      }
      else
      {
        vinteg = CIntegracion.IntSenCuad();
        listBox1.Items.Add("Integral Sen(x^2) por series Taylor = " + Math.Round(vinteg, 9).ToString()
          + ", para " + CIntegracion.n.ToString() + " elementos");
       listBox1.Items.Add(" ");
      }
    }
    catch (Exception ex)
    {
      MessageBox.Show(ex.Message);
    }
}
```

Botón Guardar Como …:

```
private void b_guardar_Click(object sender, EventArgs e)
{
  CListBox.lb = listBox1;
  CListBox.GuardarComo("c:\\");
}
```

Botón Limpiar:

```
private void b_limpiar_Click(object sender, EventArgs e)
{
  CListBox.lb = listBox1;
```

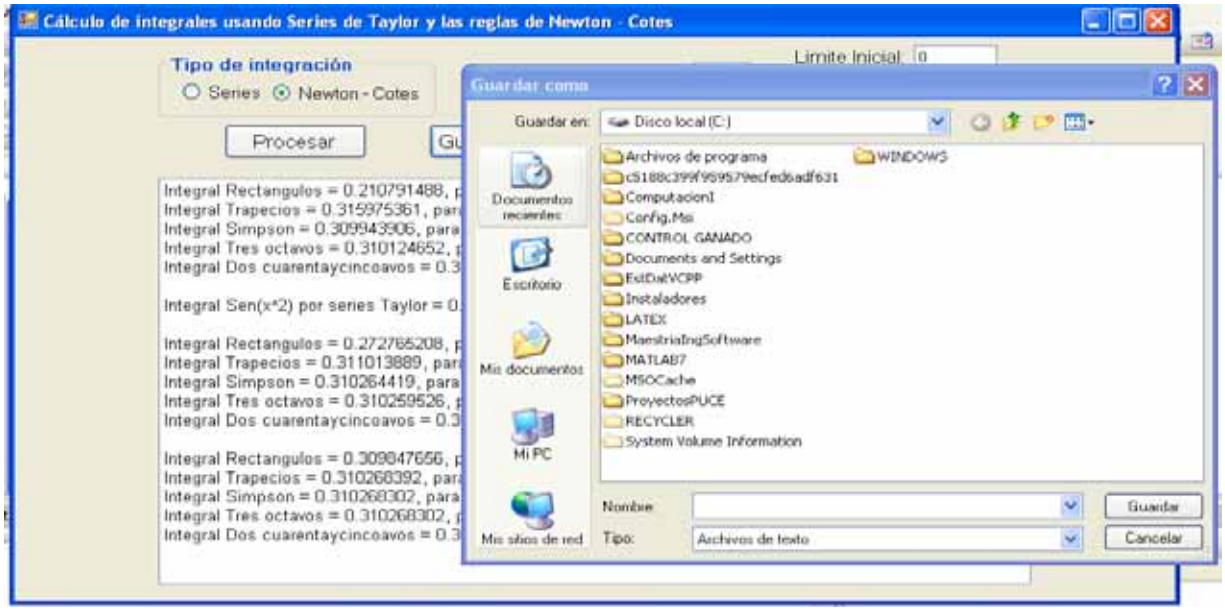
```
   CListBox.Limpiar();
 }
```

Botón Salir:

```
private void b_salir_Click(object sender, EventArgs e)
{
   this.Close();
   this.Dispose();
}
```

Las referencias a la clase CIntegracion y CListBox, provienen de las definiciones de las respectiva clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:



Calculo aproximado de integrales usando interpolación.

Este cálculo resulta adecuado cuando la longitud de paso es variable y se tiene una muestra de puntos, en este caso, se divide cada tramo de longitud de paso hi en subtramos de longitud igual y se aplica en forma combinada el proceso de interpolación de los polinomios de Lagrange o de diferencias divididas de Newton para el cálculo de las ordenadas y la regla de Newton – Cotes que resulte más eficiente para el cálculo de las subareas. La suma de subareas, produce el valor aproximado de la integral, usando interpolación.

Ejemplo 7.10

Dada la muestra:

X	1.9	2	2.1	2.2	2.3	2.4	2.5	3	3.1	3.2	3.3
Y	2.89	6.35	10.23	14.40	18.70	22.92	26.82	30.51	25.81	18.65	8.87

$$F(x) = \int_{1.9}^{3.3} f(x)dx$$

Calcular usando las reglas de Newton – Cotes e interpolación mediante diferencias divididas de Newton.

Solución:

El area a integrarse es la que se muestra en la figura 7.5

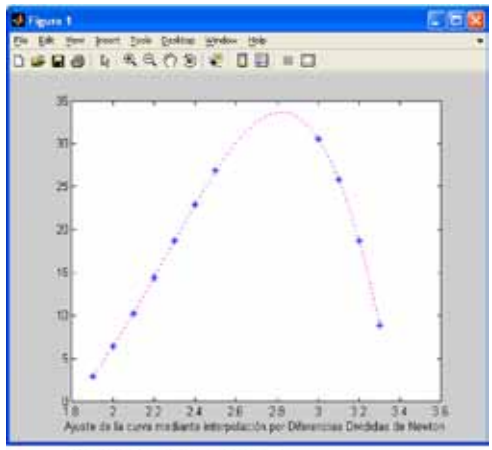


Figura 7.5 Area a integrarse por interpolación

F(X) = A_1 + A_2 + A_3, donde:
A_1 es el area en el tramo 1.9 ≤ x ≤ 2.5
A_2 es el area en el tramo 2.5 ≤ x ≤ 3
A_3 es el area en el tramo 3 ≤ x ≤ 3.3
Para A_1 se tomarán los siete puntos, h = 0.1 y se aplicará dos veces la regla de tres octavos.
Para A_2 se tomarán los once puntos para calcular siete ordenadas por interpolación,
h = (3 – 2.5) / 6 = 1 / 12 y se aplicará dos veces la regla de tres octavos.
Para A_3 se tomarán los cuatro puntos, h = 0.1 y se aplicará una vez la regla de tres octavos.
Desarrollando el proceso:
A_1 = 3*0.1*(2.89 + 3*6.35 + 3*10.23 + 2*14.40 + 3*18.70 + 3*22.92 + 26.82) / 8 = 8.74
Al efectuar la interpolación de los puntos de A_2, se obtiene la siguiente tabla:

X	2.5	2.58	2.67	2.75	2.83	2.92	3.00
Y	26.82	29.52	31.90	33.22	33.59	32.68	30.51

entonces,
A_2 = 3*(1/12)*(26.82 + 3*29.52 + 3*31.90 + 2*33.22 + 3*33.59 + 3*32.68 + 30.51) / 8
= 15.84

A_3 = 3*(0.1)*(30.51 + 3*25.81 + 3*18.65 + 8.87) / 8 = 6.48
Por tanto F(x) = 8.74 + 15.84 + 6.48 = 31.06

Para la interpolación, se ha empleado el siguiente programa en Matlab:

```
% PrbInterpolacionNewtonDis.m
% Prueba interpolación de Newton mediante diferencias divididas para una muestra.
clear;
clc;
x = [1.9 2 2.1 2.2 2.3 2.4 2.5 3 3.1 3.2 3.3];
y = [2.89 6.35 10.23 14.40 18.70 22.92 26.82 30.51 25.81 18.65 8.87];
n = length(x);
while 1
   xit = input(sprintf('Ingrese la abscisa a interpolar (salir < %f o > %f): ', x(1), x(n)));
   % Validación:
   if (xit < x(1) | xit > x(n)) break; end;
   % Cálculo de la ordenada interpolada:
   yit = InterpolacionNewton(x, y, xit, n);
   % Desplegar resultados:
   disp(sprintf('Ordenada (interpolada mediante diferencias divididas de Newton): %f',
yit));
end;
```

Cuyos resultados son:

```
Command Window
Ingrese la abscisa a interpolar (salir < 1.900000 o > 3.300000): 1.9
Ordenada (interpolada mediante diferencias divididas de Newton): 2.890000
Ingrese la abscisa a interpolar (salir < 1.900000 o > 3.300000): 2.5
Ordenada (interpolada mediante diferencias divididas de Newton): 26.820000
Ingrese la abscisa a interpolar (salir < 1.900000 o > 3.300000): 2.58
Ordenada (interpolada mediante diferencias divididas de Newton): 29.524218
Ingrese la abscisa a interpolar (salir < 1.900000 o > 3.300000): 2.67
Ordenada (interpolada mediante diferencias divididas de Newton): 31.901713
Ingrese la abscisa a interpolar (salir < 1.900000 o > 3.300000): 2.75
Ordenada (interpolada mediante diferencias divididas de Newton): 33.219590
Ingrese la abscisa a interpolar (salir < 1.900000 o > 3.300000): 2.83
Ordenada (interpolada mediante diferencias divididas de Newton): 33.594028
Ingrese la abscisa a interpolar (salir < 1.900000 o > 3.300000): 2.92
Ordenada (interpolada mediante diferencias divididas de Newton): 32.676133
Ingrese la abscisa a interpolar (salir < 1.900000 o > 3.300000): 3
Ordenada (interpolada mediante diferencias divididas de Newton): 30.510000
Ingrese la abscisa a interpolar (salir < 1.900000 o > 3.300000): 1
>>
```

Ejemplo 7.11

Implementar un programa en Matlab y otro en Visual C#, que tenga como entradas la muestra a integrarse, y el número de elementos que forman F(x) y que calcule el valor de la integral aplicando las reglas de tres octavos y de dos cuarentaicincoavos e interpolación mediante diferencias divididas de Newton. Desarrollar para la muestra del ejemplo 7.8, repetir el proceso ingresando el número de elementos en el intervalo [11, 1000]. El programa debe calcular apropiadamente el número de puntos y los errores absoluto y relativo.

Solución:

Programa en Matlab:

IntegracionXInterpolacionNewtonDis.m

```
% Calcula integral por interpolación de Newton mediante diferencias divididas
% para una muestra.
clear;
clc;
% Definición de la muestra
syms x;
F = exp(x)*cos(2*x);
vx = [1.9 2 2.1 2.2 2.3 2.4 2.5 3 3.1 3.2 3.3];
vy = [2.89 6.35 10.23 14.40 18.70 22.92 26.82 30.51 25.81 18.65 8.87];
to = [3, 3, 2]; % coeficientes regla tres octavos
dc = [32, 12, 32, 14]; % coeficientes regla dos cuarentaycincoavos
tf = length(vx);
% Proceso:
while 1
   n = input(sprintf('Ingrese el número de elementos (salir < %d o > %d): ', tf, 1000));
   % Validación:
   if (n < tf) | (n > 1000) break; end;
   % regla tres octavos:
   rg = 3;
   % Ajuste de n:
   if (mod(n, rg) ~= 0) n = n + rg - mod(n, rg); end;
   h = (vx(tf) - vx(1)) / n;
   x = vx(1);
   vinteTroc = InterpolacionNewton(vx, vy, x, tf);
   x = x + h;
   for i = 1 : n - 1
      vinteTroc = vinteTroc + to(mod(i - 1, rg) + 1) * InterpolacionNewton(vx, vy, x, tf);
      x = x + h;
   end;
   vinteTroc = 3 * h * (vinteTroc + InterpolacionNewton(vx, vy, x, tf)) / 8;
   % Calculo de la integral exacta:
   x = vx(tf);
   vintexa = eval(F);
   x = vx(1);
   vintexa = vintexa - eval(F);
  % Calculo de los errores absoluto y relativo y despliegue de resultados:
  era = abs(vintexa - vinteTroc);
  err = era / abs(vintexa) * 100;
  disp(sprintf('Integral Tres octavos = %f, Integral Exacta = %f, Error absoluto = %f,
Error relativo = %f%%, para %d elementos', vinteTroc, vintexa, era, err, n));
   % regla dos cuarentaycincoavos:
```

```
  rg = 4;
  % Ajuste de n:
  if (mod(n, rg) ~= 0) n = n + rg - mod(n, rg); end;
  h = (vx(tf) - vx(1)) / n;
  x = vx(1);
  vinteDocu = 7*InterpolacionNewton(vx, vy, x, tf);
  x = x + h;
  for i = 1 : n - 1
     vinteDocu = vinteDocu + dc(mod(i - 1, rg) + 1) * InterpolacionNewton(vx, vy, x, tf);
     x = x + h;
  end;
  vinteDocu = 2 * h * (vinteDocu + 7*InterpolacionNewton(vx, vy, x, tf)) / 45;
 % Calculo de los errores absoluto y relativo y despliegue de resultados:
 era = abs(vintexa - vinteDocu);
 err = era / abs(vintexa) * 100;
 disp(sprintf('Integral Dos cuarentaycincoavos = %f, Integral Exacta = %f, Error absoluto = %f, Error relativo = %f%%, para %d elementos\n', vinteDocu, vintexa, era, err, n));
end;
```

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 7.6
Como se puede apreciar, existen un cuadro de texto, mismo que permite ingresar el número de elementos; un cuadro lista para desplegar

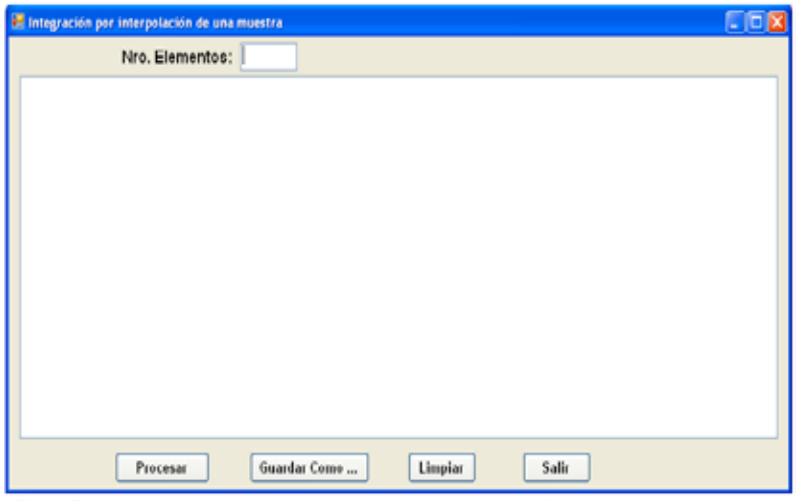


Figura 7.6 Integración por Interpolación de una muestra en Visual C#

los resultados; y cuatro botones, mismos que tienen la funcionalidad:

- Procesar: Ejecuta los métodos para calcular la integral mediante las reglas de tres octavos y dos cuarentaycincoavos, y la interpolación de Newton y despliega los resultados.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

```
// Datos miembro:
// Muestra
private static double[] vdx = {1.9, 2, 2.1, 2.2, 2.3, 2.4, 2.5, 3, 3.1, 3.2, 3.3, };
private static double[] vdy = { 2.89, 6.35, 10.23, 14.40, 18.70, 22.92, 26.82, 30.51, 25.81, 18.65, 8.87 };
private static int tf;

// Funciones miembro:

// Antiderivada:
static double F(double x)
{
  return Math.Exp(x) * Math.Cos(2 * x);
}

// Función a ser analizada:
static double InterpolacionNewton(double x)
{
  return CInterpolacion.InterpolacionNewton(x, tf);
}
```

Botón Procesar:

```
private void b_procesar_Click(object sender, EventArgs e)
{
  // Calcula integrales por interpolación de Newton de una muestra y calcula errores.
  // Declaracion de variables
  double vinteTroc, vinteDocu, vintexa, era, err;
  int n;  // Número de elementos.
  try
  {
    // Transferencia de captura de datos a variables:
    n = int.Parse(txtNroEl.Text);
    // Validación:
    if (n < tf || n > 1000) return;
    CIntegracion.n = n;
    CIntegracion.li = vdx[0];
    CIntegracion.lf = vdx[tf - 1];
    CIntegracion.punf = InterpolacionNewton;
    vinteTroc = CIntegracion.RTresOctavos();
    // Calculo de la integral exacta:
    vintexa = F(vdx[tf - 1]) - F(vdx[0]);
    // Calculo de los errores absoluto y relativo y despliegue de resultados:
```

```
        era = Math.Abs(vintexa - vinteTroc);
        err = era / Math.Abs(vintexa) * 100;
        listBox1.Items.Add("Integral Tres octavos = " + Math.Round(vinteTroc, 2).ToString()
          + ", Integral Exacta = " + Math.Round(vintexa, 2).ToString()
          + ", Error absoluto = " + Math.Round(era, 6).ToString()
          + ", Error relativo = " + Math.Round(err, 6).ToString() + "%"
          + ", para " + CIntegracion.nc.ToString() + " elementos");
        vinteDocu = CIntegracion.RDosCuarentaycincoavos();
        era = Math.Abs(vintexa - vinteDocu);
        err = era / Math.Abs(vintexa) * 100;
        listBox1.Items.Add("Integral Dos cuarentaycincoavos = "
          + Math.Round(vinteTroc, 2).ToString()
          + ", Integral Exacta = " + Math.Round(vintexa, 2).ToString()
          + ", Error absoluto = " + Math.Round(era, 6).ToString()
          + ", Error relativo = " + Math.Round(err, 6).ToString() + "%"
          + ", para " + CIntegracion.nc.ToString() + " elementos");
        listBox1.Items.Add(" ");
      }
      catch (Exception ex)
      {
        MessageBox.Show(ex.Message);
      }
    }
```

Botón Guardar Como …:

```
private void b_guardar_Click(object sender, EventArgs e)
{
  CListBox.lb = listBox1;
  CListBox.GuardarComo("c:\\");
}
```

Botón Limpiar:

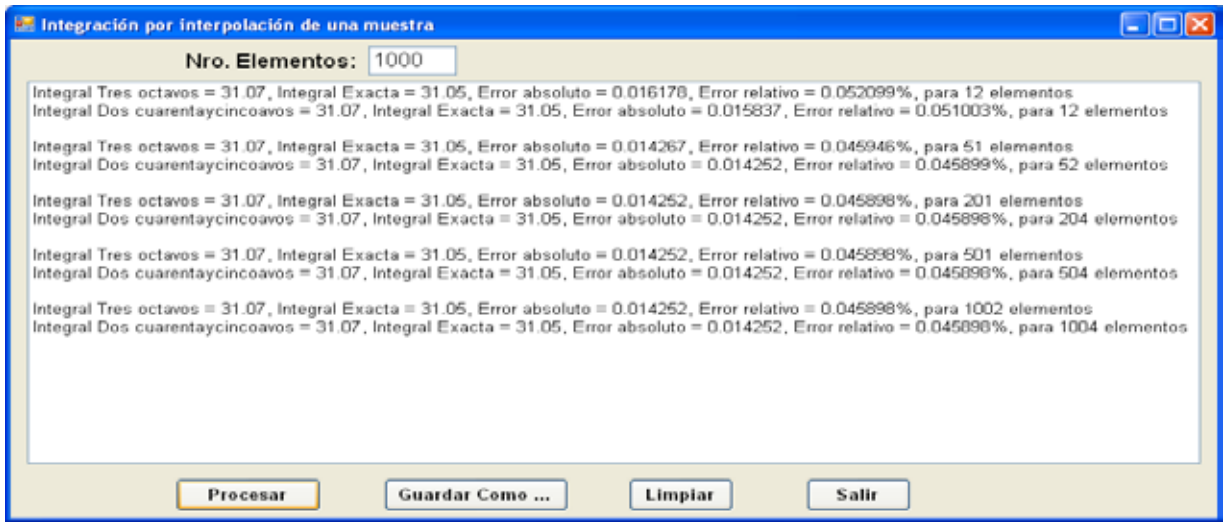
```
private void b_limpiar_Click(object sender, EventArgs e)
{
  CListBox.lb = listBox1;
  CListBox.Limpiar();
}
```

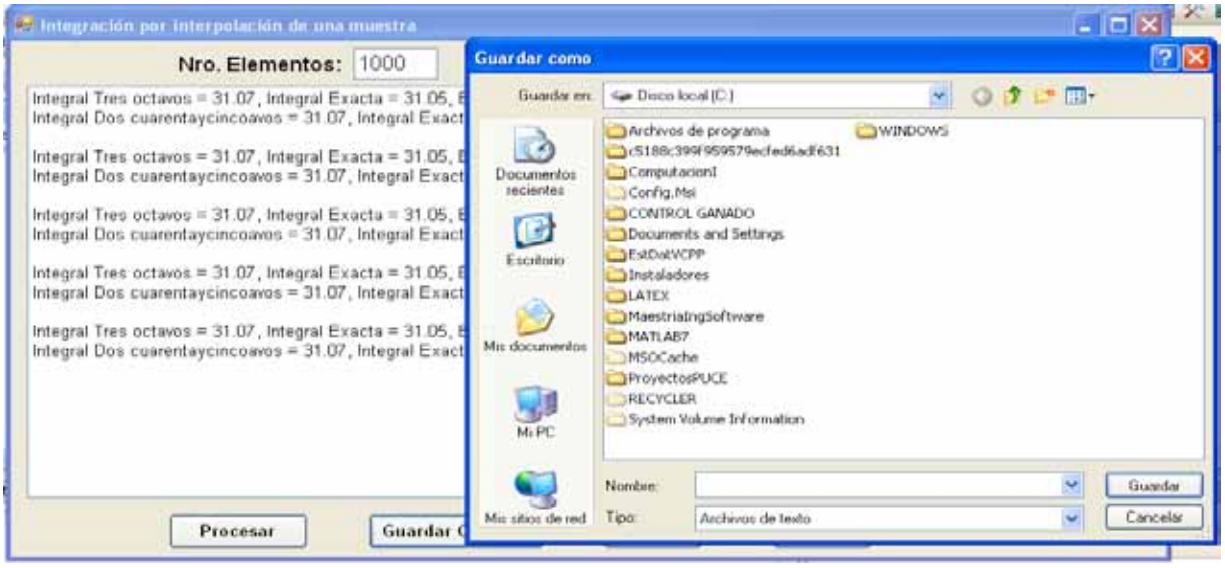
Botón Salir:

```
private void b_salir_Click(object sender, EventArgs e)
{
  this.Close();
  this.Dispose();
}
```

Las referencias a las clases CInterpolacion, CIntegracion y CListBox, provienen de las definiciones de las respectiva clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:





Calculo de integrales múltiples definidas en polinomios aplicando la regla de integración.

En este apartado, se desarrolla un procedimiemto para calcular integrales múltiples para polinomios de la forma:

$f(x) = a_0x^n + a_1x^{n-1} + a_2x^{n-2} + \ldots + a_{n-1}x + a_n$ (7.27)
Donde: $a_0, a_1, a_2, \ldots, a_{n-1}, a_n$ son valores reales, correspondientes a los coeficientes del polinomio; n >= 0 es un valor entero que corresponde al grado del polinomio.

Ejemplo 7.12

Dado el polinomio f(x) = 120; calcular: $F_1(x) = \int_{0.316238}^{0}\int_{0}^{0} f(x)dxdx$ y

$$F_2(x) = \int_{4}^{2}\int_{1}^{3}\int_{0.316238}^{0}\int_{0}^{0} f(x)dxdxdxdx$$

Solución:

$$F_1(x) = \int_{0.316238}^{0}\int_{0}^{0} 120dxdx = \int_{0.316238}^{0}(120x + (120x)\Big|_0^0)dx = \int_{0.316238}^{0}120xdx = (60x^2 + (60x^2)\Big|_{0.316238}^{0})\Big|_{0.316238}^{0}$$

$$= (60x^2 - 6)\Big|_{0.316238}^{0} = -6$$

$$F_2(x) = \int_{4}^{2}\int_{1}^{3}\int_{0.316238}^{0}\int_{0}^{0} f(x)dxdxdxdx = \int_{4}^{2}\int_{1}^{3} F_1(x)dxdx = \int_{4}^{2}\int_{1}^{3}(60x^2 - 6)dxdx =$$

$$\int_{4}^{2}((20x^3 - 6x) + (20x^3 - 6x)\Big|_1^3)dx = \int_{4}^{2}(20x^3 - 6x + 508)dx =$$

$$((5x^4 - 3x^2 + 508x) + (5x^4 - 3x^2 + 508x)\Big|_4^2)\Big|_4^2 = (5x^4 - 3x^2 + 508x - 2180)\Big|_4^2 = -2180$$

Ejemplo 7.13

Implementar un programa en Matlab y otro en Visual C#, que tenga como entradas f(x) a integrarse, y el intervalo de integración [li, lf] y que calcule el valor de la integral aplicando integración sucesiva. Desarrollar para los casos del ejemplo 7.12, repetir el proceso ingresando valores de li y lf en el intervalo [-2999, 2999].
Solución:

Programa en Matlab:

```
% PIntegralNumPol.m
% Prueba Integración numérica de polinomios.
% FUNCION PRINCIPAL:
clc; % Limpia pantalla
clear; % Libera espacio de variables
global A N
disp('Integración de polinomios: ');
[N, A] = LeePolinomio;
disp('Polinomio ingresado: ');
EscribePolinomio(A, N + 1);
while 1
  li = input('Ingrese el limite inicial (salir < -2999 o > 2999): ');
  if (abs(li) > 2999)
     break
  end;
```

```
   lf = input('Ingrese el limite final (salir < -2999 o > 2999): ');
   if (abs(lf) > 2999)
      break
   end;
   integ = IntegralNumPol(li, lf);
   disp(sprintf('Valor de la integral = %f\n', integ));
end;

% IntegralNumPol.m
% Integraciación numérica de polinomios.
function integ = IntegralNumPol(limi, limf)
global A N;
% Aplicar la regla de integración:
for i = 1 : N + 1
   A(i) = A(i) / (N - i + 2);
end;
N = N + 1;
A(N + 1) = 0;
a1 = EvaluaPolinomio(A, N, limi);
a2 = EvaluaPolinomio(A, N, limf);
A(N + 1) = a2 - a1;
disp('Polinomio resultante: ');
EscribePolinomio(A, N + 1);
a1 = EvaluaPolinomio(A, N, limi);
a2 = EvaluaPolinomio(A, N, limf);
% integ = A(N + 1);
integ = a2 - a1;
return
```

Las definiciones de las funciones LeePolinomio, EscribePolinomio y EvaluaPolinomio, se describieron anteriormente; los resultados se presentan en la siguiente pantalla:

```
Command Window
Ingrese el grado del polinomio: 0
Elemento(0) =120
Polinomio ingresado:
120.00
Ingrese el limite inicial (salir < -2999 o > 2999): 0
Ingrese el limite final (salir < -2999 o > 2999): 0
Polinomio resultante:
120.00*x
Valor de la integral = 0.000000

Ingrese el limite inicial (salir < -2999 o > 2999): 0.316238
Ingrese el limite final (salir < -2999 o > 2999): 0
Polinomio resultante:
60.00*x^2 - 6.00
Valor de la integral = -6.000388

Ingrese el limite inicial (salir < -2999 o > 2999): 1
Ingrese el limite final (salir < -2999 o > 2999): 3
Polinomio resultante:
20.00*x^3 - 6.00*x +508.00
Valor de la integral = 507.999223

Ingrese el limite inicial (salir < -2999 o > 2999): 4
Ingrese el limite final (salir < -2999 o > 2999): 2
Polinomio resultante:
 5.00*x^4 - 3.00*x^2 +508.00*x -2180.00
Valor de la integral = -2179.996116

Ingrese el limite inicial (salir < -2999 o > 2999):
```

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 7.7

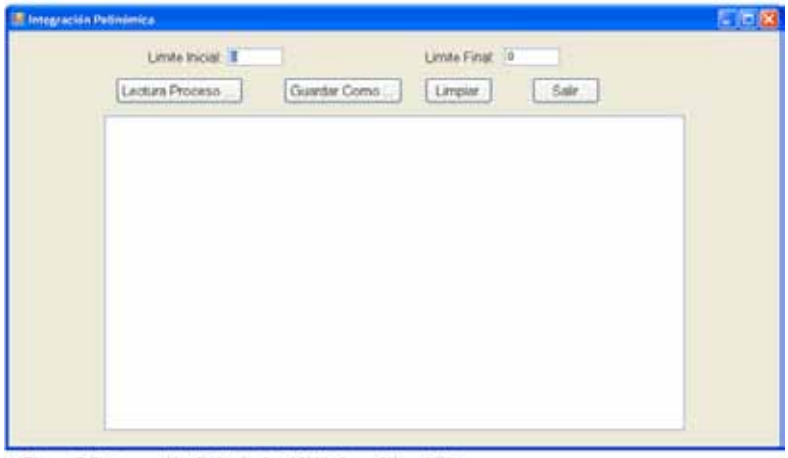


Figura 7.7 Integración Polinómica Múltiple en Visual C#

Como se puede apreciar, existen dos cuadros de texto, mismos que permiten ingresar los límites inicial y final de integración; un cuadro lista para desplegar los resultados; y cuatro botones, mismos que tienen la funcionalidad:

- Lectura Proceso…: Ejecuta los métodos para calcular la integral del polinomio mediante la regla de integración y despliega los resultados.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

```
// Datos miembro:
static bool pv = true;

// Funciones miembro:
```

Evento Load del objeto Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
  COprPolinomios.d = new double[20];
}
```

Botón Lectura Proceso...:

```
private void b_LectuProce_Click(object sender, EventArgs e)
{
  double limi, limf, it;
  // Transferir valores de los texbox a memoria:
  limi = double.Parse(txtLIni.Text);
  limf = double.Parse(txtLFin.Text);
  if (Math.Abs(limi) > 2999) return;
  if (Math.Abs(limf) > 2999) return;
  if (pv)
    try
    {
      // Abrir el archivo de entrada a traves del cuadro de dialogo:
      CEntSalArchivos objes = new CEntSalArchivos("c:\\");
      objes.AbrirArchivoEnt();
      // Leer el grado del polinomio:
      CLectEscrMatrices.n = Int16.Parse(objes.LeerLinea());
      if (CLectEscrMatrices.n >= 0)
      {
        // Definir y leer el arreglo de coeficientes de la funcion:
        // Leer el arreglo de coeficientes y generar los resultados en el listbox:
        CLectEscrMatrices.n++;
        CLectEscrMatrices.b = new double[CLectEscrMatrices.n];
        CLectEscrMatrices.obent = objes;
        CLectEscrMatrices.lb = listBox1;
        CLectEscrMatrices.LeeVector();
        // Cerrar flujo de entrada:
        objes.CerrarLectura();
        COprPolinomios.n = CLectEscrMatrices.n - 1;
        COprPolinomios.a = CLectEscrMatrices.b;
        COprPolinomios.Copiar(COprPolinomios.n);
        pv = false;
      }
    }
    catch (Exception ex)
    {
      MessageBox.Show(ex.Message);
    }
    listBox1.Items.Add("CALCULO DE LA INTEGRAL DEFINIDA: "
      + ", PARA Lim. inf. = " + limi.ToString()
      + ", Lim. sup. = " + limf.ToString());
```

```
    COprPolinomios.lb = listBox1;
    COprPolinomios.EscribePolinomio("Polinomio ingresado: ");
    it = COprPolinomios.IntegraPolinomio(limi, limf);
    listBox1.Items.Add("Valor de la integral = " + it.ToString());
}
```

Botón Guardar Como …:

```
private void b_guardar_Click(object sender, EventArgs e)
{
  CListBox.lb = listBox1;
  CListBox.GuardarComo("c:\\");
}
```

Botón Limpiar:

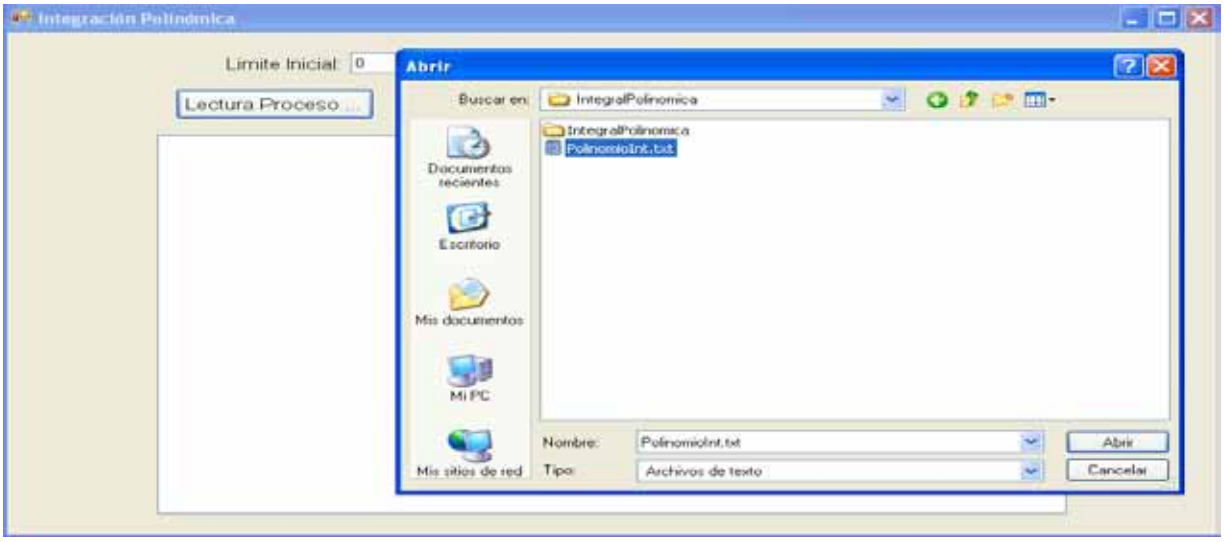
```
private void b_limpiar_Click(object sender, EventArgs e)
{
  CListBox.lb = listBox1;
  CListBox.Limpiar();
}
```

Botón Salir:

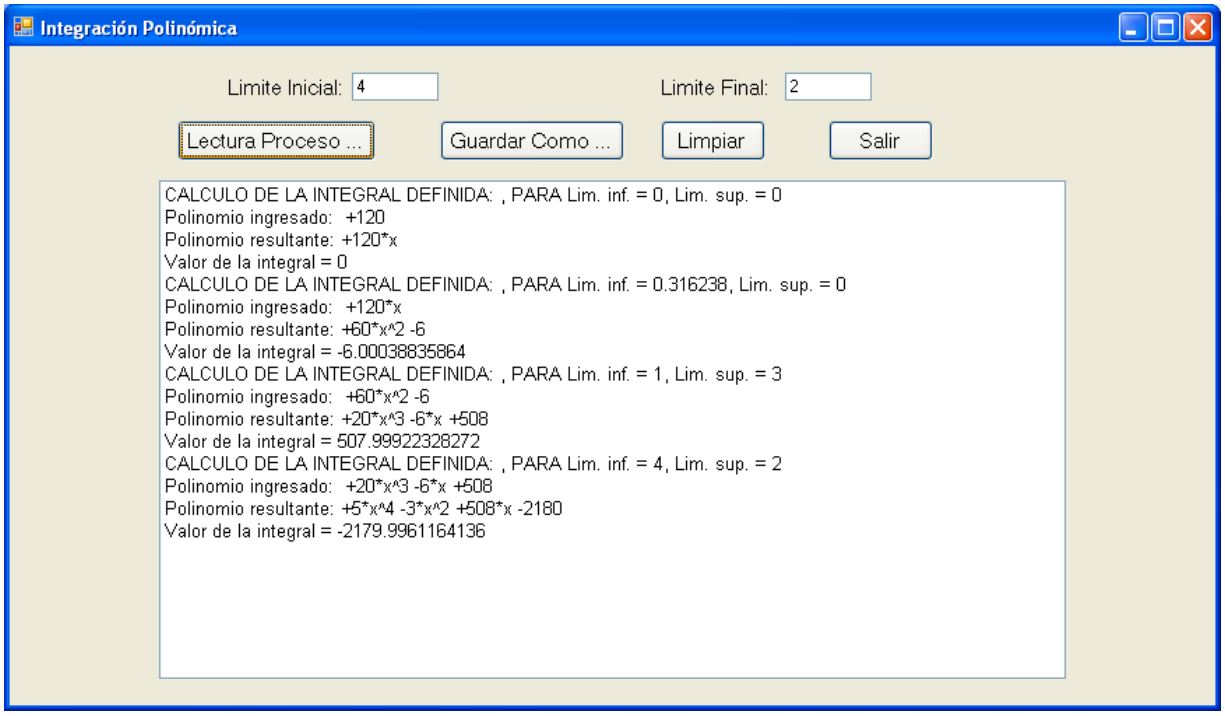
```
private void b_salir_Click(object sender, EventArgs e)
{
  this.Close();
  this.Dispose();
}
```

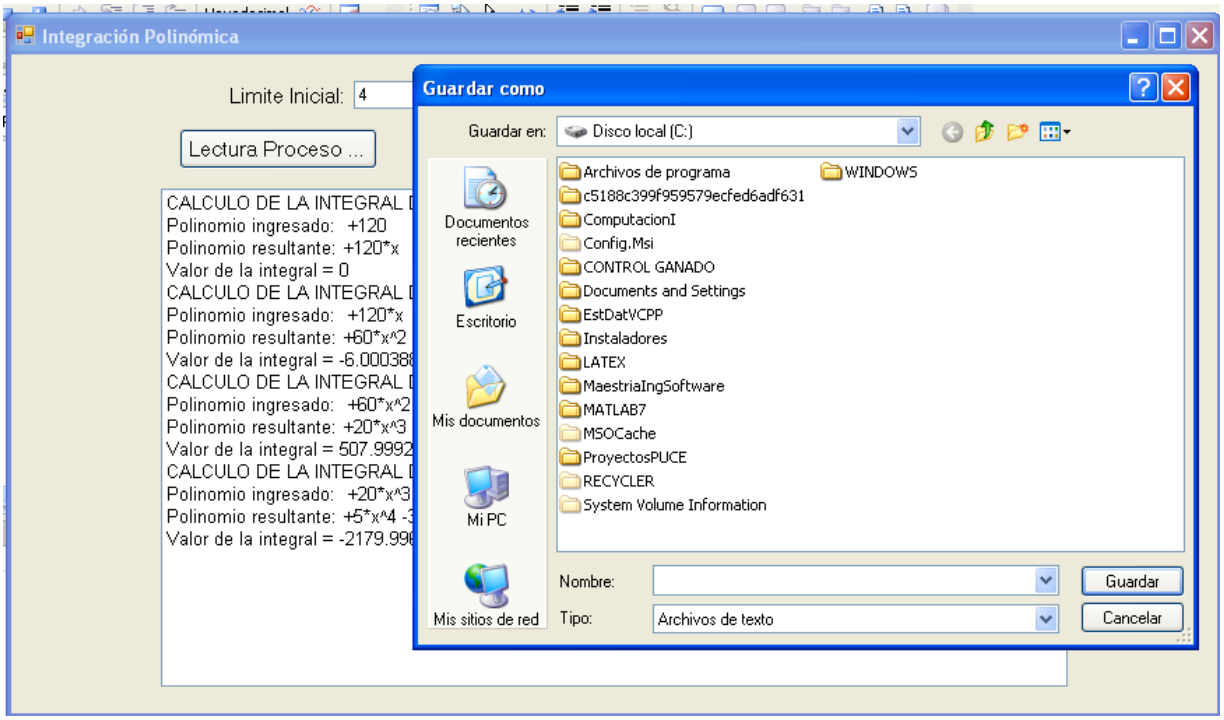
Las referencias a las clases COprPolinomios, CEntSalArchivos, CLectEscrMatrices, CIntegracion y CListBox, provienen de las definiciones de las respectiva clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:



En el archivo PolinomioInt.txt, constan los datos:
0
120





Errores en el calculo de integrales definidas.

Para determinar los errores cometidos en el calculo de integrales definidadas, se tomará como ejemplos la función a integrar f(x) = sen (x^2) y la muestra del ejemplo 7.10.
Tabulando los errores relativos para 12 y 996 elementos de f(x) = sen (x^2) para el intervalo de integración [0, 1] y [-1, 1] utilizando las series de Taylor y las reglas de Newton Cotes, se obtienen los siguientes resultados:

Errores relativos en la integración numérica para f(x) = sen(x^2), n=12, li=0, lf=1

Serie Taylor	Rectangulos	Trapecios	Simpson	Tres Octavos	Dos Cuarenta y cincoavos
0.0000060%	11.0984560%	0.2018570%	0.0012570%	0.0028340%	0.0000120%

Errores relativos en la integración numérica para f(x) = sen(x^2), n=12, li=-1, lf=1

Serie Taylor	Rectangulos	Trapecios	Simpson	Tres Octavos	Dos Cuarenta y cincoavos
0.0000080%	0.8111980%	0.8111980%	0.0202990%	0.0463060%	0.0006780%

Errores relativos en la integración numérica para f(x) = sen(x^2), n=996, li=0, lf=1

Serie Taylor	Rectangulos	Trapecios	Simpson	Tres Octavos	Dos Cuarenta y cincoavos
0.0000060%	0.1361250%	0.0000230%	0.0000060%	0.0000060%	0.0000060%

Errores relativos en la integración numérica para f(x) = sen(x^2), n=996, li=-1, lf=1

Serie Taylor	Rectangulos	Trapecios	Simpson	Tres Octavos	Dos Cuarenta y cincoavos
0.0000080%	0.0001090%	0.0001090%	0.0000080%	0.0000080%	0.0000080%

Para calcular el error relativo, se ha tomado como referencia a la función quad() de Matlab, misma que realiza el cálculo de integrales definidas.
De acuerdo a los resultados obtenidos, se puede concluir que para valores pequeños de n, los tres métodos más precisos son Serie Taylor, Dos cuarentaycincoavos y Tres octavos, en ese orden. Para valores grandes, se añade a esta lista el método de Simpson. Cuando f(x) tiende a ser una función simétrica, en precisión, la regla de los rectángulos iguala a la de los trapecios. Independientemente del método con excepción del de los rectángulos se obtiene mayor precisión cuando se toman más elementos en el cálculo.

Para el calculo del error relativo de la integral de la muestra del ejemplo 7.11, se ha considerado que la muestra tiende a la función $f(x) = e^x(\cos(2x) – 2\text{sen}(2x))$, cuya antiderivada es $F(x) = e^x\cos(2x)$.
Analizando los resultados puede apreciarse de que el error cometido al aplicar integración por interpolación de Newton y las reglas de tres octavos y dos cuarentaicincoavos para un número de elementos pequeño no supera el 0.053% y para un n grande de 0.046%, en todo caso la regla de dos cuarentaicincoavos es más precisa.

.

EJERCICIOS DEL CAPITULO VII

1.- Dada la función: f(x) = 2cos(3x) - 5e $^{-x}$ * sin(x), calcular la integral en el intervalo [0, 1] para 6 elementos aplicando las reglas de los rectángulos, trapecios, Simpson y tres octavos. Para 4 elementos aplicar la regla de dos cuarentaycincoavos

2.- Modificar los programas de los ejemplos 7.4 y 7.8 para implementar el ejercicio 1, desplegar resultados ingresando algunos intervalos.

3.- Deducir la serie de Taylor para el valor de la integral tomando los cinco términos de la serie indicada. $F(x)=\int_0^1 e^{x^2}\cos(x^2)dx$ calcular ; determinar primeros

4.- Implementar un programa en Matlab, que tenga como entradas la función f(x) a integrarse, el intervalo y el número de términos de la serie de Taylor y que calcule el valor de la integral, además el error absoluto y relativo entre el valor que produce Matlab y el calculado. Desarrollar para el f(x) del ejercicio 3, repetir el proceso ingresando valores del límite inicial y final en el intervalo [-7, 7] y del número de términos en el intervalo [3, 15].

5.- Dada la siguiente muestra de ventas (en miles de USD) de un producto en ocho meses del año:

Mes	1	2	3	4	5	9	10	12
Venta	85	93	61	73	110	91	60	87.5

calcular la integral, aplicando interpolación a través de Polinomios de Lagrange para las ordenadas y las reglas de ocho tercios y dos cuarentaicincoavos, debe completar el cálculo de las ventas en los meses que no se incluyen en la tabla. Sugerencia: hacer los cambios pertinentes al programa del ejemplo 7.10.

6.- Implementar un programa en Matlab y otro en Visual C#, que tenga como entradas la muestra a integrarse, y el número de elementos que forman F(x) y que calcule el valor de la integral aplicando las reglas de tres octavos y de dos cuarentaicincoavos e interpolación mediante los polinomios de Lagrange. Desarrollar para la muestra del ejercicio 5, repetir el proceso ingresando el número de elementos en el intervalo [11, 1000]. El programa debe calcular apropiadamente el número de puntos.

7.- Dado el polinomio f(x) = 60x - 8 calcular: $F_1(x) = \int_1^3 \int_3^2 f(x) dx dx$ y

$F_2(x) = \int_0^2 \int_4^4 \int_1^3 \int_3^2 f(x) dx dx dx dx$

8.- Implementar un programa en Matlab y otro en Visual C#, que tenga como entradas la función a integrarse f(x), y el número de elementos que forman F(x) y que calcule el valor de la integral aplicando el método de Romberg. Desarrollar para la función del ejercicio 3, repetir el proceso ingresando el número de elementos en el intervalo [3, 1000]. El programa debe calcular apropiadamente el número de puntos.

9.- Deducir la regla de dos cuarentaicincoavos para el cálculo de integrales definidas. Sugerencia: utilizar los polinomios de Lagrange para cuatro puntos.

8

RESOLUCIÓN DE ECUACIONES DIFERENCIALES DE PRIMER ORDEN

CAPITULO VIII

RESOLUCION DE ECUACIONES DIFERENCIALES DE PRIMER ORDEN.

Generalidades

Las ecuaciones diferenciales de primer orden tienen su aplicación en la física, quimica, mecánica, economía entre otros.

En este capítulo se resolverán ecuaciones diferenciales de primer orden de la forma:
$f'(x) = g(x; f(x)), (x_0; f(x_0)), y = f(x)$ (8.1)
Ejemplos:
1) $y' = (x - y) / (x + y), (x_0; f(x_0))$
2) $y' = -3y / (2x), (1; 1)$
3) $y' = x / y, (0; 2)$
4) $y' = y, (2; e^2)$

La fórmula (8.1), también puede adoptar la siguiente forma:
$e(x, y)dx + h(x, y)dy = 0, (x_0; f(x_0)), y = f(x)$ (8.2)
Ejemplos:
1) $(5x^2y - xy)dx + (2x^3y^2 + x^3y^4)dy = 0, (x_0; f(x_0))$
2) $ydx - xdy = 0, (1; 2)$
3) $e^x\cos(2x)ydx + y^3dy = 0, (x_0; f(x_0))$

Los temas que se revisarán en este capítulo son:

- Resolución de ecuaciones diferenciales de primer orden usando los métodos de Euler, Taylor, Euler Modificado, Predictor - Corrector y Runge - Kutta.
- Errores en la resolución de ecuaciones diferenciales de primer orden.

Resolución de ecuaciones diferenciales de primer orden:

Método de Euler:

Se basa en el principio de diferencias en el que $\Delta f(x) = f(x + h) – f(x)$ (8.3)
donde h es la longitud de paso en x.
Por otra parte: $\Delta f(x) / h = g(x; f(x))$ o $\Delta f(x) = hg(x; f(x))$ (8.4).
Sustituyendo (8.4) en (8.3) y despejando f(x + h):
$f(x + h) = f(x) + hg(x; f(x))$ (8.5).
Discretizando (8.5):

$$y_{k+1} = y_k + hg(x_k; y_k) \quad (8.6)$$

La fórmula (8.6) es la de recurrencia para la resolución de ecuaciones diferenciales de primer orden por el método de Euler.

Algoritmo para el método de Euler

Entradas: g(x, y), x0, y0, xn, n.
Donde g(x, y) es la función de la ecuación diferencial, (x0, y0) es el punto inicial, xn abscisa correspondiente a la solución de la ecuación diferencial, n es el numero de elementos que se requieren para encontrar la solución de la ecuación diferencial.
Salidas: yn
Donde yn es la solución numérica de la ecuación diferencial.
Proceso:
a) h = (xn - x0) / n
b) xk = x0
c) yk = y0
d) f = g(xk, yk)
e) yk = yk + h*f
f) xk = xk + h
g) Repetir desde el paso (d) hasta que xk >= xn
h) Desplegar yn

Ejemplo 8.1

Calcular f(2), para y' = x / y, (0; 2), usando el método de Euler y ocho elementos.

Solución:

g(x, y) = x / y; x0 = 0; y0 = 2; xn = 2; n = 8; h = (2 - 0) / 8 = 0.25; el resto del proceso se detalla en la siguiente tabla:

k	x_k	$g(x_k, y_k)$	y_k
0	0		2
1	0.25	0	2
2	0.5	0.125	2.03125
3	0.75	0.24615385	2.09278846
4	1	0.35837354	2.18238185
5	1.25	0.45821496	2.29693558
6	1.5	0.54420333	2.43298642
7	1.75	0.61652625	2.58711798
8	2	0.67642837	2.75622507

Por tanto yn = 2.75622507; si se compara con la solución exacta $y^2 - x^2 = 4$, es decir,

$$y = f(x) = \pm\sqrt{4 + x^2}\,; f(2) = \pm\sqrt{4 + 2^2} = \pm\sqrt{8} = \pm 2.82842712$$

Método de Taylor:

Se basa en la utilización de la fórmula (6.2), en este caso se puede sustituir en la fórmula anterior f(x+h) por y_{k+1}, f(x) por y_k, de manera que:

$$y_{k+1} = y_k + hf'(x) + \frac{h^2}{2!} f''(x) + \frac{h^3}{3!} f'''(x) + \frac{h^4}{4!} f^{(4)}(x) + ... (8.7)$$

De acuerdo al orden de la derivada en (8.7), se obtienen las fórmulas de Taylor para la resolución de ecuaciones diferenciales. Asi se tiene que coinciden el método de Euler con el método de Taylor de orden uno.

Método de Taylor de orden dos:

Tomando los tres primeros términos de (8.7), se tiene que:

$$y_{k+1} = y_k + hg(x,y) + \frac{h^2}{2!} f''(x) = y_k + hg(x,y) + \frac{h^2}{2} \left(\frac{\partial g(x,y)}{\partial x} + g(x,y) \frac{\partial g(x,y)}{\partial y}\right) (8.8)$$

La fórmula (8.8) es la de recurrencia para la resolución de ecuaciones diferenciales por el método de Taylor.

Algoritmo para el método de Taylor

Entradas: g(x, y) , gx(x, y) , gy(x, y), x0, y0, xn, n.
Donde gx(x, y) es la derivada parcial de g(x, y) respecto de x; gy(x, y) es la derivada parcial de g(x, y) respecto de y.
Salidas: yn
Proceso:
a) h = (xn - x0) / n
b) xk = x0
c) yk = y0
d) f = g(xk, yk)
e) a = gx(xk, yk)
f) b = gy(xk, yk)
g) yk = yk + h*f + h^2*(a + b*f)/2
h) xk = xk + h
i) Repetir desde el paso (d) hasta que xk >= xn
j) Desplegar yn

Ejemplo 8.2

Calcular f(2), para y' = x / y, (0; 2), usando el método de Taylor de segundo orden y ocho elementos.

Solución:

g(x, y) = x / y; gx(x, y) = 1 / y; gy(x, y) = -x / y^2; x0 = 0; y0 = 2; x1 = 2; n = 8; h = (2 - 0) / 8 = 0.25; el resto del proceso se detalla en la siguiente tabla:

k	x_k	$g(x_k, y_k)$	$\delta g(x, y)/\delta x$	$\delta g(x, y)/\delta y$	y_k
0	0				2
1	0.25	0	0.5	0	2.015625
2	0.5	0.12403101	0.49612403	-0.06153476	2.06213663
3	0.75	0.24246696	0.48493392	-0.11758045	2.13790755
4	1	0.3508103	0.46774707	-0.16409049	2.24022723
5	1.25	0.44638329	0.44638329	-0.19925804	2.36577253
6	1.5	0.52836863	0.42269491	-0.22333873	2.5110739
7	1.75	0.59735398	0.39823599	-0.23788786	2.67285727
8	2	0.65473006	0.37413146	-0.24495511	2.84823139

Por tanto yn = 2.84823139

Método de Euler Modificado:

A este método también se lo conoce con el nombre de Predictor - Corrector de Euler Se basa en tomar el promedio de dos iteraciones consecutivas del método de Euler; es decir:

$y_{k+1} = y_k + h(y_k + y_{k+1}) / 2$ (8.9); en la primera iteración se determina el valor predictor

$$p_{k+1} = y_k + hg(x_k; y_k) \quad (8.10)$$

y en la segunda se determina el valor corrector dado por:

$$c_{k+1} = y_{k+1} = y_k + \frac{h}{2}(g(x_k; y_k) + g(x_{k+1}; p_{k+1})) \quad (8.11)$$

La fórmula (8.11) es la de recurrencia para la resolución de ecuaciones diferenciales por el método de Euler modificado.

Algoritmo para el método de Euler Modificado

Entradas: g(x, y), x0, y0, xn, n.
Salidas: yn
Proceso:
a) h = (xn - x0) / n
b) xk = x0
c) yk = y0
d) f = g(xk, yk)
e) pk1 = yk + h*f
f) xk = xk + h

g) yk = yk + h*(f + g(xk, pk1)) / 2
h) Repetir desde el paso (d) hasta que xk >= xn
i) Desplegar yn
Ejemplo 8.3

Calcular f(2), para y' = x / y, (0; 2), usando el método de Euler modificado y ocho elementos.
Solución:

g(x, y) = x / y; x0 = 0; y0 = 2; x1 = 2; n = 8; h = (2 - 0) / 8 = 0.25; el resto del proceso se detalla en la siguiente tabla:

k	x_k	$g(x_k, y_k)$	p_{k+1}	$g(x_{k+1}, p_{k+1})$	y_k
0	0				2
1	0.25	0	2	0.125	2.015625
2	0.5	0.12403101	2.04663275	0.24430372	2.06166684
3	0.75	0.24252221	2.12229739	0.35339062	2.13615595
4	1	0.35109796	2.22393044	0.44965435	2.23624998
5	1.25	0.4471772	2.34804428	0.53235793	2.35869188
6	1.5	0.52995477	2.49118057	0.60212416	2.50020174
7	1.75	0.59995159	2.65018964	0.6603301	2.65773695
8	2	0.65845493	2.82235069	0.70862916	2.82862246

Por tanto yn = 2.82862246

Método de Runge – Kutta de orden dos (R – K – 2):

Este método se basa en el de Euler Modificado, en el que se introduce una variable de ajuste b,
(0 < b < 1), con lo que las fórmulas para la resolución quedan de la siguiente manera:

$$m_1 = hg(x_k; y_k)$$

$$m_2 = hg(x_k + \frac{h}{2b}; y_k + \frac{m_1}{2b})$$

$$y_{k+1} = y_k + (1 - b)m_1 + bm_2; (8.12)$$

Algoritmo para el método R – K - 2

Entradas: g(x, y), x0, y0, xn, n, b.
Donde b es la variable de ajuste.
Salidas: yn
Proceso:
a) h = (xn - x0) / n

b) xk = x0
c) yk = y0
d) m1 = h*g(xk, yk)
e) m2 = h*g(xk + h/2/b, yk + m1/2/b)
f) yk = yk + (1 - b)*m1 + b*m2
g) xk = xk + h
h) Repetir desde el paso (d) hasta que xk >= xn
i) Desplegar yn

Ejemplo 8.4

Calcular f(2), para y' = x / y, (0; 2), usando el método R-K-2 y ocho elementos.

Solución:

g(x, y) = x / y; x0 = 0; y0 = 2; x1 = 2; n = 8; h = (2 - 0) / 8 = 0.25; b = 0.5; el resto del proceso se detalla en la siguiente tabla:

k	x_k	y_k	m1	m2	y_{k+1}
0	0	2	0	0.03125	2.015625
1	0.25	2.015625	0.03100775	0.06107593	2.06166684
2	0.5	2.06166684	0.06063055	0.08834766	2.13615595
3	0.75	2.13615595	0.08777449	0.11241359	2.23624998
4	1	2.23624998	0.1117943	0.13308948	2.35869188
5	1.25	2.35869188	0.13248869	0.15053104	2.50020174
6	1.5	2.50020174	0.1499879	0.16508253	2.65773695
7	1.75	2.65773695	0.16461373	0.17715729	2.82862246
8	2	2.82862246			

Por tanto yn = 2.82862246
Como se puede observar, se obtienen los mismos resultados en este método y en el de Euler modificado para cuando b = 0.5. Si se ajusta b al valor de 0.465, manteniendo los valores en el resto de entradas, los resultados que se producen son:

k	x_k	y_k	m1	m2	y_{k+1}
0	0	2	0	0.03360215	2.015625
1	0.25	2.015625	0.03100775	0.0633023	2.06164972
2	0.5	2.06164972	0.06063106	0.09037065	2.13610968
3	0.75	2.13610968	0.08777639	0.11419194	2.2361693
4	1	2.2361693	0.11179833	0.13461494	2.35857736
5	1.25	2.35857736	0.13249512	0.15181825	2.50005773
6	1.5	2.50005773	0.14999654	0.16615824	2.65756946
7	1.75	2.65756946	0.16462411	0.1780523	2.82843768
8	2	2.82843768			

Por tanto yn = 2.82843768, el cual es un valor más cercano al exacto.

Método de Runge – Kutta de orden cuatro (R – K – 4):

Este método se basa en su similar de orden dos, en el que se realizan cuatro iteraciones, con lo que las fórmulas para la resolución quedan de la siguiente manera:

$$m_1 = hg(x_k; y_k)$$

$$m_2 = hg(x_k + \frac{h}{2}; y_k + \frac{m_1}{2})$$

$$m_3 = hg(x_k + \frac{h}{2}; y_k + \frac{m_2}{2})$$

$$m_4 = hg(x_k + h; y_k + m_3)$$

$$y_{k+1} = y_k + \frac{m_1 + 2m_2 + 2m_3 + m_4}{6}; (8.13)$$

Algoritmo para el método R – K - 4

Entradas: g(x, y), x0, y0, xn, n.
Salidas: yn
Proceso:
a) h = (xn - x0) / n
b) xk = x0
c) yk = y0
d) m1 = h*g(xk, yk)
e) m2 = h*g(xk + h/2, yk + m1/2)
f) m3 = h*g(xk + h/2, yk + m2/2)
g) xk = xk + h
h) m4 = h*g(xk, yk + m3)
i) yk = yk + (m1 + 2*m2 + 2*m3 + m4)/6
j) Repetir desde el paso (d) hasta que xk >= xn
k) Desplegar yn

Ejemplo 8.5

Calcular f(2), para y' = x / y, (0; 2), usando el método de R-K-4 y tres elementos.

Solución:

g(x, y) = x / y; x0 = 0; y0 = 2; x1 = 2; n = 3; h = (2 - 0) / 3 = 2/3; el resto del proceso se detalla en la siguiente tabla:

k	x_k	y_k	m1	m2	m3	m4	y_{k+1}
0	0	2	0	0.11111111	0.10810811	0.21082621	2.10821077
1	0.66666667	2.10821077	0.21081594	0.30116598	0.29514278	0.36985357	2.40375861
2	1.33333333	2.40375861	0.36979124	0.42922345	0.42435216	0.47145725	2.8284919
3	2	2.8284919					

Por tanto yn = 2.8284919

Implementación de los algoritmos en MATLAB y VISUAL C#:

Para cada lenguaje, se elabora un solo programa, desde el cual se ingresan el miembro derecho de la ecuación diferencial g(x, y) (solo para Matlab), el punto inicial (x0, y0), la abscisa donde se desea que se calcule la solución xn y el número de elementos, luego prueba a cada uno de los métodos para resolver la ecuación diferencial y la despliega por pantalla. Se describen a continuación una lista de funciones y subrutinas.

1) Una función que reciba los valores de x, y y calcule y devuelva g(x, y) (Para visual C#).
2) Una función que reciba los valores de x, y y calcule y devuelva la derivada parcial de g(x,y) respecto de x, gx(x, y) (Para visual C#).
3) Una función que reciba los valores de x, y y calcule y devuelva la derivada parcial de g(x,y) respecto de y, gy(x, y) (Para visual C#).
4) Una función que reciba un puntero a la función g(x, y), el punto inicial (x0, y0), la abscisa xn para la solución y el número de elementos y resuelve y devuelve el valor de la ecuación diferencial por el método de Euler.
5) Una función que reciba los punteros a las funciones g(x, y), gx(x, y), gy(x, y), el punto inicial (x0, y0), la abscisa xn para la solución y el número de elementos y resuelve y devuelve el valor de la ecuación diferencial por el método de Taylor.
6) Una función que reciba un puntero a la función g(x, y), el punto inicial (x0, y0), la abscisa xn para la solución y el número de elementos y resuelve y devuelve el valor de la ecuación diferencial por el método de Euler modificado.
7) Una función que reciba un puntero a la función g(x, y), el punto inicial (x0, y0), la abscisa xn para la solución, el número de elementos y la variable de ajuste y resuelve y devuelve el valor de la ecuación diferencial por el método de Runge - Kutta de orden dos.
8) Una función que reciba un puntero a la función g(x, y), el punto inicial (x0, y0), la abscisa xn para la solución y el número de elementos y resuelve y devuelve el valor de la ecuación diferencial por el método de Runge - Kutta de orden cuatro.

Archivos fuente en MATLAB:

```
% PrbEcDiferencial.m
clc; % Limpia pantalla
clear; % Libera espacio de variables
syms x y % Define como simbolo a x y
g = input('Ingresar la función g(x, y): ');
```

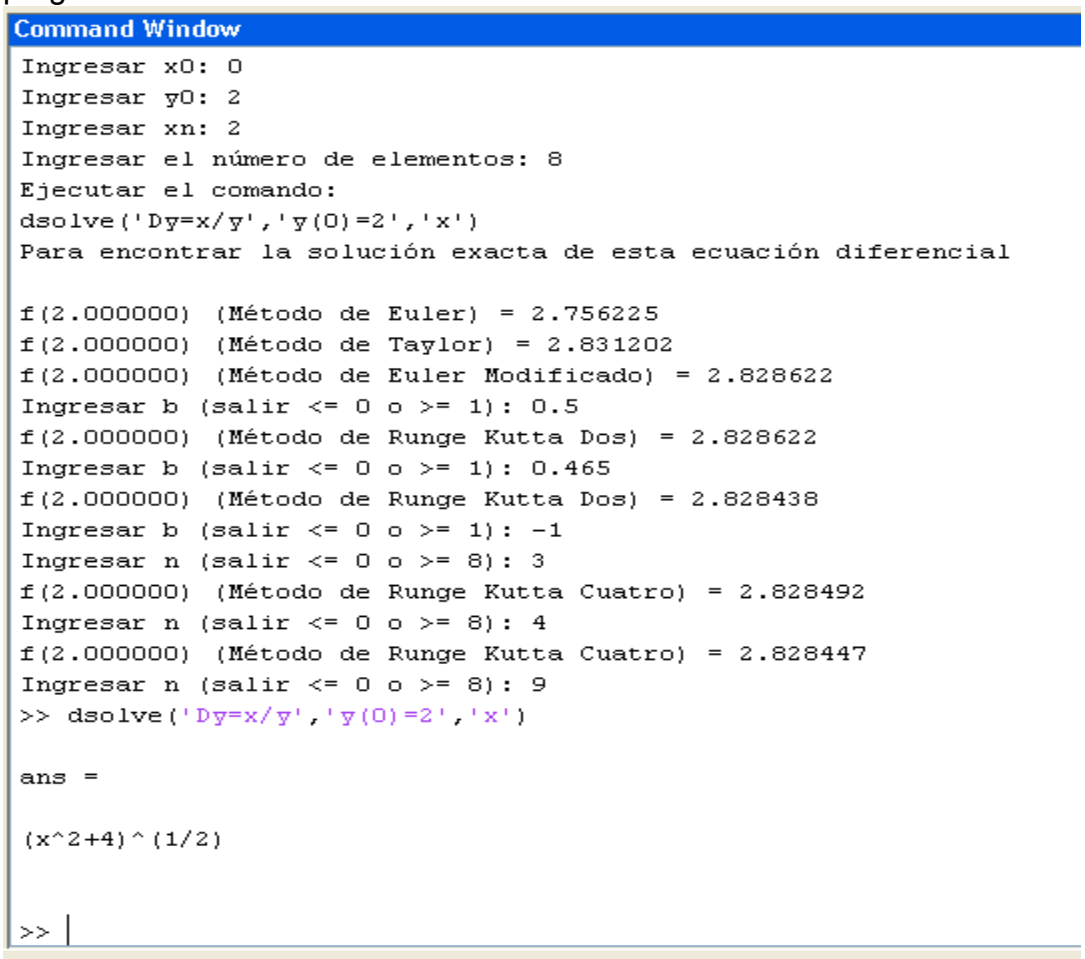
```
x = input('Ingresar x0: ');
y = input('Ingresar y0: ');
xn = input('Ingresar xn: ');
n = input('Ingresar el número de elementos: ');
sed = ['dsolve(',char(39), 'Dy=', char(g), char(39), ',', char(39), 'y(', num2str(x), ')=', num2str(y), char(39), ',', char(39), 'x', char(39),')'];
disp(sprintf('Ejecutar el comando:\n%s\nPara encontrar la solución exacta de esta ecuación diferencial\n', sed));
disp(sprintf('f(%f) (Método de Euler) = %f', xn, EcDifEuler(g, x, y, xn, n)));
disp(sprintf('f(%f) (Método de Taylor) = %f', xn, EcDifTaylor(g, x, y, xn, n)));
disp(sprintf('f(%f) (Método de Euler Modificado) = %f', xn, EcDifEulerModificado(g, x, y, xn, n)));
while (1)
  b = input('Ingresar b (salir <= 0 o >= 1): ');
  if (b <= 0 | b >= 1) break; end;
  disp(sprintf('f(%f) (Método de Runge Kutta Dos) = %f', xn, EcDifRungeKuttaDos(g, x, y, xn,n, b)));
end;
while (1)
  m = input(sprintf('Ingresar n (salir <= 0 o >= %d): ', n));
  if (m <= 0 | m >= n) break; end;
  disp(sprintf('f(%f) (Método de Runge Kutta Cuatro) = %f', xn, EcDifRungeKuttaCuatro(g, x, y, xn, m)));
end;

% EcDifEuler.m
function y = EcDifEuler(g, x0, y0, xn, n)
   h = (xn - x0) / n;
   syms x y; % Define como simbolo a x y
   x = x0;
   y = y0;
   for k = 1: n
     y = y + h * eval(g);
     x = x + h;
   end;

% EcDifTaylor.m
function y = EcDifTaylor(g, x0, y0, xn, n)
   h = (xn - x0) / n;
   syms x y; % Define como simbolo a x y
   derivadax = diff(g, x, 1); % Genera simbólicamente la derivada parcial de g respecto de x
   syms y x; % Define como simbolo a y x
   derivaday = diff(g, y, 1); % Genera simbólicamente la derivada parcial de g respecto de y
   x = x0;
```

```
  y = y0;
  for k = 1: n
    f = eval(g);
    a = eval(derivadax);
    b = eval(derivaday);
    y = y + h * f + h^2 * (a + b * f) / 2;
    x = x + h;
  end;

% EcDifEulerModificado.m
function y = EcDifEulerModificado(g, x0, y0, xn, n)
  h = (xn - x0) / n;
  syms x y; % Define como simbolo a x y
  x = x0;
  y = y0;
  for k = 1: n
    f = eval(g);
    pk1 = y + h * f;
    x = x + h;
    y1 = y;
    y = pk1;
    m = eval(g);
    y = y1 + h * (f + m) / 2;
  end;

% EcDifRungeKuttaDos.m
function y = EcDifRungeKuttaDos(g, x0, y0, xn, n, b)
  h = (xn - x0) / n;
  syms x y; % Define como simbolo a x y
  x = x0;
  y = y0;
  for k = 1: n
    m1 = h * eval(g);
    x1 = x;
    x = x + h/2/b;
    y1 = y;
    y = y + m1/2/b;
    m2 = h * eval(g);
    x = x1 + h;
    y = y1 + (1 - b) * m1 + b * m2;
  end;

% EcDifRungeKuttaCuatro.m
function y = EcDifRungeKuttaCuatro(g, x0, y0, xn, n)
  h = (xn - x0) / n;
  syms x y; % Define como simbolo a x y
```

```
x = x0;
y = y0;
for k = 1: n
  m1 = h * eval(g);
  x1 = x;
  x = x + h/2;
  y1 = y;
  y = y + m1/2;
  m2 = h * eval(g);
  y = y1 + m2/2;
  m3 = h * eval(g);
  x = x1 + h;
  y = y1 + m3;
  m4 = h * eval(g);
  y = y1 + (m1 + 2*m2 + 2*m3 + m4)/6;
end;
```

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa:



Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura 8.1. Como se puede apreciar, existen tres botones de opción para seleccionar los métodos; cinco cuadros de edición, mismos que permitirán ingresar los datos; un cuadro de lista, el cual permitirá presentar los resultados; y cuatro botones, mismos que tienen la funcionalidad:

- Procesar: Ejecuta los métodos para resolver la ecuación diferencial y despliega los resultados.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

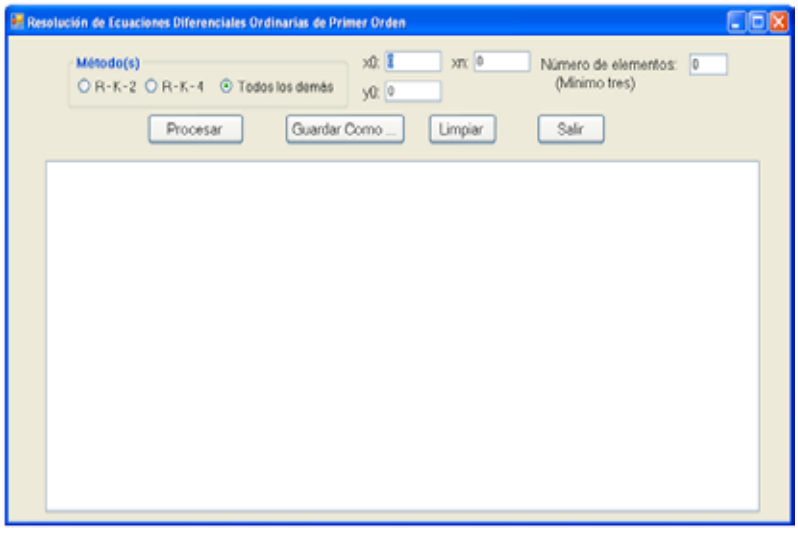


Figura 8.1 Resolución de ecuaciones diferenciales ordinarias de primer orden en Visual C#

Se describe a continuación el código fuente:

```
// Funciones miembro:
```

Evento Load del objeto Form1:

```
private void Form1_Load(object sender, EventArgs e)
{
   label4.Hide();
   txtB.Hide();
}
```

Evento CheckedChanged del radiobutton Runge - Kutta dos:

```
private void rbRk2_CheckedChanged(object sender, EventArgs e)
{
   label4.Show();
   txtB.Show();
}
```

Evento CheckedChanged del radiobutton Runge - Kutta cuatro:

```
private void rbRk4_CheckedChanged(object sender, EventArgs e)
{
    label4.Hide();
    txtB.Hide();
}
```

Evento CheckedChanged del radiobutton Todos los demás:

```
private void rbTd_CheckedChanged(object sender, EventArgs e)
{
    label4.Hide();
    txtB.Hide();
}
```

Funciones externas:

```
// Funcion a ser analizada:
static double g(double x, double y)
{
  return x / y;
}

// Solución de la EDO:
static double slg(double x)
{
  return Math.Sqrt(Math.Pow(x, 2) + 4);
}

// Derivada parcial de g respecto de x:
static double gx(double x, double y)
{
  return 1 / y;
}

// Derivada parcial de g respecto de y:
static double gy(double x, double y)
{
  return -1 / Math.Pow(y, 2);
}
```

Botón Procesar:

```
private void b_procesar_Click(object sender, EventArgs e)
{
  // Resuelve ecuaciones diferenciales ordinarias de primer orden.
  // Declaracion de variables
  double x0, y0, xn, yn, b, era, err, solexa;
  int n;  // Número de elementos.
  try
  {
    // Transferencia de captura de datos a variables:
    n = int.Parse(txtNun.Text);
```

```
// Validación:
if (n < 3 || n > 1000) return;
x0 = double.Parse(txtX0.Text);
y0 = double.Parse(txtY0.Text);
xn = double.Parse(txtXn.Text);
CEcDifOrdIOrden.n = n;
CEcDifOrdIOrden.x0 = x0;
CEcDifOrdIOrden.y0 = y0;
CEcDifOrdIOrden.xn = xn;
CEcDifOrdIOrden.pung = g;
solexa = slg(xn);
if (rbTd.Checked)
{
  CEcDifOrdIOrden.pungx = gx;
  CEcDifOrdIOrden.pungy = gy;
  yn = CEcDifOrdIOrden.Euler();
  era = Math.Abs(yn - solexa);
  err = era / Math.Abs(solexa) * 100;
  listBox1.Items.Add("f(" + xn.ToString() + ") (Método de Euler) = "
    +  Math.Round(yn, 7).ToString()
    + ", n = " + n.ToString()
    + ", Error absoluto = " + Math.Round(era, 7).ToString()
    + ", Error relativo = " + Math.Round(err, 7).ToString() + "%");
  yn = CEcDifOrdIOrden.Taylor();
  era = Math.Abs(yn - solexa);
  err = era / Math.Abs(solexa) * 100;
  listBox1.Items.Add("f(" + xn.ToString() + ") (Método de Taylor) = "
    + Math.Round(yn, 7).ToString()
    + ", n = " + n.ToString()
    + ", Error absoluto = " + Math.Round(era, 7).ToString()
    + ", Error relativo = " + Math.Round(err, 7).ToString() + "%");
  yn = CEcDifOrdIOrden.EulerModificado();
  era = Math.Abs(yn - solexa);
  err = era / Math.Abs(solexa) * 100;
  listBox1.Items.Add("f(" + xn.ToString() + ") (Método de Euler
    Modificado) = " + Math.Round(yn, 7).ToString()
    + ", n = " + n.ToString()
    + ", Error absoluto = " + Math.Round(era, 7).ToString()
    + ", Error relativo = " + Math.Round(err, 7).ToString() + "%");
}
else if (rbRk2.Checked)
{
  b = double.Parse(txtB.Text);
  if (b <= 0 || b >= 1) return;
  CEcDifOrdIOrden.b = b;
  yn = CEcDifOrdIOrden.RungeKuttaOrdenDos();
  era = Math.Abs(yn - solexa);
  err = era / Math.Abs(solexa) * 100;
  listBox1.Items.Add("f(" + xn.ToString() + ") (Método de Runge –
    Kutta Orden dos) = " + Math.Round(yn, 7).ToString()
    + ", n = " + n.ToString()
    + ", b = " + Math.Round(b, 7).ToString()
    + ", Error absoluto = " + Math.Round(era, 7).ToString()
    + ", Error relativo = " + Math.Round(err, 7).ToString() + "%");
}
```

```
      else
      {
        yn = CEcDifOrdIOrden.RungeKuttaOrdenCuatro();
        era = Math.Abs(yn - solexa);
        err = era / Math.Abs(solexa) * 100;
        listBox1.Items.Add("f(" + xn.ToString() + ") (Método de Runge –
          Kutta Orden cuatro) = " + Math.Round(yn, 7).ToString()
          + ", n = " + n.ToString()
          + ", Error absoluto = " + Math.Round(era, 7).ToString()
          + ", Error relativo = " + Math.Round(err, 7).ToString() + "%");
      }
    }
    catch (Exception ex)
    {
       MessageBox.Show(ex.Message);
    }
}
```

Botón Guardar Como …:

```
private void b_guardar_Click(object sender, EventArgs e)
{
  CListBox.lb = listBox1;
  CListBox.GuardarComo("c:\\");
}
```

Botón Limpiar:

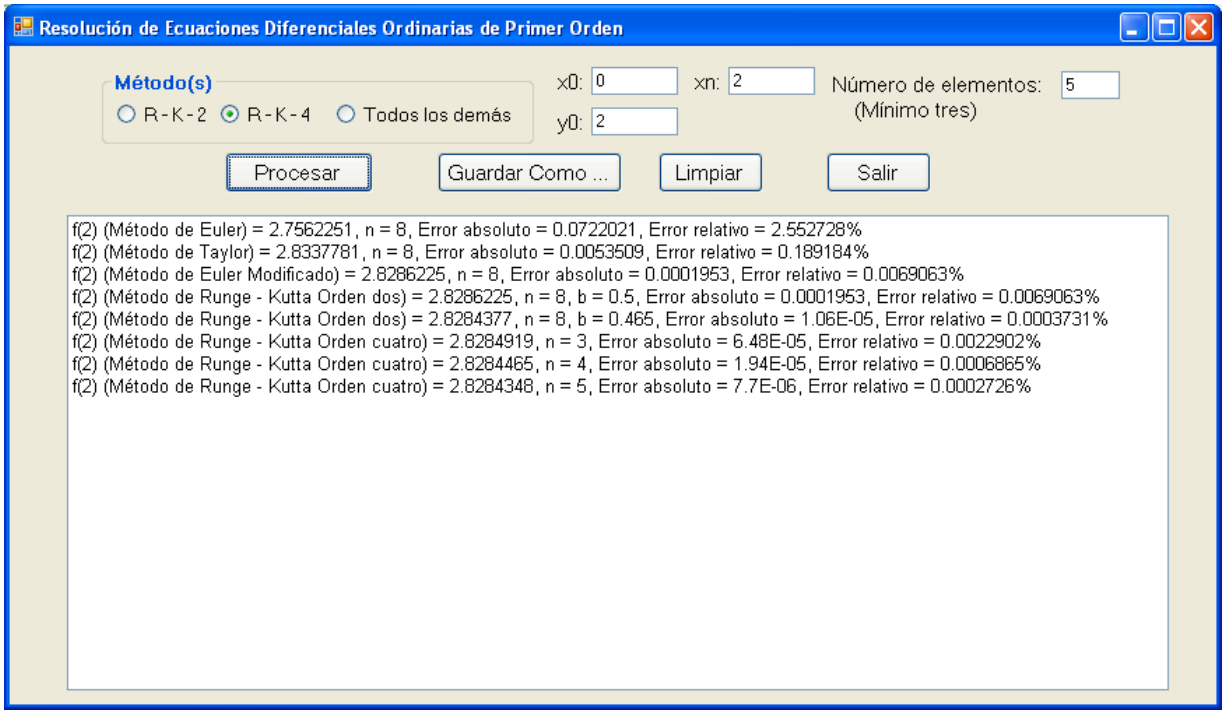
```
private void b_limpiar_Click(object sender, EventArgs e)
{
  CListBox.lb = listBox1;
  CListBox.Limpiar();
}
```

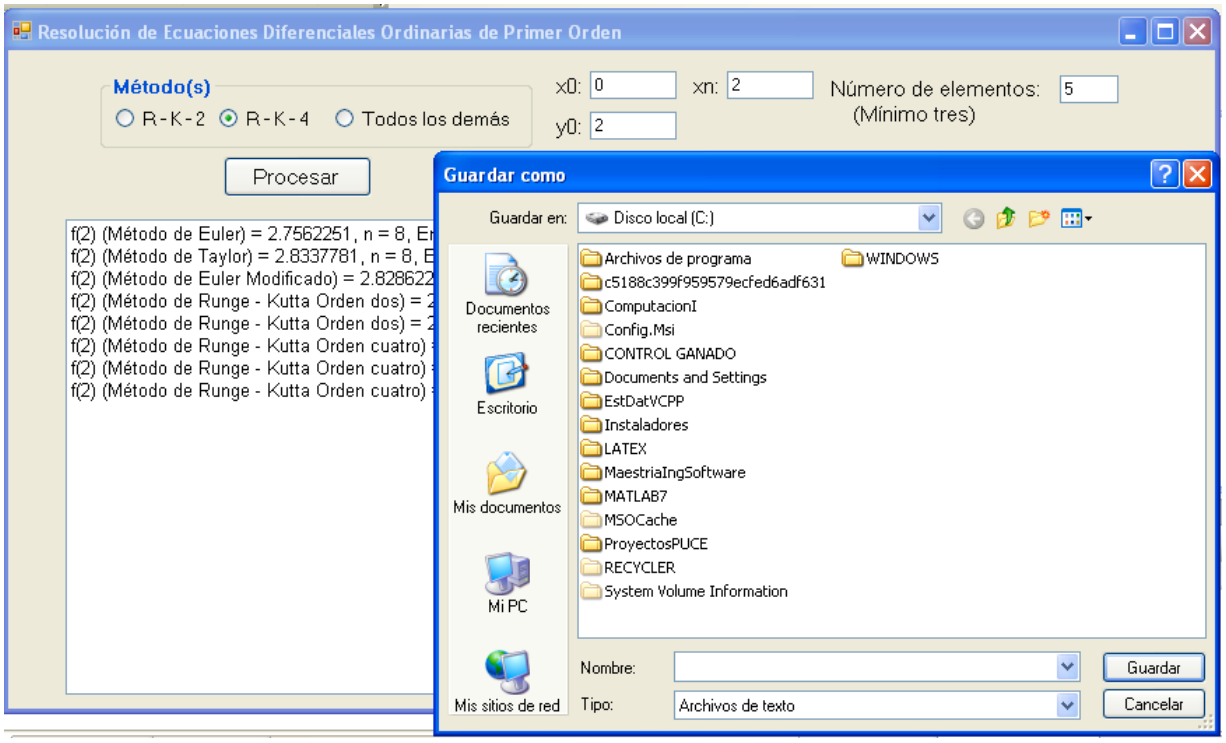
Botón Salir:

```
private void cb_salir_Click(object sender, EventArgs e)
{
  this.Close();
  this.Dispose();
}
```

Las referencias a las clases CEcDifOrdIOrden y CListBox, provienen de la definición de las respectivas clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe en el Apéndice B de este libro.

Se presenta a continuación los resultados que produce la ejecución de este programa:


Resolución de Ecuaciones Diferenciales Ordinarias de Primer Orden
Método(s)
R-K-2
R-K-4
Todos los demás
x0: 0
xn: 2
y0: 2
Número de elementos: 5
(Mínimo tres)
Procesar
Guardar Como ...
Limpiar
Salir
f(2) (Método de Euler) = 2.7562251, n = 8, Error absoluto = 0.0722021, Error relativo = 2.552728%
f(2) (Método de Taylor) = 2.8337781, n = 8, Error absoluto = 0.0053509, Error relativo = 0.189184%
f(2) (Método de Euler Modificado) = 2.8286225, n = 8, Error absoluto = 0.0001953, Error relativo = 0.0069063%
f(2) (Método de Runge - Kutta Orden dos) = 2.8286225, n = 8, b = 0.5, Error absoluto = 0.0001953, Error relativo = 0.0069063%
f(2) (Método de Runge - Kutta Orden dos) = 2.8284377, n = 8, b = 0.465, Error absoluto = 1.06E-05, Error relativo = 0.0003731%
f(2) (Método de Runge - Kutta Orden cuatro) = 2.8284919, n = 3, Error absoluto = 6.48E-05, Error relativo = 0.0022902%
f(2) (Método de Runge - Kutta Orden cuatro) = 2.8284465, n = 4, Error absoluto = 1.94E-05, Error relativo = 0.0006865%
f(2) (Método de Runge - Kutta Orden cuatro) = 2.8284348, n = 5, Error absoluto = 7.7E-06, Error relativo = 0.0002726%


Resolución de Ecuaciones Diferenciales Ordinarias de Primer Orden
Método(s)
R-K-2
R-K-4
Todos los demás
x0: 0
xn: 2
y0: 2
Número de elementos: 5
(Mínimo tres)
Procesar
Guardar como
Guardar en: Disco local (C:)
Documentos recientes
Escritorio
Mis documentos
Mi PC
Mis sitios de red
Archivos de programa
WINDOWS
c5188c399f959579ecfed6adf631
ComputacionI
Config.Msi
CONTROL GANADO
Documents and Settings
EstDatVCPP
Instaladores
LATEX
MaestriaIngSoftware
MATLAB7
MSOCache
ProyectosPUCE
RECYCLER
System Volume Information
Nombre:
Tipo: Archivos de texto
Guardar
Cancelar

Errores en la resolución de ecuaciones diferenciales de primer orden:

Para determinar los errores cometidos en la resolución de ecuaciones diferenciales ordinarias de primer orden, se tomará como ejemplo la función del miembro derecho g(x, y) = x/y en el punto inicial (0, 2), para xn = 2.
Tabulando los errores relativos para el método / nro. elementos / variable de ajuste (b), se obtienen los siguientes resultados:

Errores relativos en ecuaciones difernciales para g(x,y) = x/y, (0, 2), xn = 2

Euler/8	Taylor/8	Euler Modificado/ 8	R-K-2/8/0.5	R-K-2/8/0.465	R-K-4/3	R-K-4/5
2.5527280%	0.1891840%	0.0069063%	0.0069063%	0.0003731%	0.0022902%	0.0002726%

Errores relativos en ecuaciones difernciales para g(x,y) = x/y, (0, 2), xn = 2

Euler/100	Taylor/100	Euler Modificado/ 100	R-K-2/100/0.5	R-K-2/100/0.465	R-K-4/30
0.1969615%	0.0107034%	0.0000034%	0.0000034%	0.0000432%	0.0000002%

Para calcular el error relativo, se ha tomado como referencia la solución exacta del ejemplo 8.1.
De acuerdo a los resultados obtenidos, se puede concluir que para valores pequeños de n, el método más preciso el de Runge-Kutta de orden cuatro. Para valores grandes, se añaden a esta lista los métodos de Runge-Kutta de orden dos y de Euler Modificado. Para Runge-Kutta de orden dos, cuando n es pequeño, el método es más preciso cuando b < 0.5; en cambio cuando n es grande, es más preciso para b = 0.5. Independientemente del método con excepción del de los rectángulos se obtiene mayor precisión cuando se toman más elementos en el cálculo.

EJERCICIOS DEL CAPITULO VIII

1.- Dada la función: $g(x, y) = x^2y / (1 + x^3)$, y el punto inicial (1, 2), resolver la ecuación diferencial para 8 elementos y xn = 3 aplicando los métodos de Euler, Taylor, Euler Modificado, Runge - Kutta de orden dos y Runge - Kutta de orden cuatro. Comparar los resultados obtenidos con la solución exacta.

2.- Ejecutar el programa en Matlab para resolver ecuaciones diferenciales ordinarias de orden uno, dadas las condiciones el problema (1).

3.-Modificar el programa en Visual C# para resolver ecuaciones diferenciales ordinarias de orden uno, dadas las condiciones el problema (1).

APENDICE A

EJERCICIOS EN MATLAB

Ejercicio a.1:

Uso de la sentencia if:

Implementar un programa en Matlab para ingresar los lados de un triángulo en orden ascendente y determinar si lo es; en el caso de que lo sea, determinar y desplegar su tipo y calcular desplegando el perímetro y la superficie.

Solución:

Se representa un triángulo en la figura a.1:

Se tiene un triángulo si
a + b ≥ c
Y para determinar su tipo:
si a = c entonces
es un triángulo equilátero
caso contrario si (a = b) o
(b = c) entonces
es un triángulo isóceles
caso contrario
es un triángulo escaleno.

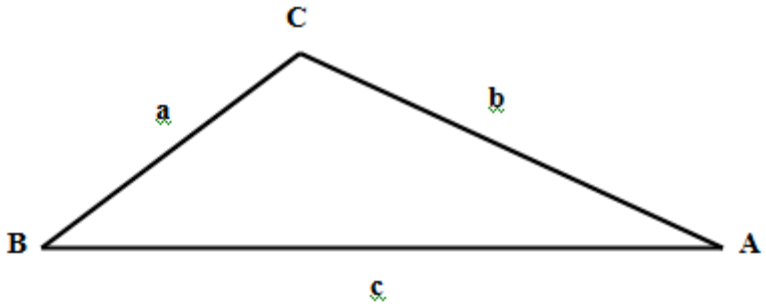


Figura a.1 Triángulo BAC

perímetro = a + b + c; s = perímetro / 2;

$$superficie = \sqrt{s\,(s-a)\,(s-b)\,(s-c)}$$

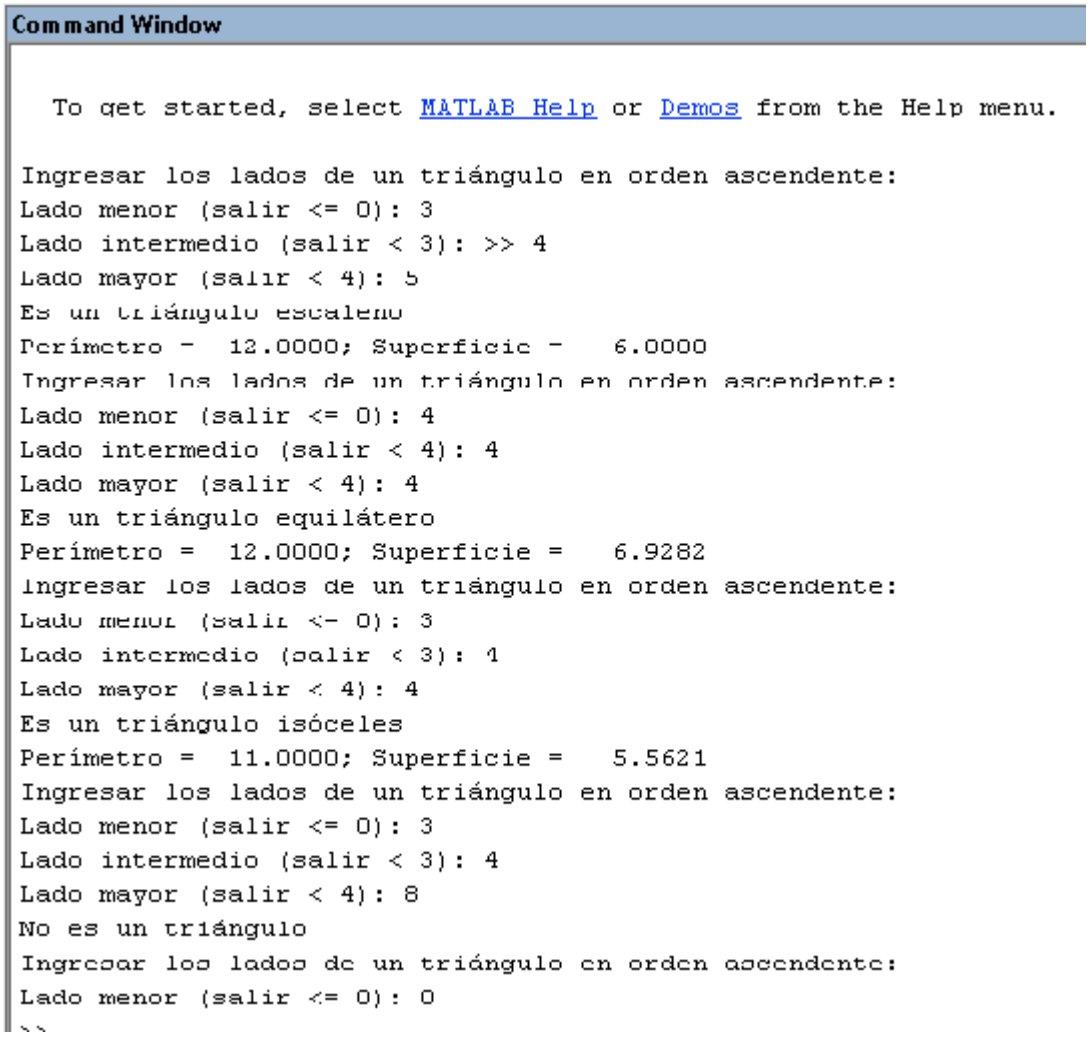
Archivo fuente en MATLAB:

```
% triangulo.m
while 1
 disp('Ingresar los lados de un triángulo en orden ascendente:');
 a = input('Lado menor (salir <= 0): ');
 if a <= 0
    break;
 end;
 b = input(sprintf('Lado intermedio (salir < %d): ', a));
 if b < a
    break;
 end;
 c = input(sprintf('Lado mayor (salir < %d): ', b));
 if c < b
    break;
 end;
 if  a + b >= c
    if a == c
       disp('Es un triángulo equilátero');
```

```
    elseif (a == b) || (b == c)
      disp('Es un triángulo isóceles');
    else
      disp('Es un triángulo escaleno');
    end;
    per = a + b + c;
    s = per / 2;
    sup = sqrt(s * (s - a) * (s - b) * (s - c));
    disp(sprintf('Perímetro = %8.4f; Superficie = %8.4f', per, sup));
  else
    disp('No es un triángulo');
  end
end
```

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa:



Ejercicio a.2:

Uso de ciclos:

Implementar un programa en Matlab para ingresar el ángulo en grados sexagesimales;
Si el ángulo se encuentra entre -180 y 180 grados, se convierte a radianes y se calcula el coseno trigonométrico de ese ángulo y se ingresa otro ángulo. El proceso se repite hasta cuando se ingresa un ángulo fuera del rango indicado.
Para el cálculo coseno trigonométrico, se usa la siguiente serie de Taylor:

$$Cos(x) = 1 - x^2 / 2! + x^4 / 4! - x^6 / 6! + \ldots$$

Donde x es el ángulo en radianes.
Para finalizar el cálculo tomar hasta el término >= 1e-5.

Solución:

Archivo fuente en MATLAB:

```
% coseno.m
precision = 1e-5;
while 1
  a = input('Ingresar un ángulo en grados sexagesimales (salir < -180 0 > 180):');
  if abs(a) > 180
     break;
  end;
  x = pi * a / 180;
  suma = 0;
  signo = 1;
  for par = 0 : 2: 1000
     %Calcular el factorial:
     fac = 1;
     for i = 2 : par
        fac = fac * i;
     end;
     termino = (x ^ par) / fac;
     suma = suma + signo * termino;
     signo = -signo;
     if termino < precision
        break;
     end
  end;
  disp(sprintf('Coseno(%8.4f) = %8.4f', a, suma));
end
```

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa:

```
Command Window

  To get started, select MATLAB Help or Demos from the Help menu.

Ingresar un ángulo en grados sexagesimales (salir < -180 0 > 180):-180
Coseno(-180.0000) =  -1.0000
Ingresar un ángulo en grados sexagesimales (salir < -180 0 > 180):0
Coseno(  0.0000) =   1.0000
Ingresar un ángulo en grados sexagesimales (salir < -180 0 > 180):45
Coseno( 45.0000) =   0.7071
Ingresar un ángulo en grados sexagesimales (salir < -180 0 > 180):90
Coseno( 90.0000) =   0.0000
Ingresar un ángulo en grados sexagesimales (salir < -180 0 > 180):180
Coseno(180.0000) =  -1.0000
Ingresar un ángulo en grados sexagesimales (salir < -180 0 > 180):200
>>
```

Ejercicio a.3:

Gráfico en tres dimensiones:

Implementar un programa en Matlab para ingresar la base y la altura de un cilindro y graficarlo

Solución:

Se usa el método de sobre posición de discos elementales.

Archivo fuente en MATLAB:

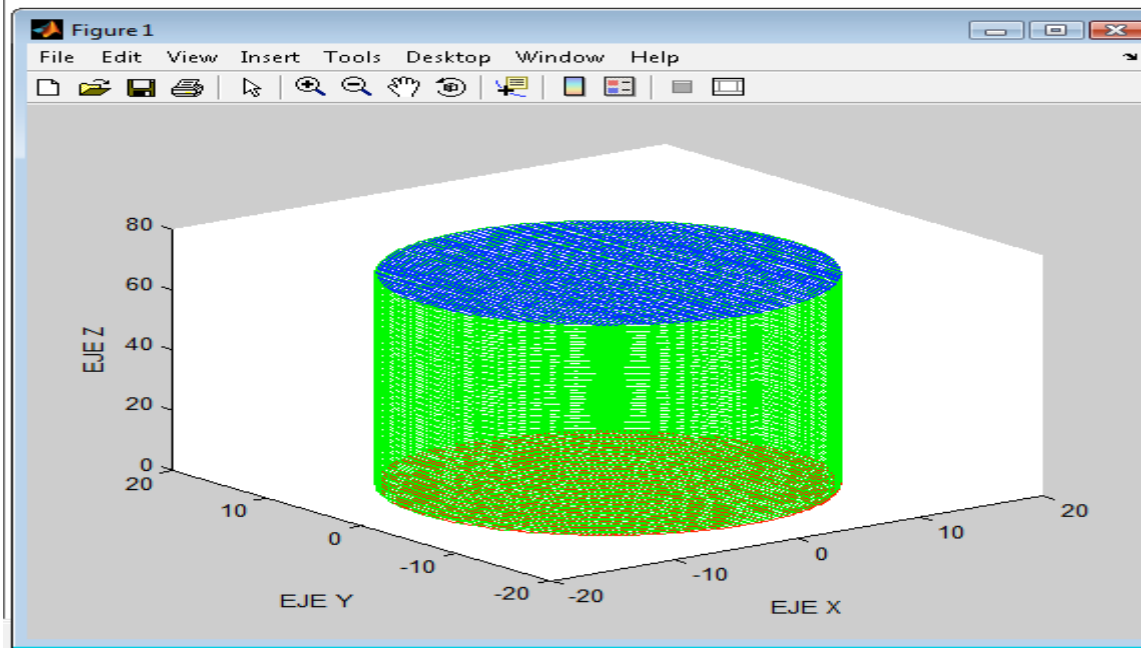
```
% cilindro.m
% PROGRAMA PARA INGRESAR DESDE EL TECLADO LAS DIMENSIONES DE UN
%  CILINDRO DE REVOLUCION (DIAMETRO DE LA BASE Y ALTURA DEL CILINDRO
%  Y DIBUJAR LA PARED LATERAL DEL CILINDRO DE COLOR VERDE,
%  LA BASE DE COLOR ROJO Y LA TAPA DE COLOR AZUL. SE USA EL METODO DE
%   LA SOBREPOSICION DE SEMIDISCOS
rango = 100;
d = input('INGRESE EL DIAMETRO DE LA BASE: ');
h = input('INGRESE LA ALTURA DEL CILINDRO: ');
r = d/2.;
x = -r : d/rango : r;
n = length(x);
% DIBUJO DE LA CARA LATERAL FRONTAL:
y = sqrt(r*r - x.*x);
% ARTIFICIO PARA QUE SE DEFINA LA DIMENSION DE z:
```

```
i = 0;
for j = 1 : n
  z(j) = i;
  i = i + h/rango;
end;
for i = 0 : h/rango : h
 for j = 1 : n
   z(j) = i;
 end;
 plot3(x, y, z, 'g');
 hold on;
end;
% DIBUJO DE LA CARA LATERAL POSTERIOR:
y = -sqrt(r*r - x.*x);
for i = 0 : h/rango : h
 for j = 1 : n
   z(j) = i;
 end;
 plot3(x, y, z, 'g');
 hold on;
end;
% DIBUJO DE LA BASE:
for i = -r : d/rango : r
   plot3([i, i], [-sqrt(r*r - i*i), sqrt(r*r - i*i)], [0, 0], 'r');
   hold on;
end;
% DIBUJO DE LA TAPA:
for i = -r : d/rango : r
   plot3([i, i], [-sqrt(r*r - i*i), sqrt(r*r - i*i)], [h, h], 'b');
   hold on;
end;
xlabel('EJE X');ylabel('EJE Y');zlabel('EJE Z')
```

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa:

```
INGRESE EL DIAMETRO DE LA BASE: 30
INGRESE LA ALTURA DEL CILINDRO: 70
>>
```



Ejercicio a.4:

Gráfico en tres dimensiones:

Implementar un programa en Matlab para ingresar el diámetro de una esfera y graficarla

Solución:

Se usa el método de sobre posición de discos elementales.

Archivo fuente en MATLAB:

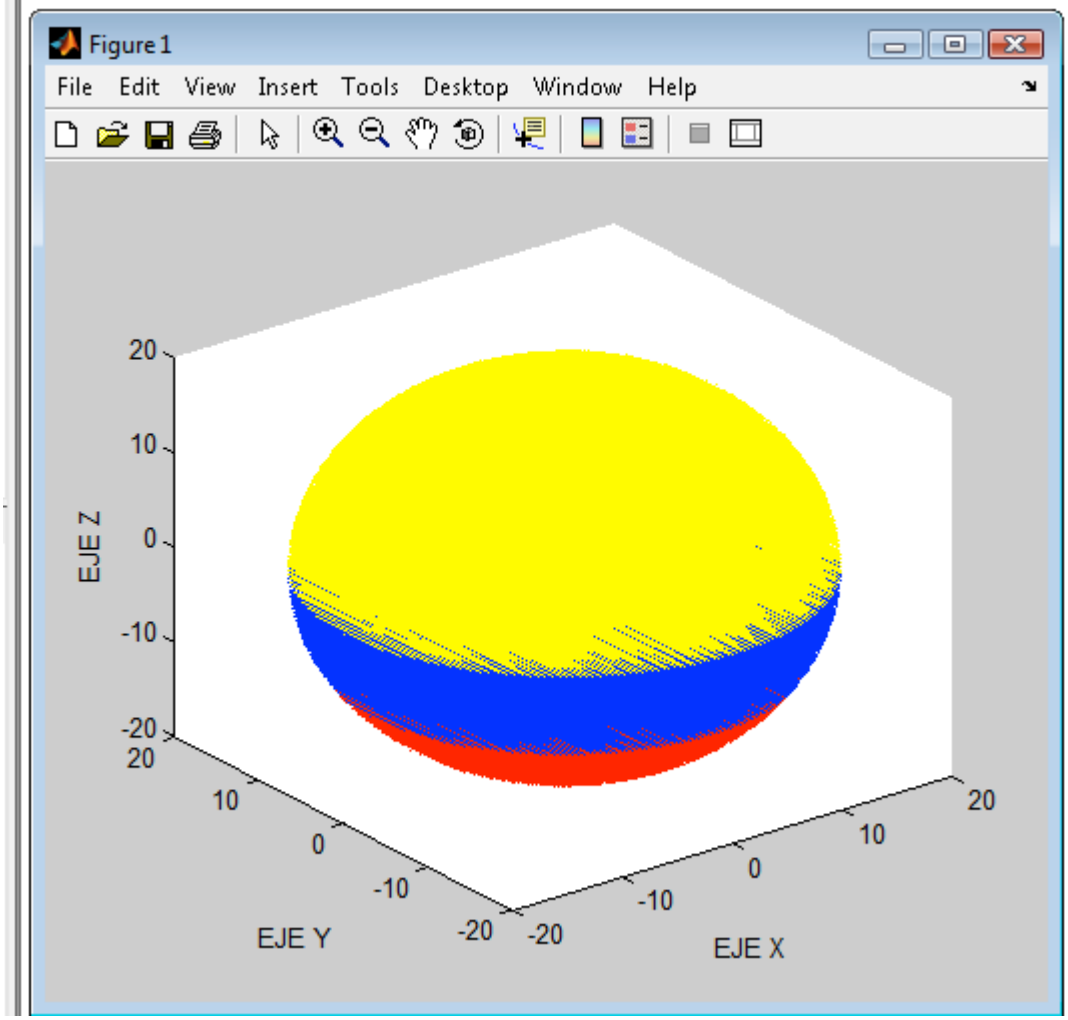
```
% esfera.m
% PROGRAMA PARA INGRESAR DESDE EL TECLADO EL DIAMETRO DE UNA
%  ESFERA Y DIBUJAR CUATRO PARTES DE LA MISMA CON LOS COLORES,
%  DE LA BANDERA NACIONAL. SE USA EL METODO DE LA SOBREPOSICION DE
%  DISCOS. ANTES DE PLOTEAR, ALMACENA EN ARREGLOS
resolhor = 100;
resolvert = 50;
d = input('INGRESE EL DIAMETRO DE LA ESFERA: ');
r = d/2.;
% GRAFICACION DE LA PARTE INFERIOR
k = 0;
for i = -r : d/resolvert : -r/2.
   rp = sqrt(r*r - i*i);
   l = 0;
   k = k + 1;
   for j = -rp : 2*rp/resolhor : rp
      l = l + 1;
      x(k, l, 1) = j;
```

```
    x(k, l, 2) = j;
    y(k, l, 1) = -sqrt(rp*rp - j*j);
    y(k, l, 2) = sqrt(rp*rp - j*j);
    z(k, l, 1) = i;
    z(k, l, 2) = i;
  end;
end;
plot3(x, y, z, 'r');
hold on;
% GRAFICACION DE LA PARTE CENTRAL
k = 0;
for i = -r/2. : d/resolvert : 0.
  rp = sqrt(r*r - i*i);
  l = 0;
  k = k + 1;
  for j = -rp : 2*rp/resolhor : rp
    l = l + 1;
    x1(k, l, 1) = j;
    x1(k, l, 2) = j;
    y1(k, l, 1) = -sqrt(rp*rp - j*j);
    y1(k, l, 2) = sqrt(rp*rp - j*j);
    z1(k, l, 1) = i;
    z1(k, l, 2) = i;
  end;
end;
plot3(x1, y1, z1, 'b');
hold on;
% GRAFICACION DE LA PARTE SUPERIOR
k = 0;
for i = 0 : d/resolvert : r
  rp = sqrt(r*r - i*i);
  l = 0;
  k = k + 1;
  for j = -rp : 2*rp/resolhor : rp
    l = l + 1;
    x2(k, l, 1) = j;
    x2(k, l, 2) = j;
    y2(k, l, 1) = -sqrt(rp*rp - j*j);
    y2(k, l, 2) = sqrt(rp*rp - j*j);
    z2(k, l, 1) = i;
    z2(k, l, 2) = i;
  end;
end;
plot3(x2, y2, z2, 'y');
hold on;
xlabel('EJE X');ylabel('EJE Y');zlabel('EJE Z')
```

Resultados obtenidos:

Se presenta a continuación los resultados que produce la ejecución de este programa:

```
INGRESE EL DIAMETRO DE LA ESFERA: 40
```



APENDICE B

EJERCICIOS EN VISUAL C#.NET

Ejercicio b.1:

Uso de la sentencia if:

Implementar el programa del ejercicio a.1

Solución:

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura b.1 Como se puede apreciar, existen seis cuadros de texto, tres que permiten ingresar los lados del triángulo en orden ascendente y los otros tres para desplegar los resultados; y dos botones, mismos que tienen la funcionalidad:

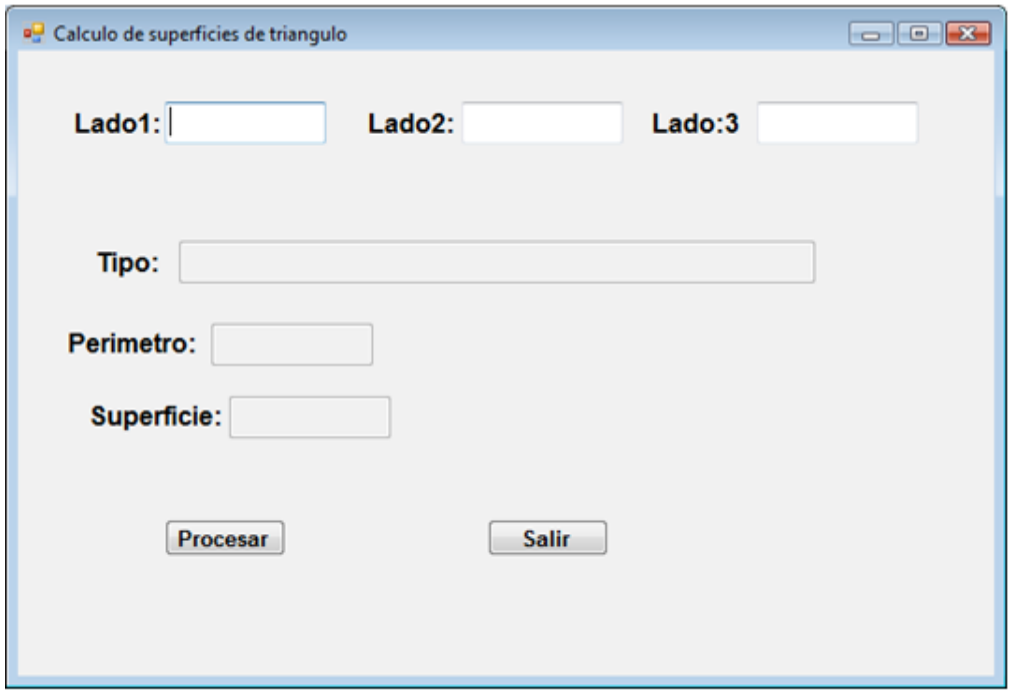


Figura b.1 Ventana para procesar triángulos en Visual C#

- Procesar: valida los lados del triángulo, determina y despliega el tipo de triángulo; calcula y despliega el perímetro y superficie.
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

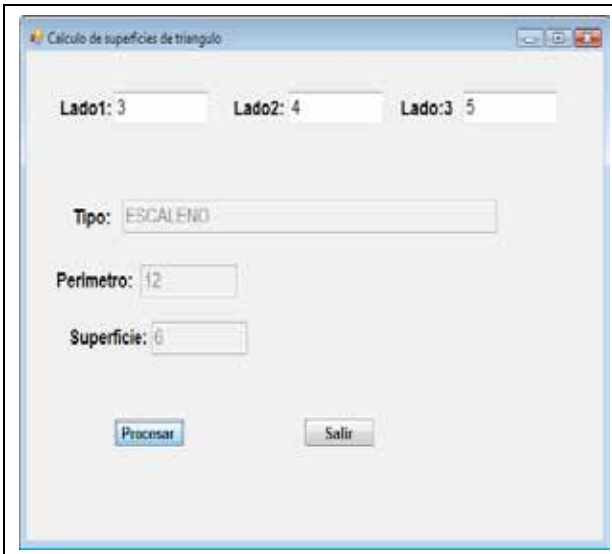
Botón Procesar:

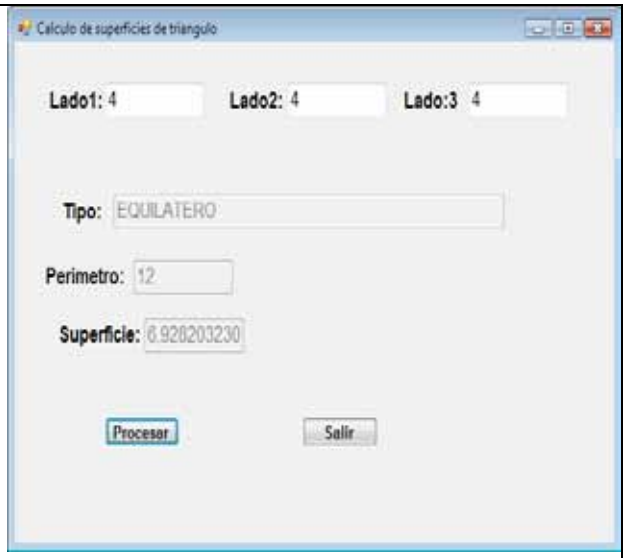
```
private void b_procesar_Click(object sender, EventArgs e)
{
   // Declaracion de variables
   double a, b, c; // Lados del triangulo
   double perimetro, superfifie, s;
   try
   {
      a = double.Parse(txtLado1.Text);
      b = double.Parse(txtLado2.Text);
      c = double.Parse(txtLado3.Text);
      if (a + b >= c)
      { // Es un triangulo
         // Determinar el tipo:
         if (a == c)
           txtTipo.Text = "EQUILATERO";
         else if (a == b || b == c)
           txtTipo.Text = "ISOCELES";
         else
           txtTipo.Text = "ESCALENO";
         // Calcular el perimetro y superficie:
         perimetro = a + b + c;
         s = perimetro / 2;
         superfifie = Math.Sqrt(s * (s - a) * (s - b) * (s - c));
         txtPerimetro.Text = perimetro.ToString();
         txtSuperficie.Text = superfifie.ToString();
      }
      else
        txtTipo.Text = "NO ES UN TRIANGULO";
   }
   catch (Exception ex)
   {
      MessageBox.Show(ex.Message);
   }
}
```

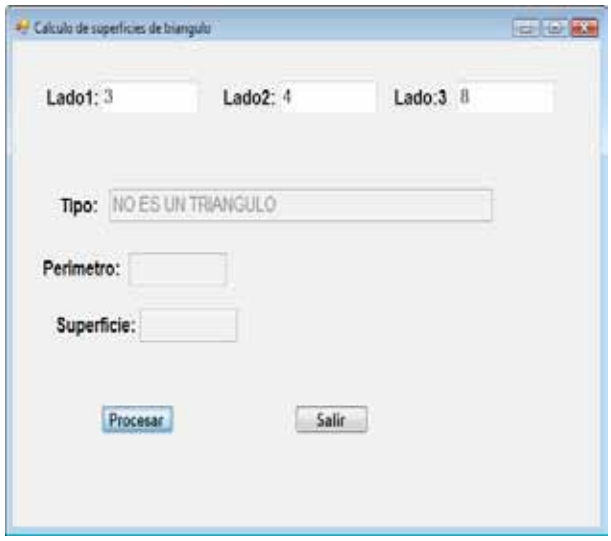
Botón Salir:

```
private void b_salir_Click(object sender, EventArgs e)
{
   this.Close();
   this.Dispose();
}
```

Se presenta a continuación los resultados que produce la ejecución de este programa:







Ejercicio b.2:

Uso de ciclos:

Implementar el programa del ejercicio a.2

Solución:

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene el diseño de la ventana de la figura b.2
Como se puede apreciar, existen cuatro cuadros de texto, uno que permite ingresar el ángulo en grados y los otros tres para desplegar los resultados; y dos botones, mismos que tienen la funcionalidad:

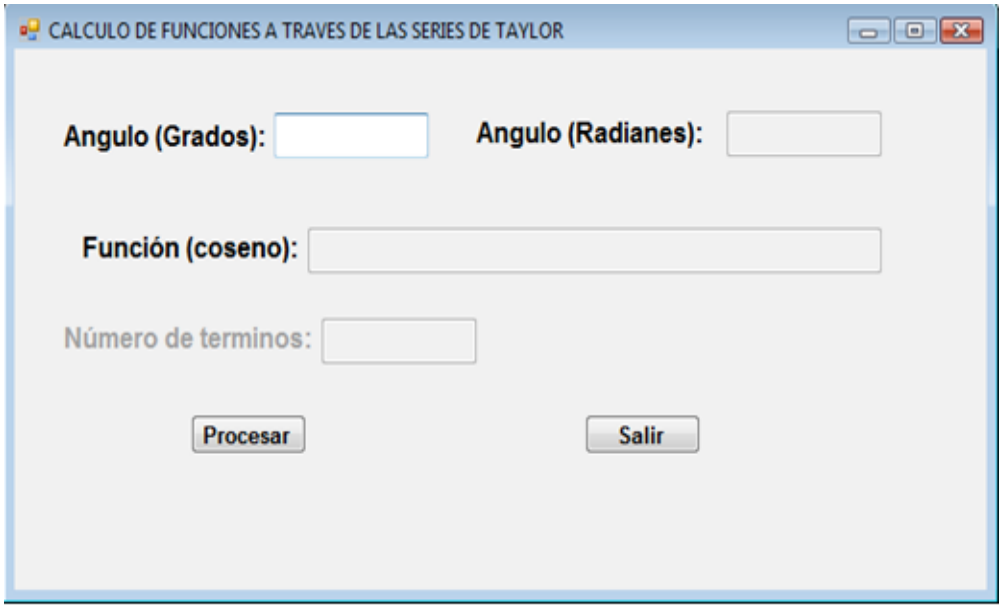


Figura b.2 Ventana para calcular el coseno mediante series de Taylor en Visual C#

- Procesar: valida ángulo en grados; calcula y despliega el ángulo en radianes, la función coseno y el número de términos utilizados para alcanzar la precisión requerida.
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Variables y funciones miembro de la clase Form1:

```
// Datos miembro:
private double precision = 1e-5;
private int numterminos;

// funciones miembro:
private double factorial(int num)
{
  double f = 1;
  for (int i = 2; i <= num; i++)
    f *= i;
  return f;
}

private double potencia(double base1, int ex)
{
   double p = 1;
```

```
    for (int i = 1; i <= ex; i++)
      p *= base1;
    return p;
}

private double coseno(double x)
{
  double suma = 0, termino;
  int signo = 1, par = 0;
  numterminos = 0;
  for (; ; )
  {
    termino = potencia(x, par) / factorial(par);
    suma += signo * termino;
    signo = -signo;
    par += 2;
    numterminos++;
    if (termino <= precision) return suma;
  }
  return suma;
}
```

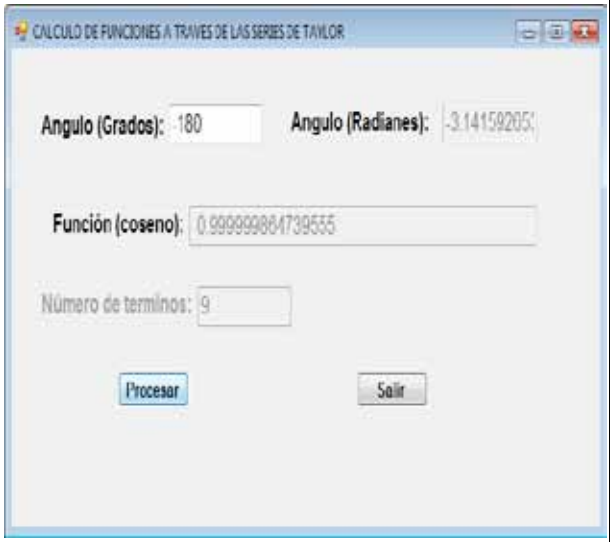
Botón Procesar:

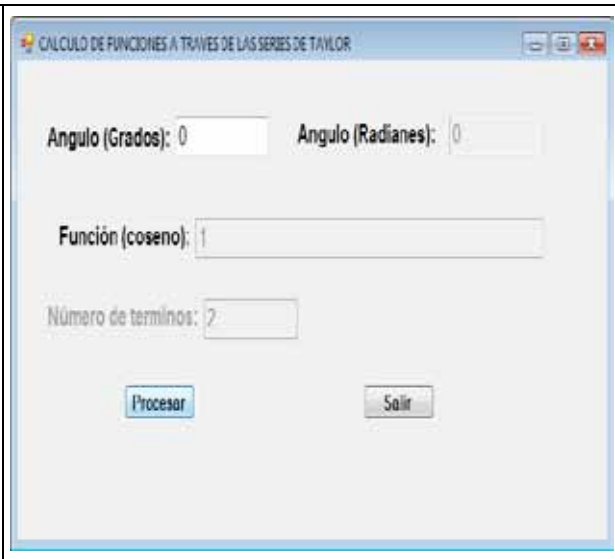
```
private void b_procesar_Click(object sender, EventArgs e)
{
  // Declaracion de variables
  double a; // Angulo en grados
  double x; // Angulo en radianes
  double resfuncion; // Resultado de la funcion
  try
  {
    // Transferir del textbox a memoria:
    a = double.Parse(txtAGrados.Text);
    // Convertir el angulo de grados sexagecimales a radianes:
    x = Math.PI * a / 180;
    // Calcular el coseno:
    resfuncion = coseno(x);
    // Desplegar resultados:
   txtARadianes.Text = x.ToString();
   txtFuncion.Text = resfuncion.ToString();
   txtNTer.Text = numterminos.ToString();
 }
 catch (Exception ex)
 {
    MessageBox.Show("Error " + ex.Message);
 }
}
```

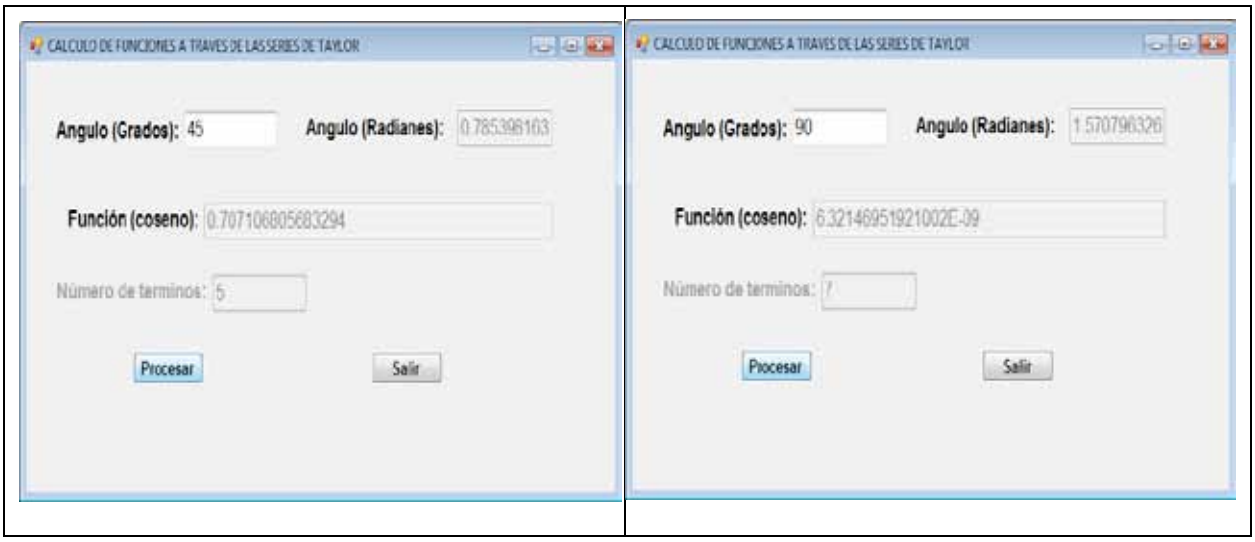
Botón Salir:

```
private void b_salir_Click(object sender, EventArgs e)
{
  this.Close();
  this.Dispose();
}
```

Se presenta a continuación los resultados que produce la ejecución de este programa:







Ejercicio b.3:

Uso de clases:

Implementar una versión del programa del ejercicio b.2, que utilice orientación a objetos.

Solución:

Archivos fuente en Visual C#:

Antes de describir el código fuente, en Visual C#, se ha implementado un proyecto tipo Windows Application, la cual contiene

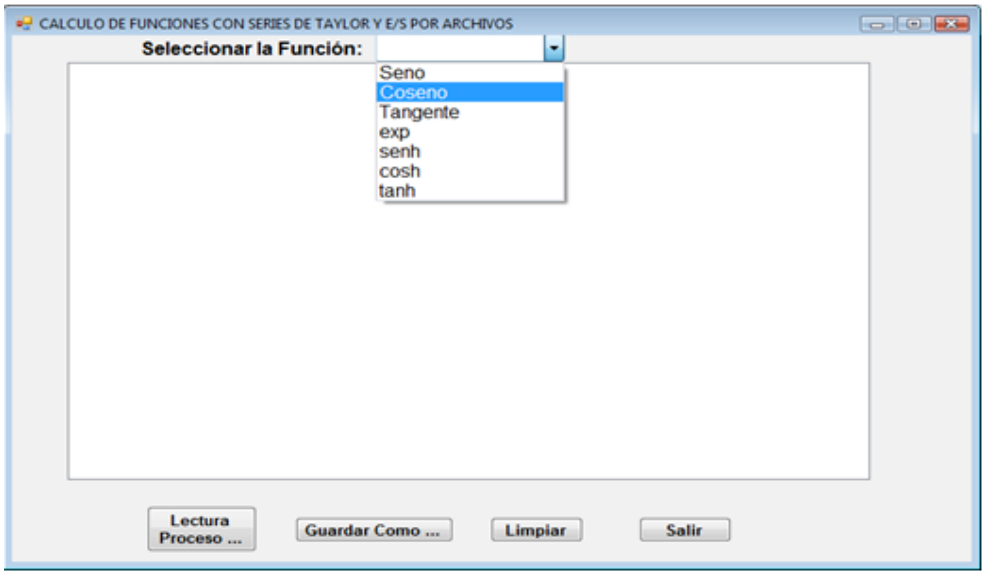


Figura b.3 Ventana para calcular funciones mediante series de Taylor en Visual C#

el diseño de la ventana de la figura b.3
Como se puede apreciar, existen un combinado para seleccionar la función a procesarse; un cuadro lista para desplegar los resultados; y cuatro botones, mismos que tienen la funcionalidad:

- Lectura Proceso…: abre una ventana para registrar el nombre del archivo el cual permite leer los datos de los ángulos en grados sexagesimales; convierte a radianes; ejecuta la función pertinente y despliega los resultados.
- Guardar Como …: Abre una ventana para registrar el nombre del archivo en el que se almacena permanentemente los resultados obtenidos en el cuadro lista.
- Limpiar: deja en blanco el cuadro lista; y
- Salir: Finaliza la ejecución del programa.

Se describe a continuación el código fuente:

Clase CSeriesTaylor:

```
using System;
using System.Collections.Generic;
using System.Text;

 class CSeriesTaylor
  {
     // Datos miembro:
     private double precision;

     // Funciones miembro:
     // Constructor:
     public CSeriesTaylor(double p)
     {
        precision = p;
     }

     public double factorial(int num)
     {
        double f = 1;
        for (int i = 2; i <= num; i++)
           f *= i;
        return f;
     }

     public double potencia(double base1, int ex)
     {
        double p = 1;
        for (int i = 1; i <= ex; i++)
           p *= base1;
        return p;
     }

     public double coseno(double x, ref int nter)
     {
        double suma = 0, termino;
```

```
        int signo = 1, par = 0;
        nter = 0;
        for (; ; )
        {
            termino = potencia(x, par) / factorial(par);
            suma += signo * termino;
            signo = -signo;
            par += 2;
            nter++;
            if (termino <= precision) return suma;
        }
        return suma;
    }

}
```

Botón Lectura Proceso…:

```
private void b_lectura_Click(object sender, EventArgs e)
{
  try
  {
    // Validar la seleccion de la funcion
    String sfun = cmbFuncion.Text;
    if (sfun == null || sfun.Length <= 0)
    {
      MessageBox.Show("Seleccionar la función a procesar");
      return;
    }
    // Abrir el archivo de entrada a traves del cuadro de dialogo:
    CEntSalArchivos objes = new CEntSalArchivos("c:\\");
    objes.AbrirArchivoEnt();
    // Leer el numero de entradas:
    int n = Int16.Parse(objes.LeerLinea());
    if (n > 0)
    {
      // Generar los resultados en el listbox:
      double ang, vfun = 0;
      int nt = -1;
      CSeriesTaylor obj = new CSeriesTaylor(1e-8);
      listBox1.Items.Add("CALCULO DE FUNCIONES MEDIANTE LAS SERIES DE TAYLOR");
      for (int i = 0; i < n; i++)
      {
        ang = double.Parse(objes.LeerLinea());
        switch (sfun)
        {
          case "Seno": break;
          case "Coseno":
            vfun = obj.coseno(Math.PI * ang / 180, ref nt);
            break;
          case "Tan": break;
          /* Poner resto de opciones */
          default:
            MessageBox.Show("Opción desconocida");
```

```
            return;
          }
          listBox1.Items.Add(sfun + "(" + ang.ToString() + ") = " + vfun.ToString() + ", Número de términos
= " + nt.ToString());
        }
        // Cerrar flujo de entrada:
        objes.CerrarLectura();
      }
    }
    catch (Exception ex)
    {
      MessageBox.Show(ex.Message);
    }
  }
```

Botón Guardar Como …:

```
  private void b_guardar_como_Click(object sender, EventArgs e)
      {
          CListBox.lb = listBox1;
          CListBox.GuardarComo("c:\\");
      }
```

Botón Limpiar:

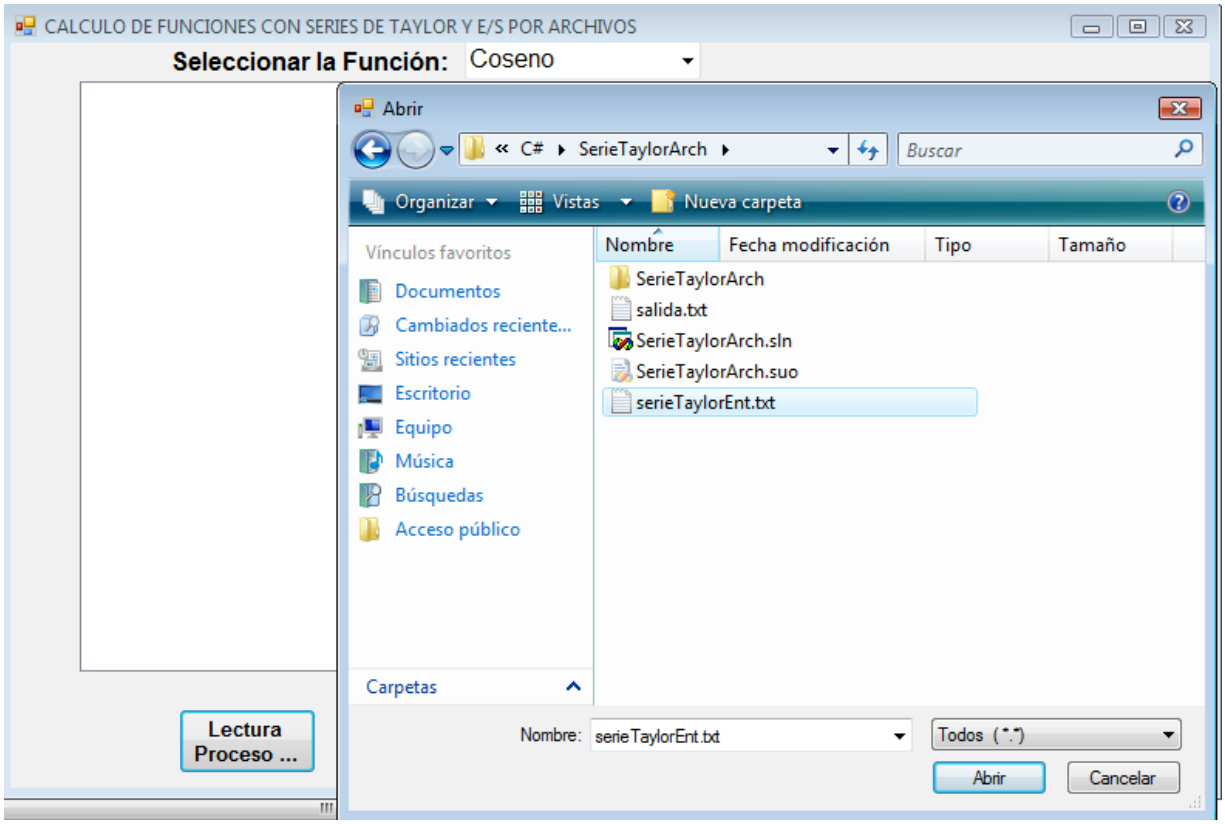
```
  private void b_limpiar_Click(object sender, EventArgs e)
      {
          CListBox.lb = listBox1;
          CListBox.Limpiar();
      }
```

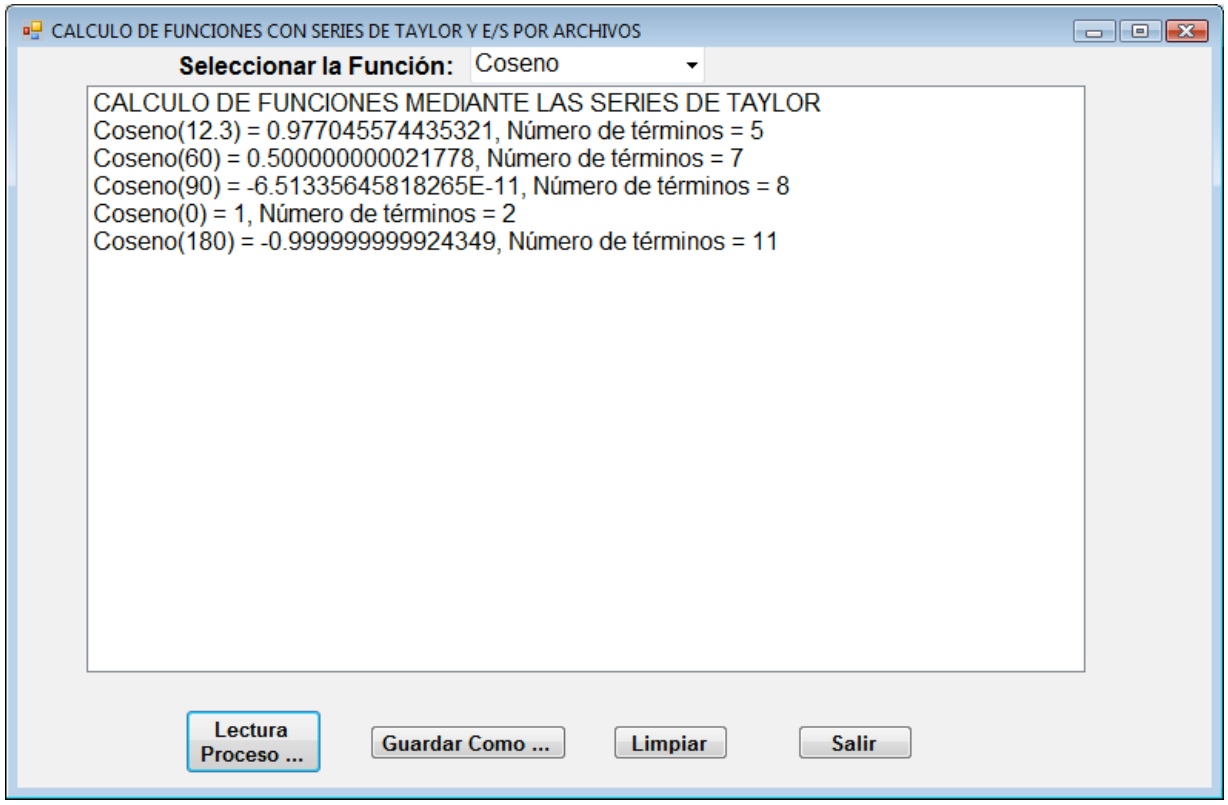
Botón Salir:

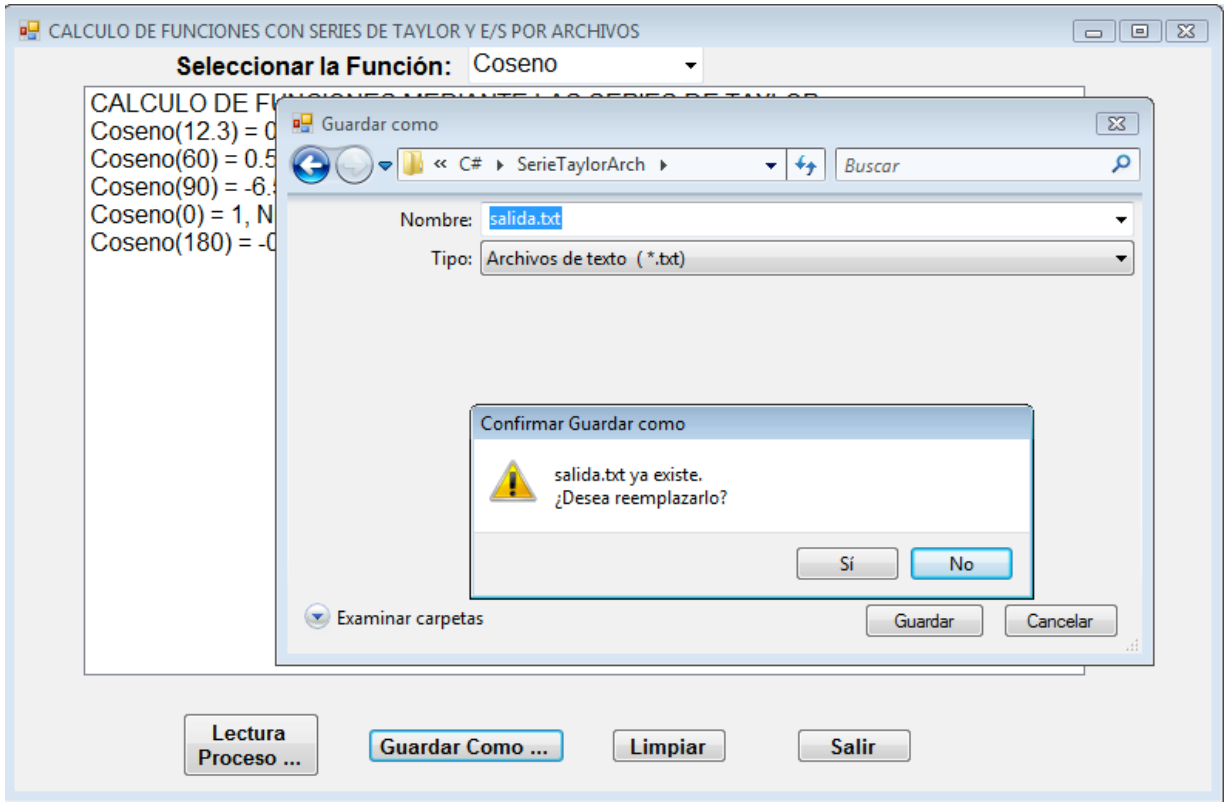
```
private void b_salir_Click(object sender, EventArgs e)
{
  this.Close();
  this.Dispose();
}
```

Las referencias a las clases CEntSalArchivos y CListBox, provienen de la definición de las respectivas clases incorporadas en la librería del archivo CLibComunClases.cs, el cual se describe más adelante, en este Apéndice.

Se presenta a continuación los resultados que produce la ejecución de este programa:




CALCULO DE FUNCIONES CON SERIES DE TAYLOR Y E/S POR ARCHIVOS
Seleccionar la Función:
Coseno
CALCULO DE FUNCIONES MEDIANTE LAS SERIES DE TAYLOR
Coseno(12.3) = 0.977045574435321, Número de términos = 5
Coseno(60) = 0.500000000021778, Número de términos = 7
Coseno(90) = -6.51335645818265E-11, Número de términos = 8
Coseno(0) = 1, Número de términos = 2
Coseno(180) = -0.999999999924349, Número de términos = 11
Lectura Proceso ...
Guardar Como ...
Limpiar
Salir


CALCULO DE FUNCIONES CON SERIES DE TAYLOR Y E/S POR ARCHIVOS
Seleccionar la Función:
Coseno
Guardar como
C# ▸ SerieTaylorArch ▸
Buscar
Nombre:
salida.txt
Tipo:
Archivos de texto (*.txt)
Confirmar Guardar como
salida.txt ya existe.
¿Desea reemplazarlo?
Sí
No
Examinar carpetas
Guardar
Cancelar
Lectura Proceso ...
Guardar Como ...
Limpiar
Salir

LIBRERÍA COMUN DE CLASES:

Esta librería ha sido usada por la mayor parte de programas descritos en este libro, desarrollados en Visual C#.NET; se la describe a continuación:

```
using System;
using System.Collections.Generic;
using System.Text;
// Importacion de cuadros de dialogo para lectura escritura:
using System.Windows.Forms;
// Importacion de objetos para flujos de entrada - salida:
using System.IO;

class idc // Clase para manejo de indices:
{
   // Convierte el índice matricial en lineal para una matriz simétrica:
   public static int ind(int i, int j) // i, j >= 0
   {
      return ((i + 1) * (i + 1) - i - 1 + 2 * (j + 1)) / 2 - 1;
   }

   // Convierte el índice matricial en lineal para una matriz simétrica en banda:
   public static int inb(int i, int j, int m)
   {
      return (i > m) ? ((2*i*m - 2*i + 2*j - m*m + m)/2) : ((i*i - i + 2*j)/2);
   }

   // Convierte el índice matricial en lineal para una matriz simétrica Skyline:
   public static int insk(int i, int j, int [] dr)
   {
      return dr[i] + j - i;
   }
}

class CEntSalArchivos
{
   // Datos Miembro:
   private String NPath, NomArchEnt, NomArchSal;
   private StreamReader stLectura;
   private StreamWriter stEscritura;

   // Metodos:

   // Constructor
   public CEntSalArchivos(String p)
   {
      NPath = p;
   }
```

```
// Metodo para abrir un cuadro de dialogo para lectura e inicializar stLectura
public void AbrirArchivoEnt()
{
   // Se define un cuadro de dialogo para apertura de archivos:
   OpenFileDialog ofd = new OpenFileDialog();
   // se establece un filtro:
   ofd.Filter = "Todos | *.*| Archivos de texto | *.txt| Archivos Word | *.doc";
   // Se fija por omisión los archivos txt:
   ofd.FilterIndex = 2;
   // Se establece como a este como directorio inicial:
   ofd.InitialDirectory = NPath;
   // Se abre el cuadro de diálogo para lectura:
   if (ofd.ShowDialog() == DialogResult.OK)
      // Se atrapan errores:
      try
      {
         // Obtener el nombre del archivo de entrada:
         NomArchEnt = ofd.FileName;
         stLectura = new StreamReader(NomArchEnt);
      }
      catch (Exception ex)
      {
         MessageBox.Show("Error: " + ex.Message);
      }
}

// Metodo para abrir un cuadro de dialogo para guardar como e inicializar stEscritura
public void AbrirArchivoSal()
{
   // Se define un cuadro de dialogo para apertura de archivos:
   SaveFileDialog sfd = new SaveFileDialog();
   // se establece un filtro:
   sfd.Filter = "Archivos de texto | *.txt| Archivos Word | *.doc";
   // Se fija por omisión los archivos txt:
   sfd.FilterIndex = 1;
   // Se establece como a este como directorio inicial:
   sfd.InitialDirectory = NPath;
   // Se abre el cuadro de diálogo para guardar como:
   if (sfd.ShowDialog() == DialogResult.OK)
      // Se atrapan errores:
      try
      {
         // Obtener el nombre del archivo de salida:
         NomArchSal = sfd.FileName;
         stEscritura = new StreamWriter(NomArchSal);
      }
      catch (Exception ex)
      {
         MessageBox.Show("Error: " + ex.Message);
      }
```

```
    }

    // Metodo para leer toda la linea desde stLectura
    public String LeerLinea()
    {
        return stLectura.ReadLine();
    }

    // Metodo para escribir toda la linea sobre stEscritura
    public void EscribirLinea(String s)
    {
        stEscritura.WriteLine(s);
    }

    // Metodo para escribir sin saltar la linea sobre stEscritura
    public void Escribir(String s)
    {
        stEscritura.Write(s);
    }

    // Metodo para cerrar el flujo de entrada:
    public void CerrarLectura()
    {
        stLectura.Close();
    }

    // Metodo para cerrar el flujo de salida:
    public void CerrarEscritura()
    {
        stEscritura.Close();
    }
}

class CListBox
{
    // Datos miembro:
    static public ListBox lb;

    // Funciones miembro:
    static public void Limpiar()
    {
        // Recuperar desde el ultimo elemento del listbox e ir eliminando:
        int n = lb.Items.Count;
        for (int i = n - 1; i >= 0; i--)
        {
            lb.Items.Remove(lb.Items[i].ToString());
        }
    }

    static public void GuardarComo(String spa)
```

```
    {
        try
        {
            // Abtir el archivo de salida a traves del cuadro de dialogo:
            CEntSalArchivos objes = new CEntSalArchivos(spa);
            objes.AbrirArchivoSal();
            // Recuperar la informacion que se encuentra en el listbox y guardarla en el archivo:
            int n = lb.Items.Count;
            for (int i = 0; i < n; i++)
            {
                objes.EscribirLinea(lb.Items[i].ToString());
            }
            // Cerrar flujo de salida:
            objes.CerrarEscritura();
        }
        catch (Exception ex)
        {
            MessageBox.Show(ex.Message);
        }
    }
}


// Declarar delegados (punteros a funcion)
public delegate double pf(double x);
public delegate double pfg(double x, double y);

class CRaicesReales
{
    // Datos miembro:
    static public double a; // Limite inferior del rango a analizar
    static public double b; // Limite superior del rango a analizar
    static public double xi; // Limite inferior del rango en el cambio de signo
    static public double xd; // Limite superior del rango en el cambio de signo
    static public double dx; // Incremento de la variable independiente
    static public int maxIter; // Maximo de iteraciones
    static public double precision; // Precisión requerida
    static public ListBox lb; // Lisbox para despliegue de resultados
    static public pf punf, punfp, punfm, punfpm;

    // Funcion que permite localizar el siguiente cambio de signo:
    static public double LocCambioSigno()
    {
        double YI, YD, P;
        xd = xi;
        YD = punf(xd);
        do
        {
            xi = xd;
            YI = YD;
```

```
            xd += dx;
            YD = punf(xd);
        }
        while ((P = YI * YD) > 0 && xd < b);
        if (P == 0)
            if (YI == 0)
                lb.Items.Add("RAIZ EXACTA (LOCALIZACION CAMBIO DE SIGNO) = " +
xi.ToString());
            else
            {
                lb.Items.Add("RAIZ EXACTA (LOCALIZACION CAMBIO DE SIGNO) = " +
xd.ToString());
                xd += dx;
            }
        return (P);
    }

    // Funcion que permite calcular una raiz por el metodo de la biseccion:
    static public double Biseccion()
    {
        int CITER = 0;
        double Xm, Ym, YI;
        String m = "";
        YI = punf(xi);
        lb.Items.Add("METODO DE LA BISECCION, RAIZ ENTRE " + xi.ToString() + " Y " +
xd.ToString());
        do
        {
            Xm = (xi + xd) / 2;
            Ym = punf(Xm);
            if (Ym * YI > 0)
            {
                xi = Xm;
                YI = Ym;
            }
            else xd = Xm;
        }
        while ((Ym = Math.Abs(Ym)) > precision && ++CITER < maxIter);
        if (Ym > precision)
            m = "NO SE ALCANZA LA PRESICION REQUERIDA: " + Ym.ToString();
        m += "RAIZ = " + Xm.ToString() + ", EN " + CITER.ToString() + " ITERACIONES";
        lb.Items.Add(m);
        return (Xm);
    }

    // Funcion que permite calcular una raiz por el metodo de aproximaciones sucesivas:
    static public double AproxSucesivas()
    {
        int CITER = 0;
        double Xm, Ym = (xi + xd) / 2, FP = 0, Prec;
```

```
        String m = "";
        lb.Items.Add("METODO DE APROXIMACIONES SUCESIVAS, RAIZ ENTRE " +
xi.ToString() + " Y " + xd.ToString());
        do
        {
            Xm = Ym;
            Ym = punfm(Xm);
        }
        while ((Prec = Math.Abs(Ym - Xm)) > precision && (FP = Math.Abs(punfpm(Xm))) <= 1
&& ++CITER < maxIter);
        if (FP > 1)
            m = "NO CUMPLE CON LA CONDICION DE CONVERGENCIA";
        else
        {
            if (Prec > precision)
                m = "NO SE ALCANZA LA PRESICION REQUERIDA: " + Prec.ToString();
            m += " RAIZ = " + Xm.ToString() + " EN " + CITER.ToString() + " ITERACIONES";
        }
        lb.Items.Add(m);
        return (Xm);
    }

    // Funcion que permite calcular una raiz por el metodo de aproximaciones sucesivas
modificado:
    static public double AproxSucesivasMod()
    {
        int CITER = 0;
        double YD = punfm(xi), YI, TGT, BETA;
        String m = "";
        lb.Items.Add("METODO DE APROXIMACIONES SUCESIVAS MODIFICADO, RAIZ
ENTRE " + xi.ToString() + " Y " + xd.ToString());
        do
        {
            YI = YD;
            YD = punfm(xd);
            TGT = (YD - YI) / (xd - xi);
            BETA = 1 / (1 - TGT);
            xi = xd;
            xd += BETA * (YD - xd);
        }
        while ((TGT = Math.Abs(xd - xi)) > precision && ++CITER < maxIter);
        if (TGT > precision)
            m = "NO SE ALCANZA LA PRESICION REQUERIDA: " + TGT.ToString();
        m += " RAIZ = " + xd.ToString() + " EN " + CITER.ToString() + " ITERACIONES";
        lb.Items.Add(m);
        return (xd);
    }

    // Funcion que permite calcular una raiz por el metodo de Newton - Raphson:
    static public double NewtonRaphson()
```

```
    {
        int CITER = 0;
        double Y;
        String m = "";
        lb.Items.Add("METODO DE NEWTON RAPHSON, RAIZ ENTRE " + xi.ToString() + " Y "
+ xd.ToString());
        xi = (xi + xd) / 2;
        do
            xi -= (Y = punf(xi)) / punfp(xi);
        while ((Y = Math.Abs(Y)) > precision && ++CITER < maxIter);
        if (Y > precision)
            m = "NO SE ALCANZA LA PRESICION REQUERIDA: " + Y.ToString();
        m += " RAIZ = " + xi.ToString() + " EN " + CITER.ToString() + " ITERACIONES";
        lb.Items.Add(m);
        return (xi);
    }

    // Funcion que permite calcular una raiz por el metodo de Regula - Falsi:
    static public double RegulaFalsi()
    {
        int CITER = 0;
        double YI = punf(xi),
          YD = punf(xd),
          Xm, Ym;
        String m = "";
        lb.Items.Add("METODO DE REGULA FALSI, RAIZ ENTRE " + xi.ToString() + " Y " +
xd.ToString());
        do
        {
            Xm = xi - YI * (xd - xi) / (YD - YI);
            Ym = punf(Xm);
            if (YI * Ym > 0)
            {
                xi = Xm;
                YI = Ym;
            }
            else
            {
                xd = Xm;
                YD = Ym;
            }
        }
        while ((Ym = Math.Abs(Ym)) > precision && ++CITER < maxIter);
        if (Ym > precision)
            m = "NO SE ALCANZA LA PRESICION REQUERIDA: " + Ym.ToString();
        m += " RAIZ = " + Xm.ToString() + " EN " + CITER.ToString() + " ITERACIONES";
        lb.Items.Add(m);
        return (Xm);
    }
```

```
    // Funcion que permite calcular una raiz por el metodo de la Secante:
    static public double Secante()
    {
        int CITER = 0;
        double YI = punf(xi), YD = punf(xd), Prec;
        String m = "";
        lb.Items.Add("METODO DE LA SECANTE, RAIZ ENTRE " + xi.ToString() + " Y " +
xd.ToString());
        do
        {
            xi = xi - YI * (xd - xi) / (YD - YI);
            YI = punf(xi);
        }
        while ((Prec = Math.Abs(YI)) > precision && ++CITER < maxIter);
        if (Prec > precision)
            m = "NO SE ALCANZA LA PRESICION REQUERIDA: " + YI.ToString();
        m += " RAIZ = " + xi.ToString() + " EN " + CITER.ToString() + " ITERACIONES";
        lb.Items.Add(m);
        return (xi);
    }

    static public double Muller()
    {
        int CITER;
        double x0 = xi, x1 = xd, x2, x3 = 0, y0, y1, y2, y3 = 0, h0, h1, d0, d1,
            a, b, c, rc, den;
        String m = "";
        lb.Items.Add("METODO DE MULLER, RAIZ ENTRE " + xi.ToString() + " Y " +
xd.ToString());
        x2 = (x0 + x1) / 2;
        y0 = punf(x0);
        y1 = punf(x1);
        y2 = punf(x2);
        for (CITER = 0; CITER < maxIter; CITER++)
        {
            h0 = x1 - x0;
            h1 = x2 - x1;
            d0 = (y1 - y0) / h0;
            d1 = (y2 - y1) / h1;
            a = (d1 - d0) / (h1 + h0);
            b = a * h1 + d1;
            c = y2;
            rc = Math.Sqrt(b * b - 4 * a * c);
            den = b - rc;
            if (Math.Abs(b + rc) > Math.Abs(b - rc))
                den = b + rc;
            x3 = x2 - 2 * c / den;
            y3 = punf(x3);
            if (Math.Abs(y3) <= precision)
                break;
```

```
            x0 = x1;
            x1 = x2;
            x2 = x3;
            y0 = y1;
            y1 = y2;
            y2 = y3;
        }
        y3 = Math.Abs(y3);
        if (y3 > precision)
          m = "No se alcanza la precisiòn requerida: " + y3.ToString();
        m += "RAIZ = " + x3.ToString() + ", EN " + CITER.ToString() + " ITERACIONES";
        lb.Items.Add(m);
        return (x3);
    }

}

class CLectEscrMatrices
{
    // Datos Miembro:
    static public int n; // Dimension del vector, matriz o sistema
    static public int m; // Ancho de banda de un sistema simetrico
    static public double[][] a; // Matriz de coeficientes
    static public double[] c; // vector matriz de coeficientes
    static public double[] b; // Vector de coeficientes o terminos independientes
    static public int[] Dir; // vector de direcciones de la matriz de coeficientes (Skyline)
    static public CEntSalArchivos obent; // Objeto de flujo de entrada
    static public ListBox lb; // listbox para despliegue de resultados

    // Metodos:
    static public void LeeVector()
    {
        for (int i = 0; i < n; i++)
            try
            {
                b[i] = double.Parse(obent.LeerLinea());
            }
            catch { }
    }

    static public void LeeMatriz()
    {
        int j;
        for (int i = 0; i < n; i++)
          for (j = 0; j < n; j++)
            try
            {
                a[i][j] = double.Parse(obent.LeerLinea());
            }
            catch { }
```

```
}

static public void EscribeVector(String ob)
{
   String cad = "";
   lb.Items.Add("VECTOR " + ob);
   for (int i = 0; i < n; i++)
      cad += Math.Round(b[i], 8).ToString() + " ";
   lb.Items.Add(cad);
}

static public void EscribeMatriz(String ob)
{
   String cad;
   int i, j;
   lb.Items.Add("MATRIZ " + ob);
   for (i = 0; i < n; i++)
   {
      cad = "";
      lb.Items.Add("FILA " + (i + 1).ToString());
      for (j = 0; j < n; j++)
         cad += a[i][j].ToString() + " ";
      lb.Items.Add(cad);
   }
}

public static void LeeMatrizSimetrica()
{
   int i, j;
   for (i = 0; i < n; i++)
      for (j = 0; j <= i; j++)
         try
         {
            c[idc.ind(i, j)] = double.Parse(obent.LeerLinea());
         }
         catch { }
}

public static void EscribeMatrizSimetrica(String ob)
{
   String cad;
   int i, j;
   lb.Items.Add("MATRIZ " + ob);
   for (i = 0; i < n; i++)
   {
      lb.Items.Add("FILA " + (i + 1).ToString());
      cad = "";
      for (j = 0; j < n; j++)
         if (i >= j)
            cad += c[idc.ind(i, j)].ToString() + " ";
```

```
            else
                cad += c[idc.ind(j, i)].ToString() + " ";
        lb.Items.Add(cad);
    }
}

public static void LeeMatrizSimBanda()
{
    int i, j, l;
    for (i = 1; i <= n; i++) {
        l = (i > m) ? (i - m + 1) : (1);
        for (j = l; j <= i; j++)
            try
            {
                c[idc.inb(i, j, m)] = double.Parse(obent.LeerLinea());
            }
            catch { }
    }
}

public static void EscribeMatrizSimBanda(String ob)
{
    String cad;
    int i, j;
    lb.Items.Add("MATRIZ " + ob);
    for (i = 1; i <= n; i++)
    {
        lb.Items.Add("FILA " + i.ToString());
        cad = "";
        for (j = 1; j <= n; j++)
          if (j <= i - m || j >= i + m)
            cad += "0 ";
          else if (i < j)
            cad += c[idc.inb(j, i, m)].ToString() + " ";
          else
            cad += c[idc.inb(i, j, m)].ToString() + " ";
        lb.Items.Add(cad);
    }
}

public static void LeeMatrizSimSkyline()
{
  int PrimerCeroDeFila = 0, Cont = 0, i, j;
  double Aux;
  for (i = 1; i <= n; i ++)
    for (j = 1; j <= i; j ++)
      try
      {
        Aux = double.Parse(obent.LeerLinea());
        if (i == j) {
```

```
                c[++ Cont] = Aux;
                Dir[i]= Cont;
                PrimerCeroDeFila = 1;
              }
              else if (Aux != 0) {
                c[++ Cont] = Aux;
                PrimerCeroDeFila = 0;
              }
              else if (PrimerCeroDeFila == 0)
                c[++ Cont] = Aux;
            }
            catch { }
        }

        public static void EscribeMatrizSimSkyline(String ob)
        {
            String cad;
            int i, j;
            lb.Items.Add("MATRIZ " + ob);
            for (i = 1; i <= n; i++)
            {
                lb.Items.Add("FILA " + i.ToString());
                cad = "";
                for (j = 1; j <= i; j++)
                  cad += RecuSkyLine(i, j).ToString() + " ";
                for (j = i + 1; j <= n; j++)
                  cad += RecuSkyLine(j, i).ToString() + " ";
                lb.Items.Add(cad);
            }
            lb.Items.Add("");
            lb.Items.Add("VECTOR DE DIRECCIONES");
            cad = "";
            for (i = 1; i <= n; i ++)
              cad += Dir[i].ToString() + " ";
            lb.Items.Add(cad);
            lb.Items.Add("");
        }

        public static double RecuSkyLine(int i, int j)
        {
          if (i + j == 2 || j > i - Dir[i] + Dir[i - 1])
            return c[idc.insk(i, j, Dir)];
          return 0;
        }

    }

    class CHorner
    {
        // Datos miembro:
```

```
    public static int n; // Grado del polinomio
    public static double[] a; // Coeficientes del polinomio

    // Metodos:

    // Metodo de Horner o división sintetica:
    public static double horner(double x)
    {
        double b = a[0];
        for (int i = 1; i <= n; i++)
            b = a[i] + b * x;
        return b;
    }

    // Calculo de la derivada a traves del metodo de Horner:
    public static double fphorner(double x)
    {
        double b = a[0];
        double yp = b;
        for (int i = 1; i < n; i++)
        {
            b = a[i] + b * x;
            yp = b + yp * x;
        }
        return yp;
    }
}

class COprPolinomios  // Operaciones con polinomios
{
    // Datos Miembro:
    static public int n; // Grado del polinomio.
    static public int na; // Grado del polinomio.
    static public double[] a; // Coeficientes del polinomio
    static public double[] d; // Coeficientes del polinomio auxiliar
    static public ListBox lb; // listbox para despliegue de resultados

    // Metodos:
    static public void Copiar(int np)
    {
        na = np;
        for (int i = 0; i <= np; i++)
            d[i] = a[i];
    }

    public static double EvaluaPolinomio(int m, double x)
    {
        double b = d[0];
        for (int i = 1; i <= m; i++)
            b = d[i] + b * x;
```

```
    return b;
}

static public void EscribePolinomio(String op)
{
    String cad = "";
    int gr = na;
    for (int i = 0; i <= na; i++)
    {
        if (d[i] != 0)
        {
            if (i == 1)
            {
                cad += Math.Round(d[i], 2).ToString();
                if (gr > 0) cad += "*";
            }
            else
            {
                if (d[i] < 0)
                    cad += " -";
                else
                    cad += " +";
                if (Math.Abs(d[i]) != 1)
                {
                    cad += Math.Round(Math.Abs(d[i]), 2).ToString();
                    if (gr > 0) cad += "*";
                }
            }
            if (gr > 0)
            {
                cad += "x";
                if (gr > 1) cad += "^" + gr.ToString();
            }
        }
        gr--;
    }
    lb.Items.Add(op + cad);
}

static public double DerivaPolinomio(int nd, double x)
{
   if (nd > n) return 0;
   // Aplicar la regla de derivación:
   int gd = n, i, j;
   Copiar(n);
   for (i = 0; i < nd; i++)
   {
      gd --;
      for (j = 0; j <= gd; j++)
         d[j] = (gd - j + 1) * d[j];
```

```
      }
      na = gd;
      EscribePolinomio("Polinomio resultante:");
      // Evaluar para x al polinomio resultante:
      return EvaluaPolinomio(gd, x);
   }
}

class CIntegracion
{
   // Datos miembro:
   public static int n, nc; // Numero de elementos o de términos
   public static double li, lf; // Límite inicial y final de la integral
   private static int[] to = { 3, 3, 2 }; // coeficientes regla tres octavos
   private static int[] dc = { 32, 12, 32, 14 }; // coeficientes regla dos cuarentaycincoavos
   static public pf punf;

   // Metodos:

   // Integral de sen(x^2) usando series de Taylor:
   public static double IntSenCuad()
   {
      double val = 0, fac;
      int i, j, signo = 1;
      for (i = 1; i <= n; i ++) {
         fac = 1;
         for (j = 2; j <= 2*i - 1; j ++)
            fac = fac * j;
         val +=  signo * (Math.Pow(lf, 4*i - 1) - Math.Pow(li, 4*i - 1)) / fac / (4*i - 1);
         signo = -signo;
      }
      return val;
   }

   // Regla de los rectángulos:
   public static double RRectangulos()
   {
      double h = (lf - li) / n, x, val = 0;
      x = li;
      for (int i = 0; i < n; i++)
      {
         val += punf(x);
         x += h;
      }
      return val * h;
   }

   // Regla de los trapecios:
   public static double RTrapecios()
   {
```

```
    double h = (lf - li) / n, x, val;
    x = li;
    val = punf(x) / 2;
    x += h;
    for (int i = 1; i < n; i++)
    {
        val += punf(x);
        x += h;
    }
    return (val + punf(x) / 2) * h;
}

// Regla de Simpson:
public static double RSimpson()
{
    double h, x, val;
    int neb = 2;
    nc = n;
    if (n % neb != 0) nc = n + neb - n % neb;
    h = (lf - li) / nc;
    x = li;
    val = punf(x);
    x += h;
    for (int i = 1; i < nc; i++)
    {
        val += neb * (i % neb + 1) * punf(x);
        x += h;
    }
    return (val + punf(x)) * h / 3;
}

// Regla Tres octavos:
public static double RTresOctavos()
{
    double h, x, val;
    int neb = 3;
    nc = n;
    if (n % neb != 0) nc = n + neb - n % neb;
    h = (lf - li) / nc;
    x = li;
    val = punf(x);
    x += h;
    for (int i = 0; i < nc - 1; i++)
    {
        val += to[i % neb] * punf(x);
        x += h;
    }
    return 3 * (val + punf(x)) * h / 8;
}
```

```
    // Regla Dos cuarentaycincoavos:
    public static double RDosCuarentaycincoavos()
    {
        double h, x, val;
        int neb = 4;
        nc = n;
        if (n % neb != 0) nc = n + neb - n % neb;
        h = (lf - li) / nc;
        x = li;
        val = 7 * punf(x);
        x += h;
        for (int i = 0; i < nc - 1; i++)
        {
            val += dc[i % neb] * punf(x);
            x += h;
        }
        return 2 * (val + 7 * punf(x)) * h / 45;
    }

}


class CEcDifOrdIOrden
{
    // Datos miembro:
    public static int n, nc; // Numero de elementos o de términos
    public static double x0, y0, xn, b; // Punto inicial, abscisa de solución, variable de ajuste
    static public pfg pung, pungx, pungy;

    // Metodos:

    // Método de Euler:
    public static double Euler()
    {
        double h = (xn - x0) / n, xk = x0, yk = y0;
        for (int k = 1; k <= n; k++)
        {
            yk += h * pung(xk, yk);
            xk += h;;
        }
        return yk;
    }

    // Método de Taylor:
    public static double Taylor()
    {
        double h = (xn - x0) / n, xk = x0, yk = y0, a, b, f;
        for (int k = 1; k <= n; k++)
        {
            f = pung(xk, yk);
```

```
        a = pungx(xk, yk);
        b = pungy(xk, yk);
        yk += h * f + Math.Pow(h, 2) * (a + b * f) / 2;
        xk += h;
    }
    return yk;
}

// Método de Euler Modificado:
public static double EulerModificado()
{
    double h = (xn - x0) / n, xk = x0, yk = y0, f, pk1;
    for (int k = 1; k <= n; k++)
    {
        f = pung(xk, yk);
        pk1 = yk + h * f;
        xk += h;
        yk += h * (f + pung(xk, pk1)) / 2;
    }
    return yk;
}

// Método de Runge Kutta de orden dos:
public static double RungeKuttaOrdenDos()
{
    double h = (xn - x0) / n, xk = x0, yk = y0, m1, m2;
    for (int k = 1; k <= n; k++)
    {
        m1 = h * pung(xk, yk);
        m2 = h * pung(xk + h / 2 / b, yk + m1 / 2 / b);
        yk += (1 - b) * m1 + b * m2;
        xk += h;
    }
    return yk;
}

// Método de Runge Kutta de orden cuatro:
public static double RungeKuttaOrdenCuatro()
{
    double h = (xn - x0) / n, xk = x0, yk = y0, m1, m2, m3, m4;
    for (int k = 1; k <= n; k++)
    {
        m1 = h * pung(xk, yk);
        m2 = h * pung(xk + h / 2, yk + m1 / 2);
        m3 = h * pung(xk + h / 2, yk + m2 / 2);
        xk += h;
        m4 = h * pung(xk, yk + m3);
        yk += (m1 + 2 * m2 + 2 * m3 + m4)/6;
    }
    return yk;
```

```
    }

}

class CRaicesRealesComplejas
{
    // Datos miembro:
    public static int n; // Grado del polinomio
    public static double[] A; // Coeficientes del polinomio
    public static double[] B; // Coeficientes del polinomio
    static public int maxIter; // Maximo de iteraciones
    static public double precision; // Precisión requerida
    static public ListBox lb; // Lisbox para despliegue de resultados

    // Metodos:

    // Metodo de Newton - Bairstow:
    public static void NewtonBairstow()
    {
      int CITER = 0, I;
      double AMax = Math.Abs(A[0]), CTS, C1 = 0, C2, C3, DT, DP,
             DQ, DPMin = 0, P1, Q1;
      /* CALCULAR LAS COTAS DEL MODULO DE LA RAIZ */
      for (I = 1; I <= n; I++)
          if (AMax < (P1 = Math.Abs(A[I]))) AMax = P1;
      CTS = (Math.Abs(A[0]) + AMax) / Math.Abs(A[0]);
      /* CALCULAR LOS COEFICIENTES INICIALES P1 Y Q1 */
      P1 = (CTS > 100) ? A[1] / A[0] : A[n - 1] / A[n - 2];
      Q1 = (CTS > 100) ? A[2] / A[0] : A[n] / A[n - 2];
      do {
        /* DEFLACIONAR EL POLINOMIO AL GRADO 2*M-2: */
        B[0] = A[0]; B[1] = A[1] - P1 * B[0];
        for (I = 2; I <= n; I++)
          B[I] = A[I] - Q1 * B[I - 2] - P1 * B[I - 1];
       /* CALCULAR C1, C2 Y C3: */
       C3 = B[0]; C2 = B[1] - P1 * B[0];
       for (I = 2; I < n; I++) {
         C1 = B[I] - Q1 * C3 - P1 * C2;
         if (I < n - 1) {
           C3 = C2; C2 = C1;
         }
       }
       /* RESOLVER EL SISTEMA:
          C1*DP+C2*DQ=B[n]
          C2*DP+C3*DQ=B[n-1]  */
       DT = C1 * C3 - C2 * C2; DP = (B[n] * C3 - B[n - 1] * C2) / DT;
       DQ = (C1 * B[n - 1] - C2 * B[n]) / DT;
       /* CALCULAR LOS NUEVOS COEFICIENTES P Y Q: */
       P1 = P1 + DP; Q1 = Q1 + DQ;
       DPMin = (DPMin < Math.Abs(DQ)) ? DPMin = Math.Abs(DQ) : Math.Abs(DP);
```

```
      }
      while (DPMin > precision && ++CITER < maxIter);
      if (DPMin > precision)
        lb.Items.Add("NO SE ALCANZA LA PRECISION REQUERIDA: "  + DPMin.ToString());
       /* VERIFICAR SI LAS RAICES SON REALES O COMPLEJAS: */
       Q1 = P1 * P1 - 4 * Q1;
       if (Q1 < 0)
         lb.Items.Add("RAICES COMPLEJAS = " + (-P1 / 2).ToString() + " +/- " +
             (Math.Sqrt(-Q1) / 2).ToString() + " * I");
       else
         lb.Items.Add("RAICES REALES MULTIPLES O CERCANAS: " +
            ((-P1 - Math.Sqrt(Q1)) / 2).ToString() + " Y " +
            ((-P1 + Math.Sqrt(Q1)) / 2).ToString());
      lb.Items.Add("EN " + CITER.ToString() + " ITERACIONES");
      /* TRANSFERIR LOS COEFICIENTES B[I] A LOS A[I]: */
      for (I = 0 ; I < n - 1; I++) A[I] = B[I];
      n -= 2;
    }

}
class CResolSisSimetricosEcuaciones
{
    // Datos miembro:
    public static int n; // Número de ecuaciones = Número de incógnitas
    public static double[] A; // matriz de coeficientes
    public static double[] B; // Vector de términos independientes

    // metodos:
    /* FUNCION QUE RESUELVE SISTEMAS DE ECUACIONES LINEALES
       SIMETRICOS, POR EL METODO DE CROUT */
    public static double CroutSimetrico(Boolean ps)
    {
        int i, j, k;
        double det = 1;
        if (ps)
        { // Primera solucion, transformar la matriz y calcular el determinante
            det = A[idc.ind(0, 0)];
            for (i = 1; i < n; i++)
            {
                for (j = 1; j <= i; j++)
                    for (k = 0; k <= j - 1; k++)
                        A[idc.ind(i, j)] -= A[idc.ind(i, k)] * A[idc.ind(j, k)] / A[idc.ind(k, k)];
                det *= A[idc.ind(i, i)];
            }
        }
        // Calculo del vector C:
        for (i = 0; i < n; i++)
        {
            for (k = 0; k <= i - 1; k++) B[i] -= A[idc.ind(i, k)] * B[k];
            B[i] /= A[idc.ind(i, i)];
```

```
        }
        // Cálculo del vector X:
        for (i = n - 2; i >= 0; i--)
            for (k = i + 1; k < n; k++) B[i] -= A[idc.ind(k, i)] * B[k] / A[idc.ind(i, i)];
        return det;
    }
}

class CResolSisSimBanda
{
    // Datos miembro:
    public static int n; // Número de ecuaciones = Número de incógnitas
    public static int m; // Ancho de banda
    public static double[] A; // matriz de coeficientes
    public static double[] B; // Vector de términos independientes

    // metodos:
    /* FUNCION QUE RESUELVE SISTEMAS DE ECUACIONES LINEALES
       SIMETRICOS EN BANDA, POR EL METODO DE CROUT */
    public static double CroutSimBanda(Boolean ps)
    {
        int i, j, k, l;
        double det = 1;
        if (ps)
        { // Primera solucion, transformar la matriz y calcular el determinante
            det = A[idc.inb(1, 1, m)];
            for (i = 2; i <= n; i++)
            {
              l = 2;
              if (i > m) l = i - m + 2;
              for (j = 1; j <= i; j++)
               for (k = l - 1; k <= j - 1; k++)
                 A[idc.inb(i, j, m)] -= A[idc.inb(i, k, m)] * A[idc.inb(j, k, m)] / A[idc.inb(k, k, m)];
               det *= A[idc.inb(i, i, m)];
            }
        }
        // Calculo del vector C:
        for (i = 1; i <= n; i++)
        {
            l = 1;
            if (i > m) l = i - m + 1;
            for (k = l; k <= i - 1; k++) B[i - 1] -= A[idc.inb(i, k, m)] * B[k - 1];
            B[i - 1] /= A[idc.inb(i, i, m)];
        }
        // Cálculo del vector X:
        for (i = n - 1; i >= 1; i--) {
            l = m + i - 1;
            if (i > n - m) l = n;
            for (k = i + 1; k <= l; k++) B[i - 1] -= A[idc.inb(k, i, m)] * B[k - 1] / A[idc.inb(i, i, m)];
        }
```

```
        return det;
    }
}

class CResolSisSimSkyline
{
    // Datos miembro:
    public static int n; // Número de ecuaciones = Número de incógnitas
    public static int[] Dir; // Vector de direcciones
    public static double[] A; // matriz de coeficientes
    public static double[] B; // Vector de términos independientes

    // metodos:
    /* FUNCION QUE RESUELVE SISTEMAS DE ECUACIONES LINEALES
       SIMETRICOS SKYLINE, POR EL METODO DE CROUT */
    public static double CroutSimSkyline(Boolean ps)
    {
        int i, j, k, lim;
        double det = 1;
        if (ps)
        { // Primera solucion, transformar la matriz y calcular el determinante
            det = A[idc.insk(1, 1, Dir)];
            for (i = 2; i <= n; i++)
            {
              lim = i - Dir[i] + Dir[i - 1] + 1;
              for (j = lim + 1; j <= i; j ++)
                for (k = lim; k <= j - 1; k ++)
                   A[idc.insk(i, j, Dir)] -= A[idc.insk(i, k, Dir)] *
                      CLectEscrMatrices.RecuSkyLine(j, k) / A[idc.insk(k, k, Dir)];
              det = det * A[idc.insk(i, i, Dir)];
            }
        }
        // Calculo del vector C:
        for (i = 1; i <= n; i++)
        {
          for (j = i - Dir[i] + Dir[i-1] + 1; j <= i - 1; j ++)
            B[i-1] -= A[idc.insk(i, j, Dir)] * B[j-1];
          B[i-1] /= A[idc.insk(i, i, Dir)];
        }
        // Cálculo del vector X:
        for (i = n - 1; i >= 1; i--)
        {
          for (j = i + 1; j <= n; j ++)
            B[i-1] -= CLectEscrMatrices.RecuSkyLine(j, i) * B[j-1] / A[idc.insk(i, i, Dir)];
        }
        return det;
    }
}

class CInterpolacion // Clase para realizar interpolación:
```

```
{
    // Datos miembro:
    public static double[] vx; // Vector de abscisas de los puntos conocidos.
    public static double[] vy; // Vector de ordenadas de los puntos conocidos.
    public static double[] vdd; // Vector para calcular las diferencias divididas.

    // metodos:
    /* ASIGNA MEMORIA A LOS ARREGLOS */
    public static void AsignaMemoria(bool t, int n)
    {
        vx = new double[n];
        vy = new double[n];
        if (t) vdd = new double[n];
    }

    /* FUNCION QUE RECIBE LAS COORDENADAS DE LOS PUNTOS INICIAL Y FINAL; Y
LA
     * ABSCISA A INTERPOLAR. REALIZA LA INTERPOLACION LINEAL Y DEVUELVE LA
       ORDENADA INTERPOLADA */
    public static double InterpolacionLineal(double x)
    {
        return (vy[1] - vy[0]) / (vx[1] - vx[0]) * (x - vx[0]) + vy[0];
    }

    /* FUNCION QUE RECIBE LAS COORDENADAS DE VARIOS PUNTOS;  LA
     * ABSCISA A INTERPOLAR Y EL NUMERO DE PUNTOS. REALIZA LA
INTERPOLACION
     * MEDIANTE LOS POLINOMIOS DE LAGRANGE Y DEVUELVE LA ORDENADA
INTERPOLADA */
    public static double InterpolacionLagrange(double x, int n)
    {
       double y = 0, L;
       int i, j;
       for (j = 0; j < n; j ++) {
         L = 1;
         for (i = 0; i < n; i ++)
             if (i != j)
                 L *= (x - vx[i]) / (vx[j] - vx[i]);
         y += vy[j] * L;
       }
       return y;
    }

    /* FUNCION QUE RECIBE LAS COORDENADAS DE VARIOS PUNTOS;  LA
     * ABSCISA A INTERPOLAR Y EL NUMERO DE PUNTOS. REALIZA LA
INTERPOLACION DE
     * NEWTON MEDIANTE DIFERENCIAS DIVIDIDAS Y DEVUELVE LA ORDENADA
INTERPOLADA */
    public static double InterpolacionNewton(double x, int n)
    {
```

```
      double y, p = 1;
      int i, j, k;
      for (i = 0; i < n; i ++) vdd[i] = vy[i];
      y = vy[0];
      for (j = 0; j < n - 1; j ++) {
        k = 0;
        p *=  x - vx[j];
        for (i = 0; i < n - j - 1; i ++) {
          vdd[i] = (vdd[i + 1] - vdd[i]) / (vx[k + j + 1] - vx[k]);
          k ++;
        }
        y += vdd[0] * p;
      }
      return y;
    }
}

class CGenMatrizYVector // Clase para ajustar polinomios:
{
    // Datos miembro:
    public static int n; // Grado del polinomio
    public static int p; // Tamaño de la muestra
    public static double[] T; // matriz de coeficientes
    public static double[] A; // Vector de términos independientes->Coeficientes del polinomio.
    public static double[] X; // Vector de abscisas de la muestra.
    public static double[] Y; // Vector de ordenadas de la muestra.

    // metodos:
    /* FUNCION QUE RECIBE UN NUMERO p DE PUNTOS Y SUS COORDENADAS
       (Xi,Yi)), Y EL GRADO DE AJUSTE n, A UN POLINOMIO; REGRESA LA MATRIZ
       T DE N FILAS POR N COLUMNAS Y EL VECTOR A DE N ELEMENTOS; QUE
CONFORMAN
       EL SISTEMA DE ECUACIONES SIMULTANEAS LINEALES PARA AJUSTAR
       UNA CURVA A UN POLINOMIO DE GRADO N. */
    public static void GeneraMatrizYVector()
    {
        int i, j, k = 0, l;
        for (i = 0; i < n; i++)
        {
            T[idc.ind(i, i)] = 0;
            A[i] = 0;
            for (j = 0; j < p; j++)
            {
                T[idc.ind(i, i)] += Math.Pow(X[j], k);
                A[i] += Y[j] * Math.Pow(X[j], i);
            }
            for (l = i + 1; l < n; l++)
            {
                T[idc.ind(l, i)] = 0;
                for (j = 0; j < p; j++)
```

```
                T[idc.ind(l, i)] += Math.Pow(X[j], k + l - i);
            }
            k += 2;
        }
    }
}
```

DIAGRAMAS UML:

A fin de motivar al lector, en el diseño orientado a objetos, se presenta a continuación un diagrama de clases y otro de secuencia, mismos que fueron diseñados para el cálculo de raíces reales, vistos en el capítulo 1 de este libro. Estos diagramas ha sido tomados del artículo “Modelado Orientado a Objetos para evaluar Métodos Numéricos utilizando Interfaces Visuales” F. Meneses, W. Fuertes de la Revista Tendencias en Computación DECC – Report.

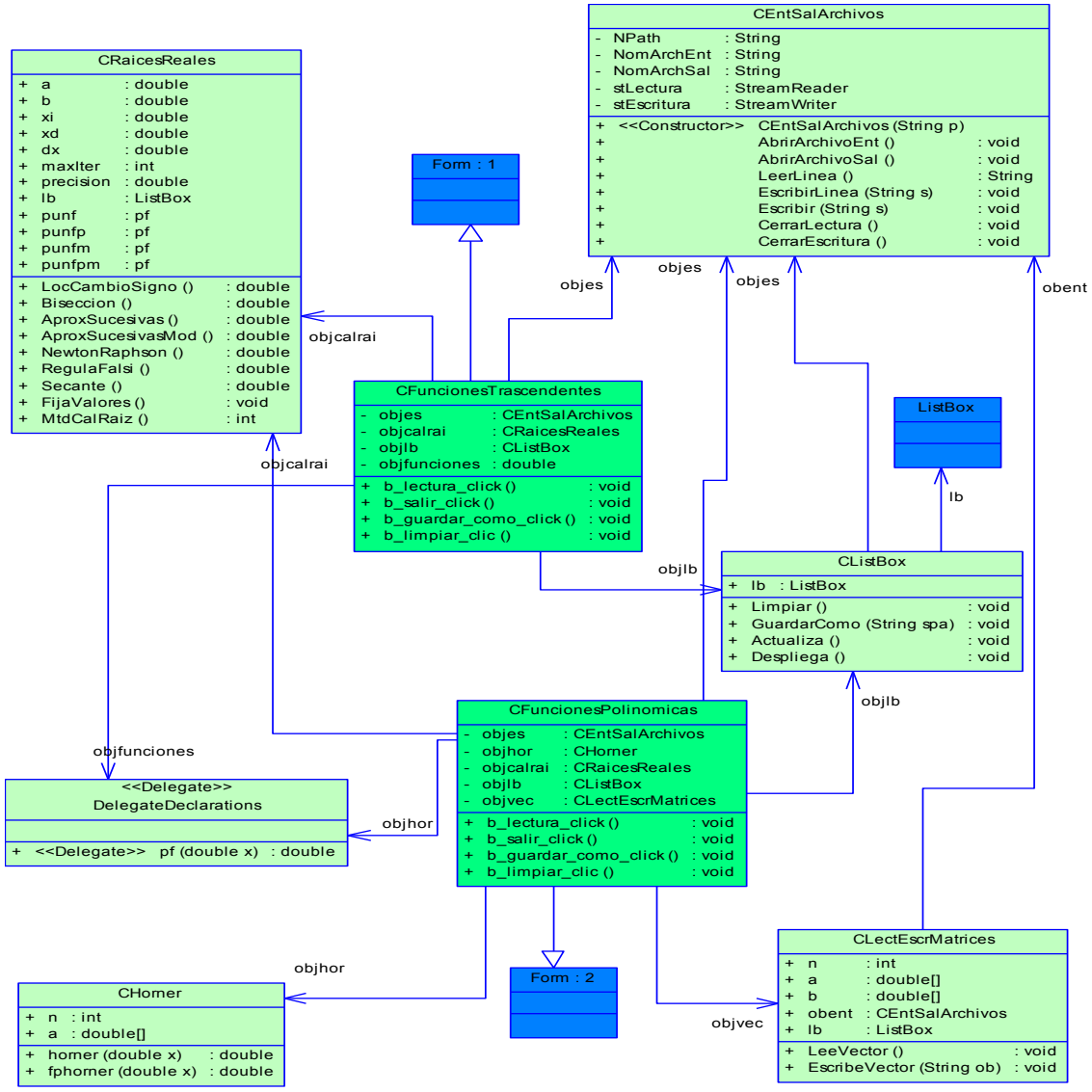


Figura b.4 Diagrama de Clases de la Interfaz Visual (Aplicación) y sus Interrelaciones

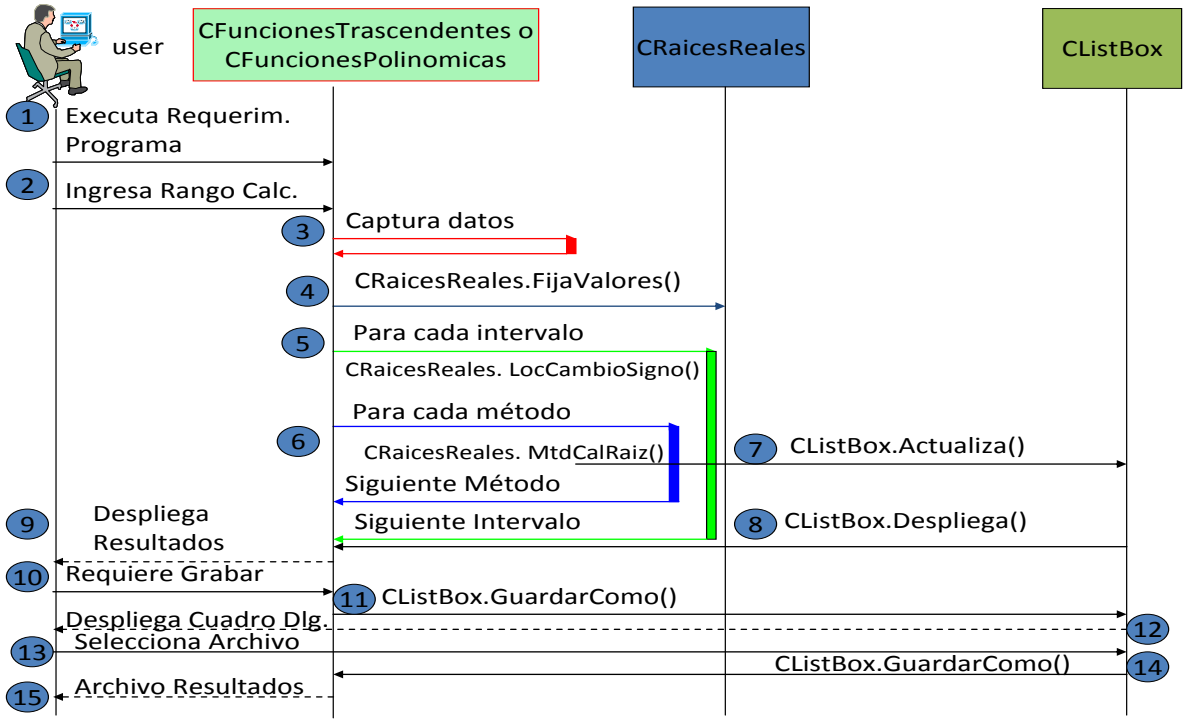


Figura b.5 Diagrama de Secuencia de la Interfaz Visual (Aplicación)

EJERCICIOS DE LOS APENDICES

1.- Implementar un programa en Matlab y en Visual C#; para ingresar los lados de un triángulo en cualquier orden y determinar si lo es; en el caso de que lo sea, determinar y desplegar su tipo y calcular desplegando el perímetro y la superficie.

2.- Implementar un programa en Matlab y en Visual C# para ingresar un valor entre -1 y 1. De ser asi, el programa debe calcular el arco seno del valor ingresado, aplicando la siguiente serie de Taylor:
$sen^{-1}(x) = x + (x^3/3)(1/2) + (x^5/5)(1*3)/(2*4) + (x^7/7)(1*3*5)/(2*4*6) + \ldots$
desplegando el resultado por pantalla e ingresando otro valor. El proceso se repite hasta cuando se ingresa un valor fuera del rango indicado.
Para finalizar el cálculo tomar hasta el término >= 1e-5 o se hayan alcanzado los 100 primeros términos de la serie indicada.

3.- Implementar un programa en Matlab para ingresar la base y la altura de un paraboloide de revolución y graficarlo

BIBLIOGRAFIA:

1.- B.P. Demidovich, I.A. Maron. “Cálculo Numérico Fundamental”. Paraninfo, ISBN: 84.283.0887 X, 1977.

2.- Smith, W Allen. “Análisis Numérico”. Prentice-Hall, ISBN: 519.4 B896, 1998.

3.- Fausto Meneses, Walter Fuertes, “Modelado Orientado a Objetos para evaluar Métodos Numéricos utilizando Interfaces Visuales” F. Meneses, W. Fuertes de la Revista Tendencias en Computación DECC – Report, ISSN 1390 - 5236.

4.- Dr. Félix Calderon Solorio
http://lc.fie.umich.mx/~calderon/programacion/Mnumericos/Muller.html.

5.- www.umng.edu.co/www/resources/INTERPOLACION.doc.

6.- Cándido Piñeiro Gómez "Apuntes de Matlab "
www.uhu.es/08003/aula_virtual/modulo_didactico/matlab.pdf.

7.- J. M. González de Durana "Introducción a MATLAB"
http://www.vc.ehu.es/depsi/jg/imatlab.pdf

8.- Paúl Medina, Ph.D., "Métodos Numéricos, Derivación e Integración".

9.- JAAN KIUSALAAS. "Numerical Methods in Engineering with MATLAB".

10.- John H. Mathews, Kurtis D. Fink . "Numerical Methods using MATLAB".

11.- Frank Ayres Jr. Ph. D. "Ecuaciones Diferenciales TEORIA y 560 problemas resueltos"

ESPE
UNIVERSIDAD DE LAS FUERZAS ARMADAS
INNOVACIÓN PARA LA EXCELENCIA

Publicaciones Científicas

ISBN: 978-9978-301-77-7