UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN

DESARROLLO DE APLICACIONES CON INTERFAZ GRÁFICA DE USUARIO PARA MÉTODOS NUMÉRICOS USANDO EL ENTORNO DE DESARROLLO INTEGRADO WXDEVC++

T E S I S

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO MECÁNICO ELECTRICISTA

P R E S E N T A:

JUAN ANTONIO DEL ÁNGEL SANTIAGO

ASESOR: DR. ROGELIO RAMOS CARRANZA

CUAUTITLÁN IZCALLI, ESTADO DE MÉXICO 2014

VNIVERSIDAD NACIONAL AVPNOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLÁN
UNIDAD DE ADMINISTRACIÓN ESCOLAR
DEPARTAMENTO DE EXÁMENES PROFESIONALES

U. N. A. M.
FACULTAD DE ESTUDIOS SUPERIORES CUAUTITLAN

ASUNTO: VOTO APROBATORIO

DRA. SUEMI RODRÍGUEZ ROMO
DIRECTORA DE LA FES CUAUTITLAN
PRESENTE

ATN: L.A. ARACELI HERRERA HERNÁNDEZ
Jefa del Departamento de Exámenes Profesionales de la FES Cuautitlán.

DEPARTAMENTO DE EXAMENES PROFESIONALES

Con base en el Reglamento General de Exámenes, y la Dirección de la Facultad, nos permitimos a comunicar a usted que revisamos **LA TESIS:**

"Desarrollo de Aplicaciones con Interfaz Gráfica de Usuario Para Métodos Numéricos Usando El Entorno de Desarrollo Integrado WxDevC++".

Que presenta el pasante: **JUAN ANTONIO DEL ÁNGEL SANTIAGO**
Con número de cuenta: **30627791-5** para obtener el Título de: **Ingeniero Mecánico Electricista**

Considerando que dicho trabajo reúne los requisitos necesarios para ser discutido en el **EXAMEN PROFESIONAL** correspondiente, otorgamos nuestro **VOTO APROBATORIO**.

ATENTAMENTE
"POR MI RAZA HABLARA EL ESPÍRITU"
Cuautitlán Izcalli, Méx. a 22 de Noviembre de 2013.

PROFESORES QUE INTEGRAN EL JURADO

	NOMBRE	FIRMA
PRESIDENTE	M.I. José Juan Contreras Espinosa	
VOCAL	M.T.I. Jorge Buendía Gómez	
SECRETARIO	Dr. Rogelio Ramos Carranza	
1er SUPLENTE	M.M. Marco Antonio Hernández Rodríguez	
2do SUPLENTE	Ing. Jorge Ramírez Rodríguez	

NOTA: los sinodales suplentes están obligados a presentarse el día y hora del Examen Profesional (art. 127).

HHA/Vc

AGRADECIMIENTOS

A mi madre por el apoyo brindado en cada etapa de mi vida y por ayudarme a terminar mi carrera profesional.

A mi asesor por brindarme su atención y apoyo en la realización de esta tésis.

A todos mis familiares y amigos que me brindaron su apoyo moral y me animaron a dar siempre lo mejor de mi.

A Dios por por ponerme en el camino de la ingeniería y haberme acompañado a través de la gran experiencia de estudiar Ingeniería Mecánica Eléctrica.

Dedico esta tésis a mi madre y a mi sobrinos Juan Carlos e Iván por ser personas muy importantes en mi vida.

ÍNDICE

INTRODUCCIÓN

Se realizaron programas para 8 métodos numéricos en el entorno de desarrollo integrado WxDevC++ de tal manera que tengan una interfaz amigable para el usuario y la mayor funcionalidad posible. Estos programas son:

Para encontrar raíces de funciones:

- Bisección.
- Newton-Raphson.

Para resolver sistemas de ecuaciones lineales.

- Descomposición LU.

Para interpolar.

- Interpolación de Newton.
- Interpolación de Lagrange.
- Interpolación Spline que incluye:
  - **I.** Natural.
  - **II.** Anclado.
  - **III.** Extrapolación.
  - **IV.** Curvatura ajustada en puntos extremos.

Para Integrar

- Integración Numérica que incluye:
  - **I.** Simpson 1/3.
  - **II.** Simpson 1/8.
  - **III.** Trapezoidal.

Para ecuaciones diferenciales ordinarias.

- Runge Kutta de cuarto orden.

Los capítulos han sido escritos de tal manera que se explique lo más claramente posible como se programaron los componentes utilizados como listas de selección y botones, para que sirvan de guía a los estudiantes que pretendan diseñar aplicaciones con ventana. Hago uso además de capturas de pantalla, secciones de código y diagramas de flujo para hacer la explicación aún más clara.

En el capítulo 1 se explica el diseño de un *Analizador de raíces* que es una herramienta que permite conocer los intervalos en los que se encuentran las raíces reales de una función para que una vez conociendo dichos intervalos se puedan conocer exactamente el valor de las raíces haciendo uso del programa *Bisección* que se explica en el capítulo 2 y *Newton-Raphson* que se detalla en el capítulo 3. Posteriormente en el capítulo 4 se presenta el programa *Descomposición LU* que fue pensado inicialmente para resolver sistemas de ecuaciones de 5x5 pero finalmente el código fue modificado para poder resolver sistemas de orden 2x2 hasta 10x10.

Los capítulos 5, 6 y 7 se enfocan a programas de interpolación que son *Interpolación de Lagrange*, *Interpolación de Newton* e *Interpolación Spline* respectivamente. Estos programas fueron diseñados para introducir hasta 10 puntos y utilizando todos los decimales que proporciona WxDevC++ con el tipo **double** se consiguen interpolaciones bastante aceptables en polinomios de orden 9 para el caso de Lagrange y Newton. En el capítulo 8 se explica el programa de *Integración Numérica* que ha sido diseñado para introducir una función o para introducir datos en forma tabular.

En el capítulo 9 se presenta el programa *Runge-Kutta* de cuarto orden. En el capítulo 10 se detallan las instrucciones para usar los programas y sacar el mayor provecho de ellos y en el capítulo 11 se presentan aplicaciones y resultados de los programas dando solución a diversos casos de estudio.

.

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

comparten una característica común: invariablemente se debe realizar un buen número de tediosos cálculos aritméticos. No es raro que con el desarrollo de computadoras digitales eficientes y rápidas, el papel de los métodos numéricos en la solución de problemas de ingeniería haya aumentado en forma considerable en los últimos años. (Chapra S.C., 2007, pág. 3).

Hoy en día, las computadoras y los métodos numéricos proporcionan una alternativa para cálculos complicados. Al usar la computadora para obtener soluciones directamente, se pueden aproximar los cálculos sin tener que recurrir a suposiciones de simplificación o a técnicas lentas. Aunque las soluciones analíticas aún son muy valiosas, tanto para resolver problemas como para proporcionar una mayor comprensión, los métodos numéricos representan alternativas que aumentan en forma considerable la capacidad para confrontar y resolver los problemas. Por consiguiente, es posible dar más importancia a la formulación de un problema, a la interpretación de la solución y a su incorporación al sistema total. (Chapra S.C., 2007, pág. 4)

Es un hecho que las computadoras son de gran ayuda para implementar los métodos numéricos y realizar miles de cálculos en muy poco tiempo. Sin embargo, muchas de las veces los códigos de programas que se encuentran en libros se ejecutan en consola, por lo que la presentación de los resultados es poco amigable para el usuario en comparación con una aplicación de ventana. Además, cuando se desea cambiar las funciones matemáticas de entrada en C++ para determinado método, es ineludible editar el código del programa para insertar las nuevas funciones, lo que les resta funcionalidad a este tipo de aplicaciones.

Por su parte las aplicaciones con ventana tienen la ventaja de que se puede hacer uso de múltiples componentes tales como: botones, tablas, listas desplegables, cajas de texto, etc., que permiten llevar a los programas a tener tanta funcionalidad como se desee. Una vez compilados los programas se generan la aplicaciones con interfaz gráfica que son ejecutables, por lo que no se necesita tener el programa fuente para poder ejecutarlos, además una vez generados cada uno tiene un tamaño de alrededor de 10 MB, es decir, son de tamaño reducido y completamente portables. Se muestra en la figura 1 una comparación entre una aplicación de consola y una aplicación de ventana del método de interpolación de Lagrange.

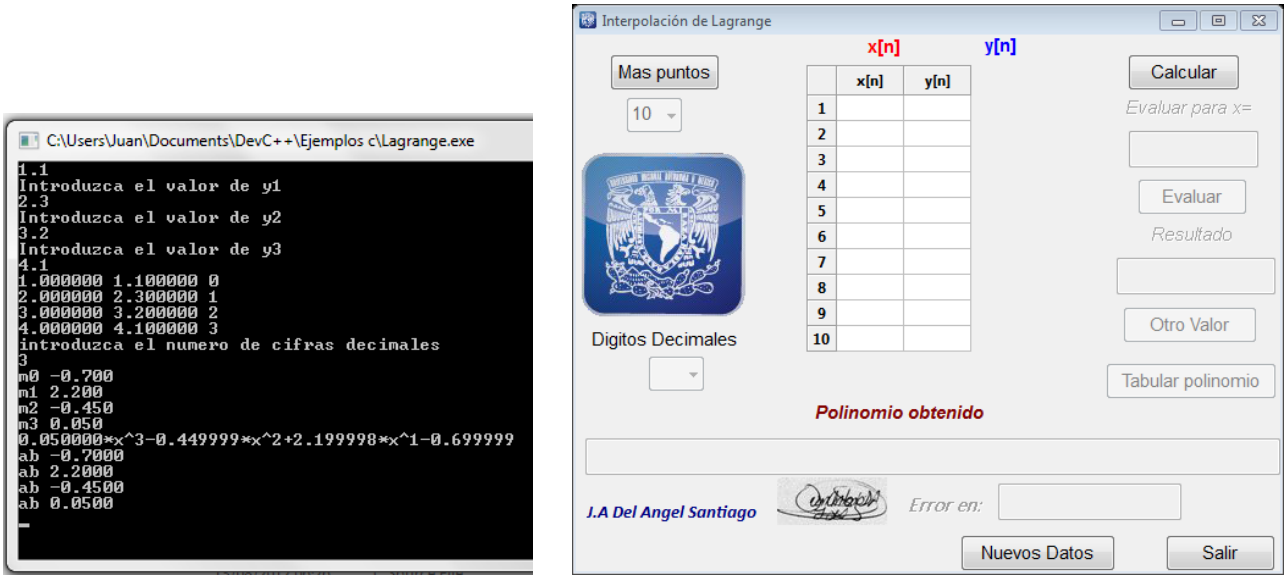


Figura 1. Comparación entre una aplicación de consola y una aplicación con interfaz gráfica de usuario (GUI).

Aunque la aplicación con interfaz gráfica tiene una presentación mucho mejor que la de consola y mejor funcionalidad tiene una desventaja con respecto a las aplicaciones de consola. Esta desventaja consiste en que el nivel de complejidad de la programación de la aplicación aumenta considerablemente ya que los componentes utilizados deben ser programados por separado.

Existen entornos como Maple orientados a modelos matemáticos, que en el caso de la programación de métodos numéricos permiten manipular fácilmente funciones pero la ejecución del programa se hace en el mismo espacio de trabajo donde se encuentra el código. Si se desea modificar los parámetros de entrada sean funciones o constantes, el código debe ser modificado.

Por todo lo anteriormente expuesto surgió la idea de realizar los programas con interfaz gráfica y debido a que la programación es más compleja decidí no sólo explicar el funcionamiento de cada programa sino también como es que se llega a esa funcionalidad. Los programas han sido desarrollados con el software WxDevC++ debido a que es un software libre que ofrece un entorno de desarrollo con el que se pueden desarrollar este tipo de aplicaciones y es semejante a Visual Studio de Microsoft.

OBJETIVOS.

- ✓ Poner al alcance de los estudiantes de las distintas carreras de Ingeniería en general y particularmente para los de la carrera de IME, un paquete de programas con interfaz gráfica de usuario para Métodos Numéricos.

- ✓ Explicar detalladamente como se creó cada uno de los programas para que los estudiantes interesados en desarrollar aplicaciones con interfaz gráfica de usuario utilicen ésta tesis como una base.

- ✓ Mostrar los resultados de las iteraciones de los métodos organizados en tablas.

- ✓ Diseñar los códigos de los programas de tal manera que tengan tanto un uso didáctico (pocas iteraciones) como un uso preciso (bastantes iteraciones con mucha precisión).

- ✓ Garantizar la portabilidad de los archivos ejecutables para que puedan ser utilizados en distintas computadoras.

- ✓ Analizar a fondo los errores que podría cometer el usuario al introducir datos a los programas, esto con el fin de programar un mensaje de error para cada uno de ellos. De esto depende la confiabilidad de los programas ya que no deberán ejecutarse si el usuario ha cometido errores, de lo contrario el programa en cuestión se perdería, no respondería y posteriormente dejaría de funcionar.

1. ANALIZADOR DE RAÍCES.

Para encontrar raíces de funciones se han realizado los programas de Bisección y Newton-Raphson. Un primer paso para aplicar dichos métodos consiste en estimar el intervalo en el que se encuentra cada una de las raíces. Se pueden utilizar las reglas de Descartes o ir dando valores a la función para observar su comportamiento pero en este caso se diseñó un programa que puede estimar dichos intervalos de una manera rápida y confiable, este programa se llama "Analizador de Raíces" y se muestra en la figura 1.1. Los parámetros de entrada requeridos por este programa son: la función, el intervalo y el incremento deseado para la variable independiente. Básicamente este programa realiza la tabulación con esos parámetros para que después mediante un algoritmo analice pares de puntos en busca de raíces. En el capítulo 10 se explica la sintaxis que se debe manejar para introducir las funciones.

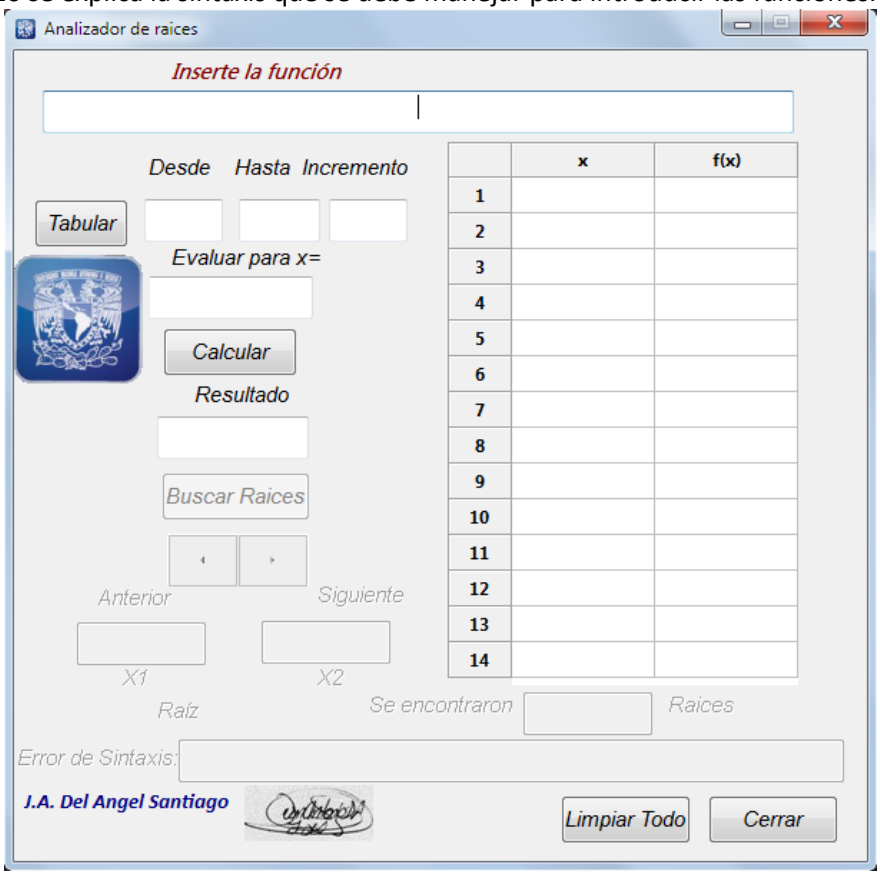


Figura 1.1. Analizador de raíces

1.1 Consideraciones de diseño del programa

- Lo ideal de acuerdo al tamaño de la ventana establecido es que la tabla cuente por defecto con 14 filas, si la cantidad de puntos a tabular es mayor que 14 entonces se agregarán cuantas filas sean necesarias.
- Siempre al iniciar la tabulación se deben de eliminar las filas agregadas anteriormente para que la tabla se ajuste dependiendo de la nueva tabulación.
- Dependiendo del valor que se obtenga al evaluar la función en cada punto el resultado se puede mostrar en los colores: rojo, azul o negro dependiendo si el valor es menor, igual o mayor que cero respectivamente.
- Al realizar una nueva tabulación se establecer color negro para el texto de la tabla.
- Se debe contar con un botón que permita evaluar la función en un punto de particular interés para el usuario que muestre al menos 8 decimales del resultado.
- Si el número de filas es igual o menor que 14 el botón *Buscar Raíces* no se habilita ya que el usuario puede determinar los intervalos de las raíces de acuerdo al color del texto en las celdas que corresponden a f(x).
- Se debe contar con un botón que permita navegar entre las raíces encontradas.

1.2 Componentes utilizados en los programas.

En esta sección se describen de una manera breve los componentes usado en los programas y en la figura 1.2 se muestran los componentes.

1. WxStaticText (Caja de Texto): Se utiliza para insertar o mostrar cadenas de caracteres.
2. WxButton (Botón): Se utilizan para ejecutar las acciones para las que sea programado.
3. WxStaticText: Muestra una etiqueta.
4. WxGrid: Tabla para mostrar o insertar datos.
5. WxMessageDialog: Diálogos que muestran al usuario un mensaje.
6. WxSpinButton: Es usado para incrementar o decrementar un valor.
7. WxChoice: Listas de selección desplegables que permiten elegir una opción de la lista.
8. WxRadioButton. Es un botón que denota una de varias opciones mutuamente exclusivas.

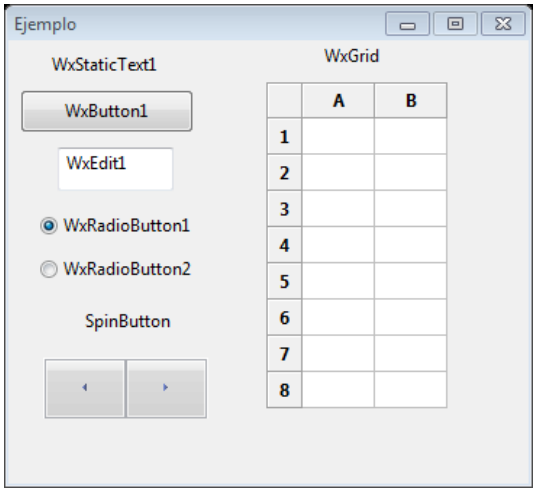


Figura 1.2 Componentes usados en los programas

Como puede observarse cuando se crean componentes los nombres predeterminados son los que aparecen en la figura 1.2. En caso de crear otro Botón su nombre sería WxButton2 y el siguiente WxButton3. Al momento de crear el código para el funcionamiento del programa sería poco práctico referirnos a los botones de esa manera, por lo tanto en los códigos que se muestran a los largo de esta tesis los nombres de los componentes tienen relación con la acción que realizan así por ejemplo el botón Calcular se llamará CalcularB y una caja de texto donde se inserte el valor de 'x' se llamará xvalCT. En este caso la terminación del nombre hace referencia a que tipo de componente se está tratando, de tal manera que las terminaciones para los componentes se muestran en la tabla 1.1.

Componente	Terminación	Ejemplo
WxEdit	CT	xvalCT
WxButton	B	CalcularB
WxStaticText	ST	ResultST
WxGrid	G	TablaG
SpinButton	SB	DesplazarSB
WxMessageDialog	DM	SintaxisDM
WxChoice	LD	DigitosLD

Tabla 1.1 Terminaciones con que se identifican los componentes

1.3 Componentes usados en el Analizador de Raíces.

- ***Cajas de Texto*** **donde:**

1. Se inserta la función: FuncionCT
2. Se inserta el parámetro **desde**: DesdeCT.

3. Se inserta el parámetro **hasta:** HastaCT
4. Se inserta el parámetro **incremento:** IncCT.
5. Se inserta un valor de **x** a evaluar: XvalCT
6. Se muestra el **resultado** de la evaluación: ResultCT
7. Se muestra el límite inferior del intervalo donde se encuentra una raíz (X_1)**:** X1CT
8. Se muestra el límite superior del intervalo donde se encuentra una raíz (X_2)**:** X2CT
9. Se muestra el número de raíces encontradas: NraizCT
10. Se muestra el número de raíz actual: RactualCT
11. Se muestran los errores de sintaxis en caso de haberlos: SintaxisCT

- **Botones**

12. Tabular: TabularB
13. Limpiar todo: LimpiarB.
14. Calcular: CalcularB
15. Buscar Raíces: RaicesB.
16. Cerrar: CerrarB.

- **Tabla**

17. Tabla: TablaG.

- **Spin Button**

18. Botón para navegar entre las raíces encontradas: DesplazarSB.

- **Textos Estáticos**

19. Inserte la función: FuncionST.
20. Desde: DesdeST.
21. Hasta: HastaST.
22. Incremento: IncST.
23. Evaluar para x=: EvaluarxST.
24. Resultado: ResultST.
25. Siguiente: SiguienteST.
26. Anterior: AnteriorST.
27. X1: X1ST.
28. X2: X2ST.
29. Se encontraron: EncontrarST.
30. Raíces: RaicesST.
31. Raíz: RaizST.
32. Error de Sintaxis: SintaxisST.

- **Diálogos de mensaje:**

33. Inserte una función: FuncionDM.
34. Introduzca un valor: XvalDM.
35. No se encontraron raíces: NoraizDM.
36. Inserte el parámetro Desde: DesdeDM.
37. Inserte el parámetro Hasta: HastaDM.
38. Inserte el parámetro incremento: IncDM.
39. Primera Raíz: PraizDM.
40. Última Raíz: UraizDM.
41. Error de Sintaxis: SintaxisDM.
42. Error de Sintaxis en los parámetros: SintaxisPDM.

Una de las ventajas de utilizar estos nombres es que si se utilizan los mismos nombres y el código mostrado al final de cada capítulo los programas pueden ser reproducidos con facilidad.

1.4 Clases, funciones miembro y objetos.

Para entender el uso del evaluador de expresiones algebraicas se debe tener noción de programación orientada objetos y si no se cuenta con ella aun así aquí se explica cómo utilizar los métodos de la clase Evaluar de la que se invocan varios métodos en el código de los programas.

"Una clase define un nuevo tipo de dato que especifica la forma de un objeto. Una clase incluye los datos y el código que operará sobre esos datos. Además, una clase enlaza datos y código. C++ usa una especificación de una clase para construir objetos. Los objetos son instancias de una clase. Además, una clase es esencialmente una serie de planes que especifican cómo construir un objeto. Es importante tener claro esto: Una clase es una abstracción lógica.

No es hasta que un objeto de esa clase sea creado que la representación física de la clase existe en la memoria. Cuando se define una clase, se declaran los datos que ésta contiene y el código que opera en esos datos. Aunque clases muy simples pueden contener sólo código o sólo datos, la mayoría de las clases contienen ambos. En conjunto, los datos se almacenan en las variables y el código en las funciones. Colectivamente, las funciones y variables que constituyen una clase son llamados 'miembros' de la clase. Una variable declarada dentro de una clase es llamada 'variable miembro', y una función declarada en una clase es llamada 'función miembro'. En ocasiones el término 'variable de instancia' es usado en lugar de variable miembro". (Schildt, 2007).

Las funciones declaradas en una clase son también conocidas como métodos de la clase y ésta última forma de llamar a las funciones será la utilizada a lo largo de esta tesis. Para que las cadenas de caracteres introducidas en los programas puedan ser interpretadas como operaciones algebraicas o funciones se necesita el "Evaluador de Expresiones Algebraicas" desarrollado por el Ing. Rafael Alberto Moreno Parra que se puede descargar como software libre de: http://darwin.50webs.com/Espanol/Evaluador.htm.

Una de las clases del evaluador de expresiones algebraicas es la clase **Evaluar** que tiene entre otros los siguientes métodos que son muy utilizados en el código de los programas:

`int EvaluaSintaxis(char *):` *se encarga de realizar 25 pruebas para garantizar que la sintaxis es correcta y si es así proceder a los cálculos. En caso de error devuelve el código de error correspondiente de acuerdo al error cometido. Los errores posibles con su respectivo código son los siguientes:*

1. *Dos o más operadores están seguidos. Ejemplo: 2++4, 5-*3*
2. *Un operador seguido de un paréntesis que cierra. Ejemplo: 2-(4+)-7*
3. *Un paréntesis que abre seguido de un operador o comienza con operador.*
4. *Los paréntesis están desbalanceados. Ejemplo: 3-(2*4))*
5. *Hay algún paréntesis vacío. Ejemplo: 2-()*3*
6. *Paréntesis que abre no corresponde con el que cierra*
7. *Un paréntesis que cierra y sigue un número. Ejemplo: (3-5)7-(1+2)*
8. *Un número seguido de un paréntesis que abre. Ejemplo: 7-2(5-6)*
9. *Doble punto en un número de tipo real. Ejemplo: 3-2..4+1 7-6.46.1+2*
10. *Un paréntesis que cierra seguido de una variable. Ejemplo: (12-4)y-1*
11. *Una variable seguida de un punto. Ejemplo: 4-z.1+3*
12. *Un punto seguido de una variable. Ejemplo: 7-2.p+1*
13. *Un número antes de una variable. Ejemplo: 3x+1*
14. *Un número después de una variable. Ejemplo: x21+4*
15. *Hay 4 o más letras seguidas. Ejemplo: 12+ramp+8.9*
16. *Función inexistente. Ejemplo: 5*alo(78)*
17. *Variable inválida (solo pueden tener una letra). Ejemplo: 5+tr-xc+5*
18. *Variable seguida de paréntesis que abre. Ejemplo: 5-a(7+3)*
19. *Después de paréntesis que cierra sigue paréntesis que abre. Ejemplo: (4-5)(2*x)*
20. *Después de operador sigue un punto. Ejemplo: -.3+7*
21. *Después de paréntesis que abre sigue un punto. Ejemplo: 3*(.5+4)*
22. *Un punto seguido de un paréntesis que abre. Ejemplo: 7+3.(2+6)*
23. *Paréntesis cierra y sigue punto. Ejemplo: (4+5).7-2*
24. *Punto seguido de operador. Ejemplo: 5.*9+1*
25. *Punto seguido de paréntesis que cierra. Ejemplo: (3+2.)*5*

En caso de no ser detectado ningún error el método EvaluaSintaxis devuelve **cero.**

void MensajeSintaxis(**int** CodigoError, **char ***): *En caso de que haya errores de sintaxis el mensaje de error se escribirá en la caja de texto correspondiente de acuerdo al código de error.*

void TransformaExpresion(**char ***, **char ***): *Este método se encarga de convertir a minúsculas y encerrar en paréntesis la expresión, luego revisar carácter por carácter filtrando sólo aquellos que son permitidos dentro de una expresión algebraica.*

void ValorVariable(**char** variableAlgebra, **double** valor): *se asigna a la variable independiente un valor.*

double Calcular(); *Una vez asignado un valor en el caso de las funciones se calcula el resultado. En caso de que solo sea una operación no es necesario asignar valor dado que no hay variable por lo que solo se usa el método Calcular .*

1.5 Funcionamiento de los componentes del programa.

En esta sección se describen el funcionamiento de los botones *Tabular, Calcular, Buscar Raíces* y de *Navegación* entre las raíces. La descripción se ha realizado por separado debido a que como se mencionó anteriormente los componentes se programan por separado. La explicación se basa en los diagramas de flujo mostrados al final del capítulo. También se muestra en el apéndice C el código de cada programa completamente comentado a manera de complementar la explicación del funcionamiento, para que se pueda observar claramente cómo funciona el programa.

En los detalles del funcionamiento de este programa se explica de una manera clara como hacer uso del evaluador de expresiones algebraicas para procesar las expresiones que son dadas como cadenas de caracteres por el usuario. En los capítulos posteriores solo se menciona el uso del evaluador para revisar sintaxis e interpretar expresiones, esto con el fin de no hacer repetitiva la explicación.

1.5.1 Funcionamiento del botón *Tabular*

Los componentes dedicados a mostrar información de las raíces están inicialmente deshabilitados hasta que se confirme que se han encontrado raíces. Si anteriormente mostraron información sobre el número de raíces encontradas y los intervalos donde se encuentran, las cajas de texto usadas para tal fin deben ser limpiadas para la siguiente búsqueda y los componentes deshabilitados en caso de que no se encuentren raíces con los parámetros dados la siguiente vez que se oprima el botón. Por tal motivo el primer paso es limpiar cajas de texto y deshabilitar componentes dedicados a mostrar información de las raíces. Los componentes deshabilitados son:

- Las cajas de texto X1CT, X2CT, NraizCT, RactualCT, SintaxisCT.
- Los textos estáticos SiguienteST, AnteriorST, X1ST, X2ST, EncontrarST, RaicesST, RaizST y SintaxisST.
- El Botón RaicesB.
- El Botón de Navegación DesplazarSB.

Además se limpian los componentes las cajas de texto mencionadas y TablaG.

Posteriormente se verifica que no estén vacías las cajas de texto solicitadas al usuario. Si están vacías se muestran mensajes de error como el que se aprecia en la figura 1.3 que se muestra cuando se ha omitido el parámetro Incremento. De lo contrario se comienza con la revisión de sintaxis de lo que se ha tecleado en cada una de las cajas de texto. Para ello se copia el contenido de la caja de texto en la cadena de caracteres *expresión* y se usa como primer parámetro un puntero al inicio de la cadena para el método **void** TransformaExpresion(**char ***, **char ***) y el segundo parámetro es un puntero al principio de *Transformado* que es una cadena inicializada en ceros que se encarga de almacenar la expresión transformada. Esté método se encarga entre otras cosas de quitar espacios, tabuladores, encerrar entre paréntesis y volver a minúsculas para luego dejar en la cadena *Transformado* la expresión modificada. Hecho esto se hace uso del método **int** EvaluaSintaxis(*** char**) que recibe como parámetro un puntero a la expresión transformada y devuelve el código de error de la verificación de sintaxis. El código de error se guarda en la variable entera verifsin para el caso de la función. Si no ocurre ningún error en

la variable `verifsin` se almacena el valor cero. De lo contrario se muestra el mensaje de error de la figura 1.4 y se habilita la caja de texto donde se específica el error que ha cometido el usuario haciendo uso del método `void MensajeSintaxis(int CodigoError, char *)` que recibe como parámetros el código de error que en este caso es `verifsin` y un puntero a la cadena *Mensaje* que es una cadena inicializado en ceros. Después de invocar al método queda el mensaje correspondiente almacenado en *Mensaje* y posteriormente se escribe en la caja de texto SintaxisCT.

De manera semejante se analiza la sintaxis de los parámetros *Desde, Hasta* e *Incremento.* Las variables que se usan para guardar los códigos de error son respectivamente *`versinp1, verifsinp2 y verifsinp3.`* Una vez que cada variable tiene su código de error se suman para observar si ocurrió error en algún parámetro, si no es así el resultado es cero. De lo contrario se muestra el mensaje de error de la figura 1.5. En este caso se podría omitir esta parte y el usuario ingresaría los parámetros como números, sin embargo, el código se ha diseñado de tal manera que se puedan introducir fracciones, funciones trigonométricas y otras operaciones de acuerdo a las necesidades del usuario. En este caso solo se muestra el diálogo de mensaje por que las expresiones insertadas como parámetros no serán tan largas a diferencia de la función para la que se muestra en la caja de texto SintaxisCT el error específico que se ha cometido.

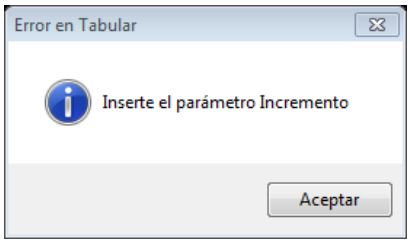


Figura 1.3 Mensaje de error

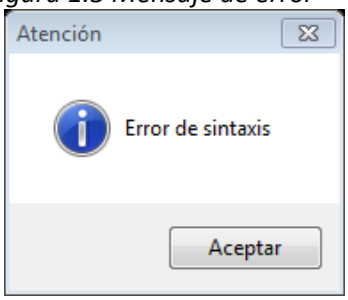


Figura 1.4 Mensaje de error del error de sintaxis en la función

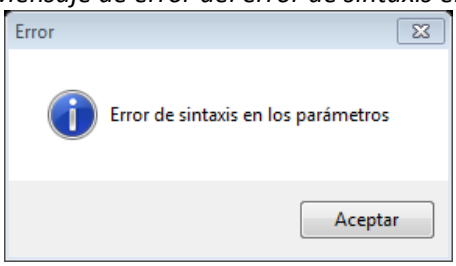


Figura 1.5 Mensaje de error de sintaxis en los parámetros

La condición que se debe de cumplir para que comience la tabulación es que no haya errores de sintaxis y las cajas de texto se hayan llenado. Una vez comprobado esto se verifica si existen más de 14 filas en la tabla para borrar las excedentes cada que inicie una nueva tabulación, debido a que se la tabla se debe ajustar a las nuevas condiciones porque si no se eliminan las filas excedentes la nueva tabla tendría más filas de las necesarias y aparecerían vacías. De la misma manera el texto de las celdas se debe volver a negro. Esta vez se usa el método **`int`** `ArreglaNegativos(`**`char *, char *`**`)` que se encarga de transformar la expresión para aceptar los menos unarios.

Se necesita como parámetro un puntero al arreglo *Transformado* y otro puntero al arreglo *ExprNegativos* inicializado en ceros donde se almacena la expresión final que será analizada una vez que se han agregado los menos unarios. Esto es necesario debido a que se considera un error sintáctico que una función comience con un operador como por ejemplo `/x+1, *x+1` pero debido a que + y – son operadores ocurre lo mismo con ellos y operaciones como `sen(-x)` tampoco serían permitidas. De tal manera que en esta versión del evaluador de expresiones algebraicas que es la más actual, se permite el operador (–) al inicio de una expresión pero es necesario el uso del método antes mencionado que se explica detalladamente en la página 86 de la referencia bibliográfica 4 que se puede consultar en http://darwin.50webs.com/Espanol/Evaluador.htm.

Después de que se ha transformado la expresión, se ha analizado la sintaxis y se han agregado los menos unarios se hace uso del método **`void`** `Analizar(`**`char`** `*expresion)` que interpreta la cadena de caracteres insertada como una expresión matemática y todo queda listo para dar valores a la variable independiente. En el caso de este programa se ha usado (x) como variable independiente pero el código del evaluador da las posibilidad de usar variables de la (a) a la (z). Para los parámetros del intervalo y el incremento el siguiente método a usar es **`double`** `Calcular()` porque los parámetros pueden ser expresiones algebraicas de las que es necesario obtener su valor y en caso de que no lo sean el valor insertado permanece inalterado. Esta acción se realiza para las variables *Desde, Hasta e Incremento* que se almacenan en las variables de tipo **double** que son `dsd, sta` e `inc` respectivamente.

Dado que se usan los mismos arreglos para todos las expresiones analizadas, cada que son utilizados los arreglos deben reiniciarse por lo que la última expresión analizada y que quedará en los arreglos es la función tecleada para que en el bucle de tabulación se le den valores a la variables independiente y no sé modifique más el contenido de los arreglos. Justo antes de entrar al bucle principal se deben inicializar las variables que corresponden al valor de la variable independiente y las iteraciones que son respectivamente **`double`** `i=dsd` e **`int`** `iter=0.` Hecho esto comienza el bucle **do while** que comienza en el límite inferior el intervalo dado y termina cuando se alcanza el valor del límite superior en el que primero se verifica si las iteraciones son más de 13 y debido que se incluye el cero si se alcanza el número máximo de filas que son 14 por defecto, es necesario insertar una fila en la tabla cada nueva iteración.

Ya que todo está listo para dar valor a la variable independiente, se le da el valor de `i` en cada iteración y se hace uso del método **`void`** `ValorVariable(`**`char`** `variableAlgebra,` **`double`** `valor)` que recibe como parámetros la variable y el valor que se le asigna (en este caso (x,i)) para que con el método **`double`** `Calcular()` se obtenga el resultado del cálculo. Si no ocurre error matemático y dependiendo del resultado se pueden dar tres posibilidades:

- El valor de la función es positivo y se muestra el número en negro.
- El valor de la función es negativo y se muestra el número en rojo.
- El valor de la función es cero y se muestra el número en color azul.

En cada iteración a la variable `i` se le suma el incremento para la siguiente iteración y la variable `iter` se incrementa en uno. El ciclo finaliza cuando se alcanza el valor del límite superior del intervalo. Si ocurre un error matemático en la celda que corresponde a f(x) se mostrará la cadena “Error”.

Dependiendo del valor de iteraciones que se realizan el botón *Buscar Raíces* será habilitado ya que si es menor de 14 el usuario puede identificar la localización de las raíces en base al color del valor de f(x), pero si es mayor a 14 el botón *Buscar Raíces* aparecerá. Otra de las razones por las que se hace esto es porque si una o más celdas de la columna que corresponde a f(x) permanecen vacías cuando se aplica el algoritmo que busca raíces, éste obtiene los números de las celdas y entiende en tal caso que las celdas vacías tienen el valor de cero y se considera una raíz exacta lo cual es un error.

1.5.2 Funcionamiento del botón *Calcular*

Primero se limpia la caja de texto que contiene el resultado del cálculo anterior y la caja de texto que muestra el mensaje de error de sintaxis. Posteriormente se verifica que las cajas de texto de la función y del valor a calcular no estén vacías. Si es así se muestra el mensaje correspondiente. De lo contrario se procede a verificar la sintaxis de la

función como se explicó en el algoritmo del botón *Tabular* y la sintaxis del valor a evaluar. De la misma manera solo se muestra el error específico de sintaxis para la función.

La condición necesaria para que se calcule el valor es que no haya cajas de texto vacías ni errores de sintaxis. Una vez confirmado esto, se toma el valor que se le dará a la variable independiente y se usa el método `void ValorVariable(char variableAlgebra, double valor)` para asignarlo. Después en una variable de tipo `double` se almacena el resultado de la evaluación y este valor es convertido a una cadena de caracteres para ser mostrado. El objetivo de contar con los componentes que permiten evaluar la función en un punto específico de acuerdo a las condiciones de diseño, es que el usuario pueda observar el valor con más precisión que los valores mostrados en la tabla, en este caso con 8 números después del punto decimal.

1.5.3 Funcionamiento del botón *Buscar Raíces*

El propósito de contar con éste botón es el de mostrar al usuario cuantas raíces se han encontrado y habilitar los componentes necesarios para mostrar los intervalos en los que se encuentran las raíces. El número de raíces encontradas, el número de raíz actual y los intervalos donde se encuentran las raíces corresponden a las cajas de texto NraizCT, RactualCT, X1CT, X2CT y deben ser limpiados cada vez que el botón es oprimido. Una vez que se ha pulsado el botón tabular se genera la tabla en la que los errores matemáticos que se puedan presentar se muestran en las celdas correspondientes con el texto "Error". El algoritmo que busca raíces se basa en analizar pares de valores de f(x) sin considerar las celdas que tienen error.

Después de limpiar las cajas de texto se obtiene el número de filas de la tabla para que ese número sea el límite el bucle *for* que se encarga de buscar raíces utilizando los valores de f(x) de la segunda columna. Antes de entrar al bucle se debe inicializar la variable entera `raíz=0` ya que dicha variable almacenará el número de raíces encontradas. Una vez en el bucle se lee el valor de la celda de acuerdo al contador y se analiza si existe o no algún error matemático. En caso de que no lo haya se comprueba si el valor actual de f(x) es igual a cero. Si es cero se encuentra una raíz exacta y el contador Raíz se incrementa en uno. El siguiente criterio usado es el cambio de signo entre el valor anterior y el valor actual. Para aplicar el criterio se realiza una multiplicación entre dichos valores para obtener una de las dos posibilidades:

- El resultado es positivo: no existe cambio de signo en el valor de la función en el par de puntos analizados, por lo tanto no se ha encontrado una raíz.
- El resultado es negativo y el cambio de signo existe por lo que se encuentra una raíz entre los dos puntos analizados, en consecuencia el contador Raíz se incrementa en uno.

Una vez analizado un par de puntos a la variable del valor anterior de tipo `float vant` se le asigna el valor actual que está almacenado en la variable de tipo `float vact` para hacer la siguiente comparación. El proceso se repite hasta que se llega al final de la tabla en cuestión. Si se ha encontrado al menos una raíz se mostrará al usuario el número de raíces y se habilitarán los componentes necesarios para ello, siendo estos:

- Los textos estáticos SiguienteST, AnteriorST, X1ST, X2ST, EncontrarST, RaicesST, RaizST.
- Las cajas de texto X1CT, X2CT, NraizCT, RactualCT.
- El botón de navegación DesplazarSB.

Si no se encuentra ninguna raíz los componentes se deshabilitaran.

1.5.4 Funcionamiento del botón de navegación.

Este algoritmo busca raíces según los criterios mencionados en el algoritmo del botón *Buscar Raíces* y almacena en arreglos los intervalos en los que se encuentran las raíces. Un vez que se han almacenado es necesario el uso de un botón que pueda ser pulsado hacia adelante y hacia atrás llevando la cuenta de las pulsaciones mediante un contador para que pueda ser usado como índice de los arreglos y mostrar en orden los valores almacenados. El

componente indicado para realizar esta función es por excelencia el componente WxSpinButton que permite obtener el valor actual del contador para ser guardado en un variable de tipo **`int.`**

El procedimiento de búsqueda es el mismo que en el caso anterior pero en este caso no solo se indicará el número de raíces si no también cada que se encuentre una raíz se almacenarán los valores de `(x)` entre los que se encuentre la raíz. Para ello se necesitan 2 arreglos que en este caso son **`float`** `mayor[50]` `y` **`float`** `menor[50]` cuyo índice será controlado por la variable entera `index.` Cuando se evalúe *condición=valor anterior * valor actual* y el resultado sea negativo se encontrará una raíz. El valor actual será guardado en el arreglo `mayor` y el valor anterior será guardado en el arreglo `menor,` ambos con el mismo índice para que cuando se muestre el intervalo en el que se encuentra una raíz se haga de manera correcta. Es importante que cada vez que se encuentre una raíz la variable `index` sea incrementada en uno para apuntar a la siguiente posición del arreglo.

Por otra parte cuando el valor de f(x) sea exactamente cero, una raíz exacta será encontrada por lo que se almacenará en ambos arreglos el valor actual de 'x' que es el que ha ocasionado que f(x) valga cero de la siguiente manera: `mayor[index]=menor[index]=xval` donde xval es el valor actual de 'x'. El ciclo termina cuando se llega al límite de la tabla y entonces los intervalos de las raíces encontradas estarán almacenados en los arreglos.

Posteriormente se obtiene el valor actual del botón de navegación, es decir, el valor del contador de las pulsaciones del botón y se almacena en una variable que en éste caso es **`int`** `svalue` para que este valor sirva de índice y ya que se incrementa y decrementa de uno en uno, los intervalos se muestren de manera ordenada. Sin embargo, existe un detalle a considerar en cuanto al número de pulsaciones ya que no puede exceder el número de raíces encontradas y tampoco ser menor a 1 debido a que si se pulsa hacia atrás una vez más devuelve -1. Por otra parte el valor almacenado en `svalue` es mostrado en una caja de texto para tener conocimiento de que número de raíz estamos observando. Por lo tanto no sería correcto que si hay sólo 3 raíces y se pulse el botón hacia adelante muestre la raíz número 4 y tampoco sería correcto que muestre la raíz -1.

En este caso si el valor del botón es -1 se fuerza a 1 para que la raíz menor que muestre sea la primera y se notifica con PraizDM al usuario que se trata de la primera raíz. Si es mayor que la cantidad de raíces se fuerza a tener el valor del número de raíces encontradas y se notifica al usuario con UraizDM que se trata de la última raíz. El primordial objetivo de esta acción es no perder control sobre el índice de los arreglos que contiene a los intervalos. En el caso de que el valor `svalue` no éste en los límites, la operación normal del código es extraer de los arreglos el intervalo de acuerdo índice, convertir a una cadena de caracteres los números y mostrarlo en las cajas de texto correspondientes.

1.6 Diagramas de flujo del Analizador de Raíces

En esta sección se presenta los diagramas de flujo de los botones *Tabular, Calcular, Buscar Raíces* y el *Botón de Navegación* entre las raíces. Cada botón cuenta con su propio diagrama de flujo ya que cada componente es programado por separado. En la figura 1.5 se muestra la simbología utilizada.

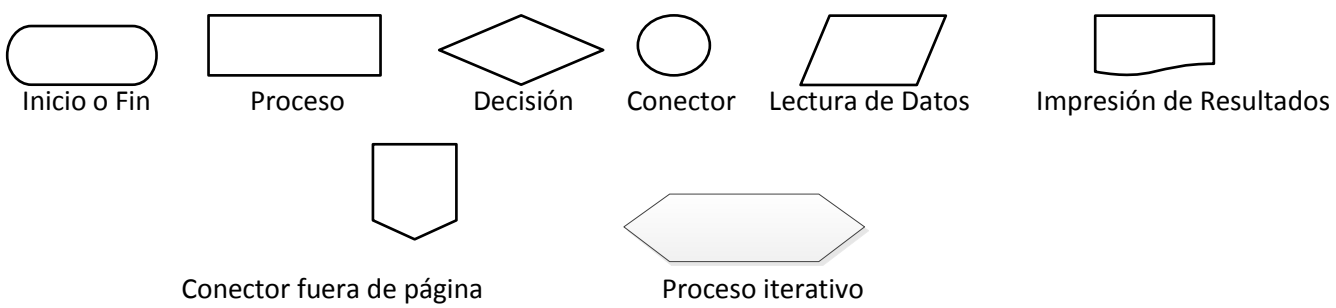
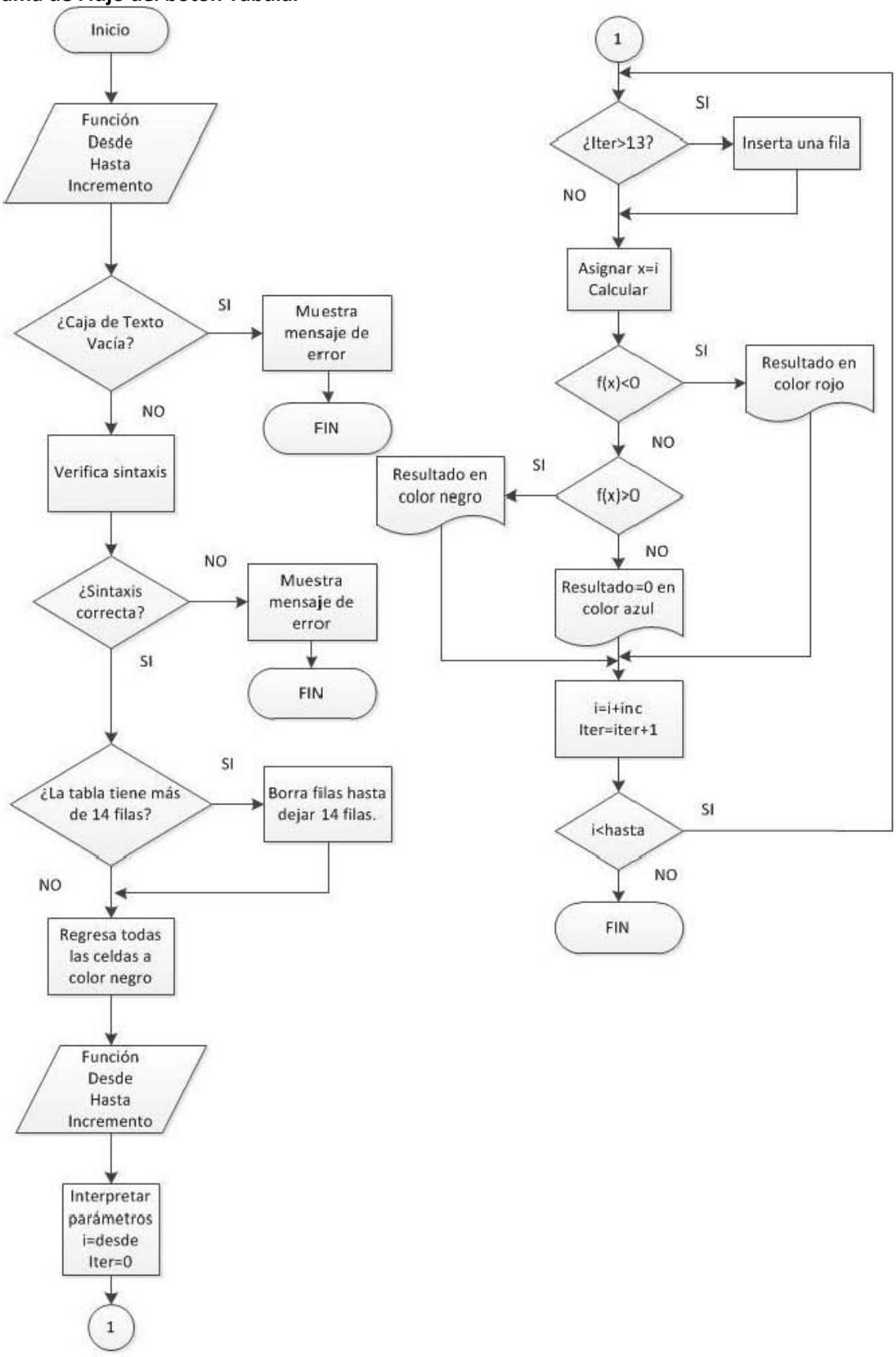


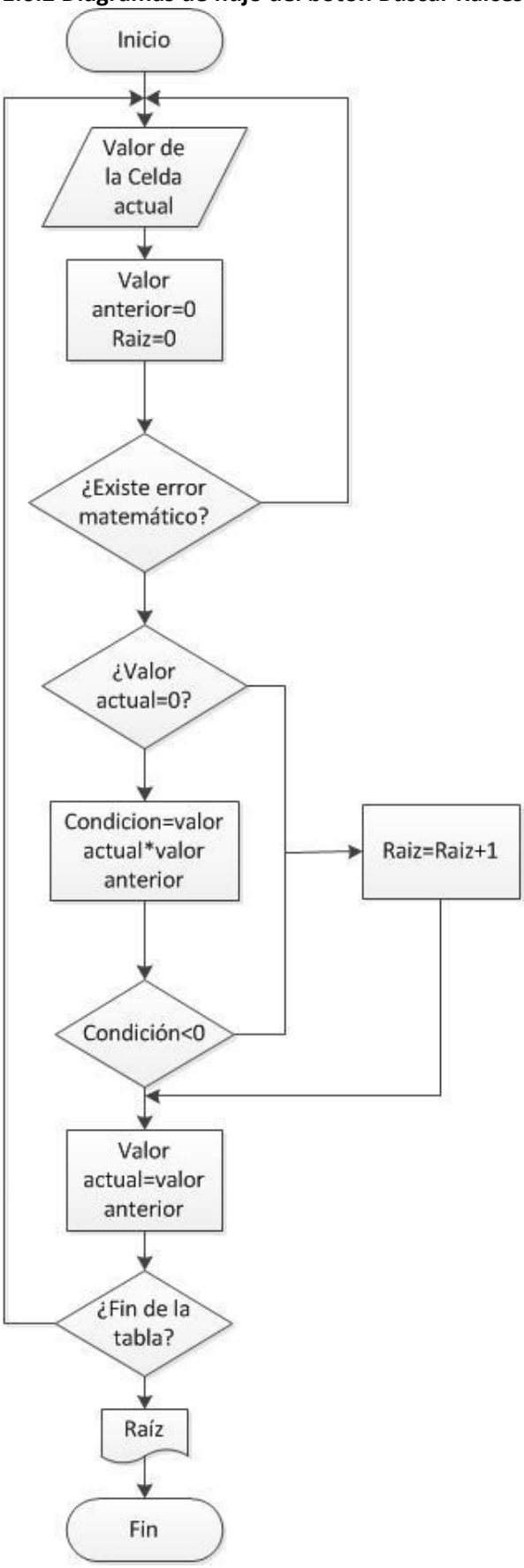
Figura 1.5 Simbología utilizada para los diagramas de flujo

Con la finalidad de no hacer excesivo el tamaño de los diagramas se han omitido pasos como la limpieza de las cajas de texto y la habilitación de componentes. Esta parte se puede ver detalladamente en la sección de código de cada programa.

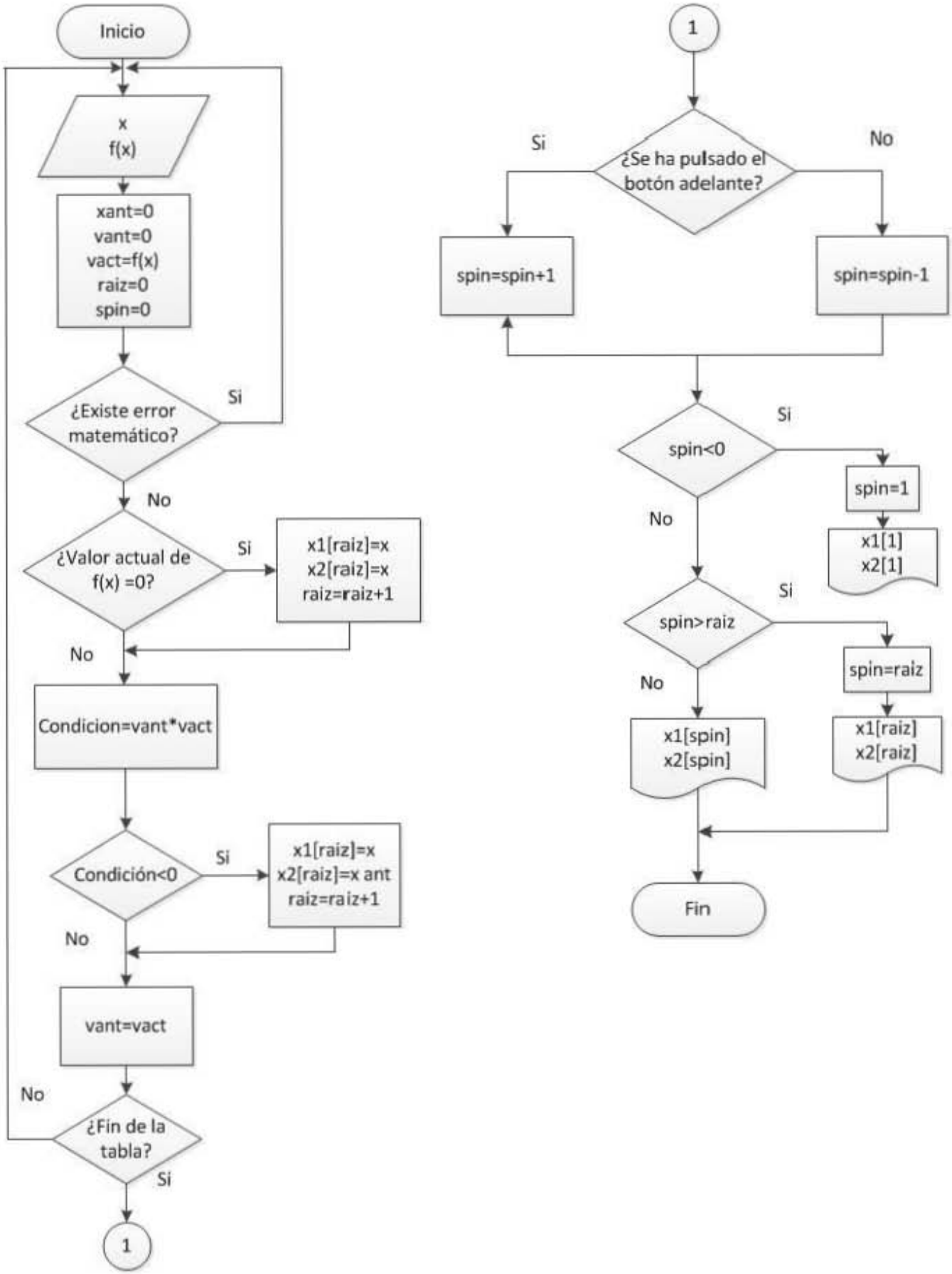
1.6.1 Diagrama de Flujo del botón Tabular



1.6.2 Diagramas de flujo del botón Buscar Raíces y del botón Calcular.



1.6.3 Diagrama de flujo del Botón de Navegación entre las raíces.



2. BISECCIÓN.

El método de bisección sirve para encontrar raíces de ecuaciones aprovechando el hecho de que una función en forma típica cambia de signo en la vecindad de una raíz. A esas técnicas se les llama *métodos de intervalos* porque se necesita de dos valores iniciales para la raíz. Como su nombre lo indica estos valores deben "encerrar" o estar sobre cualquier lado sobre la raíz. El *método de bisección,* conocido también como de corte binario, de partición en dos intervalos iguales o método Bolzano, es un método de búsqueda incremental en el que el intervalo se divide siempre en dos. Si la función cambia de signo sobre un intervalo, se evalúa la función en el punto medio. La posición de la raíz se determina situándola en el punto medio del subintervalo dentro del cual ocurre el cambio de signo. El proceso se repite hasta obtener una mejor aproximación. En la figura 2.1 se muestra un bosquejo gráfico del método.

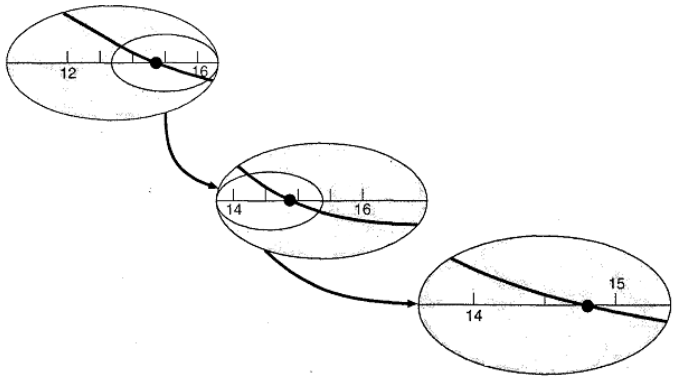


Figura 2.1 Bosquejo gráfico del método de Bisección

Este programa inicialmente fue contemplado para contar solo con los componentes necesarios para efectuar el método de bisección y con un botón el analizador de raíces fuera abierto como se muestra en la figura 2.2.

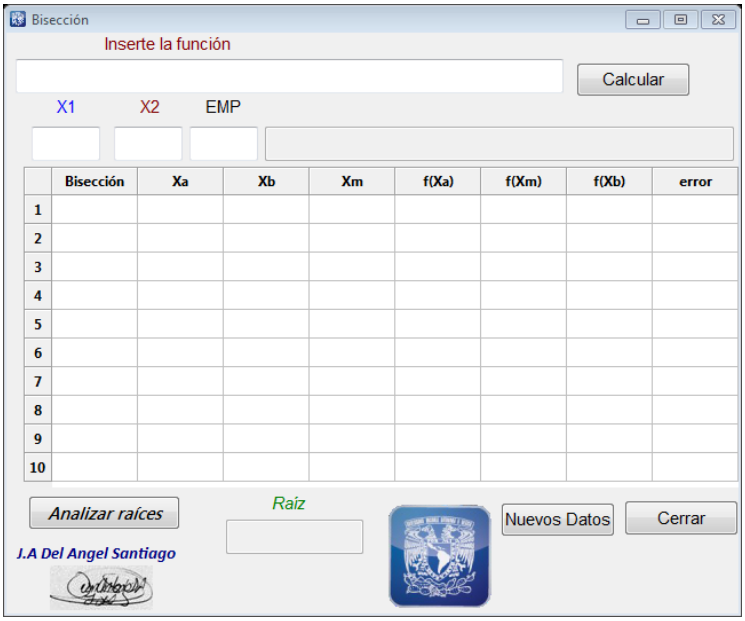


Figura 2.2 Programa de Bisección

Para evitar cualquier error y hacer al programa más completo y funcional se implementó el Analizador de Raíces en el programa Bisección para que cuando se oprima el botón *Analizar raíces* se muestren los componentes del Analizador como se aprecia en la figura 2.3.

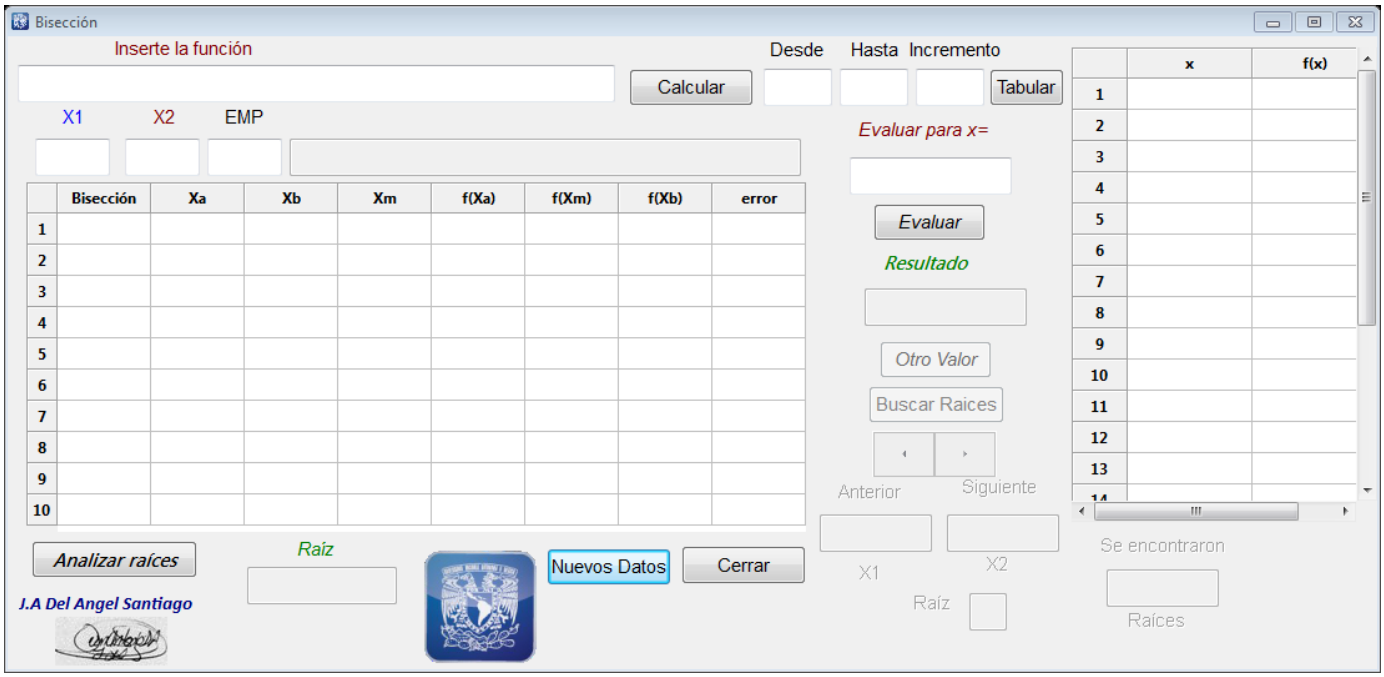


Figura 2.3 Implementación del Analizador de raíces en el programa Bisección

2.1 CONSIDERACIONES DE DISEÑO DEL PROGRAMA

- Los resultados obtenidos en cada iteración deben ser mostrados de manera ordenada en una tabla. Los resultados a mostrar son:
  - I. Número de Bisección.
  - II. Límites del intervalo.
  - III. Valor de xm (mitad del intervalo).
  - IV. El valor de la función al sustituir el valor de x del límite inferior.
  - V. El valor de la función al sustituir el valor de x del límite superior.
  - VI. El valor de la función al sustituir el valor de xm.
  - VII. El error absoluto.
- El último valor de xm obtenido en la tabla se debe mostrar en una caja de texto con al menos 8 dígitos a la izquierda del punto decimal.
- En el caso de que el usuario no sepa el intervalo donde se encuentra la raíz debe haber un botón que al ser pulsado muestre el analizador de raíces. Por lo tanto el analizador de raíces debe ser implementado dentro del programa de Bisección.
- Se debe contar con un botón que permita borrar el valor anterior sustituido en la función ya que el usuario puede estar interesado en que tan exacta es la raíz y sustituir más valores.

2.2 Componentes utilizados para el programa.

A continuación se listan los componentes usados en el programa. Se pueden observar en letra cursiva los componentes que corresponden solo al algoritmo de bisección. Los componentes del analizador se incluyen para que cuando se muestre el código de los botones *Analizar Raíces y Nuevos Datos* se pueda ver a que componentes se hacer referencia.

- **Cajas de texto donde:**

1. *Se inserta la función: FuncionCT*
2. *Se inserta el parámetro* $\boldsymbol{X_1}$*: X1BisCT.*
3. *Se inserta el parámetro* $\boldsymbol{X_2}$*: X2BisCT.*
4. *Se inserta el parámetro* ***EMP*** *(Error Mínimo Permitido): EmpCT*
5. Se inserta el parámetro **desde**: DesdeCT.
6. Se inserta el parámetro **hasta:** HastaCT.

7. Se inserta el parámetro **incremento:** IncCT.
8. Se muestra el límite inferior del intervalo donde se encuentra una raíz ($\boldsymbol{X_1}$)**:** X1CT.
9. Se muestra el límite superior del intervalo donde se encuentra una raíz ($\boldsymbol{X_2}$)**:** X2CT.
10. Se muestra el número de raíz actual: RactualCT.
11. Se muestra el número de raíces encontradas: NraizCT.
12. Se inserta un valor de **x** a evaluar: XvalCT.
13. Se muestra el **resultado** de la evaluación: ResultCT.
14. *Se muestra el error de sintaxis: SintaxisCT.*
15. *Se muestra la raíz encontrada: RaizresultCT.*

- **Botones**

16. Calcular: CalcB.
17. Tabular: TabulaB.
18. Cerrar: CerrarB.
19. Analizar Raíces: AnalizarB.
20. Nuevos Datos: NuevosB.
21. Buscar Raíces: RaicesB.
22. Evaluar: CalcularB.
23. Otro valor: OtroB.

- **Tablas**

24. Tabla donde se muestran los resultados del método Bisección: TablaResG.
25. Tabla donde se muestra la tabulación: TablaG.

- **Botón de Navegación.**

26. DesplazarSB.

- **Textos Estáticos**

27. Inserte la función: FuncionST.
28. X_1 (parámetro del método): X1BisST.
29. X_2 (parámetro del método): X2BisST.
30. EMP (Error Mínimo Permitido): EmpST.
31. Anterior: AnteriorST.
32. Siguiente: SiguienteST.
33. X1: X1ST.
34. X2: X2ST.
35. Raíz: RaizST.
36. Se encontraron: EncontrarST.
37. Raíces: RaicesST.
38. Evaluar para x=: EvaluarxST.
39. Resultado: ResultST.
40. Desde: DesdeST.
41. Hasta: HastaST.
42. Incremento: IncST.
43. Error de Sintaxis: SintaxisST
44. Raíz Calculada: RaizCalcST.

- **Diálogos de mensaje:**

45. Inserte una función: FuncionDM.
46. Introduzca el parámetro X_1: X1DM.
47. Introduzca el parámetro X_2: X2DM.

48. Introduzca el parámetro EMP: EmpDM.
49. Inserte el parámetro Desde: DesdeDM.
50. Inserte el parámetro Hasta: HastaDM.
51. Inserte el parámetro incremento: IncDM.
52. No se encontraron raíces: NoRaizDM.
53. Primera Raíz: PraizDM.
54. Última Raíz: UraizDM.
55. Inserte un valor a evaluar: XvalDM.
56. Error de Sintaxis: SintaxisDM.
57. Error de Sintaxis en los parámetros: SintaxisPDM.

2.3 FUNCIONAMIENTO DEL PROGRAMA

En esta sección se describe el funcionamiento del botón *Calcular* únicamente ya que los demás botones fueron explicados en el capítulo anterior. Dentro de la descripción del botón *Calcular* se encuentra el algoritmo del método *Bisección*. Los botones *Analizar Raíces* y *Nuevos Datos* básicamente están dedicados a mostrar y ocultar los componentes del analizador de raíces, dichos componentes se detallan en apéndice C.2. De la misma manera el botón *Otro Valor* limpia automáticamente el valor anterior insertado para ser evaluado y solo es necesario habilitar, deshabilitar y limpiar componentes mostrados en el código.

2.3.1 Funcionamiento del botón Calcular

Para comenzar se debe limpiar la caja de texto SintaxisCT y RaizresultCT que muestran información sobre el error de sintaxis y la raíz encontrada respectivamente. También se debe limpiar TablaResG para mostrar los nuevos resultados y deshabilitar el texto estático SintaxisST y la caja de texto SintaxisCT que solo son habilitados si ocurre un error de sintaxis en la función.

Se declaran los arreglos que se utilizan para interpretar las expresiones tecleadas en la función y los parámetros además de que se crea el objeto `evaluadorExpresiones` de la clase `Evaluar`. Una vez creado el objeto y si todas las cajas de texto requeridas se han llenado se hace uso de los métodos de la clase Evaluar necesarios para la revisión de sintaxis de lo contrario se muestra el mensaje de error correspondiente según la caja de texto que haya sido omitida. En esta ocasión se requieren 3 parámetros y la función para llevar a cabo el método. Sin embargo, solo se necesitan 3 variables para almacenar el código de error obtenido en la revisión de sintaxis esto debido a que el parámetro *emp* que corresponde al error mínimo permitido no necesita revisión de sintaxis ya que puede ser dado solo como un número de la forma 0.001 por ejemplo o números como 1e-9 que es equivalente a 1×10^{-9}. Él código de error de la función se almacena en la variable entera `verifsin` y el código error de los parámetros $x_1\ y\ x_2$ se almacena en las variables `verifsinp1` y `verifsinp2`.

Si no ha ocurrido ningún error de sintaxis las variables `verifsin`, `verifsinp1` y `verifsinp2` almacenan el valor 0. La variable `verifsin` está asociada al mensaje SintaxisDM que muestra "Error de Sintaxis" que se muestra en caso de que la variable no valga 0. Por otra parte las variables `verifsinp1` y `verifsinp2` se suman y se almacenan en la variable entera `verifpar` que está asociada al mensaje SintaxisPDM que muestra "Error de sintaxis en los parámetros". Además la variable entera `validator` verifica que no se haya omitido el llenado de ninguna caja de texto.

Después de revisar lo anterior se eliminan las filas sobrantes para la nueva tabulación y se obtienen los valores de $x_1\ y\ x_2$ y se almacenan en las variables de tipo **double** que son `ini` y `fin` respectivamente. Se analiza la función para dejar todo listo para sustituir valores y comenzar el algoritmo del método Bisección propiamente. Los pasos se describen a continuación:

1. Se inicializa la variable `xa` (xm anterior) y la variable `iter` igual a cero.

   $$xa = iter = 0$$

   Se repite mientras que **ea>emp:**

2. Se establece un color para cada columna teniendo en cuenta la fila actual:
   - Rojo para las columnas 1 y 4.
   - Azul para las columnas 2 y 6.
   - Café para las columnas 3 y 5.
3. Si el número de iteraciones es mayor de 9 se inserta una fila cada nueva iteración para ajustar según el número de filas necesarias para mostrar los resultados de cada iteración.
4. Se calcula el valor de xm:
$$xm = \frac{ini + fin}{2}$$
5. Se calcula el error `ea` y se obtiene su valor absoluto.
$$ea = abs(xa - xm)$$
6. Se evalúa `f(ini)` y se almacena en `fx1`.
$$fx1 = f(ini)$$
7. Se evalúa `f(fin)` y se almacena en `fx2`.
$$fx2 = f(fin)$$
8. Se evalúa `f(xm)` y se almacena en `fx3`.
$$fx3 = f(xm)$$
9. Se convierten a cadenas de caracteres los valores siguientes para ser mostrados en las columnas indicadas:
   - Número de Bisección en la columna 1.
   - Valor de x del límite inferior del intervalo en la columna 2.
   - Valor de x del límite superior del intervalo en la columna 3.
   - Valor de xm en la columna 4.
   - f(ini) en la columna 5.
   - f(xm) en la columna 6.
   - f(fin) en la columna 7.
   - Error en la columna 8.
10. Se evalúan las condiciones para asignar el nuevo intervalo:
$$cond1 = f(ini) * f(xm) \qquad cond2 = f(fin) * f(xm)$$
11. Para la asignación del nuevo intervalo se pueden dar dos situaciones:
    - El signo de la condición 1 (cond1) es negativo, significa que la raíz se encuentra entre los valores del límite inferior del intervalo y xm por lo que se asigna el nuevo límite superior $fin = xm$
    - El signo de la condición 2 (cond2) es negativo, significa que la raíz se encuentra entre los valores del límite superior del intervalo y xm por lo que el nuevo límite inferior es $ini = xm$.
12. Se asigna el valor de xm a xa (xm anterior) para calcular el error en la siguiente iteración.
13. Se incrementa el valor de las iteraciones en uno.

Una vez terminado el bucle **do while** se toma el último valor calculado de xm y se muestra en la caja de texto que corresponde a la raíz encontrada con 11 dígitos decimales.

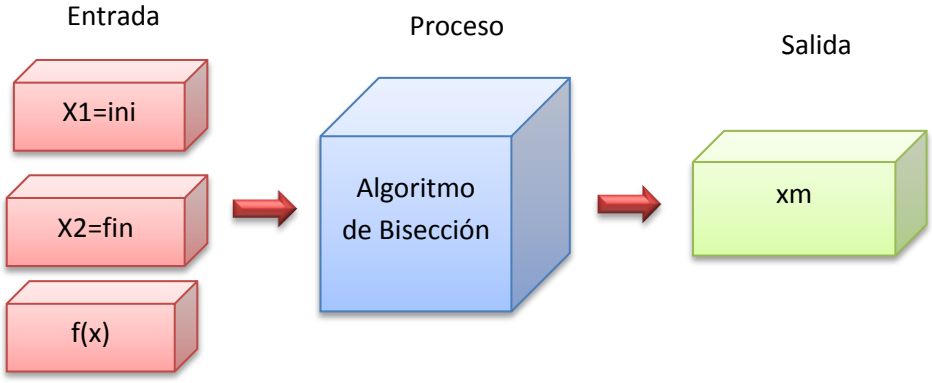
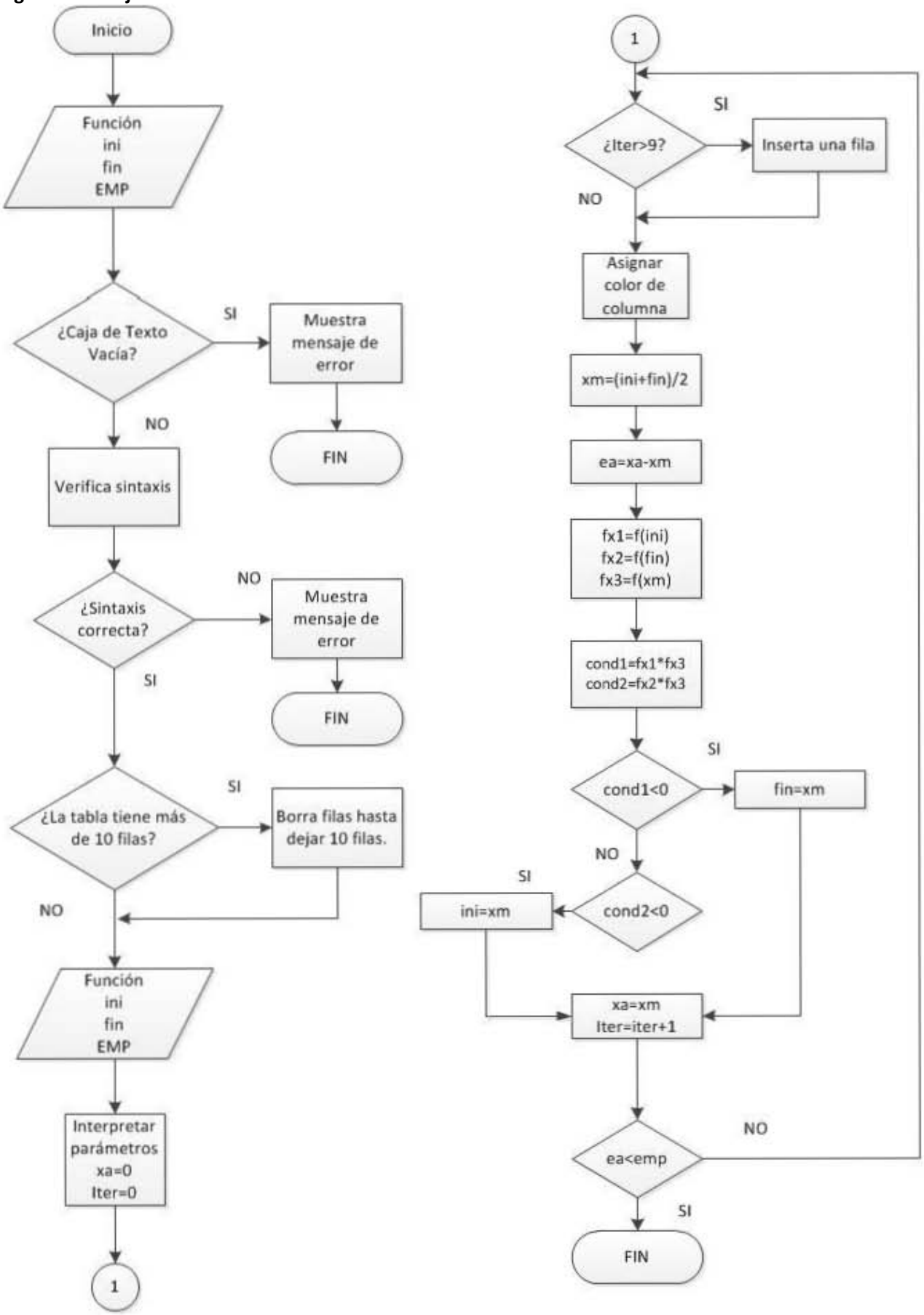


Figura 2.4 Diagrama de entrada, proceso, salida

2.4 Diagrama de flujo del botón Calcular



3. NEWTON RAPHSON

En el método anterior que usa intervalos, la raíz se encuentra dentro de estos mismos, dada por un límite inferior y otro superior. La aplicación sucesiva de estos métodos siempre genera aproximaciones cada vez más cercanas a la raíz. Tales métodos son conocidos como *convergentes,* ya que se acercan progresivamente a la raíz a medida que avanza el cálculo. En contraste, los *métodos abiertos* se basan en fórmulas que requieren únicamente de un solo valor de inicio *x* o que empiecen con un par de ellos, pero que no necesariamente encierren la raíz. Como tales alguna vez divergen o se alejan de la raíz verdadera a medida que crece el número de iteraciones (véase figura 3.1b). Sin embargo cuando los métodos abiertos convergen (véase figura 3.1c), por lo general lo hacen mucho más rápido que los que usan intervalos.

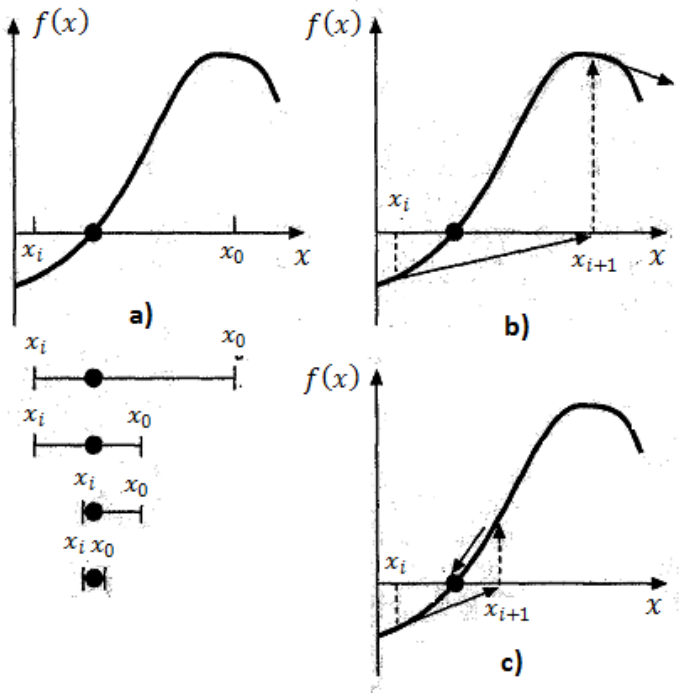


Figura 3.1 Esquema gráfico de las diferencias fundamentales entre los métodos para la localización de las raíces; los que utilizan intervalos a) y los métodos abiertos b) y c). En a), se ilustra el método de bisección, la raíz está contenida dentro del intervalo dado por x_i *y* x_0. En contraste, con los métodos abiertos ilustrados en b) y c), se usa una fórmula para proyectar de x_i a x_{i+1}, con un esquema iterativo. De esta manera cualquiera de los métodos puede b) divergir o c) converger en forma rápida dependiendo de los valores iniciales.

Tal vez, dentro de las fórmulas para localizar raíces, la fórmula de Newton-Raphson (véase figura 3.2) sea la más ampliamente usada. Si el valor inicial de la raíz es x_i entonces se puede extender una tangente desde el punto $[x_i, f(x_i)]$. El punto donde esta tangente cruza al eje x representa una aproximación mejorada de la raíz. El método de Newton Raphson se puede obtener sobre la base de una interpretación geométrica. Como en la figura 3.2, la primera derivada en x es equivalente a la pendiente:

$$f'(x_i) = \frac{f(x_i) - 0}{x_i - x_{i+1}} - - - 3.1$$

Que se puede ordenar para obtener

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} - - - 3.2$$

La cual es conocida como la fórmula de Newton-Raphson.

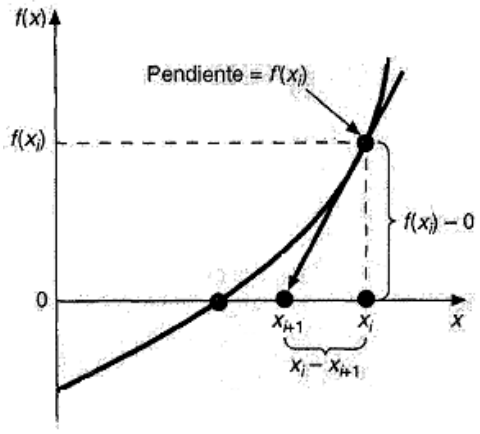


Figura 3.2 Esquema gráfico del método de Newton- Raphson. Se extrapola una tangente a la función de x_i [esto es $f'(x_i)$] para obtener una aproximación de la raíz en x_{i+1}.

Este programa requiere de menos iteraciones con respecto al método Bisección para encontrar la raíz de una función pero es necesario que el usuario inserte la derivada de la función dada. Se aprecia en la figura 3.3 la ventana del programa en la que se puede ver que este programa de igual manera que Bisección cuenta con el botón Analizar Raíces ya que se ha implementado también el Analizador de Raíces como se muestra en la figura 3.4 después de pulsar el botón.

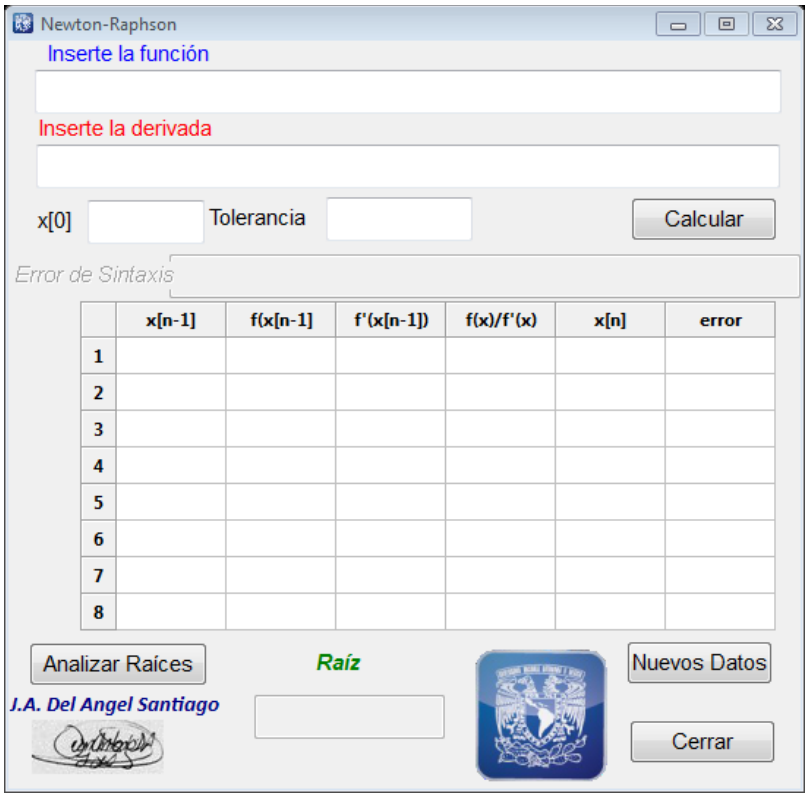


Figura 3.3 Ventana del programa Newton-Raphson

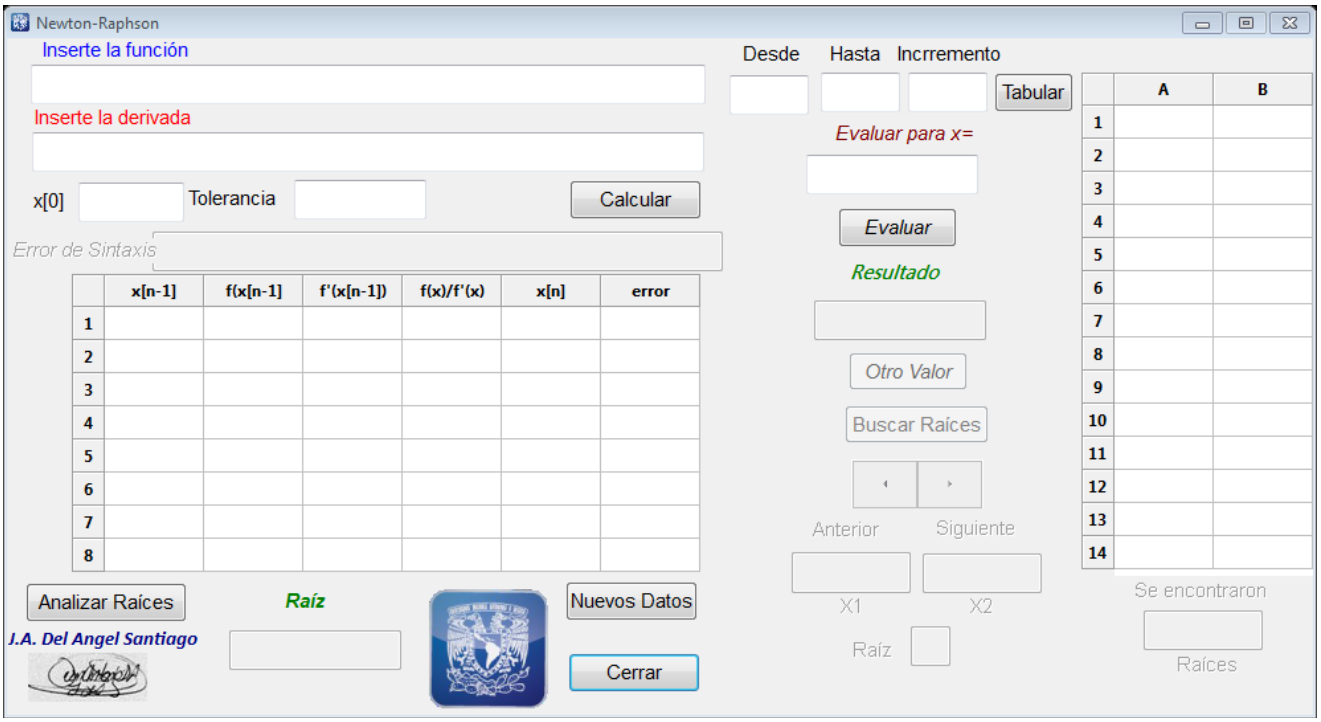


Figura 3.4 Implementación del Analizador de Raíces en el programa.

3.1 Consideraciones de diseño del programa.

- Aunque sea hará la revisión de dos funciones se conservará una sola caja de texto para mostrar errores de sintaxis por lo que se hará primero la revisión de una función y si no tiene errores se revisará la derivada.
- El usuario insertará la derivada de la función dada.
- De igual manera que para el programa de Bisección se implementará el Analizador de raíces en el programa.
- Se debe contar con una caja de texto para mostrar la raíz encontrada y de ese modo se visualice rápidamente sin la necesidad de buscarla en la tabla.
- La tabla contará con 6 columnas en las que se mostrará la siguiente información de acuerdo la expresión del método de Newton Raphson y el número de columna.

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

1. x_{n-1} 2. $f(x_{n-1})$ 3. $f'(x_{n-1})$ 4. $\frac{f(x_{n-1})}{f'(x_{n-1})}$ 5. x_n 6. error

3.2 Componentes usados para el programa.

En el capítulo anterior se mostraron los componentes del Analizador de Raíces además de los del método de Bisección, con el fin de indicar en el código que componentes se ocultaban y mostraban al presionar los botones Analizar Raíces y Nuevos Datos, razón por la cual se omiten en la siguiente lista. A continuación se muestran los componentes del método de Newton Raphson.

- **Cajas de texto donde:**

1. Se inserta la función: FuncionCT.
2. Se inserta la derivada de la función: DerivadaCT.
3. Se inserta el parámetro x[0]: X0CT.
4. Se inserta la tolerancia: TolCT.
5. Se muestra el error de sintaxis en las funciones: SintaxisCT.
6. Se muestra la raíz calculada: RaizResultCT.

- **Botones:**

7. Calcular: CalcB.
8. Analizar Raíces: AnalizarB.
9. Nuevos Datos: NuevosB.
10. Cerrar: CerrarB.

- **Tabla**

11. Se muestran los resultados de cada iteración en: TablaResG.

- **Textos Estáticos**

12. Inserte la función: FuncionST.
13. x[0]: DerivadaST.
14. Tolerancia: TolST.
15. Inserte la derivada: DerivadaST.
16. Error de Sintaxis: SintaxisST.
17. Raíz: RaizCalcST.

- **Diálogos de mensaje:**

18. Inserte una función: FuncionDM.
19. Inserte la derivada de la función dada: DerivadaDM.
20. Error de Sintaxis en la función: SintaxisDM.
21. Error de Sintaxis en la derivada: SintaxisDDM.
22. Inserte el valor de x[0]: X0DM.
23. Inserte la tolerancia: TolDM.
24. Error de Sintaxis en los parámetros. SintaxisPDM.

3.3 Funcionamiento del botón Calcular.

En el código del botón calcular se implementa el algoritmo de Newton-Raphson. En este caso la revisión de sintaxis de las funciones se hace una a la vez debido a que solo se ha considerado más práctico colocar solo una caja de texto para mostrar errores de sintaxis. De tal manera que una vez que comprueba que la función dada no tiene errores de sintaxis procede a revisar la sintaxis de la función derivada. Si se revisaran como se ha hecho en los programas anteriores y ambas tuvieran error de sintaxis se mostrarían ambos errores en la misma caja de texto lo cual sería incómodo. De la misma manera se verifica que no se haya omitido el llenado de alguna de las cajas de texto requerido y se eliminan las filas excedentes como se ha hecho anteriormente para las tablas. Posteriormente se aplica el algoritmo de Newton-Raphson que consiste en los siguientes pasos donde se incluyen pasos para mostrar resultados.

1. Se lee el valor de x[0] y el valor de la tolerancia (`tol`).
2. Se inicializa la variable del número de iteraciones `iter=0`.
3. Se lee y se calcula:
$$fe = f(x[iter])$$
4. Se lee y se calcula:
$$fpe = f'(x[iter])$$
5. Se aplica la expresión del método de Newton-Raphson:
$$x[iter+1] = x[iter] - \frac{f(x[iter])}{f'(x[iter])}$$
6. Se calcula el error:
$$ea = abs(x[iter+1] - x[iter])$$
7. Se convierten a cadenas de caracteres los valores siguientes paras ser mostrados en las columnas indicadas en las consideraciones de diseño del programa.

i. x[iter-1] en la primera columna.
ii. f(x[iter-1]) en la segunda columna.
iii. f'(x[iter-1]) en la tercera columna.
iv. El cociente f(x[iter-1]) / f'(x[iter-1]) en la cuarta columna.
v. x[iter] en la quinta columna.
vi. Error en la sexta columna

8. Se incrementa el número de iteraciones en uno.
$$iter = iter + 1$$
9. Se verifica si se cumple la condición de repetir $ea > tol$.
10. Una vez que ha acabado el proceso iterativo se toma el último valor calculado de x[iter-1] (debido a que en la última iteración se volvió a incrementar iter) y se convierte a una cadena de caracteres para ser mostrado en una caja de texto ya que es la raíz que se ha encontrado.

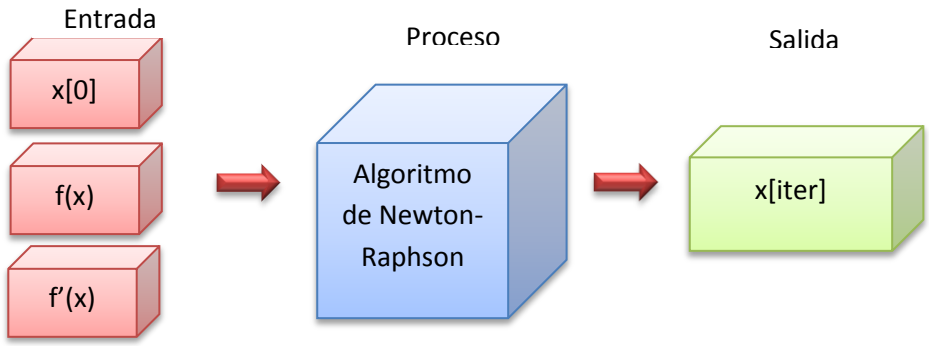
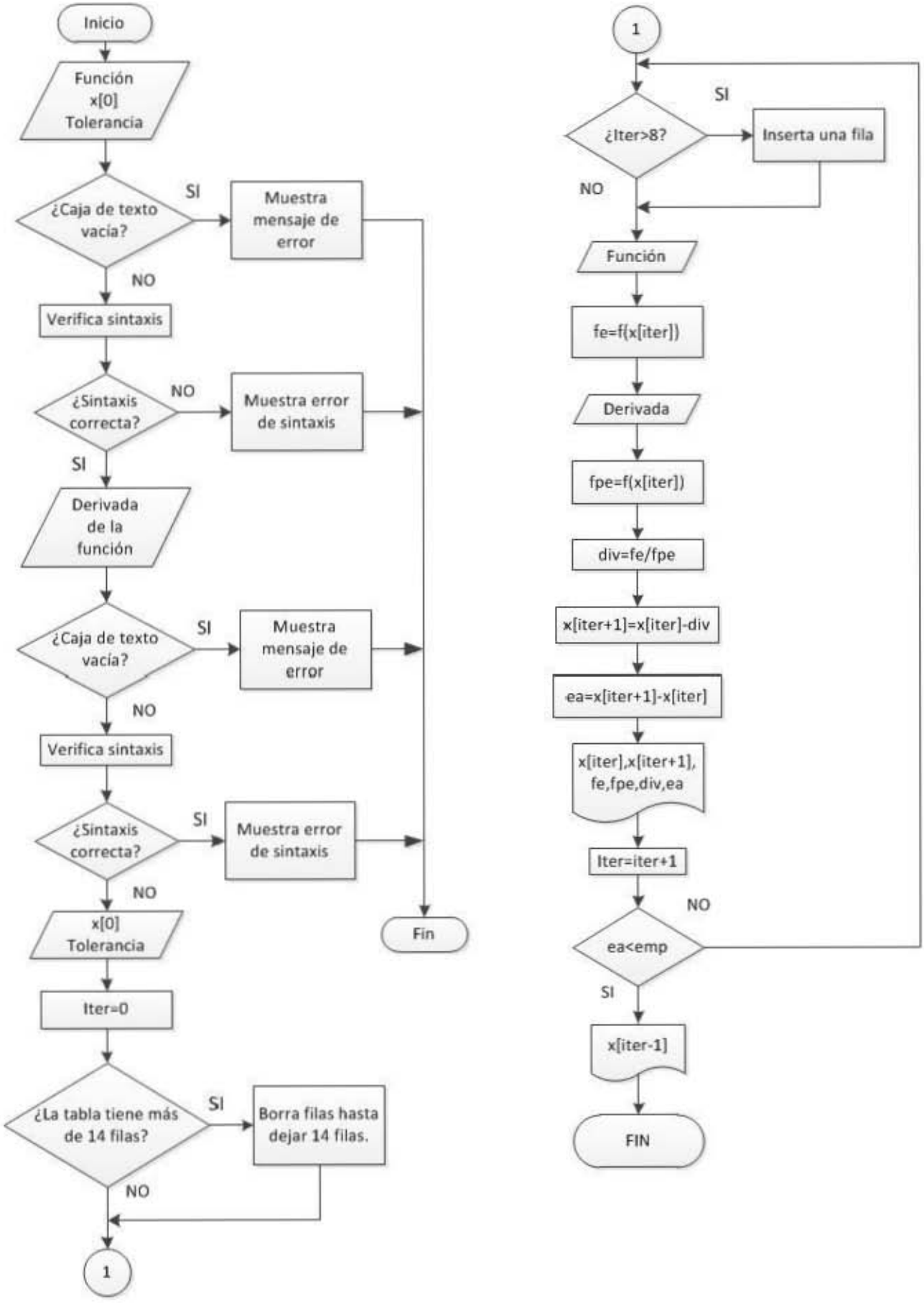


Figura 3.5 Diagrama de entrada, proceso, salida

3.4 Diagrama de flujo del botón Calcular.



4. Descomposición LU.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ l_{21} & 1 & 0 \\ l_{31} & l_{32} & 1 \end{pmatrix} \begin{pmatrix} u_{11} & u_{12} & u_{13} \\ 0 & u_{22} & u_{23} \\ 0 & 0 & u_{33} \end{pmatrix}$$

Su nombre se deriva de las palabras inglesas "Lower" y "Upper", que en español se traducen como "Inferior" y "Superior". Estudiando el proceso que se sigue en la descomposición LU es posible comprender el porqué de este nombre, analizando cómo una matriz original se descompone en dos matrices triangulares, una superior y otra inferior. La descomposición LU involucra solo operaciones sobre los coeficientes de la matriz [A], proporcionando un medio eficiente para calcular la matriz inversa o resolver sistemas de álgebra lineal.

Este programa tiene una estructura muy distinta a los programas anteriores ya que su funcionamiento se basa en 5 tablas. Dos de ellas son necesarias para introducir los datos del sistema de ecuaciones y las otras tablas muestran resultados de la aplicación del método. Además se introduce el uso de la lista desplegable WxChoice que en este caso sirve para ajustar las tablas dependiendo del orden del sistema que puede ser desde 2x2 hasta 10x10. Los ciclos **for** juegan un papel primordial para obtener los datos de las tablas, procesarlos y mostrar los resultados. Se analiza la sintaxis de cada valor introducido en las celdas ya que pueden ser introducidos como fracciones, por ejemplo. Se muestra en la figura 4.1 el programa para sistemas de 2x2 y en la figura 4.2 para sistemas de 10x10.

Figura 4.1 Programa descomposición LU para sistemas de 2x2

4.1 Consideraciones de diseño del programa.

- El programa debe funcionar para sistemas de orden 2x2 hasta 10x10 siendo el sistema de 2x2 el establecido por defecto.
- Se debe contar con una lista desplegable de la cual se elija el orden del sistema.
- Una vez seleccionado el orden la lista desplegable deberá ser desactivada para no cambiar el orden accidentalmente ya que cada que se cambia el orden todas las tablas son limpiadas.
- Se debe contar con un botón que reactive la lista desplegable.
- La posición de las tablas debe ser adecuada para que cuando aumenten de tamaño las tablas no se traslapen.
- La información que se debe mostrar es la matriz LU con color azul para los coeficientes de la matriz L y rojo para los coeficientes de la matriz U. También se debe mostrar la matriz de coeficientes c[n] del paso de la eliminación hacia adelante y el resultado del sistema en la matriz x[n].

- Para que la solución del sistema sea confiable la revisión de sintaxis es algo muy importante por lo que se debe revisar la sintaxis de cada coeficiente del sistema de ecuaciones. Se debe contar con una caja de texto que indique las coordenadas de la tabla en la que se ha cometido un error para su fácil detección y corrección.

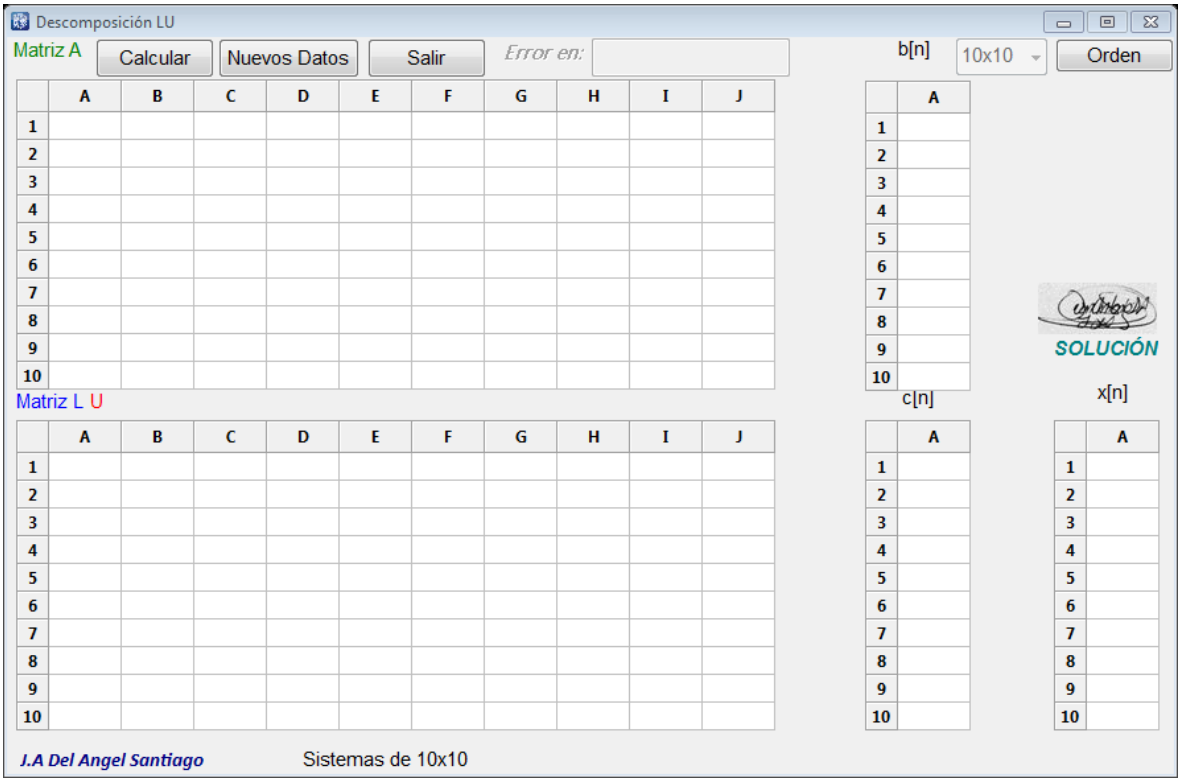


Figura 4.2 Programa de descomposición LU para sistemas de 10x10

4.2 Componentes usados para el programa

- **Caja de Texto donde:**
  1. Se muestra el error de sintaxis: SintaxisCT.

- **Botones:**
  2. Orden: OrdenB.
  3. Nuevos Datos: NuevosB.
  4. Salir: SalirB.
  5. Calcular: CalcularB

- **Tablas:**
  6. Coeficientes del sistema: CoefG.
  7. Matriz LU: LuG.
  8. Términos independientes: IndG.
  9. Coeficientes (c): MatCG.
  10. Solución del sistema: SoluciónG.

- **Lista desplegable**
  11. Elegir orden del sistema: OrdenLD.

- **Textos estáticos:**
  12. Matriz A: CoefST.

13. b[n]: IndST.
14. Matriz LU : LuST.
15. c[n]: MatCST
16. x[n]: SolucionST.
17. Solución: RespuestaST.
18. Sistemas de 2x2: 2x2ST
19. Error en: ErrorST.

- **Diálogos de mensaje.**
    20. Error de sintaxis en coeficientes: SintaxisCDM.
    21. Error de sintaxis en términos independientes: SintaxisIDM.

4.3 FUNCIONAMIENTO DEL PROGRAMA.

En esta sección se describe la forma en que trabaja la lista desplegable, el botón que habilita a la lista desplegable, y el botón Calcular en el que se implementa el método de descomposición LU. El botón Nuevos Datos solo limpia las tablas, deshabilita y limpia la caja de texto en la que se muestran las posiciones de la tabla donde ha ocurrido algún error de sintaxis.

4.3.1 FUNCIONAMIENTO DE LA LISTA DESPLEGABLE

El funcionamiento de la lista desplegable se basa en el número de opción seleccionada, las opciones posibles se observan en la figura 4.3.

Figura 4.3 Opciones de la lista desplegable

El texto que se teclea en cada opción es para indicar al usuario las opciones posibles. Sin embargo una vez que se selecciona una opción este componente devuelve un entero con el numeró de opción seleccionada. Se muestra en la figura 4.4 el valor entero que devuelve la lista desplegable al seleccionar cada opción. Para este programa se ha establecido por defecto un sistema de orden 2x2 por lo que el número de filas y de columnas es igual a 2.

Opción	*Valor devuelto*
3x3	0
4x4	1
5x5	2
6x6	3
7x7	4
8x8	5
9x9	6
10x10	7

Figura 4.4 Valores devueltos según la opción seleccionada

En este caso el número de la selección se almacena en la variable entera `sel` y el número de filas y columnas a insertar se almacena en la variable entera `ins=sel+1`. Así por ejemplo si se selecciona la opción 3x3 la selección es 0 pero se suma uno y el número de filas y columnas a insertar es `ins=1`, por lo tanto dado que el sistema establecido por defecto es de 2x2 se inserta una fila y una columna para que la tabla sea para un sistema de 3x3. De la misma manera se modificara la tabla que corresponde a la matriz LU y a las tablas que solo cuentan con una columna solo se agregarán las filas indicadas por la variable `ins`.

Una vez que se han insertado las filas y columnas necesarias se limpian las tablas y se deshabilita la lista desplegable. Sería incómodo que por error el usuario después de insertar valores en las tablas borrara accidentalmente todo lo que se ha insertado, de tal manera cuando se deshabilita la lista se habilita el botón *Orden* que sirve para habilitar nuevamente la lista teniendo en cuenta que se cambiará el orden del sistema y se limpiarán todas las tablas. Posteriormente se utiliza la función Fit() para ajustar el tamaño de la ventana al tamaño de las tablas.

4.3.2 Funcionamiento del botón *Orden*

Este botón se encarga de eliminar las filas y columnas necesarias para restablecer el orden del sistema a 2x2 y limpia todas las tablas además de que deshabilita este botón y habilita la lista desplegable. Para restablecer el orden por defecto se llevan a cabo 2 pasos:

- Se obtiene el orden del sistema, para esto se puede obtener el número de filas de cualquier tabla y se almacena en la variable entera `lim`.
- Se utiliza la variable entera `erase=lim-2`. Al número de filas se le resta 2 para obtener el número de filas y columnas que serán borradas.
- Se analiza si el número de filas es mayor a 2, en tal caso se eliminan las filas y columnas indicadas por `erase`.

Es necesario de igual manera usar la función Fit() para ajustar el tamaño de la ventana.

4.3.3 Funcionamiento del botón *Calcular*

Primero se explica la revisión de sintaxis de cada valor ingresado en las celdas de las tablas que corresponden a los coeficientes del sistema y los términos independientes. El algoritmo del método de descomposición LU es ejecutado en el código de este botón. Se explicará cada ciclo **for** utilizado para la obtención de las matrices de resultados debido a que se usan ciclos anidados.

4.3.3.1 Revisión de Sintaxis

La primera acción que se lleva a cabo al presionar el botón es limpiar las tablas LuG, MatCG y SolucionG que muestran los resultados del método. Posteriormente se declara e inicializa el arreglo **`int`** `sintax[101]` que es de 101 elementos debido a que cuando el orden seleccionado es 10x10 deberá contener 100 valores. Este arreglo se encarga de almacenar el código de error de cada una de los coeficientes insertados en la tabla 1, se declaran los arreglos necesarios para usar el evaluador de expresiones algebraicas y se crea el objeto

`evaluadorExpresiones` de la clase `Evaluar`, se obtiene el orden del sistema y se almacena en la variable entera `ord1`.

Una vez realizados los pasos anteriores y contando con el orden del sistema se aplica un algoritmo que revisa la sintaxis de lo que se ha introducido en cada celda de la tabla de coeficientes. En este algoritmo tiene dos ciclos **for** para analizar todas las celdas**,** el ciclo externo indica la fila que se está analizando y el ciclo interno recorre todos los elementos de la fila. Hecho esto el ciclo externo se incrementa en uno para indicar que se está analizando la segunda fila y de la misma manera el ciclo interno recorre todos los elementos, esto se repite hasta haber analizado todos los elementos de la matriz.

De manera específica el ciclo interno copia la expresión de cada celda para obtener su código de error y almacenarlo en el arreglo `sintax`. Se aprovecha el hecho de que con los ciclos se puede indicar la posición actual de la tabla para que en caso de que se detecte un error la posición se muestre en la caja de texto destinada a ello a manera de coordenadas para que el usuario sepa exactamente en qué celda ha cometido un error. Una vez revisada la sintaxis de la tabla se hace la sumatoria de los elementos del arreglo. Si el resultado de la sumatoria es igual a cero no se ha cometido ningún error de sintaxis de lo contrario se muestra el mensaje "Error de sintaxis en coeficientes" y se indican las posiciones donde se han cometido errores como se muestra en la figura 4.5.

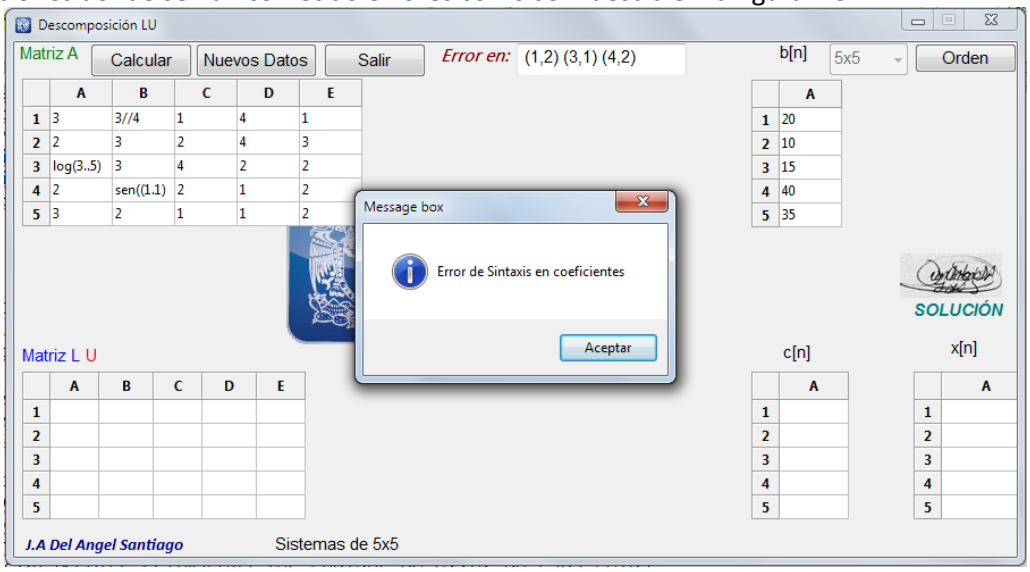


Figura 4.5 Error de sintaxis en coeficientes.

Si la sumatoria ha resultado cero se procede a limpiar el arreglo `sintax` y dado que la tabla de términos independientes cuenta solo con una columna esta vez con un solo ciclo **for** se obtienen los códigos de error. En este caso la posición de la tabla es indicada solo por un entre paréntesis ya que hay una sola columna como se observa en la figura 4.6. Se realiza la sumatoria del arreglo sintax y en caso de haber error muestra el mensaje "Error de sintaxis en términos independientes". Si todo ha ido bien y no han ocurrido errores mediante dos ciclos **for** y los métodos de la clase Evaluar se almacenan en el arreglo bidimensional **`double`** `A[ord1][ord1]`los valores de la tabla de coeficientes y se declara el arreglo **`double`** `B[ord1][ord1]`para almacenar los valores de la matriz L y la matriz U en una matriz única.

4.3.3.2 Expresiones del método

1. El n-ésimo renglón de U se obtiene de:

$$u_{rj} = a_{rj} - \sum_{k=1}^{r-1} l_{rk} u_{kj} \quad ---(4.1)$$

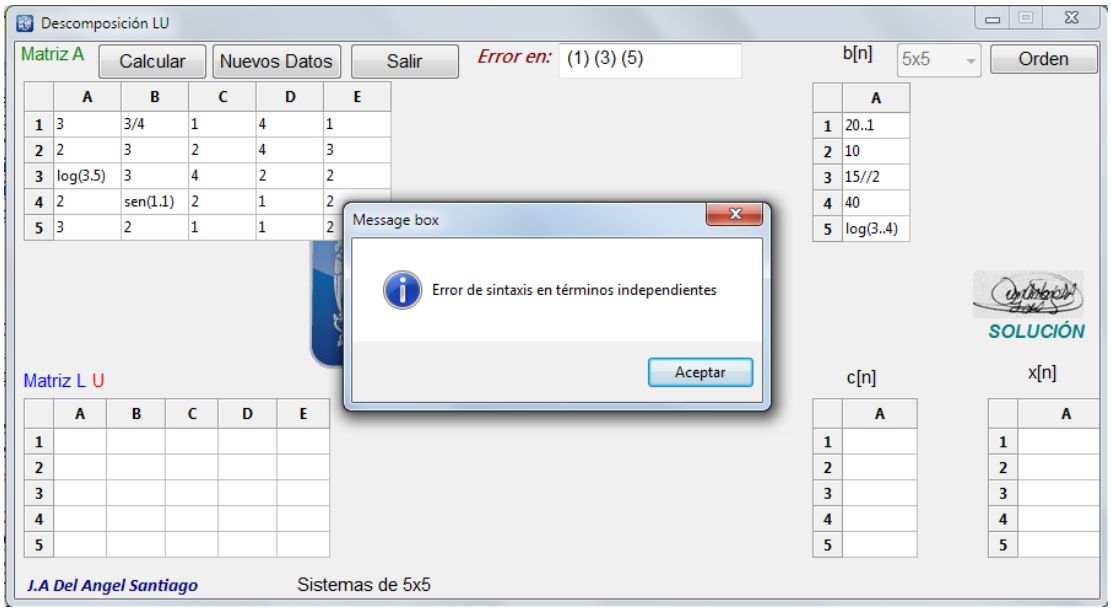


Figura 4.6 Error de sintaxis en términos independientes.

2. La n-ésima columna de L se obtiene de:

$$l_{ic} = \left[a_{ic} - \sum_{k=1}^{c-1} l_{ik}u_{kc}\right] / u_{cc} \quad ---(4.2)$$

3. Paso de la eliminación hacia adelante:

$$c_1 = y_1 \quad ---(4.3)$$

$$c_i = y_i - \sum_{j=1}^{i-1} l_{ij}z_j \quad ---(4.4)$$

4. Paso de la eliminación hacia atrás.

$$x_n = \frac{z_n}{u_{nn}} \quad ---(4.5)$$

$$x_i = \left[z_i - \sum_{j=j+1}^{n} u_{ij}x_j\right] / u_{ij} \quad ---(4.6)$$

4.3.3.3 Diseño del código para la obtención de la matriz LU

La forma en que se aplica esté método es calculando una fila de U y una columna de L. En este caso los datos obtenidos se escriben sobre una sola matriz cuadrada B de dimensiones dadas por el orden del sistema. Una forma de implementar la expresión 4.1 es la siguiente:

```
for(n=0;n<=ord-1;n++) ------------> (1)
{
  r=n; c=n; lim=n-1; i=c+1;
  for(j=n;j<=ord-1;j++) ------------> (2)
  {
       suma1=0.0;
       for(k=0;k<=lim;k++) ------------> (3)
       {
            suma1=suma1+B[r][k]*B[k][j];

       }
       B[r][j]=A[r][j]-suma1;
  }
```

```
 for(j=n;j<=ord-1;j++)-------------> (2')
{
      suma2=0.0;
      for(k=0;k<=lim;k++)-------------> (3')
      {
          suma2=suma2+B[i][k]*B[k][c];

      }
      B[i][c]=(A[i][c]-suma2)/A[c][c];
      i=i+1;
  }

}
```

En este código se han señalado los 5 ciclos **for** necesarios obtener la matriz LU.

- El ciclo **1** indica el número de fila y columna que se está calculando.
- Todos los elementos de la fila son calculados para poder pasar a la siguiente fila. El ciclo **2** indica que elemento de la fila se está calculando.
- La sumatoria indicada en la expresión 4.1 es calculada por el ciclo **3** y una vez obtenida se calcula: `B[r][j]=A[r][j]-suma1` para cada elemento de la fila que se está calculando de la matriz U.
- Después de que se calcula una fila de U el ciclo **2'** indica el elemento de la columna de L que se está calculando.
- La sumatoria indicada en la expresión 4.2 es calculada por el ciclo **3'** y una vez obtenida se calcula: `B[i][c]=(A[i][c]-suma2)/A[c][c]` para elemento de la columna de L que se está calculando.

Cuando los ciclos terminan quedan almacenados los valores de las matrices L y U en una matriz única que es la matriz B en este caso. Del código anterior se puede observar que los ciclos **2** y **2'** así como los ciclos **3** y **3'** tienen exactamente la misma estructura por lo que se pueden calcular las matrices L y U haciendo uso de 3 ciclos for. Una manera más práctica de implementar la expresión 1 es la siguiente que es la que se ha utilizado en el programa:

```
for(n=0;n<=ord-1;n++) -------------> (1)
{
  r=n; c=n; lim=n-1; i=c+1;
  for(j=n;j<=ord-1;j++) -------------> (2)
  {
      suma1=0.0; suma2=0.0;
      for(k=0;k<=lim;k++)-------------> (3)
      {
          suma1=suma1+B[r][k]*B[k][j];
          suma2=suma2+B[i][k]*B[k][c];
      }
      WxGrid2->SetCellTextColour(r,j,"red");
      B[r][j]=A[r][j]-suma1;
      WxGrid2->SetCellTextColour(i,c,"blue");
      if(i<=ord-1) B[i][c]=(A[i][c]-suma2)/B[c][c];
      i=i+1;
  }
}
```

En esta forma de implementación el ciclo 1 sigue indicando el número de fila y columna que se están calculando. El ciclo 2 indica que elemento se está calculando de la fila y columna en cuestión. En este código se calcula un elemento de la fila de U y un elemento de la columna de L hasta que se calculan todos los elementos para calcular la siguiente fila y columna. El ciclo 3 calcula las sumatorias indicadas en las expresiones 4.1 y 4.2 para que con su resultado se apliquen las expresiones 4.1 y 4.2 de la siguiente manera:

```
B[r][j]=A[r][j]-suma1;  ------------->expresión 1
if(i<=ord-1) B[i][c]=(A[i][c]-suma2)/B[c][c];------------->expresion 2
```

Considérese un sistema de ecuaciones de 5x5, los resultados se almacenan en el arreglo mostrado en la figura 4.7. Cuando n=0 para el primer ciclo for se calculan la primera fila de U y la primera columna de L de la siguiente manera:

- Cuando j=0 se calcula el elemento b00 y el elemento b10.
- Cuando j=1 se calcula el elemento b01 y b20.
- Cuando j=2 se calcula b02 y b30.
- Cuando j=3 se calcula b03 y b40.
- Cuando j=4 se calcula b04 y se pretendería almacenar algo en la posición b50 pero no existe en este caso debido a que el sistema es cuadrado y se haría referencia a un elemento fuera del arreglo que se ha declarado. Para evitar esto el cálculo de la expresión 2 está condicionado a que no se apunte a una posición fuera del arreglo como se indica en la línea de código en negritas.

b00	b01	b02	b03	b04
b10	b11	b12	b13	b14
b20	b21	b22	b23	b24
b30	b31	b32	b33	b34
b40	b41	b42	b43	b44

Figura 4.7 Matriz LU para un sistema de 5x5

4.3.3.4 Diseño del código del paso de la eliminación hacia adelante.

Es necesario obtener los términos independientes y almacenarlos en un arreglo que en este caso es **double** `b[ord1]`. Para implementar la expresión 4.3 simplemente se hace el primer elemento de c igual al primer término independiente. Para la expresión 4.4 es necesario utilizar dos ciclos for, el ciclo **1** indica que elemento de c está siendo calculado. El ciclo **2** calcula la sumatoria indicada en la expresión para que una vez calculada se aplique la expresión 4.3 que se muestra en negritas.

```
c1[0]=b[0];
for(i=1;i<=ord-1;i++)-------->(1)
{
    suma=0.0; lim=i-1;
    for(k=0;k<=lim;k++)-------->(2)
    {
        suma=suma+B[i][k]*c1[k];
    }
    c1[i]=b[i]-suma;
}
```

4.3.3.5 Diseño del código del paso de la eliminación hacia atrás.

Después de haber calculado los elementos del arreglo c estamos en condiciones para calcular los elementos del último arreglo declarado como **double** `x[ord1]`. El primer elemento del arreglo se calcula usando la expresión 4.5 y para calcular los demás elementos se hace uso de dos ciclos for. De igual manera el ciclo **1** indica el elemento que se está calculando y el ciclo **2** calcula la sumatoria indicada en la expresión 4.6. Después de calcular se calcula cada elemento con la expresión 4.6 de la manera que se indica en la línea de código en negritas.

```
index=ord-1;
x[index]=c1[index]/B[index][index];//Se calcula el primer elemento
lims=index-1;
for(i=lims;i>=0;i--)-------->(1)
{
    limi=i+1;
    suma=0.0;
    for(k=index;k>=limi;k--)-------->(2)
    {
        suma=suma+B[i][k]*x[k];
    }
    x[i]=(c1[i]-suma)/B[i][i];
}
```

Después de contar con los cálculos almacenados en los arreglos B ,c y x , los elementos de los arreglos son convertidos a cadenas de caracteres para ser mostrados en las tablas correspondientes. En el caso de la tabla para la matriz LU se establece color rojo para los elementos de U y azul para los elementos de L.

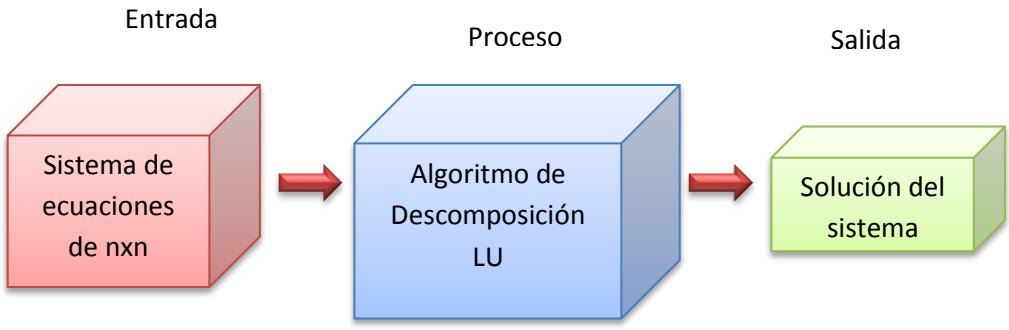
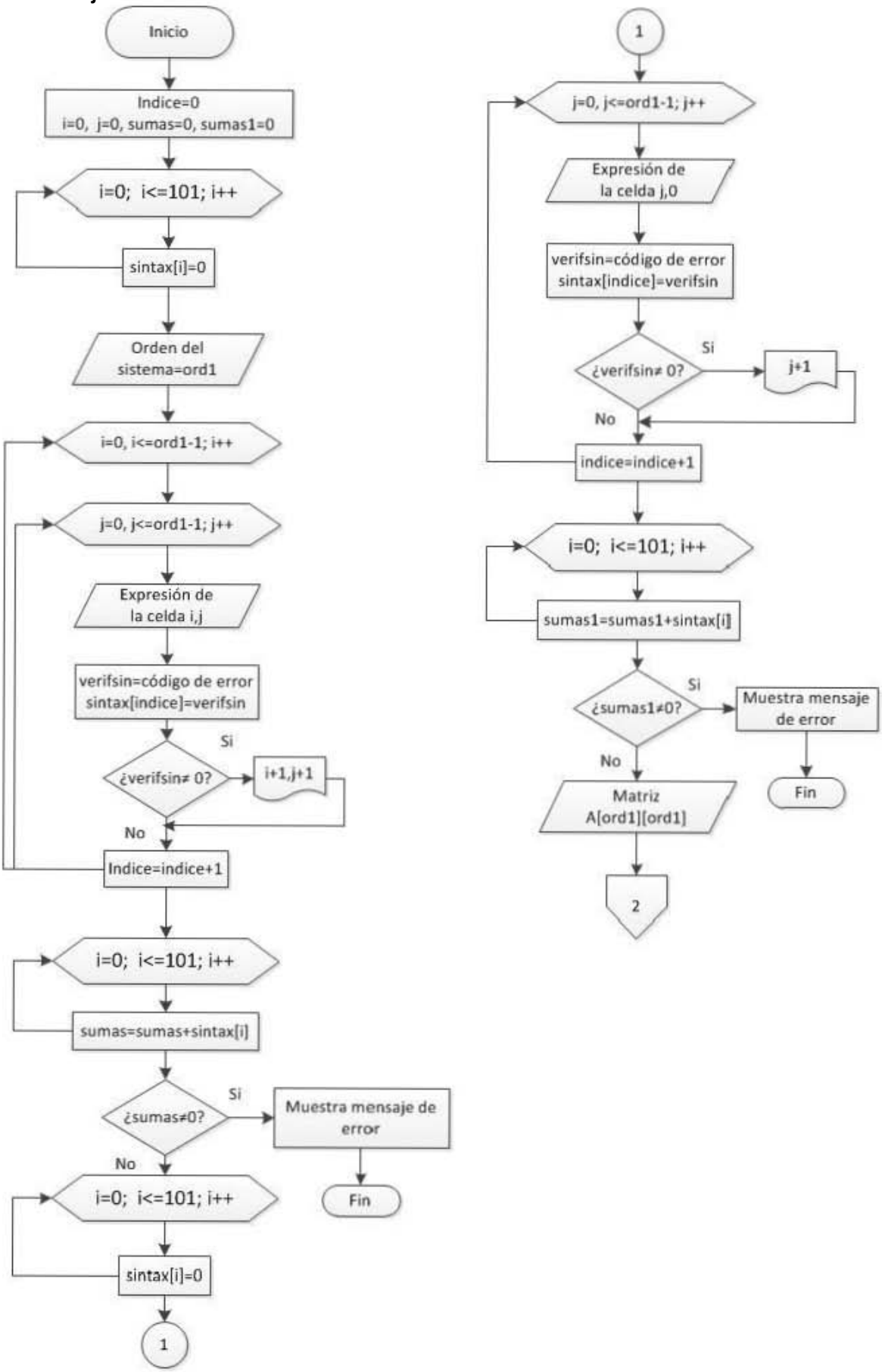
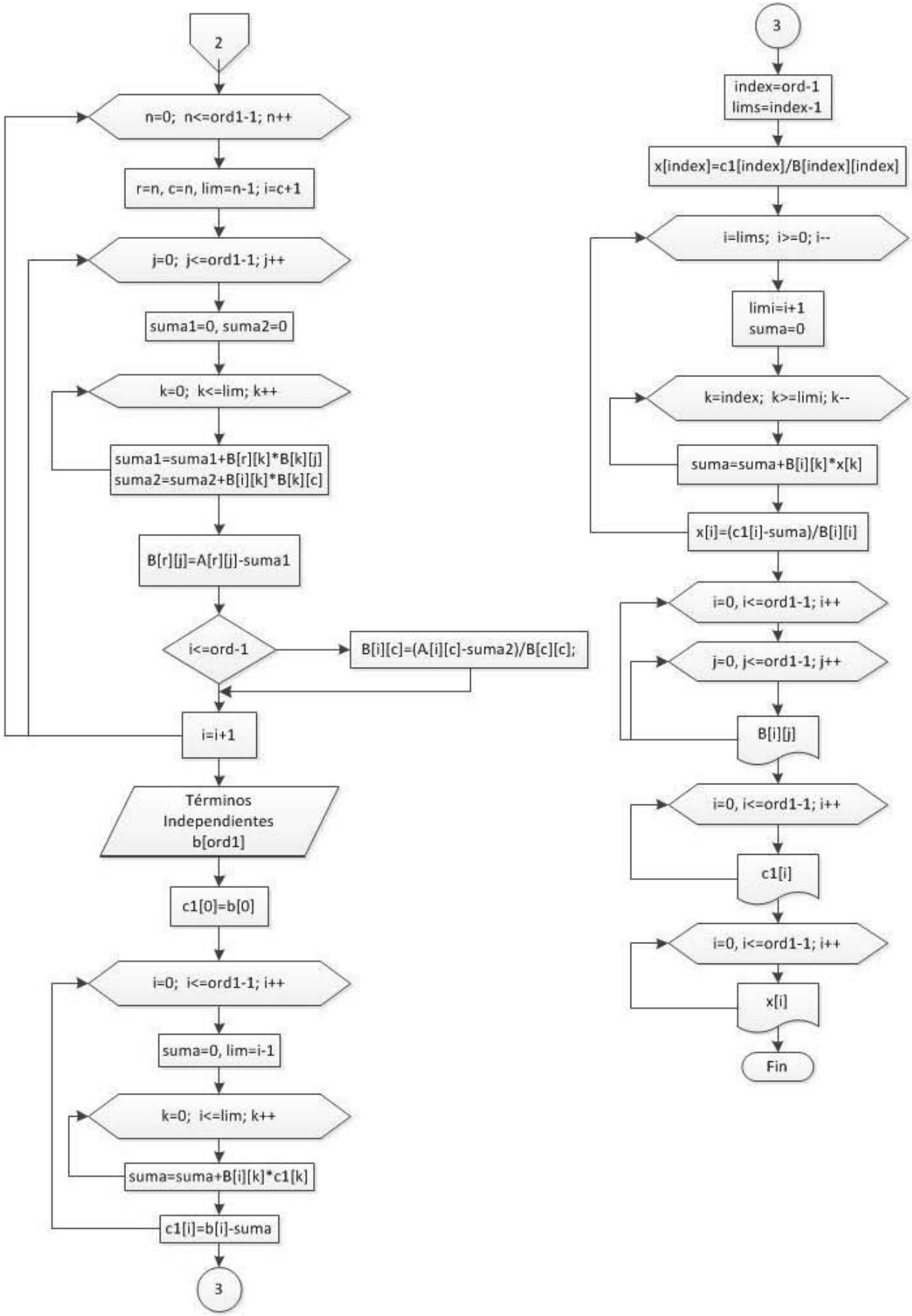


Figura 4.8 Diagrama de Entrada, Proceso Salida

4.4 Diagrama de flujo del botón Calcular




2
n=0; n<=ord1-1; n++
r=n, c=n, lim=n-1; i=c+1
j=0; j<=ord1-1; j++
suma1=0, suma2=0
k=0; k<=lim; k++
suma1=suma1+B[r][k]*B[k][j]
suma2=suma2+B[i][k]*B[k][c]
B[r][j]=A[r][j]-suma1
i<=ord-1
B[i][c]=(A[i][c]-suma2)/B[c][c];
i=i+1
Términos Independientes b[ord1]
c1[0]=b[0]
i=0; i<=ord1-1; i++
suma=0, lim=i-1
k=0; i<=lim; k++
suma=suma+B[i][k]*c1[k]
c1[i]=b[i]-suma
3
3
index=ord-1
lims=index-1
x[index]=c1[index]/B[index][index]
i=lims; i>=0; i--
limi=i+1
suma=0
k=index; k>=limi; k--
suma=suma+B[i][k]*x[k]
x[i]=(c1[i]-suma)/B[i][i]
i=0, i<=ord1-1; i++
j=0, j<=ord1-1; j++
B[i][j]
i=0, i<=ord1-1; i++
c1[i]
i=0, i<=ord1-1; i++
x[i]
Fin

5. INTERPOLACIÓN DE LAGRANGE

A menudo se necesitan estimar valores intermedios entre datos precisos. El método más común que se usa para este propósito es la interpolación del polinomio. Recuérdese que la fórmula general para un polinomio de *n-ésimo* orden es:

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots + a_nx^n$$

Para $n + 1$ puntos, hay un solo polinomio de orden n que pasa a través de todos los puntos. Por ejemplo, hay solo una línea recta (es decir, un polinomio de primer orden) que conecta dos puntos (véase la figura 5.1a). De manera similar, únicamente una parábola conecta un conjunto de tres puntos (ver figura 5.1b). *Interpolación polinomial* consiste en determinar el único polinomio de *n-ésimo* orden que ajuste n+1 puntos. Este polinomio entonces proporciona una fórmula para calcular valores intermedios. Aunque hay uno y solo un polinomio de *n-ésimo* orden que ajusta *n+1* puntos, existe una variedad de formatos matemáticos en los cuales este polinomio puede ser expresado. En esta tesis se explicaran dos alternativas; en este capítulo se describe el método de Lagrange y en el capítulo 6 el método de Newton en diferencias divididas.

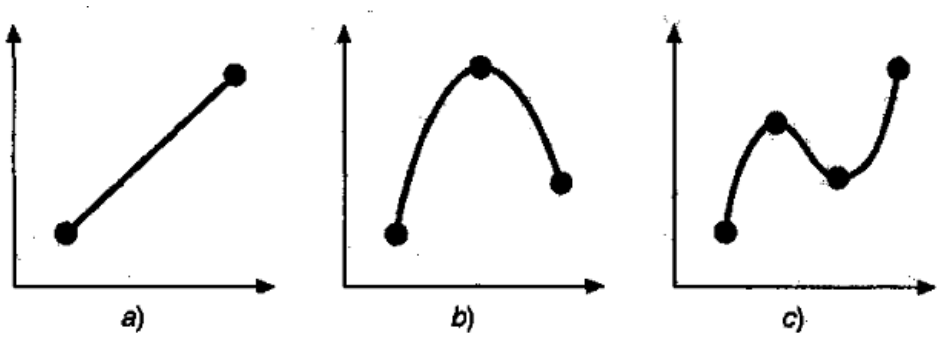


Figura 5.1 Ejemplos de interpolación polinomial. a) De primer orden (lineal) conectando dos puntos, b) de segundo orden (cuadrática o parabólica) enlazando

Este programa para interpolación permite insertar desde 2 hasta 10 puntos obteniendo como máximo un polinomio de orden 9 sin la necesidad de calcular diferencias divididas como en el caso de la interpolación de Newton. En la figura 5.1 se muestra el programa.

5.1 Consideraciones de diseño del programa

- Se debe contar con una lista desplegable que permita elegir el número de puntos a insertar desde 2 hasta 10 puntos.
- Se debe mostrar el polinomio obtenido en una caja de texto.
- En una caja de texto se insertará el valor a interpolar deseado por el usuario.
- Si son varios los valores de interés se contará con los elementos necesarios para realizar una tabulación.
- Se llevará a cabo la revisión de sintaxis de cada celda de la tabla y se indicarán las posiciones de error en una caja de texto.
- Con una lista desplegable se elegirá el número de dígitos a la izquierda del punto decimal, siendo 4 dígitos establecidos por defecto y como máximo 11 dígitos esto por dos razones:
    I. El usuario ha obtenido el polinomio manualmente y desea comprobar su resultado, por lo que no son necesarios demasiados dígitos a la derecha del punto.
    II. El usuario desea graficar el polinomio por lo que es necesario utilizar el máximo número de dígitos decimales de la lista para copiar el polinomio y colocarlo en un programa para graficar.
- Es muy importante la precisión de la interpolación por lo que se debe escribir el código de tal manera que se usen todos los decimales que permite el compilador.
- Se colocará un botón para limpiar automáticamente el valor a interpolar insertado anteriormente sin la necesidad de borrarlo manualmente.

Figura 5.2 Programa de Interpolación de Lagrange

5.2 Componentes usados para el programa.

- **Cajas de texto donde:**

1. Se muestra el polinomio obtenido: PolinomioCT.
2. Se inserta un valor a interpolar: XvalCT.
3. Se muestra el resultado de la interpolación: ResultCT.
4. Se inserta el parámetro Desde: DesdeCT.
5. Se inserta el parámetro Hasta: HastaCT.
6. Se inserta el parámetro incremento: IncCT.
7. Se muestran las posiciones de la tabla donde ha ocurrido error de sintaxis: SintaxisCT.

- **Botones.**

8. Más puntos: MasPuntosB.
9. Calcular: CalcB.
10. Nuevos Datos: NuevosB.
11. Salir: SalirB.
12. Evaluar: CalcularB.
13. Otro valor: OtroB.
14. Tabular polinomio: TabPolB.
15. Tabular: TabularB.

- **Tablas**

16. Se introducen los puntos para la interpolación: TablaResG.
17. Se muestra la tabulación: TablaG.

- **Listas desplegables**

18. Seleccionar el número de puntos: PuntosLD.
19. Seleccionar el número de dígitos a la derecha del punto decimal: DigitosLD.

- **Textos Estáticos:**

20. Tabla: TablaST.
21. Polinomio Obtenido: PolinomioST.
22. Evaluar para x=: EvaluarST.
23. Resultado: ResultST.
24. Dígitos Decimales: DigitosST.
25. Desde: DesdeST.
26. Hasta: HastaST.
27. Incremento: IncST.
28. Error en: ErrorST.

- **Diálogos de mensaje**

29. Inserte un valor para x: XvalDM.
30. Inserte el parámetro desde: DesdeDM.
31. Inserte el parámetro hasta: HastaDM.
32. Inserte el parámetro incremento: IncDM.
33. Error de sintaxis en los parámetros: SintaxisDM.

5.3 Funcionamiento del programa.

La lista desplegable PuntosLD es utilizada para cambiar el número de puntos a insertar funciona de la misma manera que la lista del programa descomposición LU, pero en este caso solo se insertan filas a la tabla y la lista es deshabilitada después de realizar la selección. De la misma forma que en el programa anterior el botón *Más puntos* se encarga de limpiar las tablas y habilitar la lista desplegable de nueva cuenta. Se ha implementado también el algoritmo de revisión de sintaxis explicado en el capítulo anterior para revisar la sintaxis de los valores insertados en la tabla. El botón *Tabular Polinomio* muestra los componentes necesarios para la tabulación y el botón Nuevos Datos limpia todo y oculta los componentes para tabular. Por lo tanto solo se explica en esta sección el funcionamiento de:

- El botón Calcular.
- La lista desplegable de selección de dígitos a la derecha del punto decimal.
- El botón Evaluar.
- El botón Tabular.

5.3.1 Funcionamiento del botón Calcular.

El primer paso a seguir es la revisión de sintaxis de los valores de los puntos insertados en la tabla. Se almacena en un arreglo el código de error obtenido de cada expresión insertada en la tabla y realiza la sumatoria de los elementos del arreglo, si o ha ocurrido error alguno la sumatoria es cero y comienza la obtención del polinomio de interpolación.

5.3.1.1 Expresión del método

La siguiente expresión representa el modelo de Lagrange:

$$p(x) = \sum_{i=0}^{n} \left[\prod_{j=0}^{n} \frac{(x - x_j)}{(x_i - x_j)} \right] y_i$$

En la que $\prod_{j=0}^{n}$ es la función producto PI que realiza los productos recorriendo *j* desde cero hasta *n,* siempre que *j* sea diferente de *i.*

5.3.1.2 Ejemplo de aplicación del método para la obtención del código

Para la tabla 5.1 obtener la aproximación polinomial de Lagrange con todos los puntos.

i	0	1	2	3
f(xi)	-3	0	5	7
xi	0	1	3	6

Tabla 5.1

Obsérvese que hay cuatro puntos en la tabla, por lo que el polinomio será de tercer grado. Al sustituir cuatro puntos en las ecuaciones generales se obtiene.

$$P_3(x) = (x-1)(x-3)(x-6)\frac{-3}{(0-1)(0-3)(0-6)} + (x-0)(x-3)(x-6)\frac{0}{(1-0)(1-3)(1-6)} + (x-0)(x-1)(x-6)\frac{5}{(3-0)(3-1)(3-6)} + (x-0)(x-1)(x-3)\frac{7}{(6-0)(6-1)(6-3)}$$

En este caso la multiplicación de polinomios se realizó en base a arreglos. Uno de los arreglos es de dos elementos en el que el segundo elemento es 1 que es el coeficiente de 'x' de cada término que se multiplica y el primer elemento del arreglo es el término independiente, así para el primer sumando los arreglos son los mostrados en la tabla 5.2:

x-1	[1 -1]
x-3	[1 -3]
x-6	[1 -6]

Tabla 5.2

La forma en que se procede es la mostrada en la figura 5.3 de manera recurrente hasta multiplicar en este caso los tres arreglos de la tabla 5.2 para el primer sumando del modelo de Lagrange.

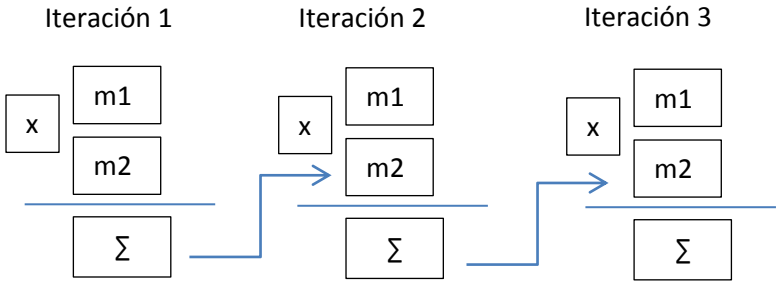


Figura 5.3 Uso de los arreglos para obtener un polinomio

- Se toma el primer arreglo de la tabla y es el multiplicando m1.
- El multiplicando m2 es un arreglo del tamaño del número de puntos que inicialmente toma los valores indicados en la tabla 5.3.

x^3	x^2	x^1	x^0	
0	0	0	1	m2

Tabla 5.3 Valores iniciales del multiplicando m2

- Se realiza la multiplicación [1 -1][0 0 0 1]:

x^3	x^2	x^1	x^0	
0	0	0	-1	r1
0	0	1	0	r2
0	0	1	-1	∑

Tabla 5.4 Primera multiplicación realizada

En el renglón r1 de la tabla 5.4 se observa el resultado de multiplicar el término independiente -1 cada elemento del arreglo m2. En el renglón r2 se muestra el resultado de multiplicar el coeficiente de 'x' por los elementos del arreglo m2 recorridos una posición a la izquierda porque aumenta en uno el grado de cada elemento del arreglo.

- El resultado de la suma se almacena en el arreglo que corresponde al multiplicando m2 para la siguiente multiplicación.

Se realiza la multiplicación (x-3)(x-1) que corresponde a [1 -3] [1 -1] en la que se observa que el resultado anterior se usa como el multiplicando m2. La lógica de este algoritmo es realizar una multiplicación y usar el resultado en la siguiente iteración como el multiplicando m2. De tal manera al comenzar $(x-1)$ que es m1 se multiplica por uno y el resultado se almacena en m2, es decir en la siguiente iteración será m2

x^3	x^2	x^1	x^0	
0	0	-1	3	r1
0	1	-3	0	r2
0	1	-4	3	∑

Tabla 5.5

En el renglón r1 de la tabla 5.5 se observa la multiplicación del multiplicando [1 -3] con el termino independiente -1 por lo que el resultado corresponde a los valores mostrados. En el renglón r2 se muestra el resultado multiplicando esta vez multiplicado por 'x', razón por la cual el resultado se desplaza una posición a la izquierda. Se necesita un arreglo bidimensional de dos filas y número de columnas según el número de puntos seleccionado para almacenar el resultado de ambas multiplicaciones. Una vez hecha la multiplicación se suma cada columna y se obtiene $x^2 - 4x + 3$ como se puede ver según los coeficientes. En el siguiente paso se multiplica [1 -6][1 -4 3] y se obtienen los resultados de la tabla 5.6.

x^3	x^2	x^1	x^0	
0	-6	24	-18	m1
1	-4	3	0	m2
1	-10	27	-18	∑

Tabla 5.6

La suma obtenida corresponde al resultado de multiplicar $(x-1)(x-3)(x-6)$ que es $x^3 - 10x^2 + 27x - 18$ según los coeficientes de la sumatoria de la tabla 5.6. Por lo tanto se necesitan dos ciclos **for** para obtener los dos renglones de la multiplicación. El **for** externo indica que elemento de uno de los arreglos de la tabla 5.2 se está multiplicando y el **for** interno multiplica dicho elemento por todos los elementos del segundo multiplicando como se muestra en la figura 5.4 para el tercer arreglo de la tabla 5.2.

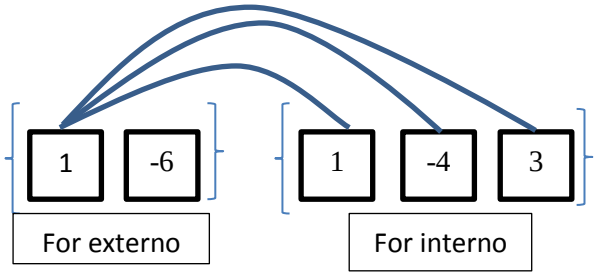


Figura 5.4 Ciclos necesarios para realizar la multiplicación término a término

Para realizar la sumatoria de cada renglón del arreglo que almacena los resultados de la multiplicación también se necesita de dos ciclos **for**. El ciclo externo indica la columna que se está sumando y el ciclo interno suma los dos elementos de cada columna, cuando la suma se ha realizado se almacena en un arreglo que funciona como el multiplicando m2 para la siguiente multiplicación que se realice ya que él primer multiplicando se toma de la tabla 5.2. Además es necesario un ciclo que vaya tomando cada arreglo de la tabla 5.2 para realizar la multiplicación de

polinomios, este ciclo es externo a los ciclos mencionados. Cada que se toma un arreglo antes de entrar a la multiplicación de los arreglos m1 y m2 se realiza la multiplicación del denominador término a término en cada iteración que es:
$(0-1)(0-3)(0-6)$ de la siguiente manera:

- Se declara e inicializa la variable `double coci=1.`
- Cuando se toma el primer arreglo se calcula `coci=1*(0-1)=-1`
- Cuando se toma el segundo arreglo se calcula `coci=-1(0-3)=3`
- Cuando se toma el tercer arreglo se calcula `coci=3(0-6)=-18`

Por lo tanto cuando se han multiplicado los arreglos de la tabla 5.2 queda almacenado en el arreglo m2 los coeficientes de la sumatoria de la tabla 5.6, es decir $m2 = [1 \quad -10 \quad 27 \quad -18]$ y tenemos $coci = -18$. Además se necesita un arreglo donde se almacenan los coeficientes finales del polinomio que se inicializa con ceros:

$$suma1 = [0 \quad 0 \quad 0 \quad 0]$$

Con estos datos se aplica el siguiente ciclo **for** para obtener los coeficientes del primer sumando:

```
for(c=0;c<=3;c++) suma1[c]=suma1[c]+(m2[c]*y[0])/coci;
```

- Cuando c=0 se calcula $suma1[0] = suma1[0] + \frac{m2[0]*(-3)}{-18} = 0 + \frac{(-18)(-3)}{-18} = -3$
- Cuando c=1 se calcula $suma1[1] = suma1[1] + \frac{m2[1]*(-3)}{-18} = 0 + \frac{(27)(-3)}{-18} = \frac{27}{6}$
- Cuando c=2 se calcula $suma1[1] = suma1[2] + \frac{m2[1]*(-3)}{-18} = 0 + \frac{(-10)(-3)}{-18} = \frac{-10}{6}$
- Cuando c=3 se calcula $suma1[1] = suma1[3] + \frac{m2[1]*(-3)}{-18} = 0 + \frac{(1)(-3)}{-18} = \frac{1}{6}$

Cuando el ciclo for termina de ejecutarse en el arreglo suma1 quedan almacenados los valores que corresponden a los coeficientes obtenidos del primer sumando:

$$suma1 = \left[\frac{1}{6} \quad \frac{-10}{6} \quad \frac{27}{6} \quad -3\right]$$

En la tabla 5.7 se observan los resultados obtenidos al sumar los coeficientes de los sumandos restantes.

x^3	x^2	x^1	x^0	iteración
0	0	0	0	0
1/6	-10/6	27/6	-3	1
1/6	-5/3	9/2	-3	2
-1/9	5/18	-17/6	3	3
-1/30	-1/30	46/15	-3	4

Tabla 5.7

Una vez que se han acabado de ejecutar los ciclos for en el arreglo sumas1 quedan almacenados los coeficientes finales del polinomio de interpolación que es:

$$p(x) = -\frac{1}{30}x^3 - \frac{1}{30}x^2 + \frac{46}{15}x - 3$$

5.3.1.3 Código para implementar el polinomio de interpolación.

En base a los pasos seguidos anteriormente se diseñó el código de la siguiente manera.

```
//Se inicializa el arreglo que almacena los coeficientes de cada sumando
for(k=0;k<=ord;k++) suma1[k]=0;
//Este ciclo indica que sumando se está calculando
for(k=0;k<=ord;k++)
{//for1
    //El coeficiente de x siempre es uno en cada término
    m1[1]=1.0;
    //Para cada sumando se reinicia el multiplicando m2
    for(c=0;c<=ord;c++) m2[c]=0;
    //Se inicializa el cociente y el primer elemento del multiplicando m2
    m2[0]=1.0; coci=1.0;
    //este ciclo toma cada uno de los términos que se van multiplicando
    for(n=0;n<=ord;n++)
    {//for2
        //si n es distinto de k toma el siguiente término
        if(n!=k)
        {//if1
            //se calcula paso a paso el valor del dividendo
            coci=coci*(x[k]-x[n]);
            //Se coloca el primer del elemento del arreglo para formar
            //cada uno de los términos (x-1)=[1 -1] por ejemplo
            m1[0]=-x[n];
            //el for externo toma los elementos del arreglo m1
            for(i=0;i<=1;i++)
            {
                //el for interno multiplica cada elemento del arreglo
                //por todos los elementos del arreglo m2
                for(j=0;j<=ord;j++)
                {
                    //j+1 se refiere a que cuando i=1 se multiplica por
                    //el coeficiente de x y debe aumentar en 1 el grado
                    //de cada término por lo que se recorren los elementos
                    //una posición a la izquierda
                    A[i][j+i]=m1[i]*m2[j];
                }
            }
            //El for externo recorre las columnas de la matriz A
            for(j=0;j<=ord;j++)
            {//for3
                //Se suma cada columna y cada iteración se resetea
                //el valor de suma
                suma=0;
                //se suman los dos elementos de cada columna
                for(i=0;i<=1;i++)
                {
                    suma=suma+A[i][j];
                }
                //cada suma se almacena en la misma posición de columna
                //indicada por j en el arreglo m2 para la siguiente
                //multiplicación
                m2[j]=suma;
            }//for3
        }//if1
    }//for2
    //Al final de los ciclos anteriores en el arreglo m2 quedan almacenados
    //los coeficientes de las multiplicaciones de los términos de cada
    //sumando y se aplica el siguiente ciclo para obtener los coeficientes
    //de cada sumando y sumarlos en cada iteración hasta obtener los
    //coeficientes finales del polinomio
    for(c=0;c<=ord;c++) suma1[c]=suma1[c]+(m2[c]*y[k])/coci;
}//for1
//Acaba la obtención de los coeficientes del polinomio
```

5.3.1.4 Presentación de los coeficientes obtenidos en forma de polinomio

Una vez aplicado el método para el ejemplo analizado quedan almacenados en el arreglo sumas1 los valores:

$$suma1 = \left[-\frac{1}{30} \quad -\frac{1}{30} \quad \frac{46}{15} \quad -3\right]$$

Para mostrar en forma de polinomio se utiliza un ciclo for que comienza en el último elemento del arreglo, es decir el coeficiente que corresponde al grado mayor y termina en él grado 1. En este caso el grado mayor es 3 y corresponde al coeficiente $-\frac{1}{30}$. El ciclo es el siguiente en el que ord = 3 para este ejemplo.

```
1) for(i=ord;i>=1;i--)
2) {
3)     {
4)         sprintf(a1,"%*.*f*x^",6,de1,suma1[i]);
5)         WxEdit1->WriteText(a1);
6)         sprintf(a1,"%d",i);
7)         WxEdit1->WriteText(a1);
8)         if(suma1[i-1]>=0) WxEdit1->WriteText("+");
9)     }
10)}
```

- En la primera iteración i = 3. En la línea 4 se convierte a cadena de caracteres el valor de suma[4] con el formato `"%*.*f*x^"` en el que el los asteriscos corresponden al números de dígitos a la izquierda y a la derecha del punto decimal, el número 6 corresponde a los dígitos a la derecha y el valor de la variables de1 corresponde a los dígitos a la izquierda. La variable de1 es obtenida de la lista desplegable 2 en la que el usuario selecciona el número de dígitos a la izquierda del punto que desea.
- La línea 5 escribe en la caya de texto $-\frac{1}{30} * x\text{^}$
- La línea 6 convierte a caracter el valor de i.
- La línea 7 escribe en la caja de texto el valor de i y queda: $-\frac{1}{30} * x\text{^}3$
- La línea 8 verifica el signo del siguiente valor, en caso de ser positivo escribe un signo “+” en la caja de texto, en caso de ser negativo no escribe nada debido a que al ser convertido a cadena de caracteres el siguiente coeficiente, incluye el signo negativo.
- Cuando i = 2 con lo que ya contenía la caja de texto queda: $-\frac{1}{30} * x\text{^}3 - \frac{1}{30} * x\text{^}2$
- Cuando i = 1 queda: $-\frac{1}{30} * x\text{^}3 - \frac{1}{30} * x\text{^}2 + \frac{46}{15} * x\text{^}1$

Sólo falta el término independiente que se escribe fuera del ciclo para no incluir x^0 y sólo escribir el término con su respectivo signo quedando así $-\frac{1}{30} * x\text{^}3 - \frac{1}{30} * x\text{^}2 + \frac{46}{15} * x\text{^}1 - 3$. Finalmente en la caja de texto se muestra en el formato: $-0.0333 * x\text{^}3 - 0.0333 * x\text{^}2 + 3.0667 * x\text{^}1 - 3.0000$

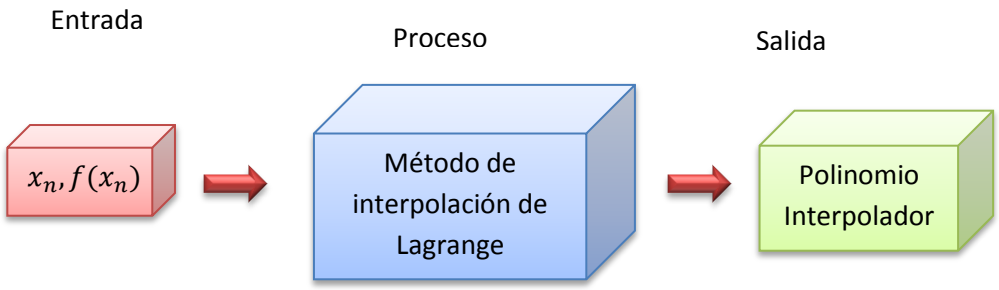


Figura 5.5 Diagrama de Entrada, Proceso, Salida del botón Calcular

5.3.2 Funcionamiento de la lista desplegable de selección de dígitos.

Inicialmente la lista desplegable de selección de dígitos a la izquierda del punto fue pensada solamente para realizar la selección y después pulsar el botón calcular nuevamente, de tal manera que como se explicó en el punto anterior la selección quedara almacenada en la variable de1. En la versión final de este programa se implementó exactamente el mismo código del botón Calcular para la lista desplegable de tal manera que con solo seleccionar un nuevo número de dígitos, el polinomio vuelve a ser calculado automáticamente sin la necesidad de presionar de nueva cuenta el botón calcular.

Cabe mencionar un aspecto importante de las listas desplegables en este entorno de desarrollo integrado. Si se pretende obtener el valor de la lista cuando el usuario no ha seleccionado ninguno el valor que devuelve es -1. En este programa la lista desplegable esta deshabilitada por defecto y solo se habilita cuando el polinomio es calculado por primera vez. Por lo tanto mientras el usuario no haya seleccionado un número de dígitos específico se fuerza a que la variable de1 = 4 por defecto ya que sería un error intentar mostrar -1 dígitos a la izquierda del punto.

5.3.3 Funcionamiento del botón Evaluar.

En los programas anteriores el botón Evaluar tomaba la expresión insertada por el usuario y mediante el evaluador de expresiones algebraicas la evaluaba para un valor particular. Para este programa en un principio se utilizó el evaluador de expresiones para interpretar la el polinomio obtenido y de esa manera se pudiera interpolar para un valor de interés para el usuario, sin embargo como resultado las interpolaciones no eran precisas. Es decir, el polinomio debe pasar por todos los puntos dados, por ejemplo para $x = 2, y = 3.4$ en polinomios de grado nueve fue obtenido un valor aproximadamente de 3.28 que claramente no corresponde al valor de $f(x)$ para el valor de x dado. Por lo tanto en el código de este botón se implementa la parte del código del botón Calcular hasta donde se obtienen los coeficientes del polinomio de interpolación y de esa manera no se pierdan decimales y así se obtengan interpolaciones más precisas.

De la misma manera que en los programas anteriores se verifica que sea insertado un valor a interpolar y que dicho valor no tenga errores de sintaxis. Una vez confirmado lo anterior, el valor se almacena en la variable **`double`** `m` y mediante un ciclo for se sustituye en el polinomio de la siguiente manera:

```
1)          for(i=1;i<=ord;i++)
2)          {
3)              suma2[i]=suma1[i]*pow(m,i);
4)          }
5)          suma=0;
6)          suma2[0]=suma1[0];
7)          for(i=0;i<=ord;i++)
8)          {
9)              suma=suma+suma2[i];
10)         }
```

En la línea 1 se observa que el ciclo comienza en el elemento 1 de arreglo que corresponde al coeficiente de x^1 y termina en el coeficiente del exponente más grande. En este caso interpolemos para x = 3, valor que se almacena en este caso en la variable m, es decir m = 3:

$$suma1 = \left[-\frac{1}{30} \quad -\frac{1}{30} \quad \frac{46}{15} \quad -3\right]$$

- Cuando i = 1 $\text{suma2}[1] = \text{suma1}[1] * \text{pow}(3,1) = \frac{46}{15}(3^1) = \frac{46}{5}$
- Cuando i = 2 $\text{suma2}[2] = \text{suma1}[2] * \text{pow}(3,2) = -\frac{1}{30}(3^2) = -\frac{3}{10}$
- Cuando i = 3 $\text{suma2}[3] = \text{suma1}[3] * \text{pow}(3,1) = -\frac{1}{30}(3^3) = -\frac{9}{10}$

El valor del término independiente permanece igual por lo que $\text{suma2}[0] = \text{suma1}[0] = -3$

Posteriormente se aplica un ciclo for para obtener la sumatoria del arreglo suma2 que es el valor de la interpolación.

$$p(3) = \frac{46}{5} - \frac{3}{10} - \frac{9}{10} - 3 = 5$$

Lo cual es correcto según la tabla de puntos dada.

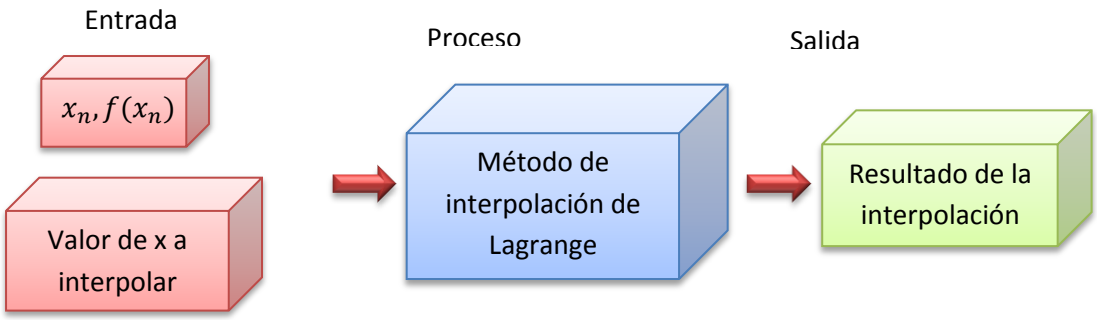


Figura 5.6 Diagrama de Entrada, Proceso, Salida del botón Evaluar

5.3.4 Funcionamiento del botón Tabular

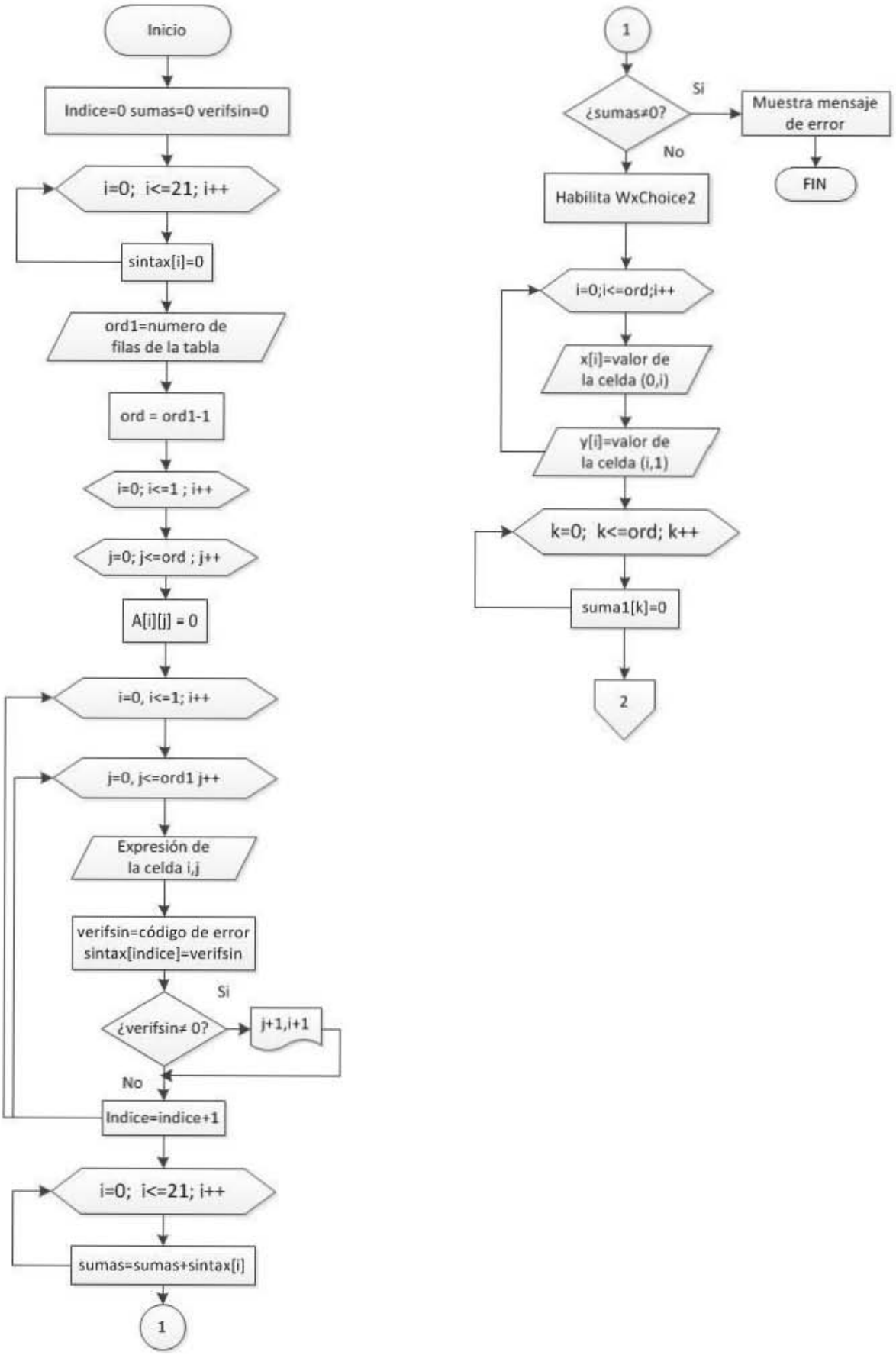
Este botón funciona de manera semejante al botón tabular del Analizador de raíces, la diferencia radica en que esta vez no se utiliza el evaluador de expresiones para sustituir los valores en la función por la razón mencionada en el algoritmo del botón evaluar. Además antes de comenzar con la tabulación se aplica el método nuevamente para obtener el arreglo sumas1 con los coeficientes del polinomio de interpolación. Después se aplica el mismo código para tabular presentado en el capítulo 1 con la siguiente diferencia:

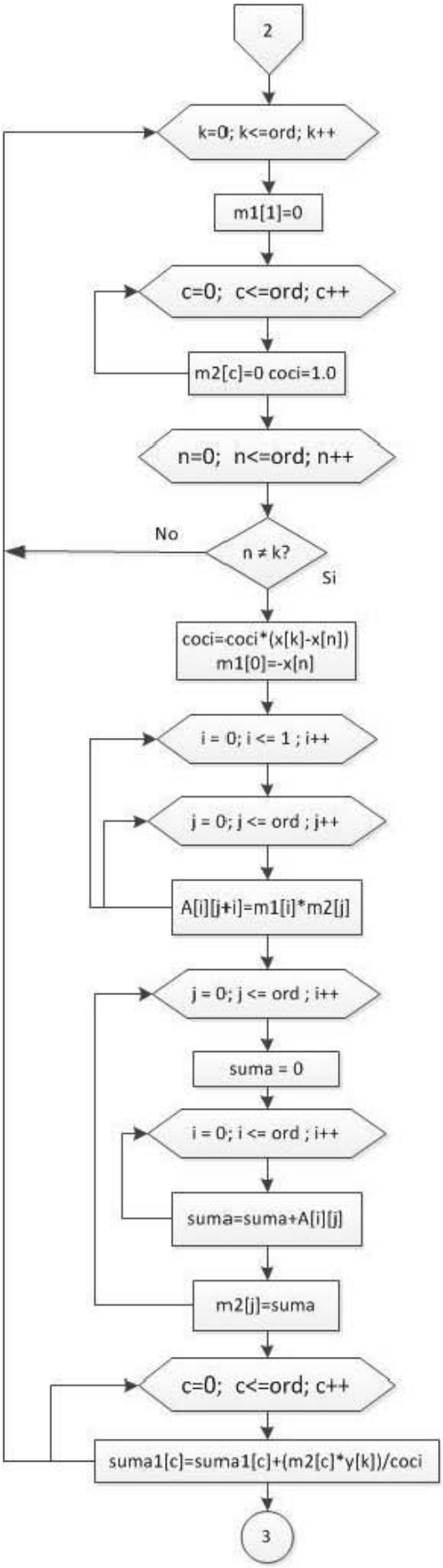
```
for(i=1;i<=ord;i++)
{
    suma2[i]=suma1[i]*pow(i1,i);
}
suma=0;
suma2[0]=suma1[0];
for(i=0;i<=ord;i++)
{
   suma=suma+suma2[i];
}

evaluadorExpresiones.ValorVariable('x', i);
valor = evaluadorExpresiones.Calcular();
```

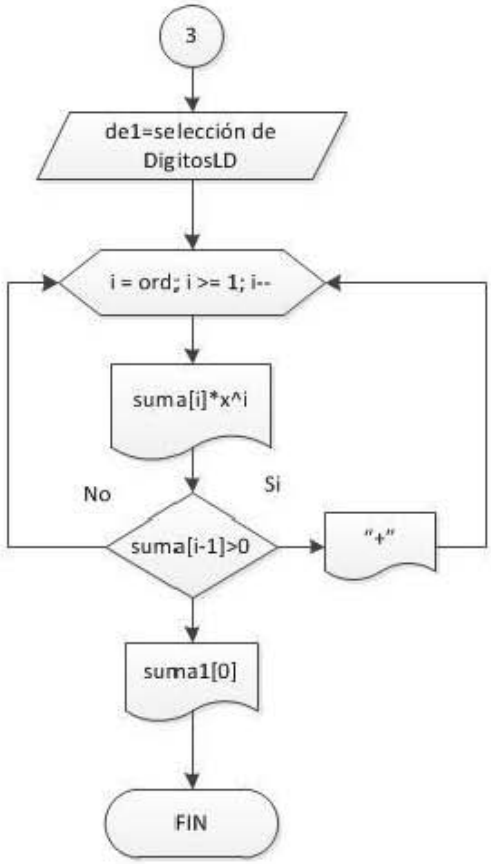
Esta sección de código utiliza la misma forma explicada en el botón Calcular para sustituir los valores que toma la variable independiente, el resultado se almacena en la variable suma que posteriormente se convierte a cadena de caracteres y se muestra en la columna de la tabla que corresponde a f(x). La sección de código reemplazada es la que se ha resaltado en negritas, en la que se le daba el valor a la variable independiente y se calculaba el resultado.

5.4 Diagramas de flujo

Se muestra el diagrama de flujo del botón calcular únicamente debido a que el comportamiento de los componentes restantes se ha mostrado en diagramas de flujo en los capítulos anteriores.


Inicio
Indice=0 sumas=0 verifsin=0
i=0; i<=21; i++
sintax[i]=0
ord1=numero de filas de la tabla
ord = ord1-1
i=0; i<=1 ; i++
j=0; j<=ord ; j++
A[i][j] = 0
i=0, i<=1; i++
j=0, j<=ord1 j++
Expresión de la celda i,j
verifsin=código de error sintax[indice]=verifsin
Si
¿verifsin≠ 0?
j+1,i+1
No
Indice=indice+1
i=0; i<=21; i++
sumas=sumas+sintax[i]
1
1
¿sumas≠0?
Si
Muestra mensaje de error
FIN
No
Habilita WxChoice2
i=0;i<=ord;i++
x[i]=valor de la celda (0,i)
y[i]=valor de la celda (i,1)
k=0; k<=ord; k++
suma1[k]=0
2


2
k=0; k<=ord; k++
m1[1]=0
c=0; c<=ord; c++
m2[c]=0 coci=1.0
n=0; n<=ord; n++
No
n ≠ k?
Si
coci=coci*(x[k]-x[n])
m1[0]=-x[n]
i = 0; i <= 1 ; i++
j = 0; j <= ord ; j++
A[i][j+i]=m1[i]*m2[j]
j = 0; j <= ord ; i++
suma = 0
i = 0; i <= ord ; i++
suma=suma+A[i][j]
m2[j]=suma
c=0; c<=ord; c++
suma1[c]=suma1[c]+(m2[c]*y[k])/coci
3


3
de1=selección de DigitosLD
i = ord; i >= 1; i--
suma[i]*x^i
No
Si
suma[i-1]>0
"+"
suma1[0]
FIN

6 INTERPOLACIÓN DE NEWTON EN DIFERENCIAS DIVIDIDAS

Este programa permite obtener el polinomio de interpolación mediante el método de Newton en diferencias divididas. A diferencia del programa de Interpolación de Lagrange, en este programa se ha añadido la posibilidad de adaptar el polinomio a todos los puntos insertados o a diferente cantidad de puntos para observar el comportamiento de los polinomios de interpolación. El programa se muestra en la figura 6.1.

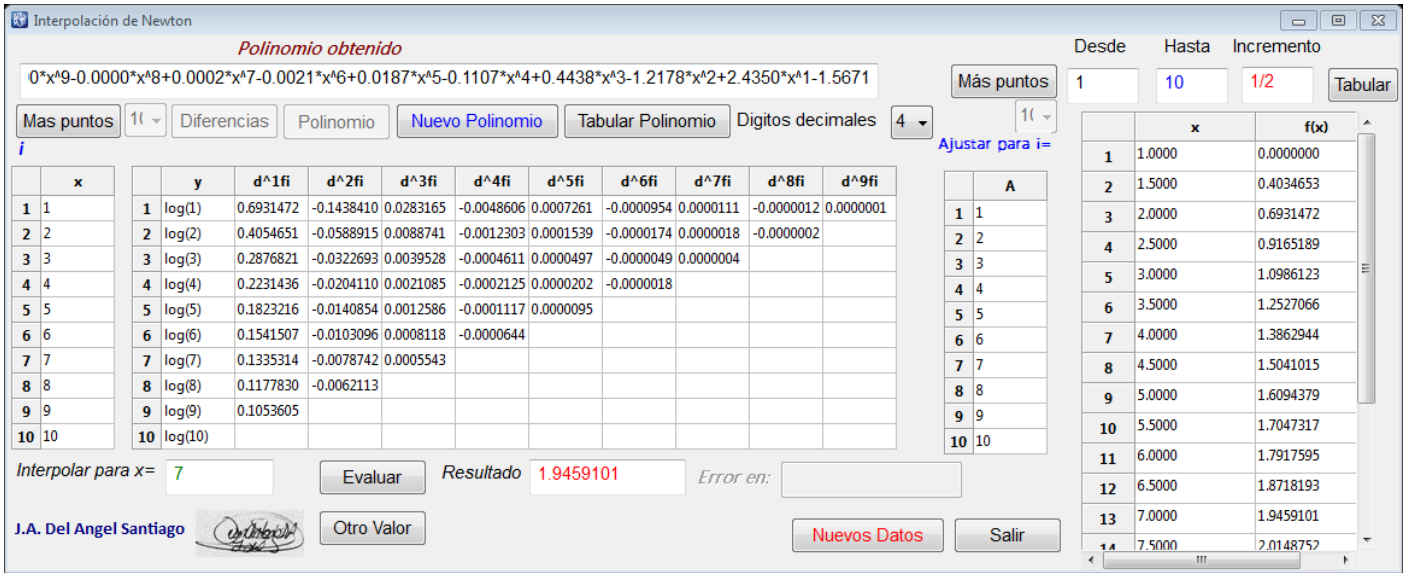


Figura 6.1 Programa de Interpolación de Newton

6.1 Consideraciones de diseño del programa

- Se debe contar con una lista desplegable que permita elegir el número de puntos a insertar desde 2 hasta 10 puntos.
- Se debe mostrar el polinomio obtenido en una caja de texto.
- En una caja de texto se insertará el valor a interpolar deseado por el usuario.
- Si son varios los valores de interés se contará con los elementos necesarios para realizar una tabulación.
- Se llevará a cabo la revisión de sintaxis de cada celda donde se inserten datos y se indicarán las posiciones de error en una caja de texto.
- Con una lista desplegable se elegirá el número de dígitos a la izquierda del punto decimal, siendo 4 dígitos establecidos por defecto y como máximo 11 dígitos.
- Es muy importante la precisión de la interpolación por lo que se debe escribir el código de tal manera que se usen todos los decimales que permite el compilador.
- Se debe contar con una lista desplegable que permita elegir el número de puntos a los que se va a ajustar el polinomio.
- Se necesita una tabla del mismo número de filas y columnas para insertar en la primera columna los valores de ‘y’ proporcionados por el usuario y en el resto de la tabla se muestren las diferencias divididas.

6.2 Componentes usados en el programa

- **Cajas de texto donde:**

1. Se muestra el polinomio obtenido: PolinomioCT.
2. Se inserta el valor de x a interpolar: XvalCT.
3. Se muestra el resultado de la interpolación: ResultCT.
4. Se inserta el parámetro desde: DesdeCT.
5. Se inserta el parámetro hasta: HastaCT.

6. Se inserta el parámetro incremento: IncCT.
7. Se muestran las posiciones de error: SintaxisCT.

- **Botones:**

8. Más puntos (insertar): MasPuntosB.
9. Diferencias: DiferenciasB.
10. Nuevos Datos: NewDataB.
11. Salir: SalirB.
12. Más puntos (ajuste): AjusteB.
13. Polinomio: PolinomioB.
14. Evaluar: EvaluarB.
15. Otro Valor: OtroValB.
16. Nuevo Polinomio: NpolinomioB.
17. Tabular Polinomio. TabPolB.
18. Tabular: TabularB.

- **Tablas:**

19. Tabla de diferencias divididas: DiferenciasG.
20. Tabla de puntos de ajuste: XvalG.
21. Valores de x: AjusteG.
22. Tabulación: TablaG.

- **Listas de selección:**

23. Elección de puntos a insertar: PuntosLD.
24. Elección de puntos de ajuste: AjusteLD.
25. Elección de número de dígitos decimales: DigitosLD.

- **Textos Estáticos:**

26. Interpolar para x= : InterpolarST.
27. Polinomio obtenido: PolinomioST.
28. Ajustar para i= : AjustarST.
29. Resultado: ResultST.
30. Desde: DesdeST.
31. Hasta: HastaST.
32. Incremento: IncST.
33. i : Ist.
34. Dígitos Decimales: DigitosST.
35. Error en: SintaxisST.

- **Diálogos de Mensaje:**

36. Opción invalida para la tabla actual: OpcionInvalidaDM.
37. Complete la tabla de puntos para el ajuste: CompleteTablaDM.
38. Inserte un valor a evaluar: XvalDM.
39. Inserte el parámetro Desde: DesdeDM.
40. Inserte el parámetro Hasta: HastaDM.
41. Inserte el parámetro Incremento: IncDM.
42. Los valores deben ser consecutivos: ConsecutvosDM.
43. Verifique los valores: VerifiqueDM.
44. Error de sintaxis en los valores de y: SintaxisYDM.
45. Error de sintaxis en los valores de x: SintaxisXDM.
46. Error de sintaxis en los parámetros: SintaxisDM.

6.3 Funcionamiento del programa

Conforme se ha ido avanzando en los capítulos se han explicado algoritmos como la revisión de sintaxis ya sea de cajas de texto o de valores insertados en tablas debido a que siempre es necesario revisar la sintaxis de los datos introducidos por el usuario de tal manera que los algoritmos que se han usado anteriormente y se implementan en este programa son:

1. La lista de selección 1 funciona de la misma manera que la lista utilizada en el programa de Interpolación de Lagrange pero insertando filas en la tabla de los valores de x e insertando el mismo número de filas y columnas en la tabla donde se insertan los valores de f(x) y se muestran las diferencias divididas calculadas.
2. Los botones **Más puntos** de las listas de selección se ocultan después de ser presionados para habilitar las listas de selección. De la misma manera después de realizar una selección se deshabilitan las listas desplegables y habilitan los botones **Más puntos** por la razón explicada en la sección 4.3.1.
3. Este programa también almacena los coeficientes finales del polinomio en el arreglo `suma1` por lo que para el botón Evaluar se utiliza la parte de código del botón **Polinomio** hasta donde son obtenidos todos los elementos del arreglo suma1. Una vez obtenidos se aplica el algoritmo explicado en la sección 5.4.3 del capítulo anterior para realizar la interpolación del valor de interés para el usuario.
4. Para el funcionamiento del botón Tabular véase la sección 5.4.4 del capítulo anterior.
5. El botón **Otro Valor** se encarga de borrar el valor a interpolar insertado anteriormente, limpia y deshabilita la caja de texto que muestra el resultado y se deshabilita a sí mismo para ser habilitado cuando la próxima vez que el usuario inserte un valor y pulse **Evaluar.**
6. Una vez que se ha pulsado el botón **Diferencias** el mismo botón se deshabilitará debido a que se llena la tabla con las diferencias dividas. Si se desea calcular diferencias nuevamente para otros datos se debe pulsar el botón **Nuevos Datos** que se encarga de ocultar los componentes para tabular y limpia todos los componentes de la ventana.
7. El botón **Nuevo Polinomio** sirve para limpiar los componentes de interpolación para un valor particular, borra el polinomio anterior y oculta los componentes para tabular además de deshabilitarse a sí mismo y habilitar el botón **Polinomio**. Esto con el fin de preparar todo en caso de que el usuario desee calcular un polinomio ajustado a otros puntos de la tabla.

Por lo tanto en esta sección se explicarán a fondo los algoritmos de los botones **Diferencias** y **Polinomio.**

6.3.1 Funcionamiento del botón Diferencias

Se explica en esta sección la forma en que se aplica la expresión de Newton para obtener la tablas de diferencias divididas con la que posteriormente es obtenido el polinomio de interpolación.

6.3.1.1 Expresiones del método.

Se puede ajustar un polinomio de *n*-ésimo grado a *n+1* datos. El polinomio de *n*-ésimo grado es:

$$f_n(x) = b_0 + b_1(x - x_0) + \cdots + b_n(x - x_0) \dots (x - x_{n-1}) - - - (6.1)$$

Para un polinomio de *n*-ésimo grado se requieren *n+1* puntos: $[x_0, f(x_0)], [x_1, f(x_1)], [x_0, f(x_0)], \dots, [x_n, f(x_n)]$.

Usamos estos datos y las siguientes ecuaciones para evaluar los coeficientes:

$$b_0 = f(x_0) - - - (6.2)$$
$$b_1 = f[x_1, x_0] - - - (6.3)$$
$$b_2 = f[x_2, x_1, x_0] - - - (6.4)$$
$$\vdots$$
$$b_n = f[x_n, x_{n-1}, \dots, x_1, x_0] - - - (6.5)$$

Donde las evaluaciones de la función colocadas entre paréntesis son diferencias divididas finitas. Por ejemplo, *la primera diferencia dividida finita* en forma general se representa como:

$$f[x_i, x_j] = \frac{f(x_i - x_j)}{x_i - x_j} - - - (6.6)$$

La *segunda diferencia dividida finita,* que representa la diferencia de las dos primeras diferencias divididas, se expresa en forma general como:

$$f[x_i, x_j, x_k] = \frac{f[x_i, x_j] - f[x_j, x_k]}{x_i - x_k} - - - (6.7)$$

En forma similar, la *n-ésima diferencia dividida finita* es:

$$f[x_n, x_{n-1}, \dots, x_1, x_0] = \frac{f[x_n, x_{n-1}, \dots, x_1] - f[x_{n-1}, x_{n-2}, \dots, x_0]}{x_n - x_0} - - - (6.8)$$

Estas diferencias sirven para calcular los coeficientes en las ecuaciones 2 a 5, los cuales se sustituirán en la ecuación 1 para obtener el polinomio de interpolación.

$$f_n(x) = f(x_0) + (x - x_0)f[x_1, x_0] + (x - x_0)(x - x_1)f[x_2, x_1, x_0] + \cdots + (x - x_0)(x - x_1)\dots(x - x_{n-1})f[x_n, x_{n-1}, x_0] - - - (6.9)$$

que se conoce como *polinomio de interpolación de Newton en diferencias divididas.*

Debe observarse que no se requiere que los datos utilizados en la ecuación (6.9) estén igualmente espaciados o que los valores de la abscisa estén en orden ascendente.

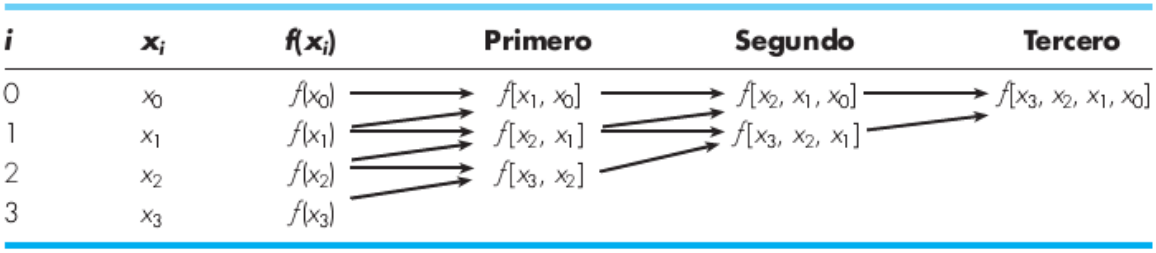


Figura 6.2 Representación gráfica de la naturaleza recursiva de las diferencias divididas finitas.

6.3.1.2 Ejemplo de aplicación del método para la explicación del código.

Se utilizará el mismo ejemplo de la sección 5.3.1.2 que consiste en obtener la aproximación polinomial con todos los puntos agregando la pareja de puntos x=9, f(x)=8.5 a la tabla 6.1

i	0	1	2	3	4
f(xi)	-3	0	5	7	8.5
xi	0	1	3	6	9

Tabla 6.1

En este caso de acuerdo a la forma en que se calculan las diferencias divididas se generó el código:

```
for(j=0;j<=ord-1;j++)
{
  lim=ord-j-1;
  for(i=0;i<=lim;i++)
  {

      C[i][j+1]=(C[i+1][j]-C[i][j])/(x[i+j+1]-x[i]);
      sprintf(a,"%*.*f",3,7,C[i][j+1]);
      WxGrid1->SetCellValue(i,j+1,a);
  }
 }
```

En el que el ciclo externo indica que diferencias están siendo calculadas, es decir, que columna de la tabla se está obteniendo. El ciclo interno calcula cada valor de la columna indicada. Una vez que se han calculado las diferencias se obtiene la tabla 6.2 de diferencias divididas. Los valores de x son almacenados en un arreglo x[4] y los valores de f(x) son almacenados en la primera columna del arreglo C[4][4], el resto del arreglo es usado para almacenar las diferencias divididas.

	columna	0	1	2	3	4
renglón	x	f(x)	D1	D2	D3	D4
0	0	-3	3	-0.16666667	-0.03333333	0.00841049
1	1	0	2.5	-0.36666667	0.04236111	
2	3	5	0.66666667	-0.02777778		
3	6	7	0.5			
4	9	8.5				

Tabla 6.2 Diferencias divididas para los puntos dados

Veamos cómo funciona el código. La variable `ord1` almacena el número de puntos de la tabla, en este caso `ord1=5` y la variable `ord=ord1-1=5-1=4` se encarga de controlar el ciclo externo.
En la primera iteración $j = 0,\ lim = ord - j - 1 = 4 - 0 - 1 = 3$. La variable `lim` indica en el ciclo for interno la variable i realizará cuatro iteraciones ya que comienza en cero. En la siguiente iteración tendremos $lim = 2$ y asi sucesivamente, esto es lo que le da el aspecto escalonado a la tabla de diferencias. Resumiendo, para la primera iteración de j tenemos: $j = 0 \quad lim = 3$ por lo que a continuación se calculan las primeras diferencias.

$$Cuando\ i = 0 \quad f[x_1, x_0] = C[0][0+1] = \frac{C[0+1][0] - C[0][0]}{x[0+0+1] - x[0]}$$

$$f[x_1, x_0] = C[0][1] = \frac{C[1][0] - C[0][0]}{x[1] - x[0]} = \frac{0 - (-3)}{1 - 0} = \mathbf{3}$$

$$Cuando\ i = 1 \quad f[x_2, x_1] = C[1][0+1] = \frac{C[1+1][0] - C[1][0]}{x[1+0+1] - x[1]}$$

$$f[x_2, x_1] = C[1][1] = \frac{C[2][0] - C[1][0]}{x[2] - x[1]} = \frac{5 - 0}{3 - 1} = \mathbf{2.5}$$

$$Cuando\ i = 2 \quad f[x_3, x_2] = C[2][0+1] = \frac{C[2+1][0] - C[2][0]}{x[2+0+1] - x[2]}$$

$$f[x_3, x_2] = C[2][1] = \frac{C[3][0] - C[2][0]}{x[3] - x[2]} = \frac{7 - 5}{6 - 3} = \frac{\mathbf{2}}{\mathbf{3}} \approx \mathbf{0.6667}$$

$$Cuando\ i = 3 \quad f[x_4, x_3] = C[3][0+1] = \frac{C[3+1][0] - C[3][0]}{x[3+0+1] - x[3]}$$

$$f[x_4, x_3] = C[3][1] = \frac{C[4][0] - C[3][0]}{x[4] - x[3]} = \frac{8.5 - 7}{9 - 6} = \frac{\mathbf{1.5}}{\mathbf{3}} = \mathbf{0.5}$$

Los valores obtenidos hasta este punto corresponden a la columna de primeras diferencias D1 de la tabla 6.2. Una vez que se han calculado los 4 valores indicados por `lim` la variable j se incrementa en uno por lo que para la siguiente iteración de j tenemos: $j = 1 \quad lim = ord - j - 1 = 4 - 1 - 1 = 2.$

$$Cuando\ i = 0 \quad f[x_2, x_1, x_0] = C[0][1+1] = \frac{C[0+1][1] - C[0][1]}{x[0+1+1] - x[0]}$$

$$f[x_2, x_1, x_0] = C[0][2] = \frac{C[1][1] - C[0][1]}{x[2] - x[0]} = \frac{2.5 - 3}{3 - 0} = \frac{-\mathbf{1}}{\mathbf{2}} = -\frac{\mathbf{1}}{\mathbf{6}} \approx \mathbf{-0.1667}$$

$$Cuando\ i = 1 \quad f[x_3, x_2, x_1] = C[1][1+1] = \frac{C[1+1][1] - C[1][1]}{x[1+1+1] - x[1]}$$

$$f[x_3, x_2, x_1] = C[1][2] = \frac{C[2][1] - C[1][1]}{x[3] - x[1]} = \frac{\frac{2}{3} - 2.5}{6 - 1} = \frac{-\frac{\mathbf{11}}{\mathbf{6}}}{\mathbf{5}} = -\frac{\mathbf{11}}{\mathbf{30}} \approx \mathbf{-0.3667}$$

$$Cuando\ i = 2 \quad f[x_4, x_3, x_2] = C[2][1+1] = \frac{C[2+1][1] - C[2][1]}{x[2+1+1] - x[2]}$$

$$f[x_4, x_3, x_2] = C[2][2] = \frac{C[3][1] - C[2][1]}{x[4] - x[2]} = \frac{0.5 - \frac{2}{3}}{9 - 6} = -\frac{\frac{\mathbf{1}}{\mathbf{6}}}{\mathbf{3}} = -\frac{\mathbf{1}}{\mathbf{36}} \approx \mathbf{-0.02778}$$

Los valores obtenidos hasta este punto corresponden a la columna de segundas diferencias D2 de la tabla 6.2. Una vez que se han calculado los 3 valores indicados por `lim` la variable j se incrementa en uno por lo que para la siguiente iteración de j tenemos: $j = 2 \qquad lim = ord - j - 1 = 4 - 2 - 1 = 1.$

$$Cuando\ i = 0 \quad f[x_3, x_2, x_1, x_0] = C[0][3] = \frac{C[0+1][2] - C[0][2]}{x[0+2+1] - x[0]}$$

$$f[x_3, x_2, x_1, x_0] = C[0][3] = \frac{C[1][2] - C[0][2]}{x[3] - x[0]} = \frac{-\frac{11}{30} - \left(-\frac{1}{6}\right)}{6 - 0} = \frac{-\frac{\mathbf{1}}{\mathbf{5}}}{\mathbf{6}} = -\frac{\mathbf{1}}{\mathbf{30}} \approx \mathbf{-0.0333}$$

$$Cuando\ i = 1 \quad f[x_4, x_3, x_2, x_1] = C[1][3] = \frac{C[1+1][2] - C[1][2]}{x[1+2+1] - x[1]}$$

$$f[x_4, x_3, x_2, x_1] = C[1][3] = \frac{C[2][2] - C[1][2]}{x[4] - x[1]} = \frac{-\frac{1}{36} - \left(-\frac{11}{30}\right)}{9 - 1} = \frac{\frac{\mathbf{61}}{\mathbf{180}}}{\mathbf{8}} = \frac{\mathbf{61}}{\mathbf{1440}} \approx \mathbf{0.04236}$$

Los valores obtenidos hasta este punto corresponden a la columna de terceras diferencias D3 de la tabla 6.2. Una vez que se han calculado los 2 valores indicados por `lim` la variable j se incrementa en uno por lo que para la siguiente iteración de j tenemos: $j = 3 \qquad lim = ord - j - 1 = 4 - 3 - 1 = 0$ por lo que el for interno solo se ejecutará una vez para calcular el último valor del arreglo de diferencias divididas.

$$Cuando\ i = 0 \quad f[x_4, x_3, x_2, x_1, x_0] = C[0][4] = \frac{C[0+1][3] - C[0][3]}{x[0+3+1] - x[0]}$$

$$f[x_4, x_3, x_2, x_1, x_0] = C[0][4] = \frac{C[1][3] - C[0][3]}{x[4] - x[0]} = \frac{\frac{61}{1440} - \left(-\frac{1}{30}\right)}{9 - 0} -= \frac{\frac{\mathbf{109}}{\mathbf{1440}}}{\mathbf{9}} = \frac{\mathbf{109}}{\mathbf{12960}} \approx \mathbf{0.0084105}$$

Las dos líneas de código resaltadas en negritas se encargan de escribir cada valor obtenido en la posición correspondiente de la tabla DiferenciasG.

6.3.2 Funcionamiento del botón Polinomio.

Como se mencionó al principio de este capítulo este programa da la posibilidad de ajustar el polinomio a distintos puntos para observar el comportamiento de los polinomios obtenidos. En la primera parte se explica el caso general en el que el polinomio se ajusta a todos los puntos y en la segunda parte se explica cómo se ajusta el polinomio a diferente cantidad de puntos.

6.3.2.1 Funcionamiento del botón polinomio para todos los puntos insertados.

Para este caso continuemos con el ejemplo en el que hemos almacenado en la matriz C las diferencias divididas. Ahora bien, los elementos de la matriz que son de interés para calcular el polinomio de interpolación son los que están almacenados en la primera fila de la matriz:

$$C[0][0] = f(x_0) = \mathbf{-3}\ \ C[0][1] = 3\ \ C[0][2] = -\frac{1}{6} \approx \mathbf{-0.1667}\ \ C[0][3] = -\frac{1}{30} \approx \mathbf{-0.0333}$$
$$C[0][4] = \frac{109}{12960} \approx \mathbf{0.0084105}$$

Que corresponden a las diferencias:

$$f[x_1, x_0] = C[0][1]\ \ f[x_2, x_1, x_0] = C[0][2]\ f[x_3, x_2, x_1, x_0] = C[0][3]\ \ f[x_4, x_3, x_2, x_1, x_0] = C[0][4]$$

Y aplicando la ecuación (9) queda:

$$f_4(x) = f(x_0) + (x - x_0)f[x_1, x_0] + (x - x_0)(x - x_1)f[x_2, x_1, x_0] + (x - x_0)(x - x_1)(x - x_2)f[x_3, x_2, x_1, x_0] + (x - x_0)(x - x_1)(x - x_2)(x - x_3)f[x_4, x_3, x_2, x_1, x_0] - - - (10)$$

Sustituyendo valores el polinomio es:

$$f_4(x) = -3 + 3(x - 0) - \frac{1}{6}(x - 0)(x - 1) - \frac{1}{30}(x - 0)(x - 1)(x - 3) + \frac{109}{12960}(x - 0)(x - 1)(x - 3)(x - 6)$$

Efectuando productos y simplificando queda:

$$f_4(x) = \frac{109}{12960}x^4 - \frac{761}{6480}x^3 + \frac{31}{160}x^2 + \frac{2099}{720}x - 3 = -0.0084x^4 - 0.1174x^3 + 0.1938x^2 + 2.9153x - 3$$

Que es el polinomio de interpolación para este caso. A continuación se muestra el código que obtiene los coeficientes del polinomio de interpolación:

```
01    for(i=0;i<=ord;i++)
02    {
03        r[i]=C[0][i];
04    }
05    // Obtención del polinomio interpolador de Newton
06    //Se inicializa el arreglo suma1
07    for(k=0;k<=ord;k++) suma1[k]=0;
08    //El ciclo externo indica el sumando que se está trabajando
09    for(k=0;k<=ord;k++)
10    {//for1
11        //El segundo elemento del multiplicando 1 es igual a 1
12        m1[1]=1.0;
13        //Se limpia el arreglo del multiplicando 2
14        for(c=0;c<=ord;c++) m2[c]=0;
15        //Se inicializa el elemento 0 de m2
16        m2[0]=1.0;
17        //indica cuantos términos serán multiplicados
18        ope=k-1;
19        //cuando k=0 solo se coloca el primer valor del renglón de C
20        if(ope==-1) suma1[0]=r[0];
21        //Cuando k>0 se realiza multiplicación de términos
22        else
23        {
24        //este ciclo indica el término que se toma
25        for(n=0;n<=ope;n++)
26        {//for2
27                //para el multiplicando se hace el elemento cero igual
28                //al valor de x indicado por el índice n
29                m1[0]=-x[n];
30                //Estos ciclos multiplican m1*m2 elemento a elemento
31                for(i=0;i<=1;i++)
32                {//for3
33                    for(j=0;j<=ord;j++)
34                    {//for4
35                        A[i][j+i]=m1[i]*m2[j];
```

```
36                  }//for4
37              }//for3
38              //Estos ciclos suman las columnas de A
39              for(j=0;j<=ord;j++)
40              {
41                  suma=0;
42                  for(i=0;i<=1;i++)
43                  {
44                      suma=suma+A[i][j];
45                  }
46                  //Se almacena la suma de cada columna en el arreglo m2
47                  //para la siguiente multiplicación
48                  m2[j]=suma;
49              }
50
51      }//for2
52      //Se multiplica cada coeficiente por cada elemento del 1er renglón
53      //de las diferencias divididas
54      for(c=0;c<=ord;c++) suma1[c]=suma1[c]+m2[c]*r[k];
55      }//else
56      }//for1
```

Veamos cómo funciona el código para implementar el método. Primero las líneas 1 a 4 almacenan los valores de la primera fila de la matriz en otro arreglo que en este caso es el arreglo r[4], que almacena los valores:

$$r[0] = f(x_0) = -\mathbf{3}\ \ r[1] = \mathbf{3}\ \ r[2] \approx -\mathbf{0.1667}\ \ r[3] \approx -\mathbf{0.0333}\ \ r[4] = \approx \mathbf{0.0084105}$$

El ciclo for de la línea 9 se encarga de calcular los coeficientes de cada uno de los sumandos de la ecuación 10. En este caso el ciclo se repite desde `k=0` hasta `k=4` ya que son 5 sumandos. Además tenemos la variable entera `ope=k-1` que controla el ciclo for de la línea 25 que corresponde al número de términos que se multiplican. Así ope toma los valores mostrados en la tabla 6.3.

k	ope	Iteraciones del ciclo
0	-1	0
1	0	1
2	1	2
3	2	3
4	3	4

Tabla 6.3 Valores que toma la variable ope

Cuando ope=-1 no se realiza multiplicación alguna y solo se almacena el valor de $f(x_0)$ en suma$[0] = -3$ que corresponde a la posición del término independiente del polinomio.

$$cuando\ k = 1\ ope = 0\ las\ líneas\ 25\ a\ 40\ multiplican\ 1 * (x - 0)\ y\ m2 = [0\ 0\ 0\ 1\ 0]$$

La línea 25 multiplica cada elemento de m2 por r[k], en este caso r[1]=3 y los suma a los coeficientes anteriores de suma1.

$$suma1 = [0\ 0\ 0\ 0 - 3]\ \ m2[0\ \ 0\ \ 0 - 3\ \ \ 0]\ \ \ suma1 = suma1 + m2 = [0\ \ 0\ \ 0\ \ -3\ \ -3]$$

$$cuando\ k = 2\ ope = 1\ se\ multiplica\ 1 * (x - 0) * (x - 1)\ y\ m2 = [0\ \ \ 0\ \ \ 1 - 1\ \ \ 0]$$

La línea 25 multiplica cada elemento de m2 por r[k], en este caso r[2]=-1/6 y los suma a los coeficientes anteriores de suma1.

$$suma1 = [0\ 0\ 0 - 3 - 3]\ \ m2\left[0\ \ 0\ \ -\frac{1}{6}\ \ \frac{1}{6}\ \ 0\right]\ \ \ suma1 = suma1 + m2 = [0\ \ 0\ \ -\frac{1}{6}\ \ \frac{19}{6}\ \ -3]$$

$$cuando\ k = 3\ ope = 2 se\ multiplica\ 1 * (x - 0) * (x - 1)(x - 3)\ y\ m2 = [0\ \ \ 1\ \ -4\ \ \ 3\ \ \ 0]$$

La línea 25 multiplica cada elemento de m2 por r[k], en este caso r[3]=-1/30 y los suma a los coeficientes anteriores de suma1.

$$suma1 = [0 \quad 0 \quad -\frac{1}{6} \quad \frac{19}{6} \quad -3]\ m2\left[0 \quad -\frac{1}{30} \quad \frac{2}{15} \quad -\frac{1}{10} \quad -3\right]$$
$$suma1 = suma1 + m2 = \left[0 \quad -\frac{1}{30} \quad -\frac{1}{30} \quad \frac{46}{15} \quad -3\right]$$

$$cuando\ k = 4\ ope = 3\ se\ multiplica\ 1 * (x-0) * (x-1)(x-3)(x-6)\ y\ m2 = [1 \quad -10 \quad 27 \quad -18 \quad 0]$$

La línea 25 multiplica cada elemento de m2 por r[k], en este caso r[4]= $\frac{109}{12960}$ y los suma a los coeficientes anteriores de suma1.

$$suma1 = \left[0 \quad -\frac{1}{30} \quad -\frac{1}{30} \quad \frac{46}{15} \quad -3\right]\ m2\left[\frac{109}{12960} \quad -\frac{109}{1296} \quad \frac{109}{480} \quad -\frac{109}{720} \quad 0\right]$$
$$\boldsymbol{suma1 = suma1 + m2 = \left[\frac{109}{12960} \quad -\frac{761}{6480} \quad \frac{31}{160} \quad \frac{2099}{720} \quad -3\right]}$$

Que corresponde a los coeficientes del polinomio de interpolación.
La multiplicación de términos realizadas de las líneas 25 40 funcionan de la manera detallada en la sección 5.3.1.1.

6.3.2.2 Ajuste del polinomio para diferentes puntos.

El código mostrado en la sección anterior se encarga de obtener los coeficientes del polinomio, pero antes de ese código se encuentra otra sección de código que realiza la revisión de los datos de la tabla WxGrid2. Primero se encarga de revisar que toda la tabla ha sido llenada. Después revisa que los valores insertados sean consecutivos y la diferencia entre cada uno de ellos sea uno. Una vez hecho esto, aunque ya existe una tabla de diferencias divididas, se calcula y se almacena en la matriz C las diferencias dividas en base a los datos de la tabla de ajuste de puntos. Ahora veamos cómo funciona el ajuste de puntos.

	columna	0	1	2	3	4
renglón	x	f(x)	D1	D2	D3	D4
0	0	-3	3	-0.16666667	-0.03333333	0.00841049
1	1	0	2.5	-0.36666667	0.04236111	
2	3	5	0.66666667	-0.02777778		
3	6	7	0.5			
4	9	8.5				

Supongamos que el usuario desea ajustar para los pares de puntos x, f(x) de los renglones 1, 2 y 3. El programa almacena en la variable entera `ord` el número de filas menos uno de la tabla de puntos de ajuste, para este caso que queremos ajustar a los tres puntos de la tabla 6.4 `ord=3-1=2` y en la variable entera `ini` se almacena el valor inicial que es `ini=1`.

i=	x	f(x)
1	1	0
2	3	5
3	6	7

Tabla 6.4 Ajustar el polinomio para los puntos 2, 3 y 4.

Con esos datos se obtienen los valores de los puntos insertados haciendo uso de un ciclo for para los valores de x y otro para los valores de f(x) que tiene la forma: `for(i=ini1;i<=ini1+ord;i++)`. De esta manera se obtendrán los puntos de interés y se calculara la matriz de diferencias divididas que se muestra en la tabla 6.4.

i=	x	f(x)	D1	D2
1	1	0	2	-0.3667
2	3	5	0.6667	
3	6	7		

Tabla 6.5 Diferencias divididas para los puntos de ajuste.

Una vez hecho esto se almacenan en el arreglo r[2] los valores del primer renglón:

$$r[0] = f(x_0) = \mathbf{0}\ \ r[1] = \mathbf{2}\ \ r[2] \approx -\mathbf{0.3667}$$

Y con ésta información se procede de la manera explicada en la sección anterior. Finalmente hay que notar que la variable `ord` controla la ejecución del código que obtiene los coeficientes del polinomio. Esto se puede observar en la línea 1 del código de la sección anterior. De tal manera que para este ejemplo `ord=2` y se calcularán tres sumandos. Para mostrar los coeficientes en forma de polinomio se utiliza el algoritmo del programa de Lagrange que escribe paso a paso el polinomio en la caja de texto correspondiente.

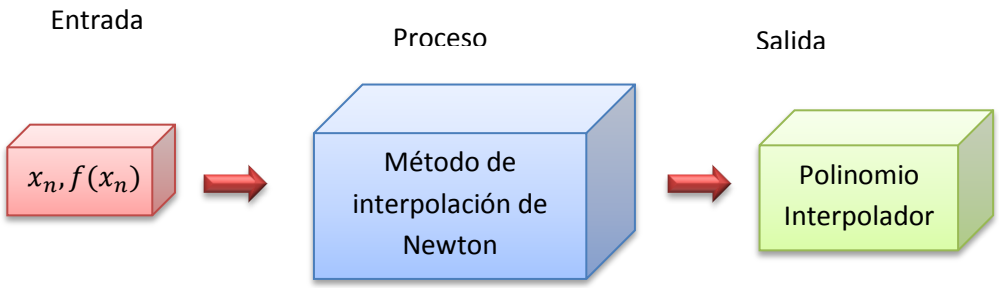
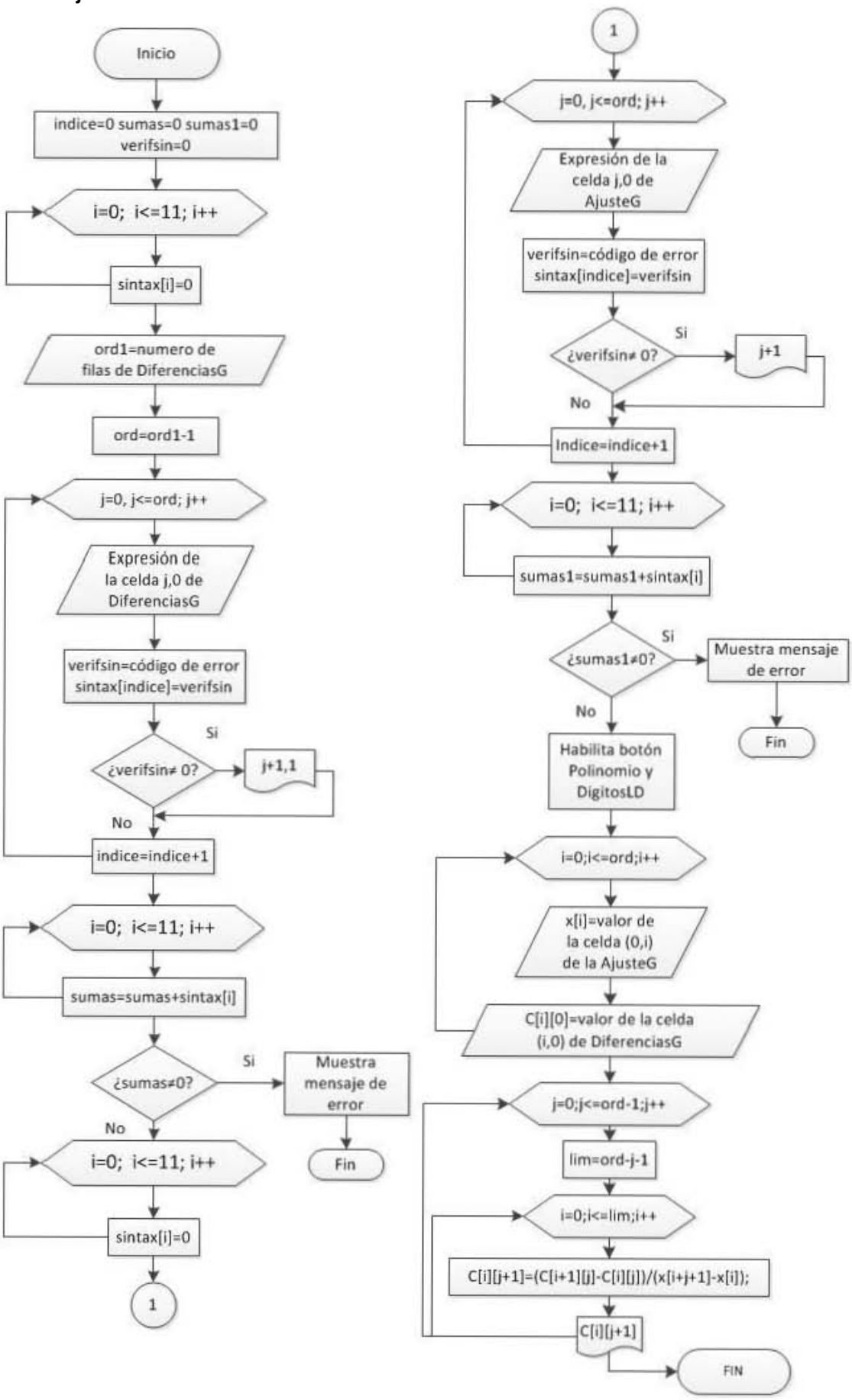
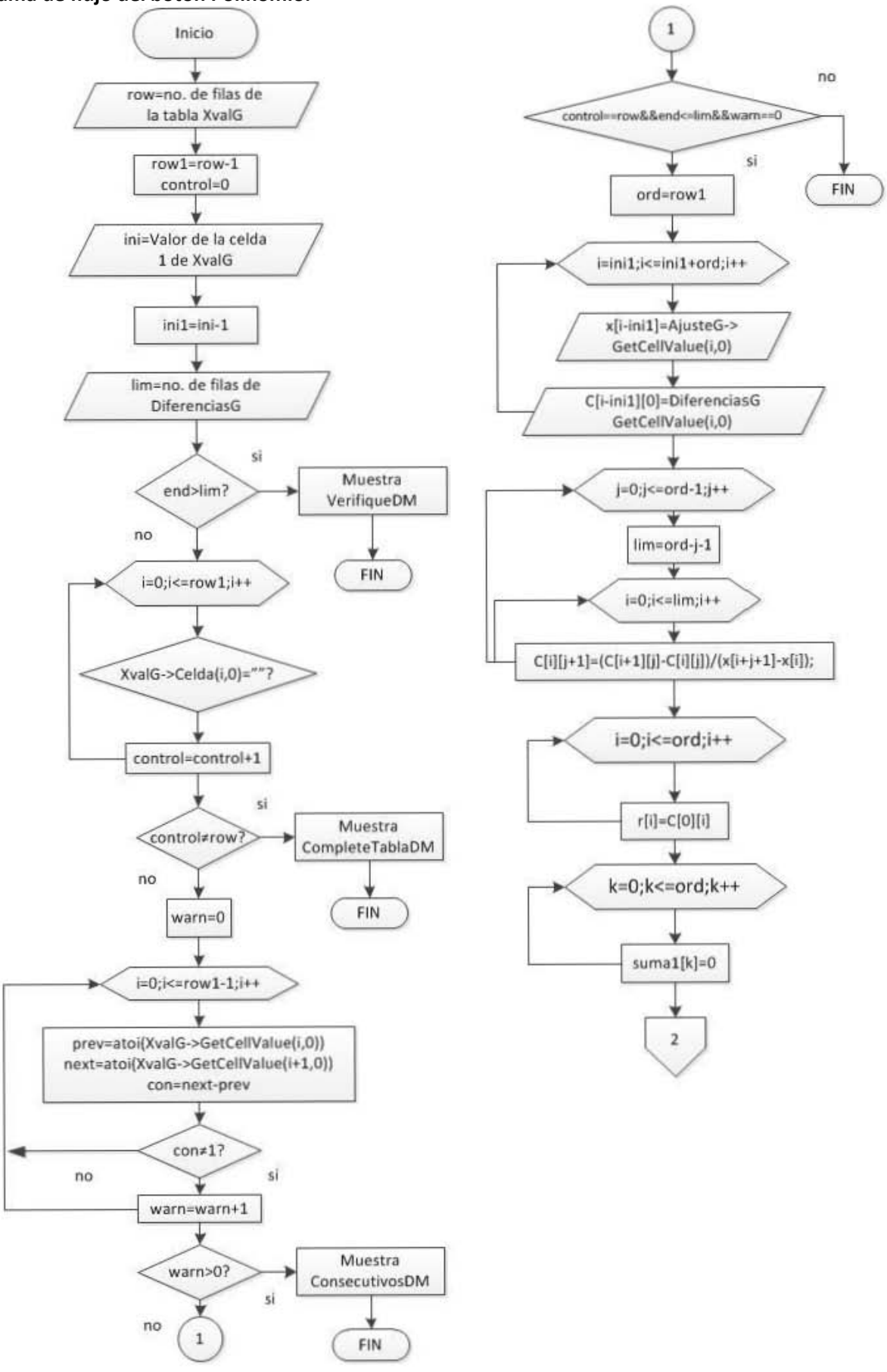


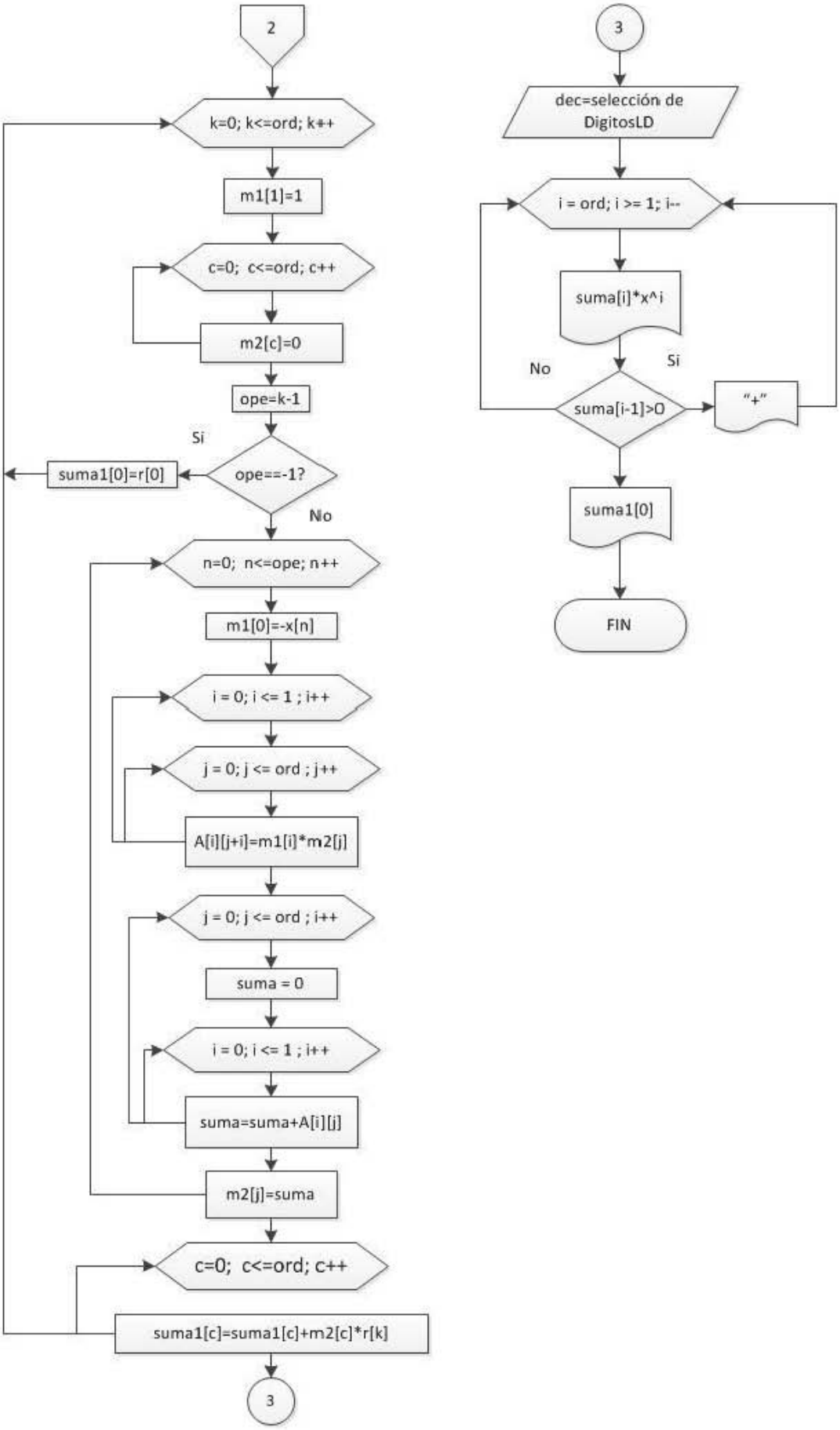
Figura 6.2 Diagrama de Entrada, Proceso, Salida del botón Calcular

6.4 Diagrama de flujo del botón Diferencias



6.5 Diagrama de flujo del botón Polinomio.




2
k=0; k<=ord; k++
m1[1]=1
c=0; c<=ord; c++
m2[c]=0
ope=k-1
Si
ope==-1?
suma1[0]=r[0]
No
n=0; n<=ope; n++
m1[0]=-x[n]
i = 0; i <= 1 ; i++
j = 0; j <= ord ; j++
A[i][j+i]=m1[i]*m2[j]
j = 0; j <= ord ; i++
suma = 0
i = 0; i <= 1 ; i++
suma=suma+A[i][j]
m2[j]=suma
c=0; c<=ord; c++
suma1[c]=suma1[c]+m2[c]*r[k]
3
3
dec=selección de DigitosLD
i = ord; i >= 1; i--
suma[i]*x^i
No
Si
suma[i-1]>0
"+"
suma1[0]
FIN

7. INTERPOLACIÓN SPLINE

La interpolación polinomial no es una técnica adecuada para representar globalmente una función en un intervalo largo, todo lo contrario, se trata de un procedimiento para buscar un polinomio de grado reducido, que represente localmente a una función de forma adecuada dentro de un pequeño intervalo. En términos muy generales, una función spline es una función polinomial por tramos que es continua y posee derivadas continuas hasta un cierto orden. Además de las condiciones de continuidad y suavidad, el spline deberá satisfacer algunas otras condiciones adecuadas al problema que se desea resolver. Para lograr todas estas condiciones, el spline contiene un conjunto de parámetros cuyos valores se escogen de forma que se satisfagan todas las condiciones deseadas. En este caso se han contemplado los casos:

- Natural.
- Anclado.
- Extrapolación.
- Curvatura ajustada en puntos extremos.

El programa se puede apreciar en la figura 7.1. Donde que se observan las 4 opciones mencionadas que se presentan con botones circulares, además de que cuenta con un componente SpinButton para desplazarse entre los polinomios encontrados con la posibilidad de modificar el número de dígitos decimales según el usuario lo requiera.

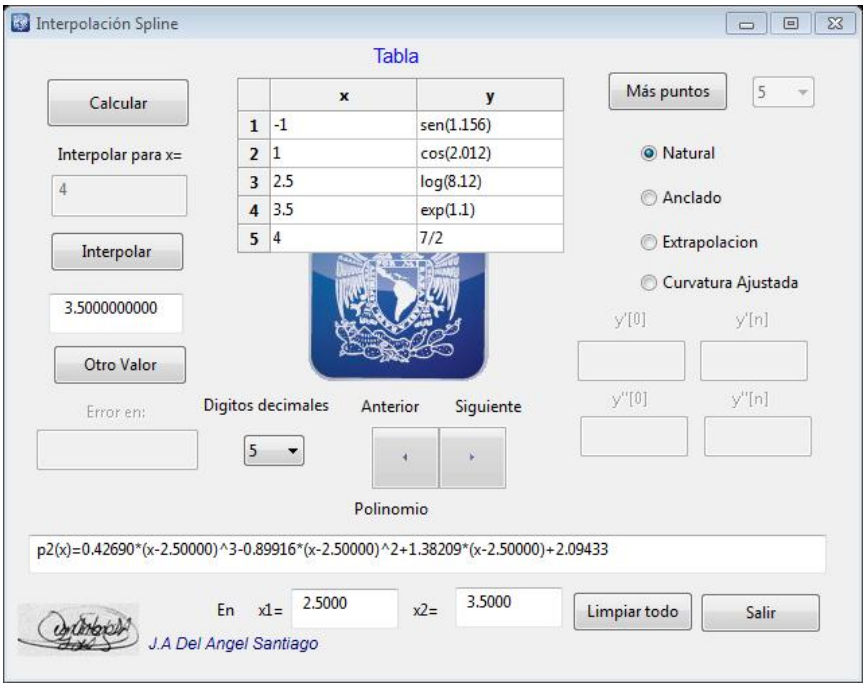


Figura 7.1 Interpolación Spline

7.1 CONSIDERACIONES DE DISEÑO DEL PROGRAMA.

- Se podrán insertar hasta 10 puntos en la tabla de valores.
- Se debe contar con una lista de selección para seleccionar el número de puntos a insertar.
- Se debe contar con una lista de selección que permita elegir el número de dígitos decimales que serán mostrados en el polinomio.
- Se contará con una caja de texto que muestre los polinomio uno a la vez por lo que es necesario utilizar el componente SpinButton para navegar entre los polinomios encontrados.
- Se debe contar con una caja de texto para insertar el valor a interpolar y otra para mostrar el resultado.
- En una caja de texto se mostrarán las posiciones de la tabla donde se cometan errores de sintaxis.
- Se debe contar con 2 cajas de texto que muestren el intervalo al que pertenece el polinomio.
- En caso de que se seleccione la opción Anclado se deben habilitar dos cajas de texto para insertar los valores de $y'(0)$ y $y'(n)$.

- En caso de que se selecciona la opción Curvatura se deben habilitar dos cajas de texto para insertar los valores de $y''(0)\ y\ y''(n)$.
- Se debe contar con un botón para limpiar todas las cajas de texto y la tabla.

7. 2 COMPONENTES USADOS PARA EL PROGRAMA

- **Cajas de texto donde:**

1. Se inserta el valor a interpolar: xvalCT.
2. Se muestra el resultado de la interpolación: ResultCT.
3. Se muestran las posiciones de error: ErrorCT.
4. Se muestra el Polinomio: PolinomioCT.
5. Se muestra el límite inferior del intervalo: X1CT.
6. Se muestra el límite superior del intervalo: X2CT.
7. Se inserta el valor de y'(0): Ypd0CT.
8. Se inserta el valor de y'(n): YpdnCT.
9. Se inserta el valor de y''(0): Ysd0CT.
10. Se inserta el valor de y''(n): YsdnCT.

- **Tablas.**

11. Tabla donde se insertan los puntos: TablaG.

- **Botones.**

12. Calcular: CalcularB.
13. Interpolar: InterpolarB.
14. Otro Valor: OtroValB.
15. Mas Puntos: PuntosB.
16. Limpiar Todo. LimpiarB.
17. Salir: SalirB.

- **Textos Estáticos.**

18. Interpolar para x=: InterpolarST.
19. Posiciones de error: ErrorST.
20. Dígitos decimales: DigitosST.
21. Anterior Siguiente: SpinST.
22. Polinomio: PolinomioST.
23. En x1=: X1ST.
24. x2: X2ST.
25. y'[0]: Ypd0ST.
26. y'[n]: YpdnST.
27. y''[0]: Ysd0ST.
28. y''[n]: YsdnST.

- **Listas de selección.**

29. Dígitos decimales: DigitosLD.
30. Número de puntos a insertar: MasPuntosLD

- **Botón de navegación.**

31. Navegación entre los polinomios encontrados: DesplazarSB.

- **Botones circulares:**

32. Natural: NaturalRB.
33. Anclado: AncladoRB.

34. Extrapolación: ExtrapolacionRB.
35. Curvatura: CurvaturaRB.

- **Diálogos de mensaje.**

36. Error de sintaxis: SintaxisDM.
37. Primer Polinomio: PrimeroDM.
38. Último Polinomio: Ultimo DM.
39. Valor fuera de rango: RangoDM.
40. Inserte los valores de y'(0) y y(n): PderivaDM.
41. Inserte los valores de y''(0) y y''(n): SderivaDM.
42. Error de sintaxis en los parámetros: ErrorparDM.
43. Inserte un valor a interpolar: XvalDM.

7.3 Expresiones del método.

Se denomina ajuste cúbico "spline" a una función definida en el intervalo $[x_0, x_1]$. Si existen polinomios cúbicos $p_0(x), p_1(x), p_2(x), ..., p_{n-1}(x)$, tal que, se cumple que dicha función definida en forma discreta se expresa mediante un conjunto de polinomios; es decir:

$$y(x) = p_i(x) \quad sobre \quad [x_i, x_{i+1}], \quad i = 0,1,2,...,n-1$$

Y además se cumple que:

$$. \quad p'_{i-1}(x_i) = p'_i(x_i) \ , \qquad i = 1,2,...,n-1 \ (igual \quad derivada \ ) \quad ...(7.1)$$

$$. \quad p''_{i-1}(x_i) = p''_i(x_i) \ , \qquad i = 1,2,...,n-1 \ (igual \quad curvatura \ ) ...(7.2)$$

$$. \quad p_i(x_i) = y_i \ , \quad p_i(x_{i+1}) = y_{i+1} \ ; \ i = 0,1,2,...,n-1 \ ...(7.3)$$

Podría notarse que en los puntos extremos x_0 , x_n , la continuidad sobre la pendiente y la curvatura no son asignadas. Las condiciones son asignadas en esos puntos, en general dependiendo de las aplicaciones.
Sea $[x_i, x_{i+1}]$, el intervalo que denota al i-ésimo intervalo. Así mismo, se denota el espaciamiento entre abscisas:

$$h_i = x_i - x_{i-1} \ , i = 1,2,3,...,n$$

Y se denota a las segundas derivadas de la función tratada, en los distintos puntos o nodos.

$$M_i = y''(x_i) \ , i = 0,1,2,...,n$$

El Spline cúbico para el i-ésimo intervalo es:

$$y(x) = a_i(x - x_i)^3 + b_i(x - x_i)^2 + c_i(x - x_i)^3 + d_i \ ... (7.4)$$

Para encontrar a_i, b_i, c_i se utiliza:

$$6\left(\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i} \right) = h_i M_{i-1} + 2(h_i + h_{i+1}) M_i + h_{i+1} M_{i+1} \quad ...(7.5)$$

Esta relación es cierta para $i = 1,2,3,\ldots,(n-1)$. Entonces se dispone de (n-1) ecuaciones para resolver las (n+1) cantidades desconocidas: $M_0, M_1 M_2 \ldots M_n$. Luego entonces, se requieren 2 condiciones más, para resolver el problema del número de ecuaciones necesarias. Si tenemos:

$$A_i = h_i \quad \ldots(7.6)$$

$$B_i = 2(h_i + h_{i+1}) \quad \ldots(7.7)$$

$$C_i = h_{i+1} \quad \ldots(7.8)$$

$$D_i = 6\left(\frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i}\right) \quad \ldots(7.9)$$

La ecuación (7.5), quedaría expresada como:

$$A_i M_{i-1} + B_i M_i + C_i M_{i+1} = D_i \;\; ; \;\; i = 1,2,3,\ldots,(n-1) \quad \ldots(7.10)$$

La ecuación (7.10) representa básicamente un sistema de ecuaciones tri-diagonal. Para el que de acuerdo a las distintas condiciones, se tendrá un sistema de ecuaciones tri-diagonal específico. Las condiciones pueden ser obtenidas considerando las formas siguientes:

i. Para el Spline natural

$$\begin{bmatrix} B_1 & C_1 & 0 & 0 & \ldots & 0 & 0 & 0 \\ A_2 & B_2 & C_2 & 0 & \ldots & 0 & 0 & 0 \\ 0 & A_3 & B_3 & C_3 & \ldots & 0 & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & \ldots & 0 & A_{n-1} & B_{n-1} \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ . \\ . \\ M_{n-1} \end{bmatrix} = \begin{bmatrix} \mathrm{D}_1 \\ \mathrm{D}_2 \\ . \\ . \\ \mathrm{D}_{n-1} \end{bmatrix} \quad \ldots(7.11)$$

Con $M_0 = M_n = 0$

ii. Spline cúbico anclado.

$$\begin{bmatrix} 2 & 1 & 0 & 0 & \ldots & 0 & 0 & 0 \\ A_1 & B_1 & C_1 & 0 & \ldots & 0 & 0 & 0 \\ 0 & A_2 & B_2 & C_2 & \ldots & 0 & 0 & 0 \\ \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\ 0 & 0 & 0 & 0 & \ldots & A_{n-1} & B_{n-1} & C_{n-1} \\ 0 & 0 & 0 & 0 & \ldots & 0 & 1 & 2 \end{bmatrix} \begin{bmatrix} M_0 \\ M_1 \\ . \\ . \\ M_{n-1} \\ M_n \end{bmatrix} = \begin{bmatrix} \mathrm{D}_0 \\ \mathrm{D}_1 \\ . \\ . \\ \mathrm{D}_{n-1} \\ \mathrm{D}_n \end{bmatrix} \quad \ldots(7.12)$$

$$2M_0 + M_1 = D_0 \ldots(7.13) \quad \text{y} \quad M_{n-1} + 2M_n = D_n \ldots(7.14)$$

Donde $D_0 = \frac{6}{h_1}\left(\frac{y_1 - y_0}{h_1} - y'_0\right) \ldots(7.15)$ y $D_0 = \frac{6}{h_n}\left(y'_0 - \frac{y_n - y_{n-1}}{h_n}\right) \ldots(7.16)$

iii. Extrapolación.

$$\begin{bmatrix} B'_1 & C'_1 & 0 & 0 & \dots & 0 & 0 & 0 \\ A_2 & B_2 & C_2 & 0 & \dots & 0 & 0 & 0 \\ 0 & A_3 & B_3 & C_3 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & A'_{n-1} & B'_{n-1} \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ . \\ . \\ M_{n-1} \end{bmatrix} = \begin{bmatrix} D_1 \\ D_2 \\ . \\ . \\ D_{n-1} \end{bmatrix} \dots (7.17)$$

Los valores de M_0 y M_n estan dados por las relaciones:

$$M_0 = M_1 - \frac{h_1(M_2 - M_1)}{h_2} \dots (7.18) \quad y \quad M_n = M_{n-1} + \frac{h_n(M_{n-1} - M_{n-2})}{h_{n-1}} \dots (7.19)$$

La primera expresión se puede volver a escribir como:

$$M_1\left[A_1 + B_1 + \frac{A_1 h_1}{h_2}\right] + M_2\left[C_1 - \frac{A_1 h_1}{h_2}\right] = D_1 \quad \text{ó} \quad M_1 B'_1 + M_2 C'_1 = D_1 \dots (7.20)$$

Dónde:

$$B'_1 = A_1 + B_1 + \frac{A_1 h_1}{h_2} \dots (7.21) \quad y \quad C'_1 = C_1 - \frac{A_1 h_1}{h_2} \dots (7.22)$$

De manera similar la segunda expresión se puede volver a escribir como:

$$M_{n-2}A'_{n-1} + M_{n-1}B'_{n-1} = D_{n-1} \dots (7.23)$$

Dónde:

$$A'_{n-1} = A_{n-1} - \frac{C_{n-1}h_n}{h_{n-1}} \dots (7.24) \quad y \quad B'_{n-1} = B_{n-1} + C_{n-1} + \frac{h_{n1}C_{n-1}}{h_{n-1}} \dots (7.25)$$

iv. Spline de curvatura ajustada en puntos extremos.

$$\begin{bmatrix} B_1 & C_1 & 0 & 0 & \dots & 0 & 0 & 0 \\ A_2 & B_2 & C_2 & 0 & \dots & 0 & 0 & 0 \\ 0 & A_3 & B_3 & C_3 & \dots & 0 & 0 & 0 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & 0 & \dots & 0 & A_{n-1} & B_{n-1} \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ . \\ . \\ M_{n-1} \end{bmatrix} = \begin{bmatrix} D'_1 \\ D'_2 \\ . \\ . \\ D_{n-2} \\ D'_{n-1} \end{bmatrix} \dots (7.26)$$

Donde $D'_1 = D_1 - A_I y''_0 \dots (7.27)$ $\qquad D'_{n-1} = D_{n-1} - C_{n-1}y''_n \dots (7.28)$

7.4 Algoritmo del método.

Este algoritmo encuentra el spline cúbico para cada uno de los intervalos $[x_i, x_{i+1}], i = 0,1, \dots, n-1$. El (i) Spline natural, (ii) Spline Anclado, (iii) Spline de Extrapolación y (iv) Spline de curvatura ajustada en puntos extremos son incorporados aquí. El espacio para x_i no necesita sere igual y se asume que $x_0 < x_1 < \cdots < x_n$.

- Cálculo de h_i

Leer $x_i, y_i, \quad i = 0, 1, 2, \dots, n;$

for $i = 1$ to n do
  $h_i = x_i - x_{i-1};$
end for.

- Cálculo de A_i, B_i, C_i y D_i.

for $i = 1$ to $n - 1$ do

$A_i = h_i \ldots (7.6) \quad B_i = 2(h_i + h_{i+1}) \ldots (7.7) \quad C_i = h_{i+1} \ldots (7.8) \quad D_i = 6\left(\frac{y_{i+1}-y_i}{h_{i+1}} - \frac{y_i-y_{i-1}}{h_i}\right) \ldots (7.9)$

end for;

Caso:

(i) *Spline Natural.*

Para encontrar $M_1, M_2, \ldots, M_{n-1}$; se resuelve el sistema de ecuaciones tri-diagonal definido en (7.11).
Con $M_0 = M_n = 0$

(ii) *Spline Anclado.*

Lee y'_0, y'_n //primera derivada de y en $x = x_0, x_n$//

Calcula: $D_0 = \frac{6}{h_1}\left(\frac{y_1-y_0}{h_1} - y'_0\right) \ldots (7.15)$ y $D_0 = \frac{6}{h_n}\left(y'_0 - \frac{y_n-y_{n-1}}{h_n}\right) \ldots (7.16)$

Para encontrar $M_0, M_1, M_2 \ldots, M_n$ se resuelve el sistema tri-diagonal de ecuaciones definido en (7.12).

(iii) *Spline de Extrapolación.*

Se calcula: $B'_1 = A_1 + B_1 + \frac{A_1 h_1}{h_2} \ldots (7.21) \quad y \quad C'_1 = C_1 - \frac{A_1 h_1}{h_2} \ldots (7.22)$

$$A'_{n-1} = A_{n-1} - \frac{C_{n-1} h_n}{h_{n-1}} \ldots (7.24) \quad y \quad B'_{n-1} = B_{n-1} + C_{n-1} + \frac{h_{n1} C_{n-1}}{h_{n-1}} \ldots (7.25)$$

Para encontrar $M_1, M_2, \ldots, M_{n-1}$ se resuelve el sistema tri-diagonal de ecuaciones definido en (7.17). Se calcula:

$$M_0 = M_1 - \frac{h_1(M_2 - M_1)}{h_2} \ldots (7.18) \quad y \quad M_n = M_{n-1} + \frac{h_n(M_{n-1} - M_{n-2})}{h_{n-1}} \ldots (7.19)$$

(iv) *Spline curvatura ajustada en puntos extremos.*

Lee y''_0, y''_n //segunda derivada de y en $x = x_0, x_n$//

Se calcula: $D'_1 = D_1 - A_I y''_0 \ldots (7.27) \qquad D'_{n-1} = D_{n-1} - C_{n-1} y''_n \ldots (7.28)$

Para encontrar $M_1, M_2, \ldots, M_{n-1}$ se resuelve el sistema tri-diagonal de ecuaciones definido en (7.26).
Con $M_0 = y''_0, \; M_n = y''_n$.

Fin de Caso.

//Se calculan los coeficientes a_i, b_i, c_i, d_i de cada polinomio de (7.4).

for i $= 0$ hasta $n - 1$

$$a_i = \frac{M_{i+1} - Mi}{6h_{i+1}} \qquad b_i = \frac{M_i}{2}$$

$$c_i = \frac{y_{i+1} - y_i}{h_{i+1}} - \frac{2h_{i+1}M_i + h_{i+1}M_{i+1}}{6} \qquad y_i = d_i$$

end for

//Se muestran los splines//

for i $= 0$ hasta $n - 1$

"Coeficientes del" i "esimo spline" a_i, b_i, c_i, d_i.

end do.

FIN DEL ALGORITMO

7.5 Funcionamiento de los componentes

Se explica en esta sección el funcionamiento de los componentes:

- Calcular
- Desplazar
- Interpolar
- Botones Circulares

La lista de selección ejecuta exactamente el mismo código del botón desplazar por la razón mencionada en la sección 5.3.2.

7.5.1 Funcionamiento del botón Calcular.

El código de este programa al igual que el algoritmo y las expresiones fueron extraídos de la referencia bibliográfica 5, (para consultar el código original véase la página 151, el código modificado se muestra al final de este capítulo). El código original está orientado a una aplicación de consola, en este caso se realizaron algunas modificaciones para adaptarlo al funcionamiento de la aplicación con ventana mostrada en la figura 7.1. El código de este botón realiza los siguientes pasos:

1. Aplica el algoritmo de revisión de sintaxis de tablas para verificar que todos los datos introducidos tengan una sintaxis correcta.
2. Verifica que si el usuario ha seleccionado los casos Anclado o de Curvatura Ajustada en puntos extremos. Dependiendo del caso verifica la sintaxis de los valores insertados en las cajas de texto y verifica que no se omita el llenado de alguna de ellas. En caso de cometer errores de sintaxis u omitir una caja se mostraran los diálogos de mensaje correspondientes.
3. Verifica cuál de los cuatro casos ha sido seleccionado y en base a eso ejecuta el código correspondiente para obtener los coeficientes a_i, b_i, c_i, d_i.
4. Muestra en la caja de texto PolinomioCT el primer polinomio obtenido.

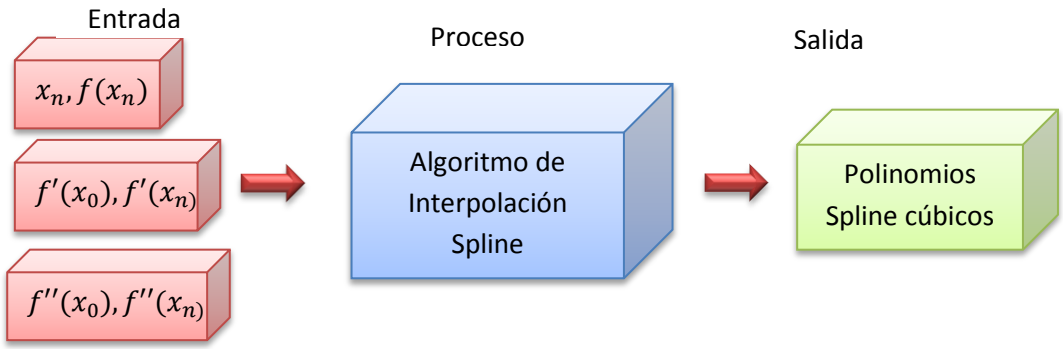


Figura 7.2 Diagrama de Entrada, Proceso, Salida del botón Calcular.

7.5.2 Funcionamiento del botón de navegación.

El código del botón de Navegación es el mismo que el usado para el botón Calcula con una pequeña variación. Para entender como se muestra el siguiente o el anterior Polinomio veamos primero el formato usado para mostrar el primer polinomio cuando se pulsa el botón Calcular:

```
"p0(x)=%7.*f*(x%c%7.*f)^3%c%7.*f*(x%c%7.*f)^2%c%7.*f*(x%c%7.*f)%c%7.*f",dig, a[0],s[1],dig,
temp,s[2],dig,fabs(b[0]),s[1],dig,temp,s[3],dig,fabs(c[0]),s[1],dig,temp, s[4],dig,fabs(d[0]));
```

La parte resaltada en negritas es lo que se escribe en la caja de texto y lo que no está resaltado son los parámetros que necesita para mostrar cada polinomio. En cursiva se observan los valores de a_i, b_i, c_i, d_i en los que se hace referencia al primer elemento de cada arreglo por lo que cuando se pulse el botón Calcular se mostrará únicamente el primer polinomio. Ahora analicemos el siguiente código:

```
1. int spin=DesplazarSB->GetValue();
```

```
2.  if(spin<=0)
3.  {
4.      DesplazarSB->SetValue(0);
5.      PrimeroDM->ShowModal();
6.  }
7.  if(spin>=n)
8.  {
9.      DesplazarSB->SetValue(n-1);
10.     UltimoDM->ShowModal();
11.     spin=n-1;
12. }
13. temp=fabs(x[spin]);
14. dig=DigitosLD->GetSelection();
15. if(dig==-1) dig=3;
16. dig=dig+1;
17. sprintf(w,"p%d(x)=%7.*f*(x%c%7.*f)^3%c%7.*f*(x%c%7.*f)^2%c%7.*f*(x%c%7.*f)%c%7.*f",spin,di
    g,a[spin],s[1],dig,temp,s[2],dig,fabs(b[spin]),s[1],dig,temp,s[3],dig,fabs(c[spin]),s[1],d
    ig,temp,s[4],dig,fabs(d[spin]));
```

En la línea 1 se obtiene el valor del botón de Navegación y se almacena en la variable entera spin, hay que recordar que cada que se pulsa hacia adelante su valor se incrementa en uno y cuando se pulsa hacia atrás a su valor se le resta 1. En la línea 2 si se está mostrando el primer polinomio y se pulsa hacia atrás el valor del botón sería menor a cero lo cual es un error debido a que se estará intentando acceder al elemento a[-1], por ejemplo. Por lo tanto en la línea 4 se establece el valor del botón igual a cero para que no ocurra tal error y en la línea 5 se notifica al usuario que es el primer polinomio.

Si hemos seleccionado insertar 5 puntos entonces se deben mostrar 4 polinomios que son $p_0(x), p_1(x), p_2(x), p_3(x)$ y como límite se accederá al cuarto elemento de los arreglos: a[3], b[3], c[3] y d[3]. En la línea 7 se comprueba si se ha pulsado Adelante una vez más, en tal caso la línea establece el valor del botón de desplazamiento igual a n-1, en este caso n=4 y n-1=3. Además en la línea 10 se notifica al usuario que se trata del último polinomio calculado y en la línea 11 también se establece el valor de n-1 para la variable spin que es la que almacena el valor del botón. En la línea 13 se obtiene el valor absoluto de x[spin] y en la línea 14 se obtiene el número de dígitos decimales que el usuario desea que se muestren. Si no se ha seleccionado ningún valor desde que se inicia el programa la selección de la Lista tiene el valor predeterminado -1 pero no es adecuado este valor para el número de dígitos decimales y se establece 3 por defecto en este caso.

En la línea 14 se suma 1 a la selección de la lista ya que el primer elemento de la lista devuelve el valor cero. Finalmente en la línea 17 se escribe cada polinomio en la caja de texto. Se puede notar que en este caso no se hace referencia al primer elemento si no a los elementos indicados por la variable spin que son a[spin], b[spin], c[spin] y d[spin]. Por lo tanto si se pulsa adelante o atrás se mostrará el polinomio correspondiente. Podría funcionar correctamente sin el botón calcular pero se prefirió que ese botón realizara toda la revisión de sintaxis y en caso de que sea correcta se habilite el botón desplazar para mostrar el resto de los polinomios.

7.5.3 Funcionamiento del botón Interpolar.

El botón interpolar utiliza el mismo código pero en este caso no muestra polinomio si no que una vez calculados los valores de a_i, b_i, c_i, d_i se verifica la sintaxis del valor a interpolar y se utiliza el código siguiente.

```
for(int i=0;i<=n-1;i++)
{
    if(xg<=x[i+1]&&xg>=x[i])
    {
        temp=xg-x[i];
        yc=a[i]*temp*temp*temp+b[i]*temp*temp+c[i]*temp+d[i];
        char w[3];
        sprintf(w,"  %7.10f",yc);
        ResultCT->WriteText(w);
    }
}
```

El valor a interpolar insertado en la caja de texto XvalCT se almacena en la variable de tipo **double** xg y se recorren los intervalos mediante un ciclo for para encontrar a que intervalo pertenece el valor de x mediante la condición if mostrada, una vez que se ha encontrado el intervalo adecuado se sustituye el valor y se almacena en yc para posteriormente convertirse a cadena de caracteres y mostrar el resultado en ResultCT.

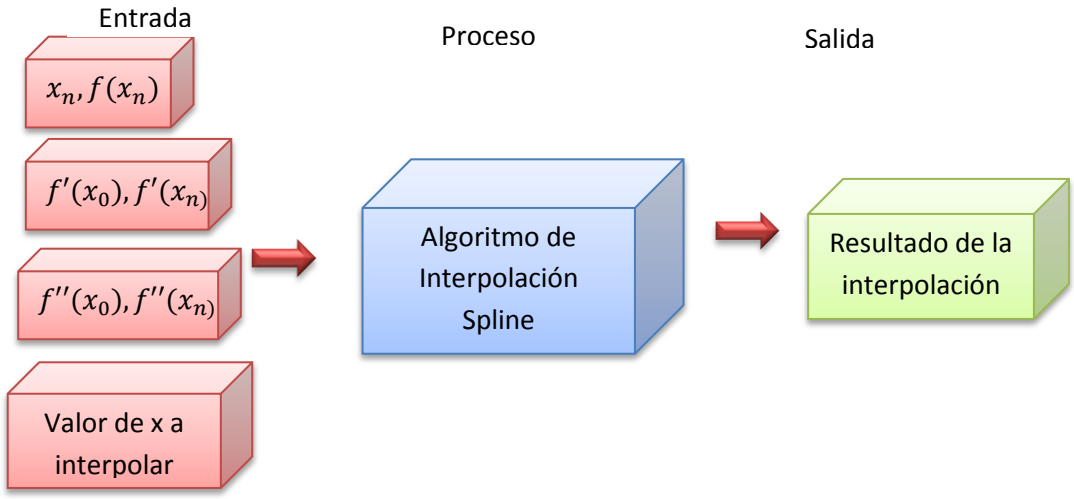


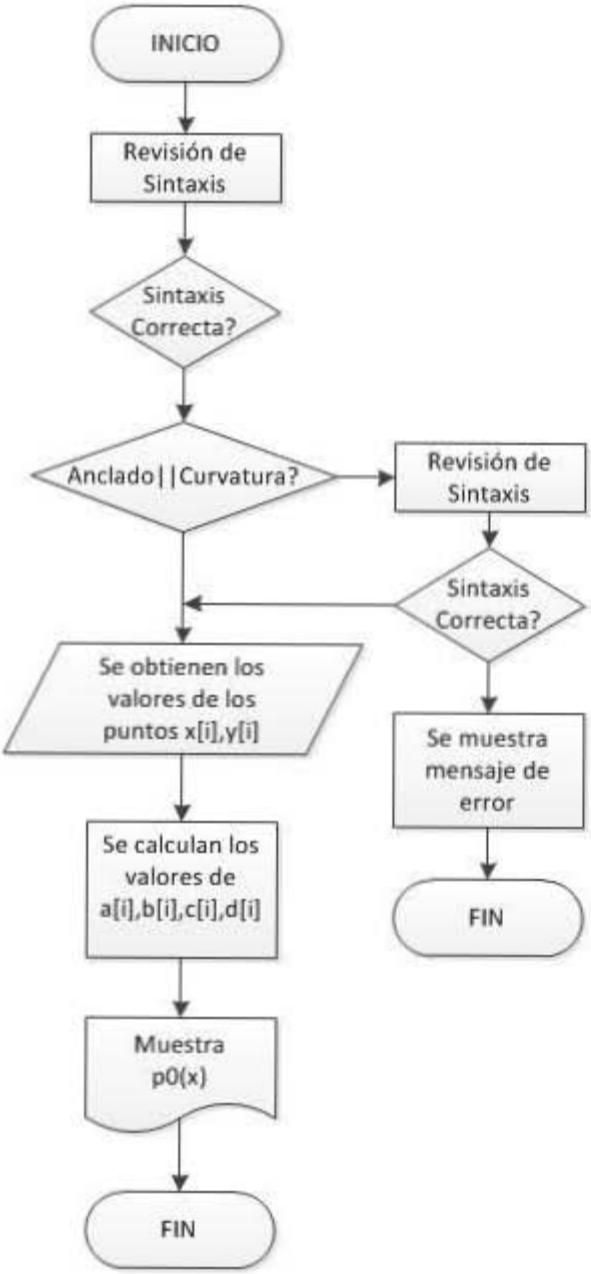
Figura 7.3 Diagrama de Entrada, Proceso, Salida del botón Interpolar.

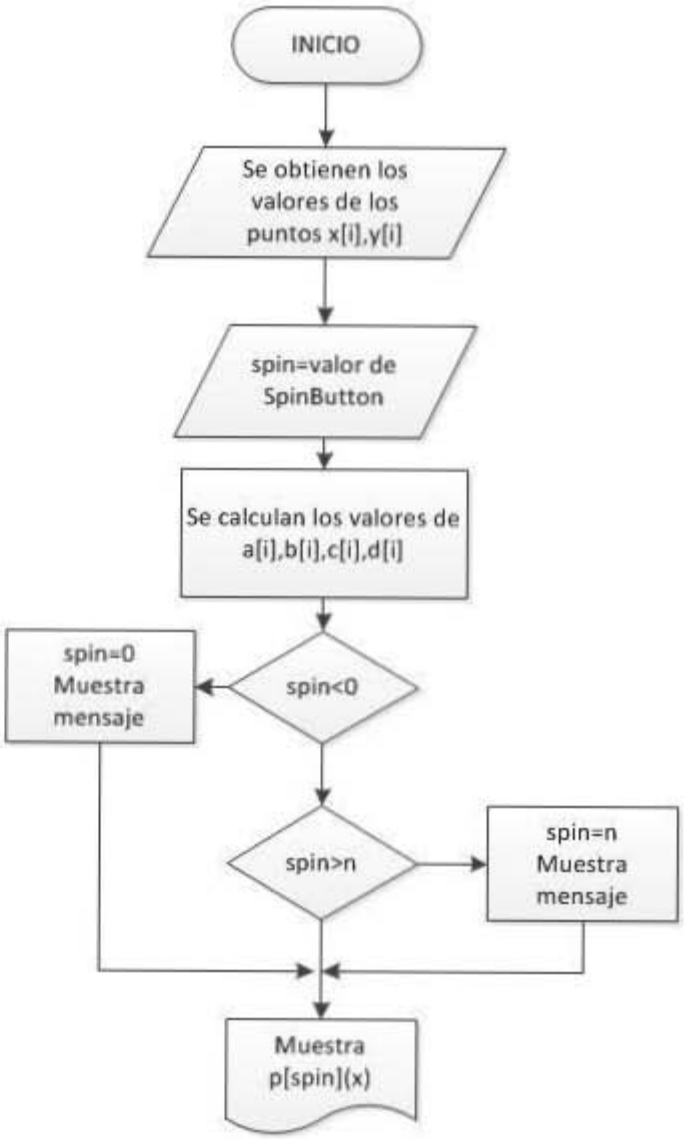
7.5.4 Funcionamiento de los botones circulares.

El funcionamiento de los botones circulares se basa en ocultar y mostrar componentes. Cuando se pulsa el botón Anclado se deben habilitar las cajas para insertar los valores de $y'(0), y'(n)$, pero se deben deshabitar las cajas de texto donde se insertan los valores de $y''(0), y''(n)$ en caso de que la opción Curvatura haya sido seleccionada anteriormente. De la misma manera si se pulsa la opción Curvatura se deben habilitar las cajas de texto se deben habilitar las cajas de texto donde se insertan los valores de $y''(0), y''(n)$ y deshabilitar las cajas que corresponde a los valores de $y'(0), y'(n)$ en caso de que la opción Anclado se haya seleccionado antes. Para las opciones de Natural y Extrapolación se deben deshabilitar las cuatro cajas de texto de las derivadas en caso de que alguna de ellas haya sido pulsada antes. De tal manera que el código es el siguiente para cada botón circular:

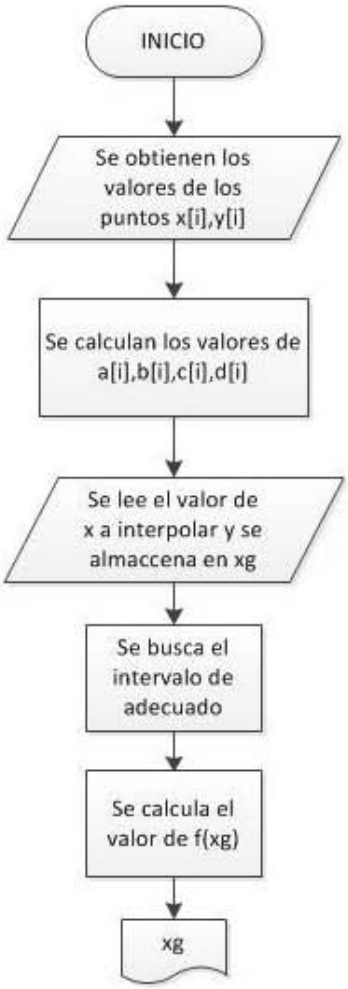
```
NATURAL, EXTRAPOLACIÓN
    YsdnCT->Enable(false);
    ysdnCT->Enable(false);
    Ysd0ST->Enable(false);
    YsdnST->Enable(false);
    Ypd0CT->Enable(false);
    YpdnCT->Enable(false);
    Ypd0ST->Enable(false);
    YpdnST->Enable(false);
ANCLADO
    Ypd0CT->Enable(true);
    YpdnCT->Enable(true);
    Ypd0ST->Enable(true);
    YpdnST->Enable(true);
    YsdnCT->Enable(false);
    ysdnCT->Enable(false);
    Ysd0ST->Enable(false);
    YsdnST->Enable(false);
CURVATURA
    YsdnCT->Enable(true);
    ysdnCT->Enable(true);
    Ysd0ST->Enable(true);
    YsdnST->Enable(true);
    Ypd0CT->Enable(false);
    YpdnCT->Enable(false);
    Ypd0ST->Enable(false);
    YpdnST->Enable(false);
```

7.6 Diagramas de flujo del botón Calcular y Desplazar





7.7 Diagrama de flujo del botón Interpolar



8 INTEGRACIÓN NUMÉRICA

La integración numérica es una herramienta esencial que se usa en la ciencia y en la ingeniería para obtener valores aproximados de integrales definidas que no pueden calcularse analíticamente. Matemáticamente la integración se representa por:

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

Los métodos de integración numérica se usan cuando f(x) es difícil o imposible de integrar analíticamente o cuando f(x) está dada como un conjunto de valores tabulados. La estrategia acostumbrada para desarrollar fórmulas para la integración numérica consiste en hacer pasar un polinomio por puntos definidos de la función y luego integrar la aproximación polinomial de la función.

Regla Trapezoidal

Geométricamente la regla trapezoidal es equivalente a aproximar el área del trapezoide bajo la línea recta que conecta *f(a)* y *f(b)* en la figura 8.1. Recuerde de geometría que la fórmula para calcular el área de un trapezoide es la altura por el promedio de las bases. En este caso, el concepto es el mismo, pero el trapezoide esta sobre su lado. Por lo tanto la integral se representa como:

$$I \cong ancho \times altura\ promedio$$

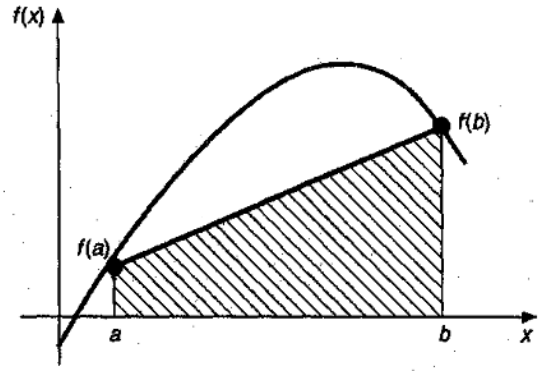


Figura 8.1 Ilustración gráfica de la regla trapezoidal

Reglas de Simpson

Además de aplicar la regla trapezoidal con segmentación más fina, otra forma de obtener una estimación más exacta de una integral es con el uso de polinomios de orden superior para conectar los puntos. Por ejemplo si hay un punto extra a la mitad del camino entre *f(a)* y *f(b),* los tres puntos se pueden conectar con una parábola (véase figura 8.2a) . Si hay dos puntos igualmente espaciados entre *f(a)* y *f(b),* los cuatro puntos se pueden conectar con un polinomio de tercer orden (véase figura 8.2b). Las fórmulas que resultan al tomar las integrales bajo esos polinomios son conocidas como *reglas de Simpson.*

En este programa de integración numérica se han incluido tres métodos de integración que son:

1. Integración Trapezoidal.
2. Simpson 1/3.
3. Simpson 3/8.

El programa se muestra en la figura 8.3. Puede apreciarse que inicialmente los tres métodos están deshabilitados debido a que se habilitarán dependiendo del número de ordenadas de la tabulación. Por otra parte puede notarse

que se cuenta con la opción de insertar una función o puede obtenerse el área bajo la curva de los puntos insertados en la tabla. Para estas opciones se hace uso de los botones circulares WxRadioButton.

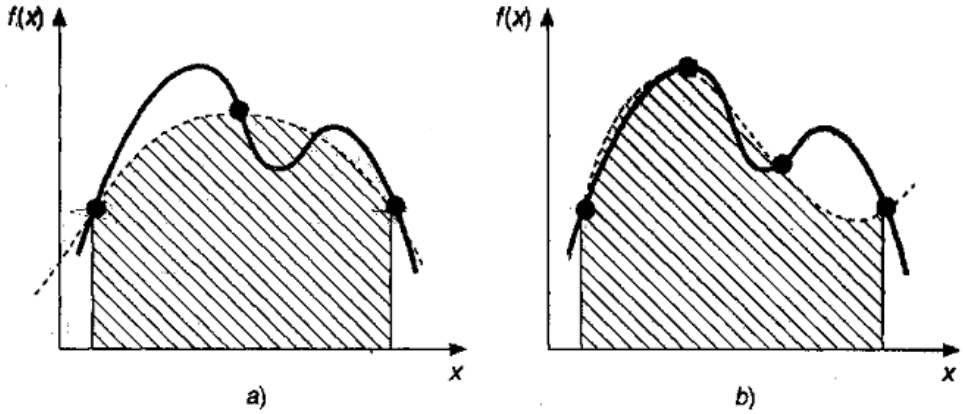


Figura 8.2 Ilustración gráfica de la regla de Simpson 1/3: consiste en tomar el área bajo una parábola que conecta tres puntos. b) Ilustración gráfica de la regla de Simpson 3/8 consiste en tomar el área bajo una ecuación cúbica que conecta cuatro puntos.

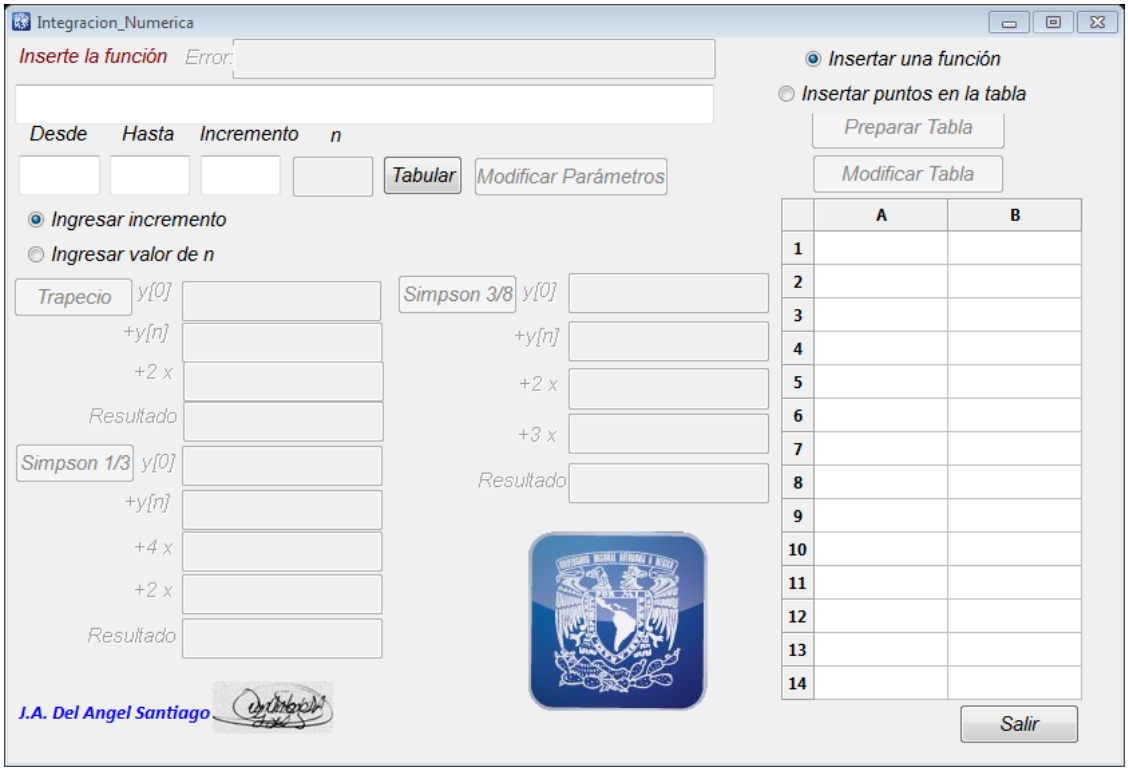


Figura 8.3 Programa de Integración Numérica.

8.1 CONSIDERACIONES DE DISEÑO DEL PROGRAMA.

- Se debe contar con la opción de insertar una función o calcular el área bajo la curva de puntos insertados en la tabla.
- Es muy importante alcanzar el valor del límite superior del intervalo deseado. Se puede notar en el *Analizador de Raíces*, que si el intervalo de tabulación es desde x=0 hasta x=3 con incremento de 0.3, el valor que se alcanza sin salir del intervalo es x=3.9 lo cual es inaceptable en el caso de la integración ya que se debe tomar la ordenada que corresponda al límite superior del intervalo. Se debe contar con dos opciones:
  - **I.** Insertar el valor del incremento. Se debe verificar que se alcanza el límite superior, de lo contrario se notificará al usuario que verifique el incremento.

II. Se puede insertar el valor d **n**, es decir en cuantas partes se dividirá el intervalo con lo que no se tendrá ningún problema ya que el incremento es calculado automáticamente para alcanzar el valor del límite superior.

- Inicialmente los métodos de integración deben estar deshabilitados hasta el momento en que insertan los parámetros para conocer el valor de **n** y se habiliten los métodos disponibles para ese valor.
- En caso de existir errores matemáticos en la tabulación no habilitará ningún método de integración y se notificará al usuario que para el intervalo indicado no es posible realizar la integración.
- Para los tres métodos se deben mostrar en cajas de texto los resultados parciales y el resultado final de la integración para que el usuario pueda corroborar sus cálculos.
- Si se elige la opción de insertar puntos en la tabla, en base al intervalo e incremento o valor de n se debe preparar la tabla colocando los valores de x y habilitando los métodos de integración para que el usuario solo inserte los datos de f(x) y oprima el método que desee.
-

8.2 COMPONENTES USADOS PARA EL PROGRAMA.

- **Cajas de texto donde:**

1. Se inserta la función: FuncionCT.
2. Se inserta el parámetro desde: DesdeCT.
3. Se inserta el parámetro Hasta: HastaCT.
4. Se inserta el parámetro Incremento: IncCT.
5. Se muestra resultado del método Trapezoidal: TansCT.
6. Se muestra resultado Simpson 1/3: S13ansST.
7. Se muestra el resultado de Simpson 3/8: S38ansCT.
8. Se inserta el valor de **n**: NCT.
9. Se muestra el valor de y[0] en trapezoidal: Ty0CT.
10. Se muestra el valor de y[n] en trapezoidal: TynCT.
11. Se muestra la sumatoria del resto de las ordenadas en trapezoidal: TsumaCT.
12. Se muestra el valor de y[0] en Simpson 1/3: S13y0CT.
13. Se muestra el valor de y[n] en Simpson 1/3: S13ynCT.
14. Se muestra la sumatoria de las ordenadas con índice impar de Simpson 1/3: S13siCT.
15. Se muestra la sumatoria de las ordenadas con índice par de Simpson 1/3: S13spCT.
16. Se muestra el valor de y[0] en Simpson 3/8: S38y0CT.
17. Se muestra el valor de y[n] en Simpson 3/8: S38ynCT.
18. Se muestra la sumatoria de las ordenadas con índice múltiplo de 3 en Simpson 3/8: S38s3CT.
19. Se muestra la sumatoria del resto de las ordenadas Simpson 3/8: S38resCT.
20. Se muestra el error de sintaxis: SintaxisCT.

- **Botones**

21. Tabular: TabularB.
22. Trapezoidal: TrapezoidalB.
23. Simpson 1/3: Simpson13B.
24. Simpson 3/8: Simpson38B.
25. Modificar Parámetros: ModParB.
26. Modificar Tabla: ModTabB.
27. Prepara tabla: PreparaB.
28. Salir: SalirB.

- **Tablas:**

29. Insertar puntos o mostrar tabulación: TablaG.

- **Botones Circulares:**

30. Ingresar incremento: IncrementoB.

31. Ingresar valor de n: IngresarnRB.
32. Insertar puntos en la tabla: InsPuntosRB.
33. Insertar una función: InsFuncionRB.

- **Textos Estáticos:**

34. Inserte la función: FuncionST.
35. Desde: Desde.
36. Hasta: HastaST.
37. Incremento: IncST.
38. n: NST.
39. y[0] (trapezoidal): Ty0ST.
40. y[n] (trapezoidal): TynST.
41. +2x (trapezoidal): TsumaST.
42. Resultado (trapezoidal): TansST.
43. y[0] (Simpson 1/3): S13y0ST.
44. y[n] (Simpson 1/3): S13ynST.
45. +4x (Simpson 1/3): S13spST.
46. +2x (Simpson 1/3): S13siST.
47. Resultado (Simpson 1/3): S13ansST.
48. y[0] (Simpson 3/8): S38y0ST.
49. y[n] (Simpson 3/8): S38ynST.
50. +2x (Simpson 3/8): S38s3ST.
51. +3x (Simpson 3/8): S38resST.
52. Resultado (Simpson 3/8): S38ansST.
53. Error: SintaxisST.

- **Diálogos de Mensaje**

54. Inserte una función: FuncionDM.
55. Inserte el parámetro Desde: DesdeDM.
56. Inserta el parámetro Hasta: HastaDM.
57. Inserte el parámetro Incremento: IncDM.
58. No se ha alcanzado el límite superior. Verifique Incremento: LimSupDM.
59. Inserte el valor de n: InsNDM.
60. Error de Sintaxis en la función: SintaxisFDM.
61. Error de Sintaxis en los parámetros: SintaxisPDM.
62. Imposible integrar existen errores matemáticos: ImposibleIntDM.

8.3 Funcionamiento del programa.

En esta sección se describe el funcionamiento de cada botón de integración y se menciona de manera breve el funcionamiento de otros componentes que forman parte del programa.

8.3.1 Funcionamiento del botón Tabular.

La forma en que tabula este botón es exactamente la misma que con la se han hecho las tabulaciones anteriores pero este botón es muy importante debido a que es el encargado de analizar el valor del número de divisiones del intervalo para que en base a ello muestre el o los métodos de integración disponibles. Primero deshabilita los componentes del método de integración Trapezoidal que es el método que está disponible para cualquier valor de n para verificar si se dan las condiciones para mostrarlos. Más adelante se comprueba si se deben deshabilitar o habilitar los otros métodos.

Se debe verificar que opción ha seleccionado el usuario:

I. Ingresar el valor de n.

II. Ingresar incremento.

Los botones circulares funcionan de tal manera que cuando se selecciona una opción se desactiva la otra. Por lo tanto cuando se selecciona Ingresar el valor de n se habilita la caja de texto para ingresar el valor y se desactiva la caja de texto del incremento. Si se selecciona ingresar incremento habilita la caja de texto del incremento y deshabilita la caja de texto del valor de n. Posteriormente como de costumbre se revisa la sintaxis de la función y de los parámetros ingresados además de que se verifica que ninguna caja de texto haya sido omitida.

Una vez revisada la sintaxis se obtienen los valores de los parámetros. Para el valor de n o el incremento se toma en cuenta la opción seleccionada. Si se ha seleccionado ingresar el incremento se calcula el valor de n:

$$n = \frac{Límite\ superior - Límite\ inferior}{Incremento}$$

Este valor se muestra en la caja de texto que corresponde al valor de n. Por lo tanto independientemente de que opción se seleccione el valor de n se obtiene directamente o se calcula, ya que este valor indica que método de integración se habilita.

Si se ha insertado el valor del incremento debe verificarse si se alcanza el límite superior del intervalo como se mencionó en las consideraciones de diseño del programa. Para hacer esto se utiliza una variable de verificación:

$$verif = Límite\ inferior + n * Incremento$$

Analicemos un caso en el que no se alcanza el límite superior con:

$$Límite\ inferior = 0 \quad Límite\ superior = 2 \quad Incremento = 0.3$$

Se calcula n:

$$n = \frac{2 - 0}{0.3} = 6.6667$$

Y se muestra en la caja de texto correspondiente el valor de n redondeado que en este caso es:

$$n = 7$$

El valor de la caja de texto se toma nuevamente y se calcula el valor de la variable de verificación:

$$verif = 0 + 7 * 0.3 = 0 + 2.1 = 2.1$$

Donde se puede apreciar que no corresponde el límite insertado por el usuario. Posteriormente se calcula la diferencia entre la variable `verif` y el Límite superior del intervalo:

$$dif = verif - Límite\ superior = 2.1 - 2 = 0.1$$

Si `dif` es diferente de cero se muestra el mensaje de error al usuario indicando que se debe verificar el incremento y la ejecución del código del botón termina. De lo contrario se procede a realizar la tabulación de la función. Inicialmente la tabulación se utilizó para que una vez que se contaba con los valores de f(x) tomarlos de la tabla y emplearlos en los métodos de integración pero surge el mismo problema que en los programas de interpolación en cuanto a la perdida de dígitos decimales por lo que los cálculos no se realizan con los valores de la tabla, en lugar de esto se almacenan los valores de f(x) en un arreglo con todos sus decimales como se explicará más adelante. De tal manera que la tabla se conservó para que el usuario pueda comprobar los resultados obtenidos sin tener que tabular la función ya que la tabulación se muestra al usuario.

Después de realizar la tabulación se analiza el valor de n. Si el valor n es par, se muestran los componentes del método de Simpson 1/3, de lo contrario se ocultan. Si el valor de n es múltiplo de 3 se muestran los componentes del

método Simpson 3/8. Los componentes del método Trapezoidal se muestran siempre y cuando se alcance el valor del límite superior.

8.3.2 Funcionamiento del botón Trapezoidal

Él código de este botón requiere de los parámetros **Desde**, **Hasta** y el valor de **n**, con los que calcula el valor del incremento. Se limpian las cajas de texto que muestran resultados del método y posteriormente el código de este botón se divide en dos partes. La primera parte se ejecuta en caso de que se haya seleccionado la opción insertar una función, la segunda parte se ejecuta en caso de que se haya seleccionado la opción de insertar puntos en la tabla.

8.3.2.1 Expresión la regla Trapezoidal.

$$\int_{x_0}^{x_n} F(x)dx = \frac{h}{2}\left[y_0 + y_n + 2\sum_{i=1}^{n-1} y_i\right] + e_r$$

Una vez que se ha pulsado el botón Tabular, en la caja de texto NCT está contenido el valor de n. **h** es valor del parámetro incremento en este caso, que se calcula como:

$$h = incremento = \frac{límite\ superior - límite\ inferior}{n}$$

8.3.2.2 Funcionamiento cuando se ha insertado una función.

Primero mediante un ciclo **for** se evalúa la función en todos los puntos como en la tabulación pero en esta ocasión no se muestran los resultados solo se almacenan en un arreglo como se muestra a continuación:

```
for(i=0;i<=n;i=i++)
{
       evaluadorExpresiones.ValorVariable('x', dsd);
       valor = evaluadorExpresiones.Calcular();
      y[i]=valor;
      //Se incrementa dsd para el siguiente valor de x
      dsd=dsd+inc;

}
```

Éste código almacena en el arreglo **`double`** `y[n]` los valores de f(x) de todos los puntos mostrados en la tabla y se obtiene la sumatoria de los elementos sin incluir y[0], y[n]:

```
n1=n-1;
for(i=1;i<=n1;i++)
{
       suma=suma+y[i];
}
```

Y se calcula el valor de la integral:

$$inte = \frac{1}{2} * inc * (y_0 + y_n + 2 * suma);$$

Después se convierten a cadenas de caracteres los valores de inte, y[0], y[n], suma para mostrarlos en las cajas de textos correspondientes de tal manera que el usuario pueda comprobar sus cálculos.

8.3.2.3 Funcionamiento si se han insertado puntos en la tabla.

Si se han insertado puntos en la tabla se debe aplicar el algoritmo de revisión de sintaxis de tablas explicado en capítulos anteriores para revisar la sintaxis de los valores de f(x) insertados por el usuario. Después se obtienen los valores de f(x) y se almacenan en el arreglo y[n] en lugar de ser calculados como en el caso de insertar una función.

Una vez que se han almacenado los valores en el arreglo se procede exactamente de la misma manera que en la sección anterior para obtener el resultado.

8.3.3 Funcionamiento del botón Simpson 1/3

El código de este botón funciona de manera semejante al botón Trapezoidal. Al principio se limpian los componentes que muestran los resultados del método y se obtienen los valores de los parámetros **Desde**, **Hasta** y el valor de **n**, con los que calcula el valor del incremento. Ahora que se ha visto como se calcula la integral cuando se insertan los puntos en la tabla solo se explica la primera parte en la que se inserta la función.

8.3.3.1 Expresión de la regla Simpson 1/3.

$$\int_{x_0}^{x_n} F(x)dx = \frac{h}{3}\left[y_0 + y_n + 4\sum ordenadas\ con\ índice\ par + 2\sum ord.con\ indice\ impar\right] + e_r$$

8.3.3.2 Funcionamiento si se ha insertado una función.

Se almacenan los valores de f(x) en el arreglo y[n] y se obtiene la sumatoria de las ordenadas con índice impar:

```
n1=n-1;
for(i=1;i<=n1;i=i+2)
{

      suma=suma+y[i];
}
```

En la primera línea se observa la estructura del ciclo for que obtiene la sumatoria. Inicia desde y[1] y solo toma los valores impares sumando 2 al índice en cada iteración sin tomar el valor de y[n]. Después se obtiene la sumatoria de las ordenadas con índice par:

```
for(i=2;i<=n1;i=i+2)
{
    suma1=suma1+y[i];
}
```

En la estructura del ciclo se observa que se comienza en y[2] que es la primera ordenada par sin tomar el valor de y[n]. Una vez que se cuenta con las sumatoria se aplica la expresión del método:

$$inte = (inc/3) * (y[0] + y[n] + 4 * suma + 2 * suma1);$$

Solo resta convertir a cadenas de caracteres y mostrar los resultados al usuario.

8.3.4 Expresión de la regla de Simpson 3/8

$$\int_{x_0}^{x_n} F(x)dx = \frac{3h}{8}\left[y_0 + y_n + 2\sum ordenadas\ con\ índice\ múltiplo\ de\ 3 + 3\sum resto\ de\ las\ ordenadas\right] + e_r$$

8.3.4.1 Funcionamiento cuando se inserta una función.

Después de obtener los valores de f(x) y almacenarlos en el arreglo y[n] se obtiene la sumatoria de las ordenadas con índice múltiplo de 3:

```
n1=n-1;
for(i=3;i<=n1;i=i+3)
{
    suma=suma+y[i];
}
```

En la primera línea se observa que se toman los valores empezando desde y[3] y sumando 3 al índice en cada iteración se toman las ordenadas con índice múltiplo de 3. Después se obtiene la sumatoria del resto de las ordenadas:

```
suma1=0;
veri=0;
for(i=1;i<=n1;i++)
{
    veri=veri+1;
    if(veri!=3)
    {
        suma1=suma1+y[i];
    }
    if(veri==3) veri=0;
}
```

En éste código la variable `veri` se inicia en cero y en cada iteración de i se incrementa en uno. Mientras `veri` sea distinta de 3 se realiza la sumatoria por lo que se toman todos los valores restantes de esta manera.

8.3.5 Funcionamiento de IncrementoRB e IngresarnRB

Estos botones circulares se encargan de habilitar las cajas de texto para insertar el valor del incremento o el valor de n respectivamente y deshabilitar la caja que no esté en uso según la opción seleccionada.

8.3.6 Funcionamiento de InsFuncionRB

Este botón básicamente se encarga de restaurar las condiciones predeterminadas del programa, es decir al pulsar este botón el programa toma el mismo aspecto que al ser abierto, habilitando la caja de texto donde se inserta la función

8.3.7 Funcionamiento de InsPuntosRB.

Este botón deshabilita la caja de texto donde se inserta la función y limpia todas las cajas de texto mostrando los componentes necesarios para preparar la tabla donde se el usuario insertara los valores de f(x).

8.3.8 Funcionamiento del botón Modificar Parámetros

Este botón limpia y deshabilita los componentes de todos los métodos. Al modificar los parámetros se modifica el valor de n y por consiguiente los métodos de integración disponibles pueden cambiar. Funciona de la misma manera si se insertan puntos en la tabla o se inserta una función.

8.3.9 Funcionamiento del botón Preparar tabla.

Este botón se encarga de analizar el valor de n de la misma manera que el botón tabular para mostrar los métodos de integración disponibles. Además en base al intervalo y el incremento se colocan en la tabla los valores de x para que el usuario solo inserte los valores de f(x).

8.4 DIAGRAMAS DE FLUJO.

En esta sección se muestran los diagramas de flujo de los botones Tabular, Trapezoidal, Simpson 1/3 y Simpson 3/8. En el diagrama del botón Trapezoidal se muestra detalladamente como se obtienen los valores de los puntos y se realiza la revisión de sintaxis cuando se insertan valores en la tabla. En los diagramas de Simpson 2/3 y 3/8 solo se indica que se toman los puntos insertados por el usuario.

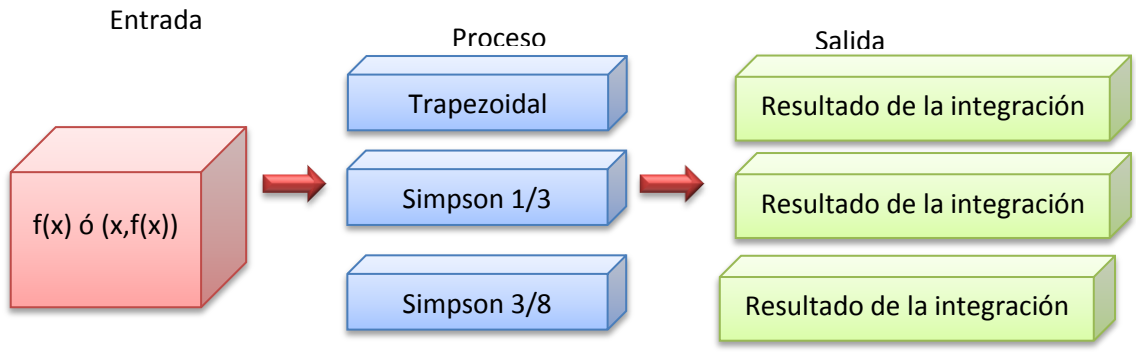
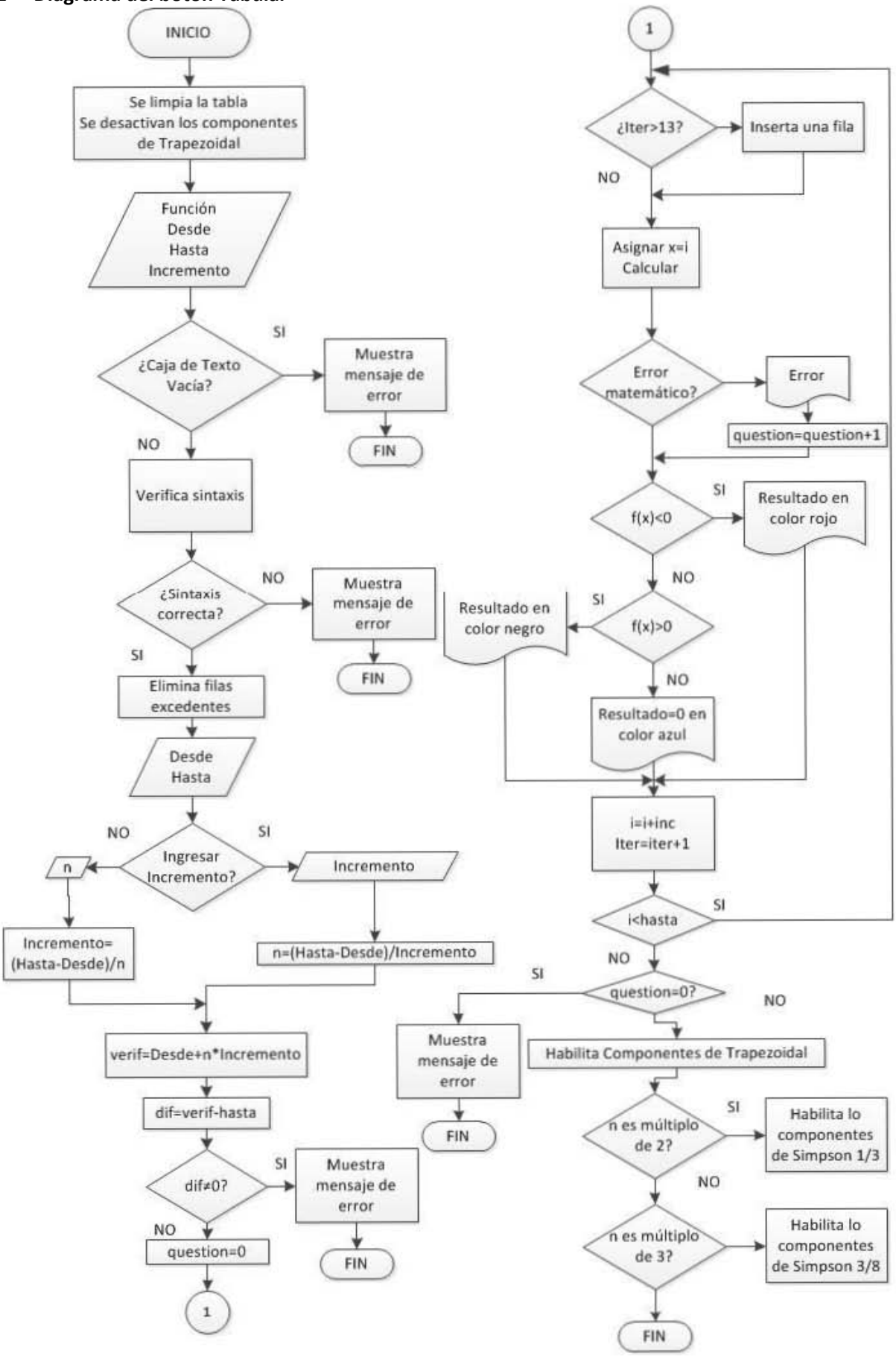
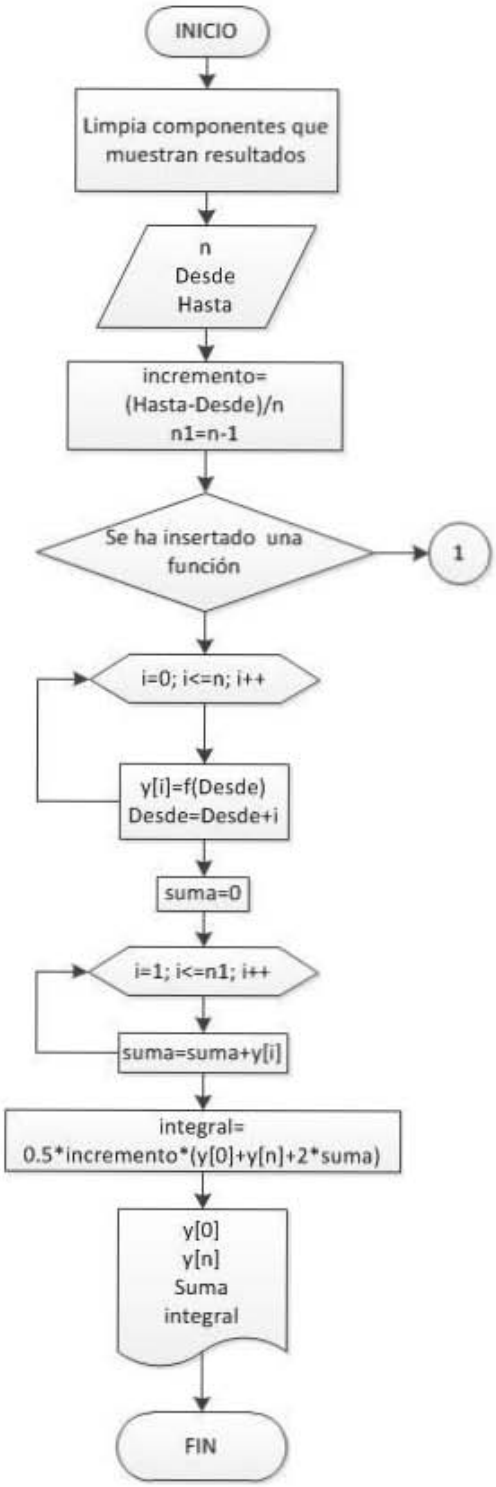


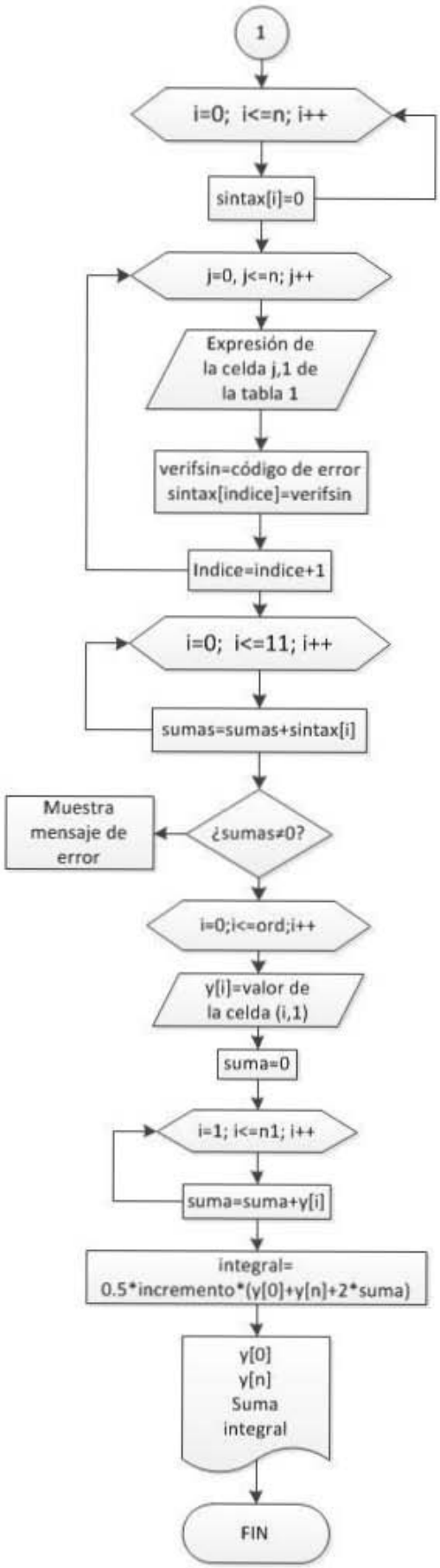
Figura 8.4 Diagrama de entrada, proceso, salida

8.4.1 Diagrama del botón Tabular

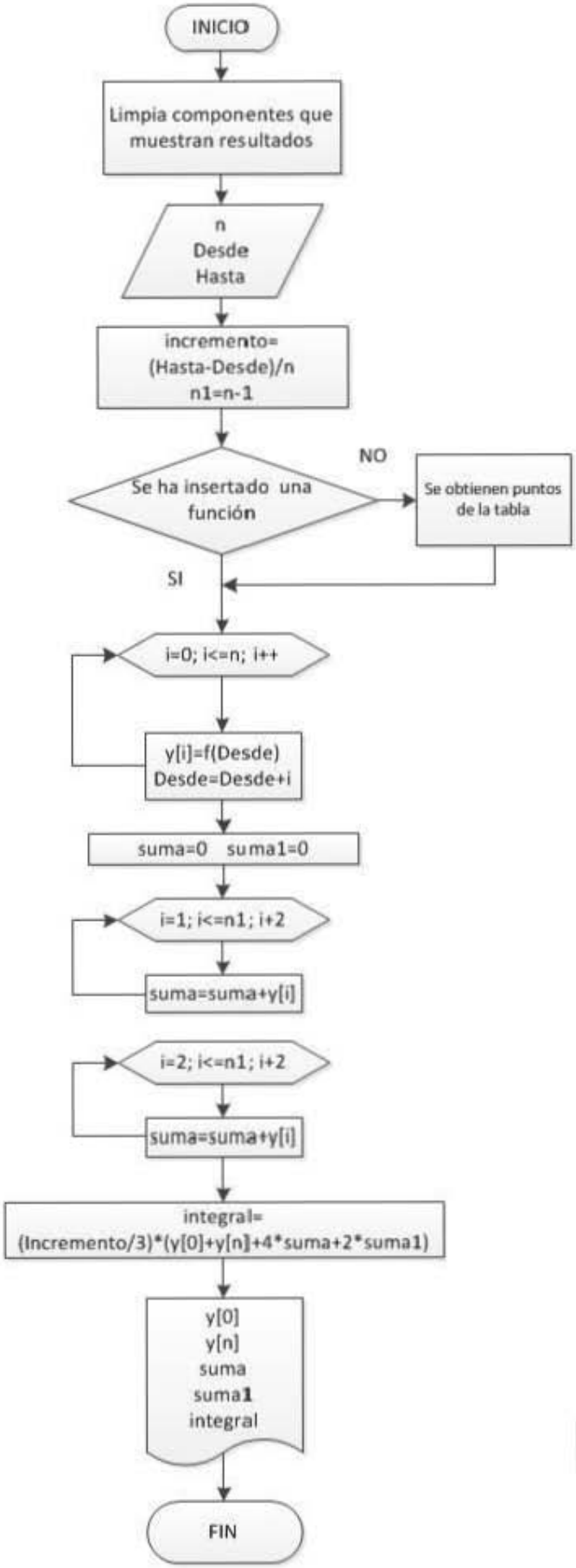


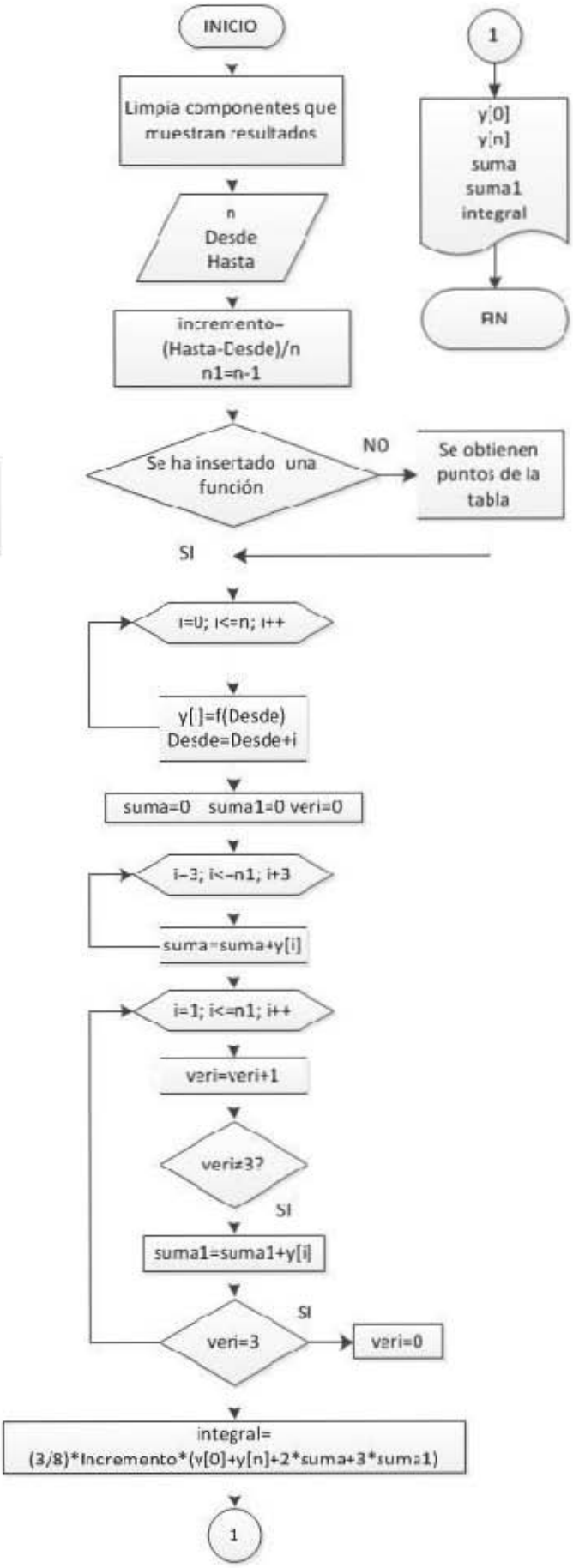
8.4.2 Diagrama del botón Trapezoidal





8.4.3 Diagrama de Simpson 1/3 y Simpson 3/8





9. RUNGE KUTTA DE CUARTO ORDEN

El estudio de los procesos dinámicos y sus sistemas de control, debe iniciarse con la obtención de una representación matemática de las relaciones existentes entre las diferentes variables involucradas en el proceso a controlar, a la que usualmente se denomina modelo del sistema. El proceso de modelado de un sistema dinámico, puede llevar a la obtención de una representación para el mismo por medio de una ecuación diferencial de orden alto, o por un conjunto de ecuaciones diferenciales de primer orden no lineales, cuya solución se debe obtener para conocer la respuesta temporal del sistema, a partir un conjunto de condiciones iniciales y una entrada dada. La solución analítica de una ecuación diferencial lineal puede ser fácil pero en ocasiones no se podrán resolver por métodos analíticos.

Éste programa sirve para resolver ecuaciones diferenciales de primer orden con condiciones iniciales. El más popular de los métodos de Runge Kutta es el de cuarto orden, ésta es la forma más común y por tanto se le conoce como *método de Runge Kutta clásico de cuarto orden*. Este programa se muestra en la figura 8.1.

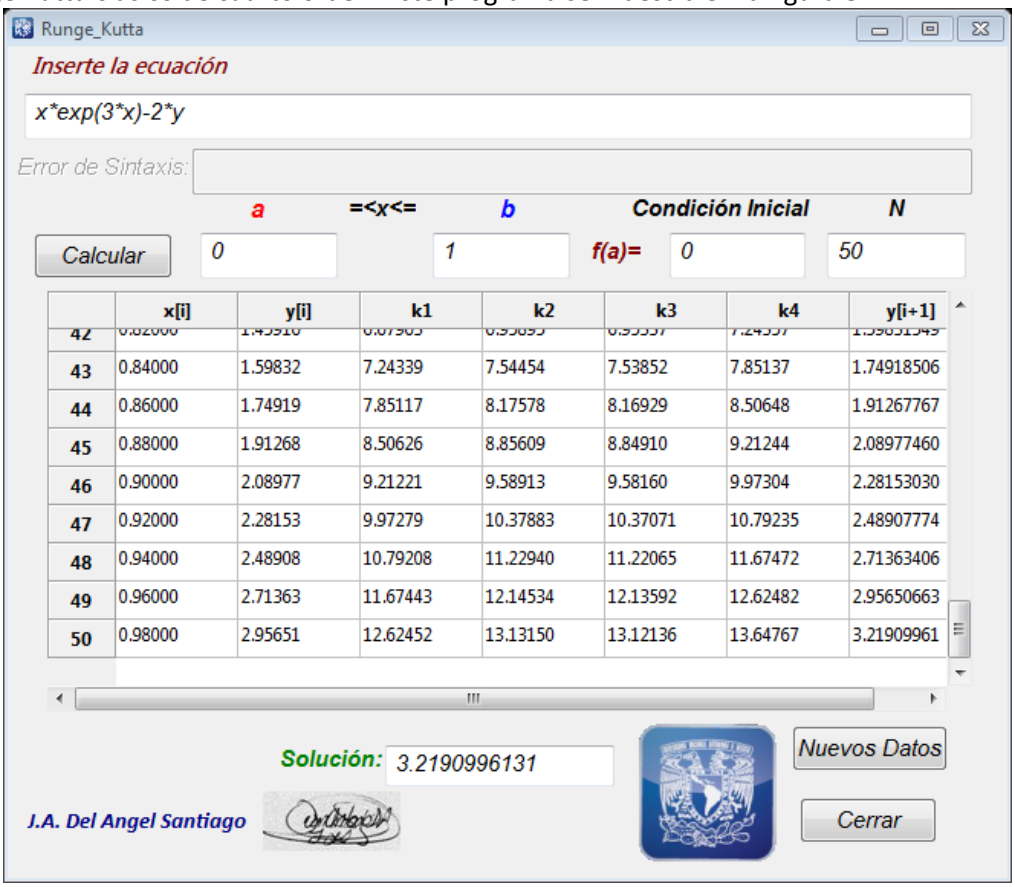


Figura 9.1 Programa de Runge Kutta de cuarto orden

9.1 CONSIDERACIONES DE DISEÑO DEL PROGRAMA.

- Las literales que se utilizan para las ecuaciones insertadas son (x) y (y).
- En una tabla se deben mostrar los resultados de cada una de las iteraciones.
- El resultado obtenido se mostrara en la última fila de la columna y[i+1]. Además, el resultado también se mostrará en una caja de texto para que se pueda visualizar rápidamente por el usuario.
- Se debe contar con una caja de texto que muestre errores de sintaxis en la función en caso de haberlos.
- De la misma manera que en los programas anteriores se debe llevar a cabo la revisión de sintaxis de todos los parámetros insertado.
- Se debe contar con un botón que limpie todos los componentes, restaurando así las condiciones predeterminadas del programa.

9.2 COMPONENTES USADOS PARA EL PROGRAMA.

- **Cajas de Texto donde:**

1. Se inserta la función: EcuacionCT.
2. Se inserta el límite inferior del intervalo (a): AvalorCT.
3. Se inserta el límite superior del intervalo (b): BvalorCT.
4. Se inserta el valor de N: NCT.
5. Se inserta la condición inicial: IcsvalorCT.
6. Se muestra el error de sintaxis: SintaxisCT.
7. Se muestra el resultado obtenido: ResultCT.

- **Botones.**

8. Calcular: CalcularB.
9. Nuevos Datos: NewDataB.
10. Cerrar: CerrarB.

- **Textos Estáticos**

11. =<x<=: IntervaloST.
12. a: AvalorST.
13. b: BvalorST.
14. f(a)= : FdeaST.
15. Condición Inicial: IcsvalorST.
16. N: NvalorST.
17. Inserte la ecuación: EcuacionST.
18. Error de Sintaxis: SintaxisST.
19. Solución: ResultST.

- **Tablas.**

20. Tabla de resultados: TablaG.

- **Diálogos de Mensaje.**

21. Inserte una ecuación: EcuacionDM.
22. Inserte el parámetro a: AvalorDM.
23. Inserte el parámetro b: BvalorDM.
24. Inserte la condición Inicial: IcsvalorDM.
25. Inserte el parámetro N: NvalorDM.
26. Error de Sintaxis en la función: SintaxisEDM.
27. Error de sintaxis en los parámetros: SintaxisPDM.

9.3 Expresiones del método.

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

$$k_1 = hf(x_i, y_i) - - - (2)$$

$$k_2 = hf\left(x_i + \frac{h}{2}, y_i + \frac{k_1}{2}\right) - - - (3)$$

$$k_3 = hf\left(x_i + \frac{h}{2}, y_i + \frac{k_2}{2}\right) - - - (4)$$

$$k_4 = hf(x_i + h, y_i + k_3) - - - (5)$$

9.4 Funcionamiento del botón calcular.

El algoritmo consiste en los siguientes pasos:

1. Se limpia la tabla, la caja de texto que muestra el resultado, y se deshabilitan los componentes que muestran error de sintaxis ya que solo se muestran en caso de que hay error.

2. Se revisa la sintaxis de la función y los parámetros además de que se verifica que ninguna caja de texto haya sido omitida.

3. Se eliminan las celdas excedentes de 10, para que cada que se pulse el botón se agreguen solo las celdas necesarias.

4. Se obtiene los valores de xa, xb y N.

5. Se calcula el valor de h:
$$h = \frac{x_b - x_a}{N}$$
6. Se almacenan en el arreglo x todos los valores que tomara x.

7. Se obtiene el valor de y[0] que es el primer elemento del arreglo y.

8. Es necesario almacenar los valores de x[i] y y[i] (donde i es el número de iteración) en valores temporales debido a que para cada valor de k los valores sustituidos en la ecuación son distintos, esto con el objetivo de hacer más claro el código aunque no es obligatorio:
$$x[i] = m \quad y[i] = n$$
9. Se evalúa: $k_1 = f(m, n)$

10. Se calcula:
$$m = x[i] + \frac{h}{2.0} \qquad n = y[i] + \frac{hk_1}{2.0}$$
11. Se evalúa: $k_2 = f(m, n)$

12. Se calcula:
$$m = x[i] + \frac{h}{2.0} \quad n = y[i] + \frac{hk_2}{2.0}$$
13. Se evalúa: $k_3 = f(m, n)$

14. Se calcula:
$$m = x[i] + h \quad n = y[i] + hk_3$$
15. Se evalúa: $k_3 = f(m, n)$

16. Se calcula el siguiente valor de y:
$$y[i+1] = y[i] + (h/6.0 * (k1 + 2.0 * k2 + 2.0 * k3 + k4))$$
17. Se realiza la siguiente iteración hasta el último valor del arreglo x.

18. Después de acabar las iteraciones se convierte a cadena de caracteres y se muestra en una caja de texto el resultado obtenido.

El botón Nuevos Datos se encarga de limpiar todos los componentes del programa.

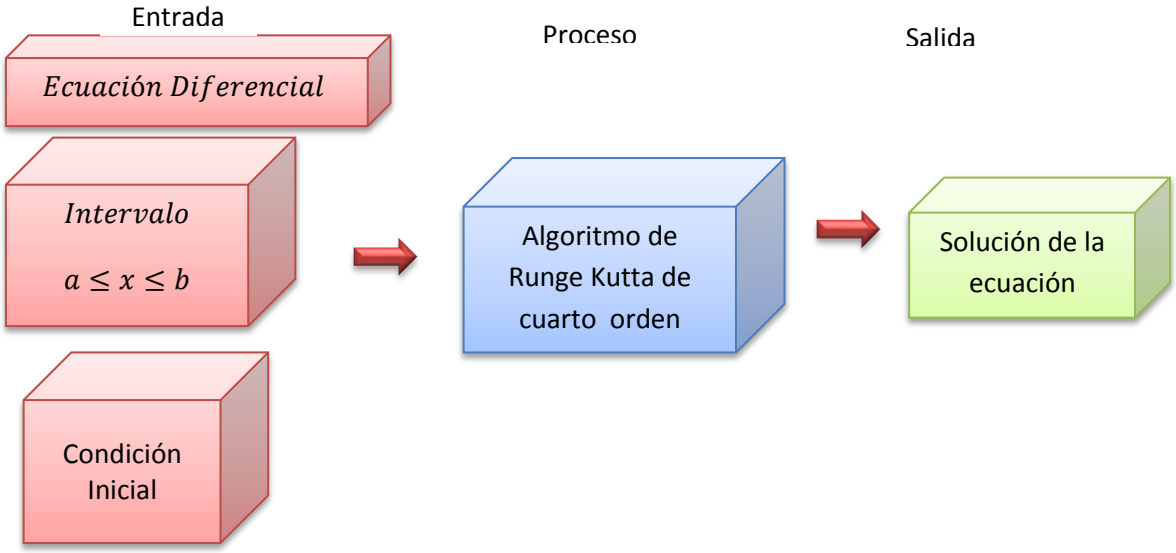
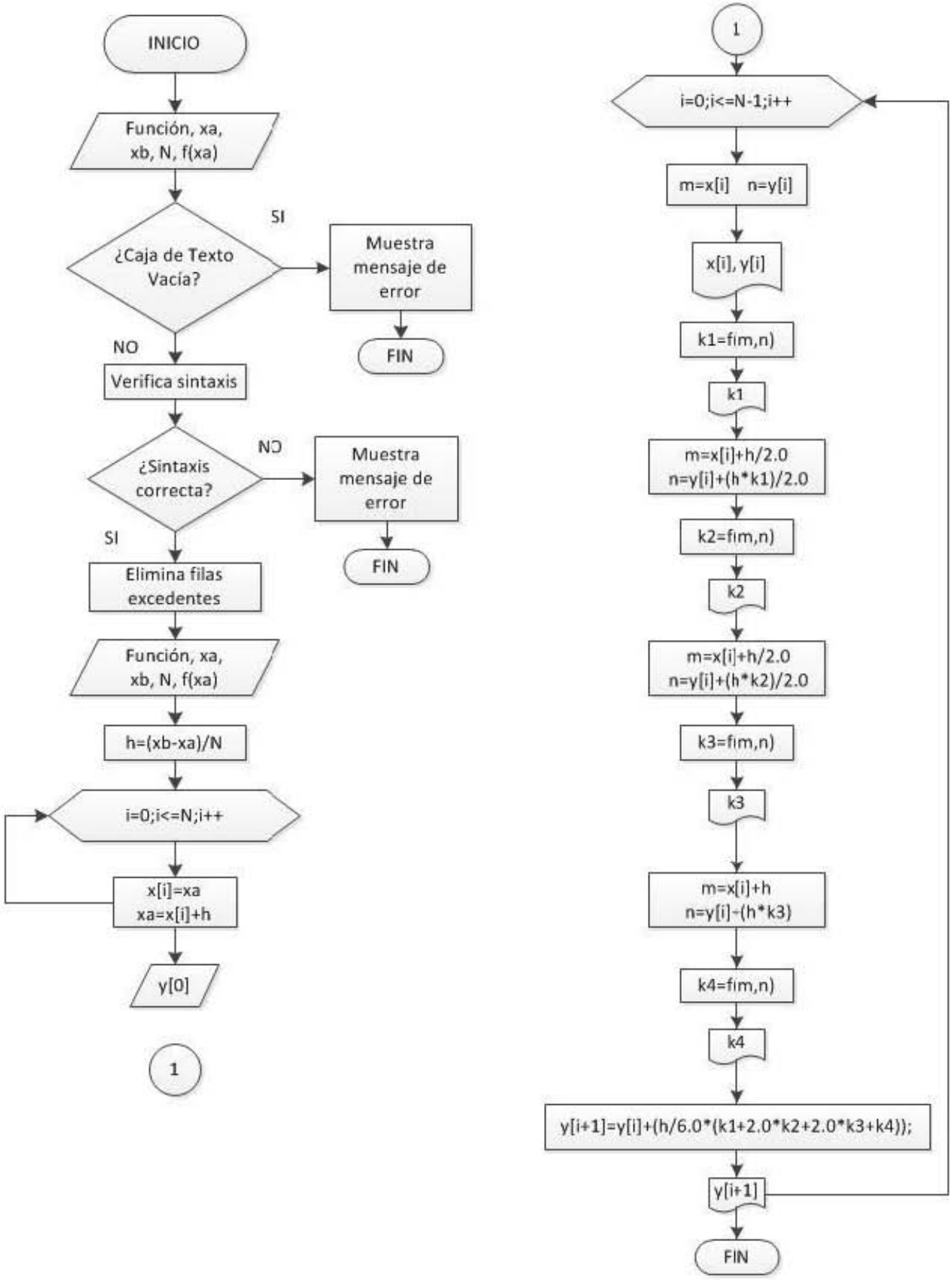


Figura 9.2 Diagrama deEntrada, Proceso Salida

9.5 Diagrama de flujo del botón Calcular



10. INSTRUCCIONES DE USO DE LOS PROGRAMAS.

En este capítulo se explica detalladamente como hacer uso de los programas. Se presentarán algunos casos en los que se aplique cada programa con el fin de detallar el uso de cada programa. Las expresiones que maneja el evaluador de expresiones y los operadores se muestran en la tabla 10.1. Además el valor exacto de $\boldsymbol{\pi}$ se inserta como $\mathbf{2} * \boldsymbol{asn}(\mathbf{1})$ y el valor de $\boldsymbol{e}$ como $\mathbf{exp}(\mathbf{1})$. **Se deberá anteponer un cero las expresiones que comience como $-\boldsymbol{x}^{\boldsymbol{2n}}$ como $\boldsymbol{sen}(\mathbf{0}-\boldsymbol{x}^{\mathbf{2}}) = \boldsymbol{sen}(-\boldsymbol{x}^{\mathbf{2}})\,, \boldsymbol{cos}(\mathbf{0}-\boldsymbol{x}^{\mathbf{4}}+\mathbf{0.1}) = \boldsymbol{cos}(-\boldsymbol{x}^{\mathbf{4}}+\mathbf{0.1}),\ \mathbf{0}-\boldsymbol{x}^{\mathbf{6}}+\boldsymbol{x} = -\boldsymbol{x}^{\mathbf{6}}+\boldsymbol{x}$.**

Función	Sintaxis	Ejemplo	Operación	Operador
seno	sen ó sin	sen(x)	Adición	+
coseno	cos	cos(x)	Sustracción	-
tangente	tan	tan(x)	Multiplicación	*
asn	arco seno	asn(x)	División	/
acs	arco coseno	acs(x)	Exponente	^
atn	arco tangente	atn(x)		
log. Natutal	log	log(x)		
Redondeo	cei	cei(x)		
exponencial	exp	exp(x)		
raiz cuadrada	sqr	sqr(x)		
raiz cúbica	rcb	rcb(x)		

Tabla 10.1 Expresiones y operadores soportadoñs por el evaluador de expresiones.

10.1 Bisección

Es muy importante insertar la variable independiente con la literal *x* ya que ha sido programado de esa manera. Considere la función $f(x) = x^3 + 2\pi x^2 + 0.1435x^2 + 0.1435\pi x - 1.1435\pi$ que tiene tres raíces reales. Se inserta un intervalo de -100 a 100 con incremento de 1/2 y se pulsa Tabular. Se ha cometido un error de sintaxis intencionalmente asn ya que se ha colocado ans y además se ha insertado 1//2 en lugar de 1/2. Primero se muestra el diálogo de mensaje que indica error de sintaxis en la función y posteriormente se muestra el error de sintaxis cometido además del mensaje de error en los parámetros como se observa en la figura 10.1

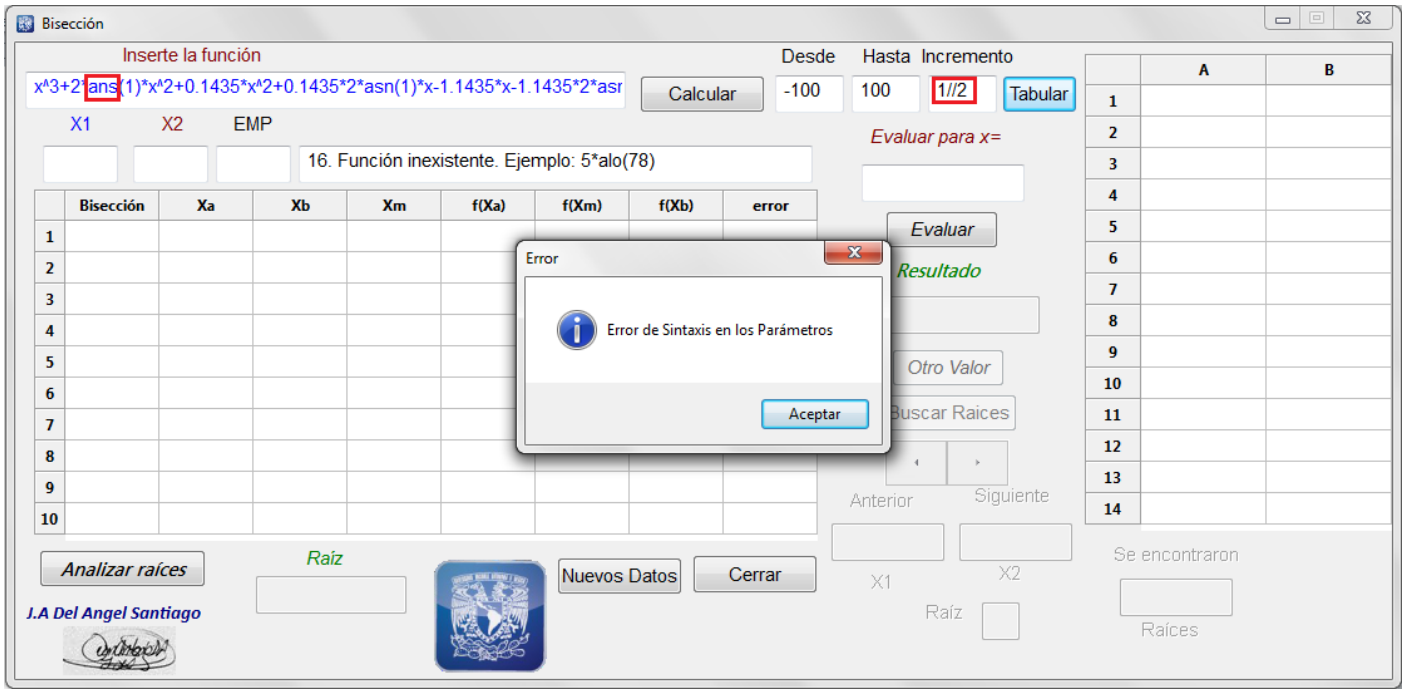


Figura 10.1 Mensaje de error de sintaxis

Después de corregir los errores se muestra la tabulación, se pulsa el botón buscar raíces y se indica que se han encontrado tres raíces, pero aún no se muestra ninguna. Para mostrar cada una de las raíces se pulsa en el botón

siguiente, y pulsando Anterior se muestra la raíz previamente mostrada. Pulsando tres veces Siguiente se obtienen los resultados de la figura 10.2. Si se muestra la primera raíz y se pulsa anterior se muestra un mensaje que informa esta situación al usuario. Si se muestra la última raíz y se pulsa Siguiente también se muestra un mensaje de información. Estos mensajes se observan en la figura 10.3

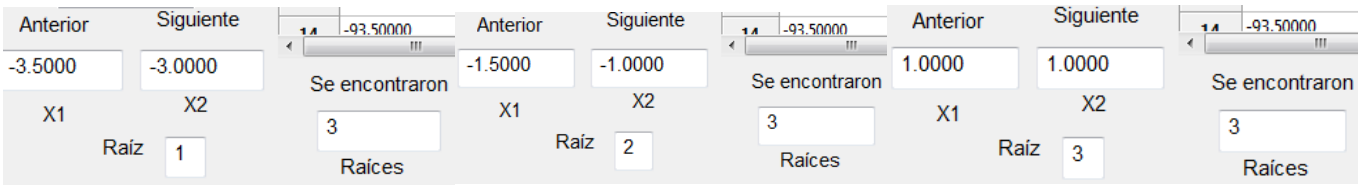


Figura 10.2 Intervalos de las raíces

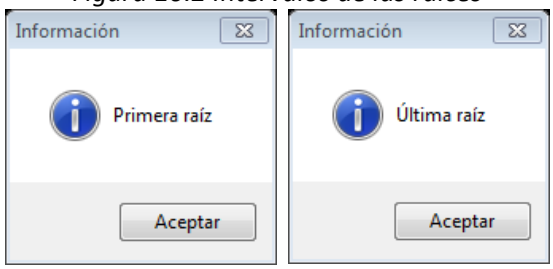


Figura 10.3 Mensajes de información

Los intervalos son $[-3{,}5, -3], [-1.5, -1], [1\ ,1]$. Se puede notar que el tercer intervalo tiene los mismos límites, esto ha sido programado con la intención de indicar al usuario que la raíz es exacta por lo que:

$x_3 = 1$.

Ahora se inserta el primer intervalo con un error mínimo permitido de$EMP = 1 \times 10^{-11}$ y se pulsa Calcular. La raíz encontrada se copia y se pega en la caja de texto bajo la etiqueta *Evaluar para x=* y se pulsa Evaluar para observar el resultado al sustituir el valor de la raíz en la función como se observa en la Figura 10.4.

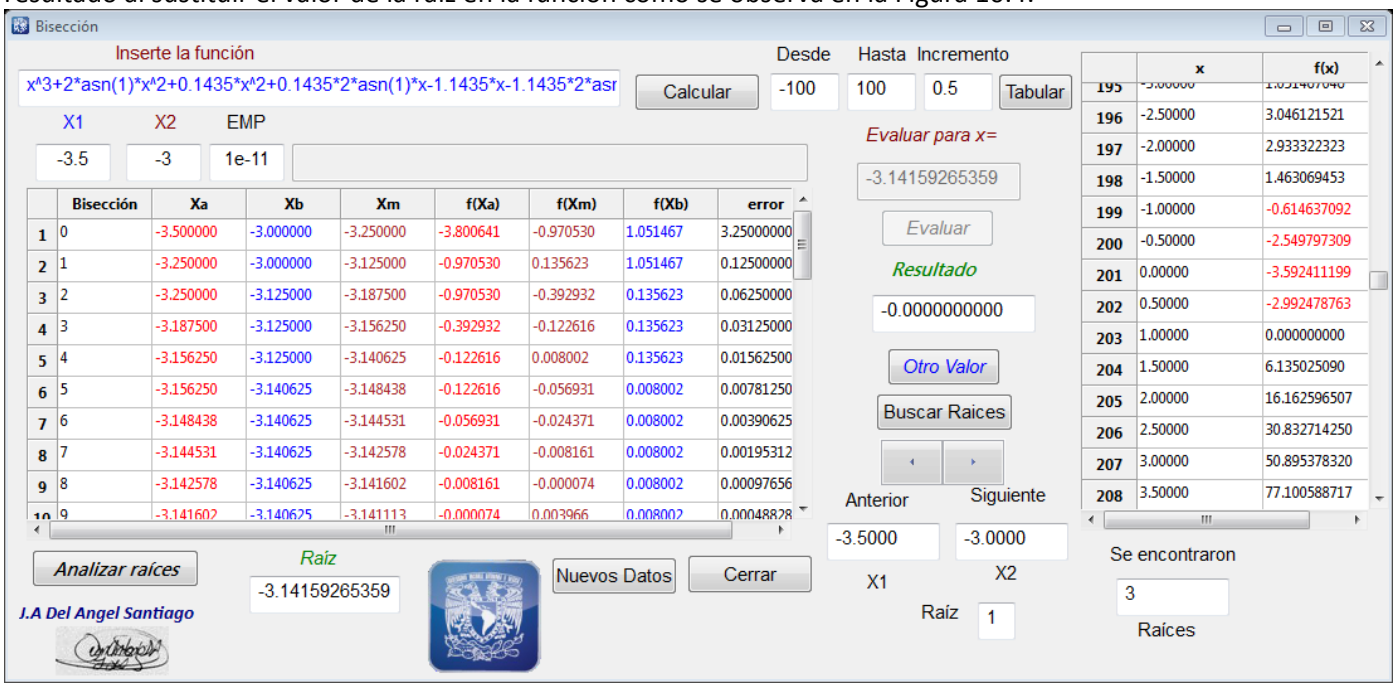


Figura 10.4 Cálculo de la primera raíz

Se puede observar que el valor de la primera raíz es $x_1 = -3.141592 \approx -\pi$. Se inserta el segundo intervalo y se pulsa calcular como se observa en la figura 10.5.

La segunda raíz es $x_2 = -1.1435$. El valor de tolerancia que se ha insertado de tal manera que se obtenga una buena aproximación de la raíz, pero se pueden insertar diferentes valores de EMP para observar más aproximaciones de la raíz. Se puede notar que la revisión de sintaxis es muy importante en todos los programas y no se permite la ejecución del código de los botones en caso de existir errores de sintaxis o se haya omitido el llenado de alguna caja de texto correspondiente a los parámetros. *En todas las cajas de texto se pueden insertar expresiones soportadas por*

el evaluador excepto en la que corresponde a EMP ya que será un número de la forma 1×10^{-n}. Existe un punto importante que debe ser considerado cuando se inserta el intervalo de la raíz debido a que si el intervalo es simétrico por ejemplo [-1,1] se genera una división sobre cero y no se calcula la raíz. Este problema se puede resolver insertando el intervalo como [-1,1.01] de tal manera que no ocurra división sobre cero.

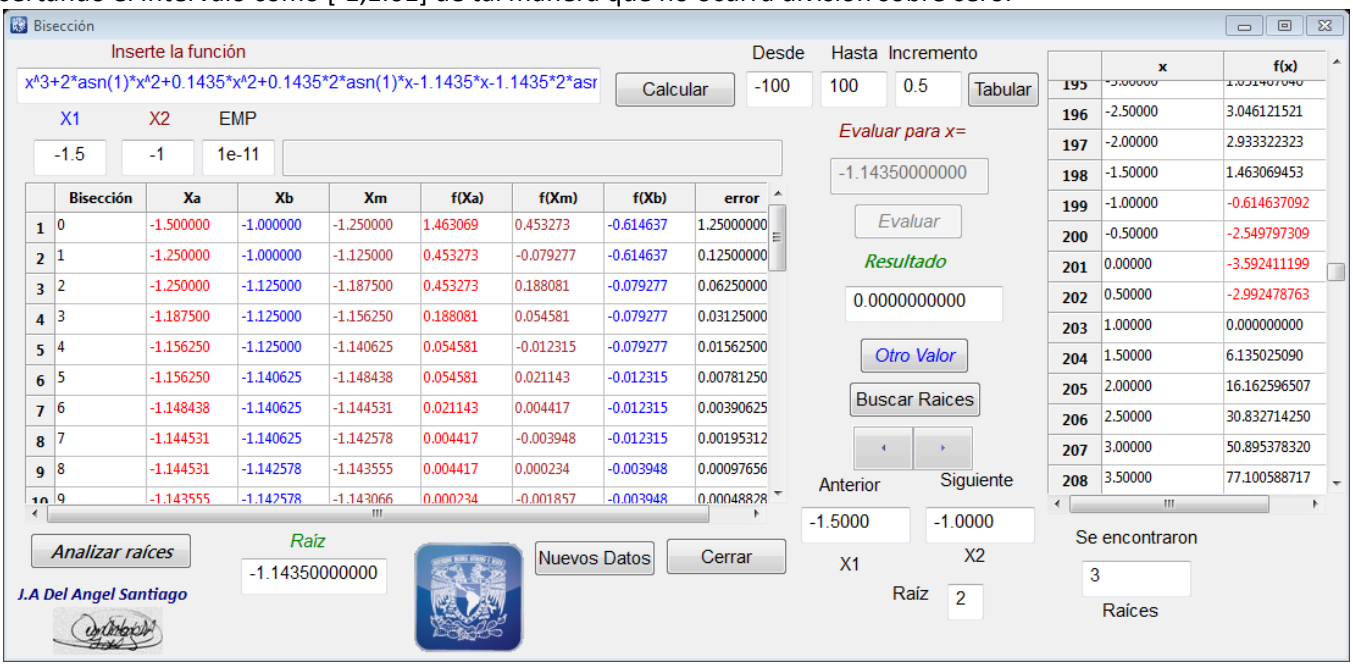


Figura 10.5 Cálculo de la segunda raíz

10.2 NEWTON RAPHSON

Considere la función $f(x) = \frac{xsen(x)}{x+3}$ para la que se encontrarán las raíces en el intervalo [2, 10]. Primero se inserta la función y se tabula de 2 hasta 10 con el incremento deseado, en este caso con un incremento de 1. En la figura 10.6 se observan los intervalos en los que se encuentran las raíces. Debe notarse que en este caso no aparece el botón buscar raíces debido a que no se insertan más filas a la tabla. En base a los colores en que aparece f(x) el usuario puede determinar los intervalos.

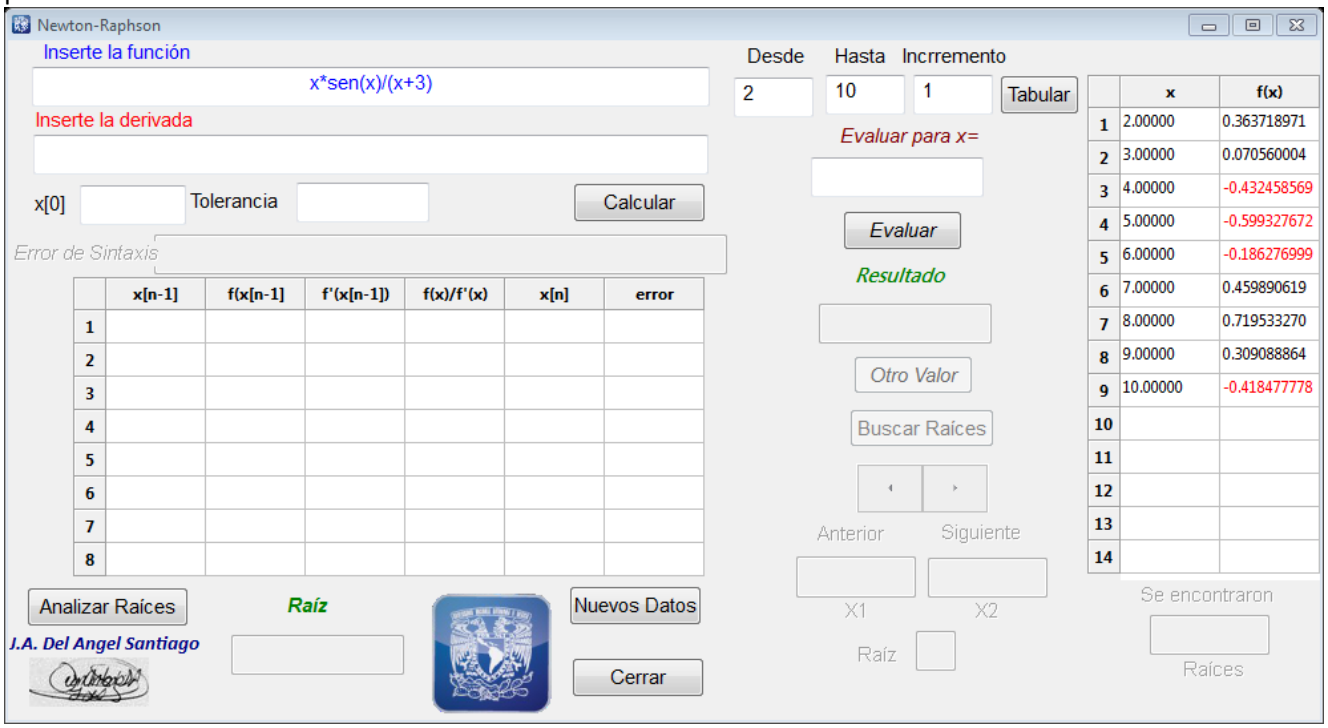


Figura 10.6 Intervalos de las raíces

Los intervalos son [3, 4], [6, 7] y [9,10]. Es conveniente insertar en x[0] el valor de la mitad del intervalo y se puede insertar la operación en la caja de texto como se observa en la figura 10.7 La derivada puede insertarse sin llevar a cabo simplificaciones, en este caso se insertara como $f'(x) = \frac{(x+3)(xcos(x)+sen(x))-xsen(x)}{(x+3)^2}$. Por otra parte se omitirá el llenado de la caja de texto que corresponde a la tolerancia intencionalmente para observar el mensaje de Información. En todos los programas se mostrarán mensajes semejantes cuando se omite una caja de texto.

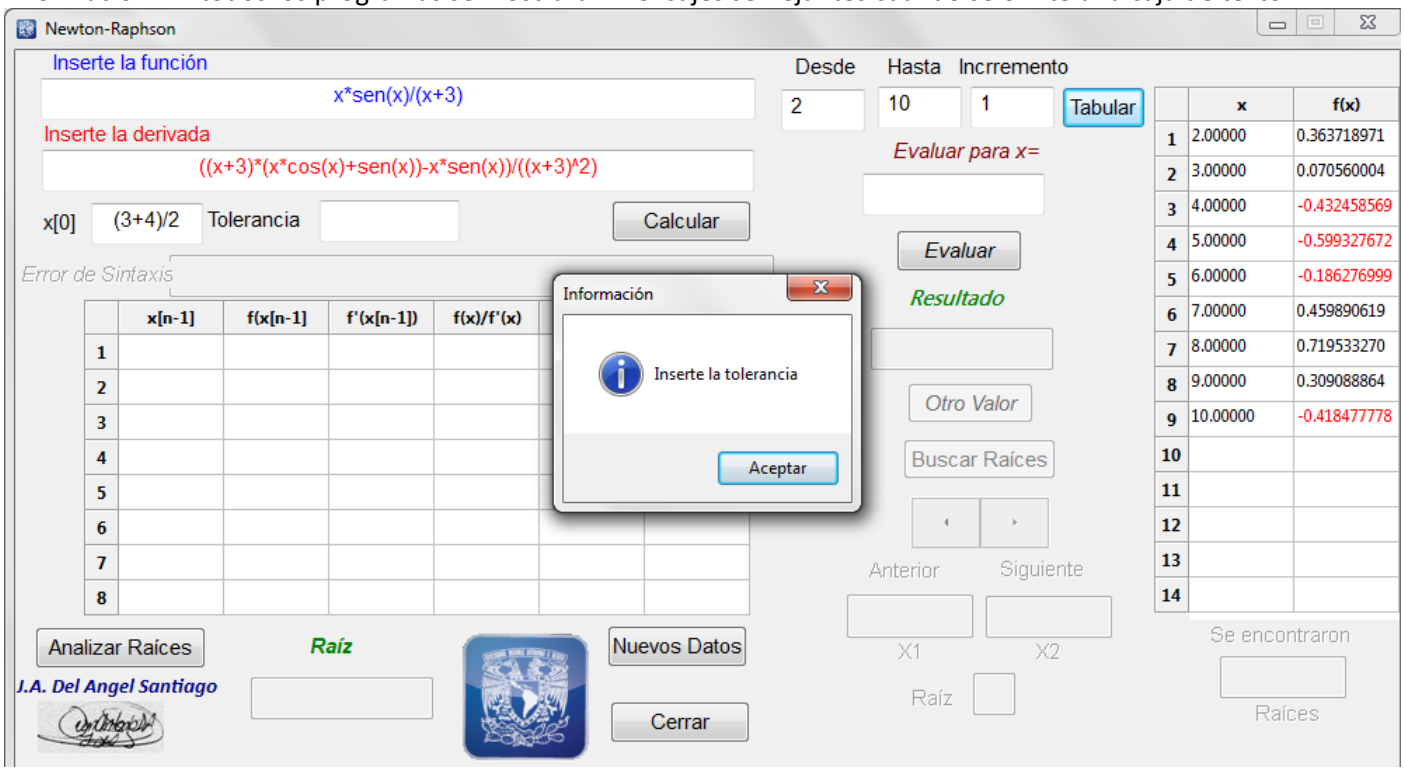


Figura 10.7 Mensaje de información al omitir la tolerancia

A continuación se inserta una tolerancia de $1x10^{-6}$ y se obtiene la raíz mostrada en la figura 10.8

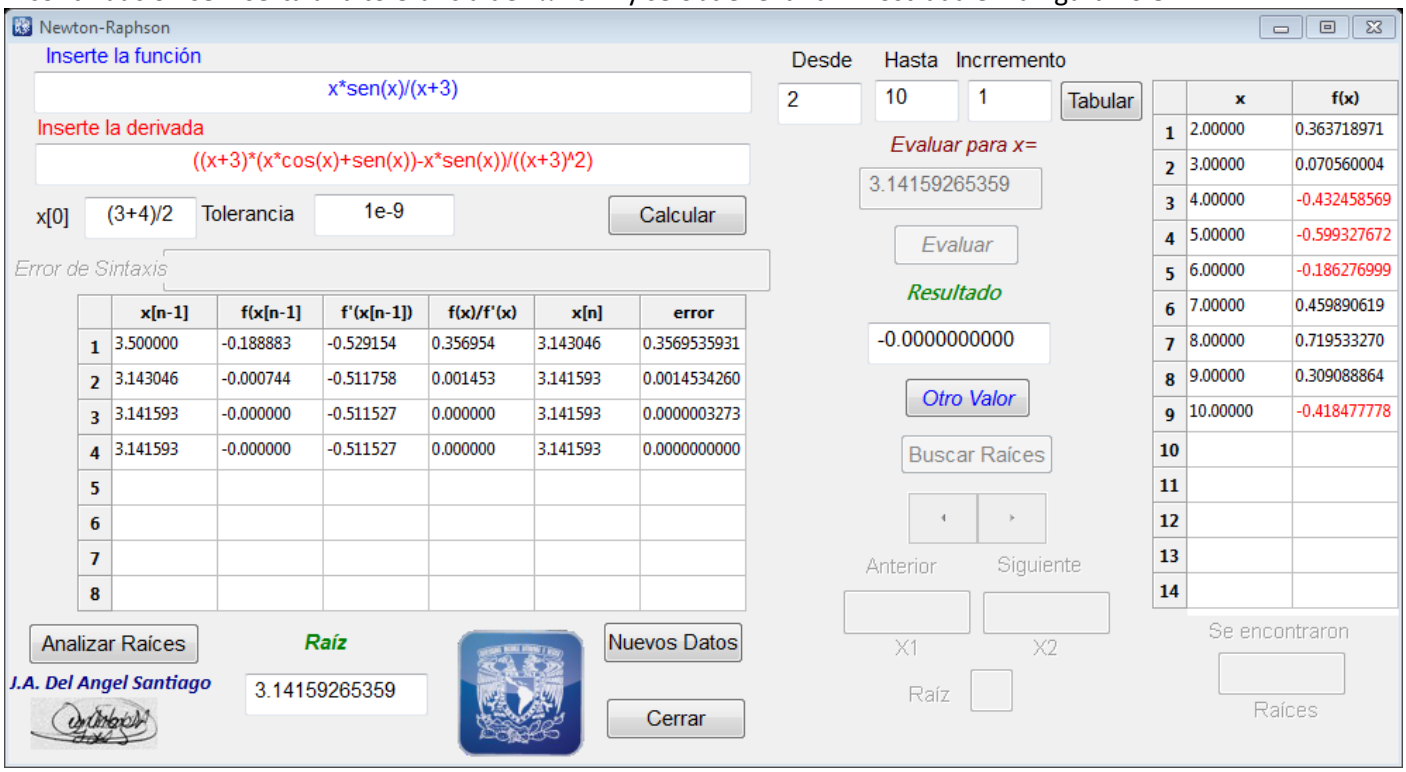
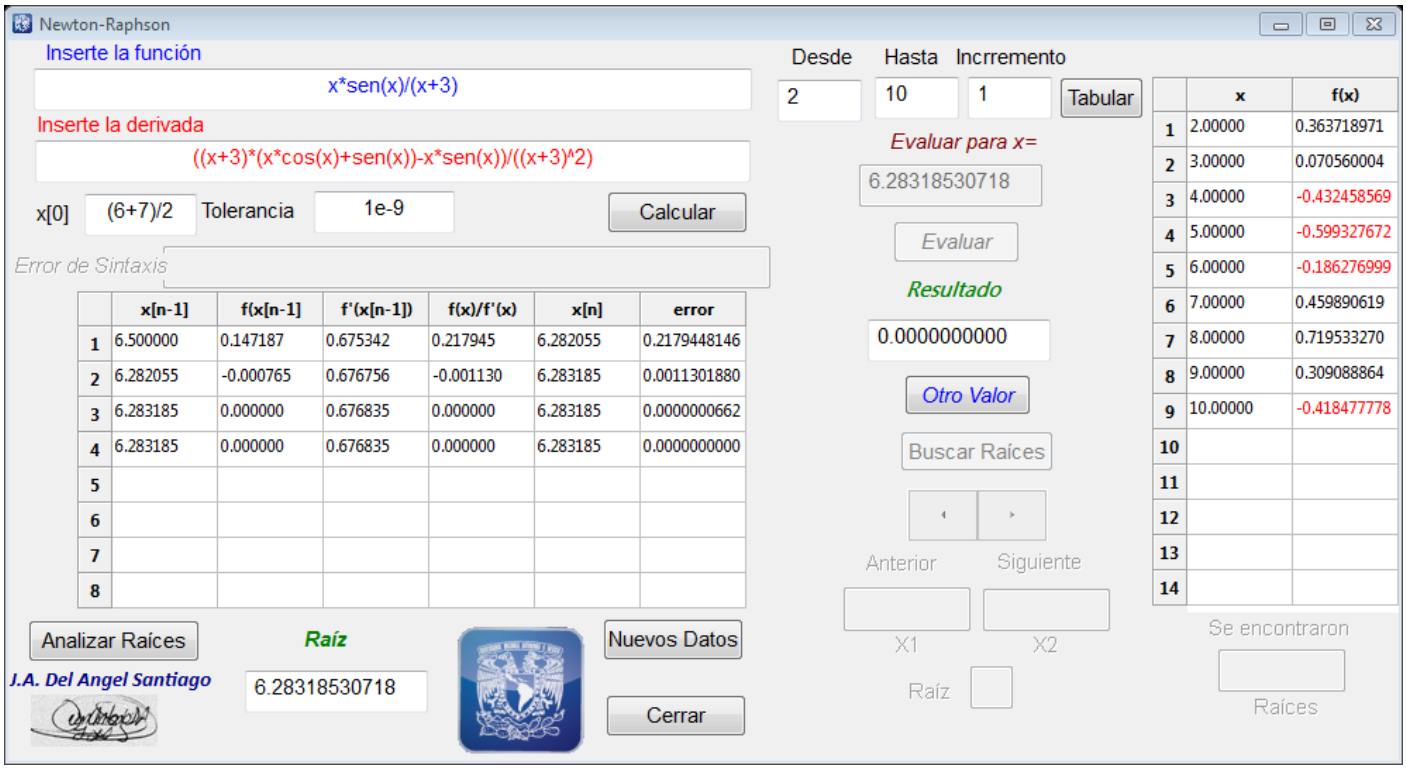


Figura 10.8 Cálculo de la primera raíz

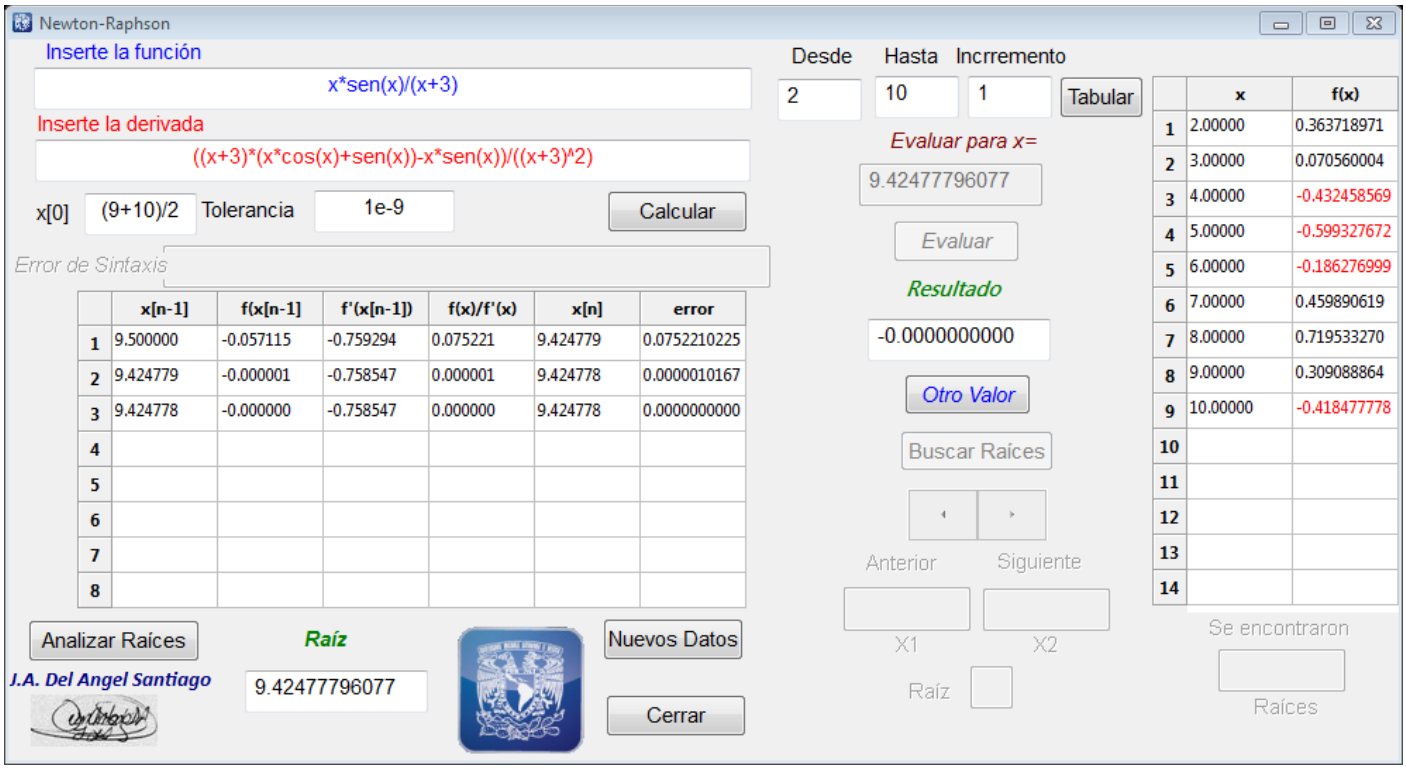
Se obtiene el valor de $x_1 = 3.141592 \approx \pi$. De la misma manera se calculan las raíces 2 y 3 que se muestran en las figuras 10.9 y 10.10.



	x[n-1]	f(x[n-1]	f'(x[n-1])	f(x)/f'(x)	x[n]	error
1	6.500000	0.147187	0.675342	0.217945	6.282055	0.2179448146
2	6.282055	-0.000765	0.676756	-0.001130	6.283185	0.0011301880
3	6.283185	0.000000	0.676835	0.000000	6.283185	0.0000000662
4	6.283185	0.000000	0.676835	0.000000	6.283185	0.0000000000
5						
6						
7						
8						

	x	f(x)
1	2.00000	0.363718971
2	3.00000	0.070560004
3	4.00000	-0.432458569
4	5.00000	-0.599327672
5	6.00000	-0.186276999
6	7.00000	0.459890619
7	8.00000	0.719533270
8	9.00000	0.309088864
9	10.00000	-0.418477778
10		
11		
12		
13		
14		

Figura 10.9 Cálculo de la segunda raíz.



	x[n-1]	f(x[n-1]	f'(x[n-1])	f(x)/f'(x)	x[n]	error
1	9.500000	-0.057115	-0.759294	0.075221	9.424779	0.0752210225
2	9.424779	-0.000001	-0.758547	0.000001	9.424778	0.0000010167
3	9.424778	-0.000000	-0.758547	0.000000	9.424778	0.0000000000
4						
5						
6						
7						
8						

	x	f(x)
1	2.00000	0.363718971
2	3.00000	0.070560004
3	4.00000	-0.432458569
4	5.00000	-0.599327672
5	6.00000	-0.186276999
6	7.00000	0.459890619
7	8.00000	0.719533270
8	9.00000	0.309088864
9	10.00000	-0.418477778
10		
11		
12		
13		
14		

Figura 10.10 Cálculo de la tercera raíz

Por lo tanto $x_2 = 6.283185 \approx 2\pi$ y $x_3 = 9.424777 \approx 3\pi$

10.3 Descomposición LU.

Considere el sistema e ecuaciones:

$$\begin{aligned} x_1 \quad + x_2 + x_3 + x_4 &= 1 \\ 0x_1 \quad + x_2 \; + \; x_3 + 2x_4 &= 1 \\ 0x_1 + 3x_2 + 3x_3 + 12x_4 &= 4 \\ 0x_1 + 0x_2 + 3x_3 + 6x_4 &= 2 \end{aligned}$$

Se selecciona un orden de 4x4 para insertar los coeficientes y los términos independientes en las tablas correspondientes y se pulsa calcular. En la figura 10.11 se observa que surge una indeterminación.

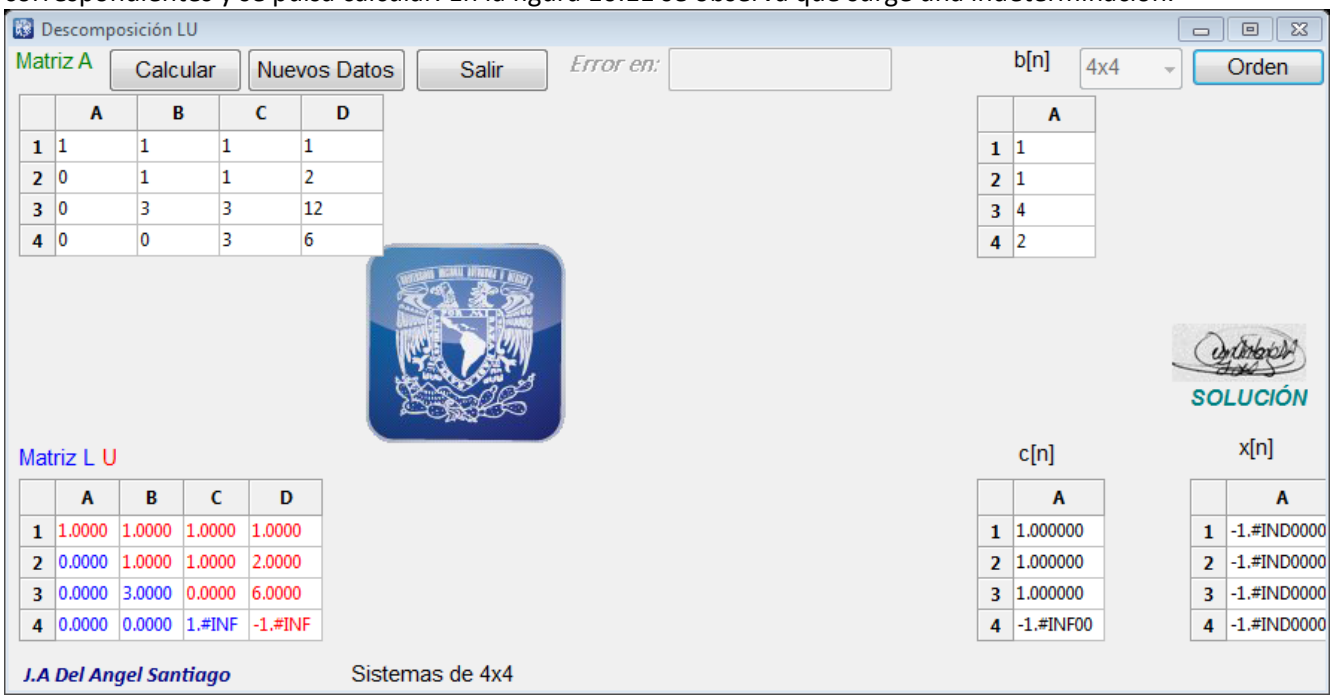


Figura 10.11 Indeterminación en el programa de LU

Para evitar la indeterminación en estos casos se pueden intercambiar renglones. Para este sistema se intercambian las filas 3 y 4. Intencionalmente se cometerán errores de sintaxis y se pulsará calcular después de intercambiar las filas.

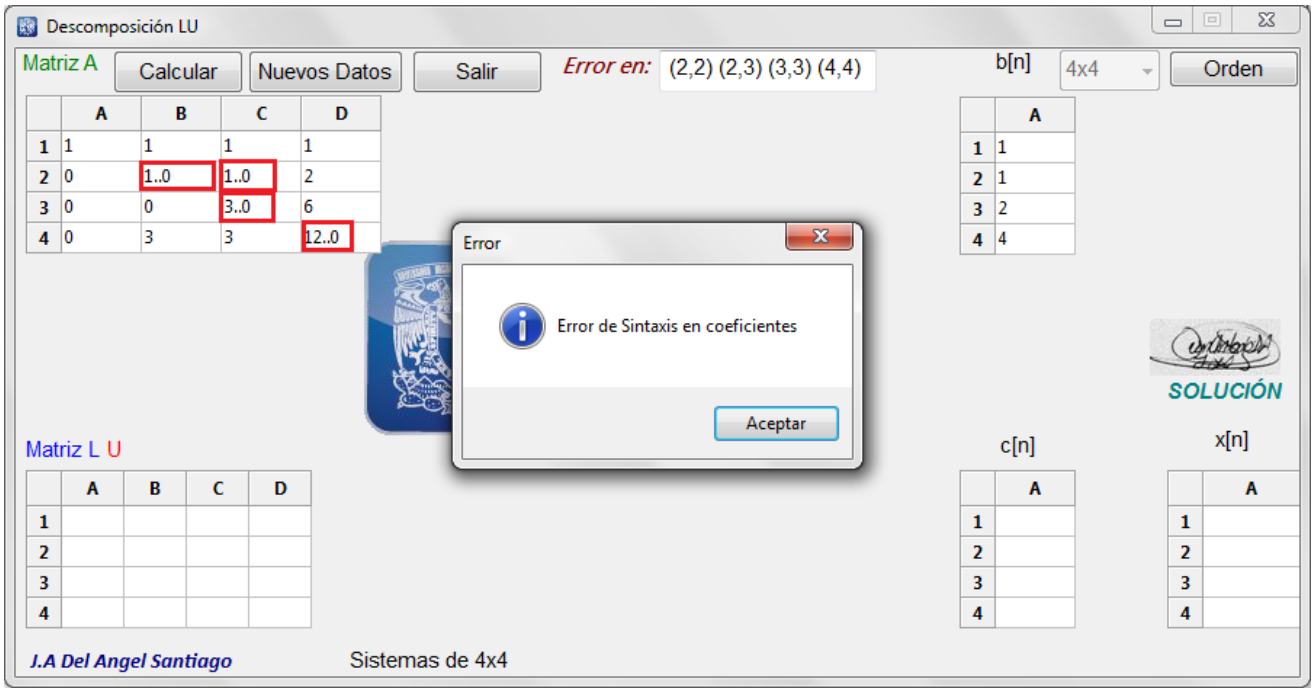


Figura 10.12 Errores de sintaxis en los coeficientes

Después de corregir los errores se pulsa calcular y se obtiene la solución del sistema que se muestra en la figura 10.13.

Figura 10.13 Solución del sistema de ecuaciones

La solución del sistema es $x_1 = x_4 = 0.1667 = \frac{1}{6}$ $x_2 = x_3 = 0.3333 = \frac{1}{3}$. En caso de cometer errores de sintaxis en los términos independientes se indicará con un número entre paréntesis el término en que se ha cometido error y se mostrara el mensaje de error como se muestra en la figura 10.14.

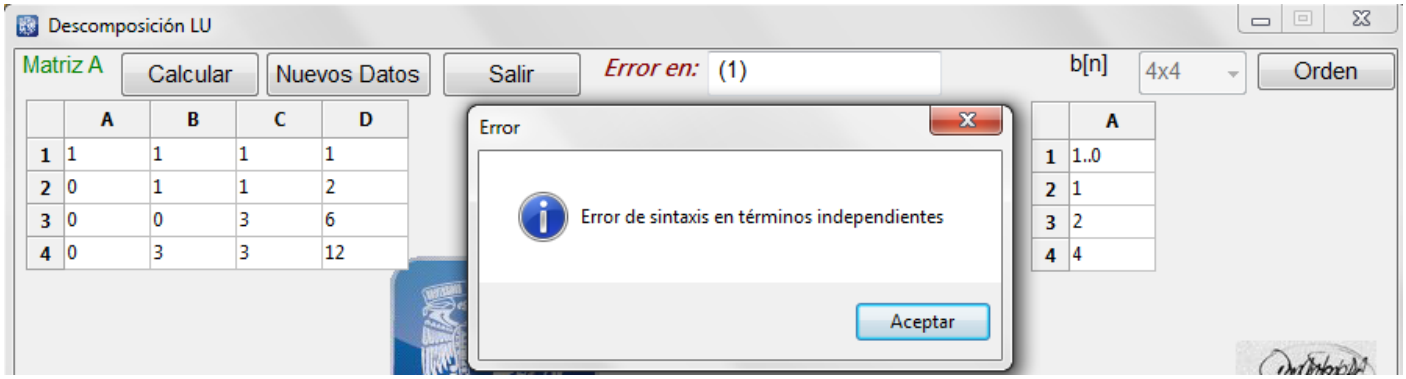


Figura 10.14 Error de sintaxis en términos independientes

10.4 Interpolación de Lagrange

Considere los puntos de la tabla 10.2 para los que se debe de obtener el polinomio de interpolación.

i	0	1	2	3
f(xi)	-3	0	5	7
xi	1/10	sen(1.5)	3	6

Tabla 10.2

Se deben seleccionar 4 puntos en la lista de selección para este caso como se muestra en la figura 10.15. A continuación se insertan los puntos y se pulsa el botón calcular, además se seleccionan 6 dígitos decimales y se interpola para x=5, esto se observa en la figura 10.16.

Figura 10.15 Selección de número de puntos

Figura 10.16 Obtención del polinomio de interpolación

Para comprobar que el polinomio pasa por los puntos insertado se realiza una tabulación pulsando el botón Tabular polinomio y se inserta un intervalo desde -3 hasta 7 con un incremento de uno como se aprecia en la figura 10.17.

10.5 Interpolación de Newton

Considere los puntos de la tabla 10.3.

i	1	2	3	4	5	6	7	8
f(xi)	sen(1)	sen(2)	sen(3)	sen(4)	sen(5)	sen(6)	sen(7)	sen(8)
xi	1	2	3	4	5	6	7	8

Tabla 10.3

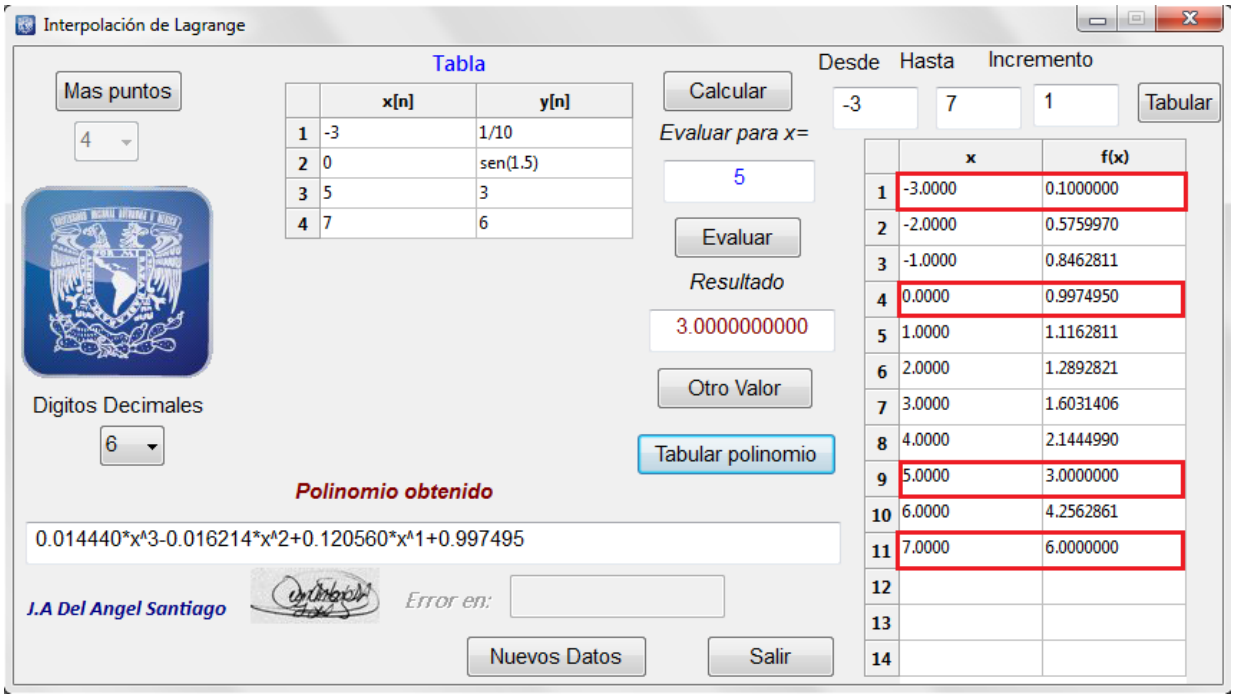


Figura 10. 17 Tabulación del polinomio obtenido.

Se debe obtener un polinomio que se ajuste a todos los puntos y otro que se ajuste a los puntos i= 5,6,7,8 e interpolar en ambos casos para x=6.5. Debe notarse que en este caso no se selecciona un número de dígitos decimales en especial por lo que se muestran 4 por defecto.

Debido a que se insertaran 8 puntos se selecciona ese valor de la lista de selección, se insertan los puntos y se pulsa diferencias, la tabla de diferencias se observa en la figura 10.18.

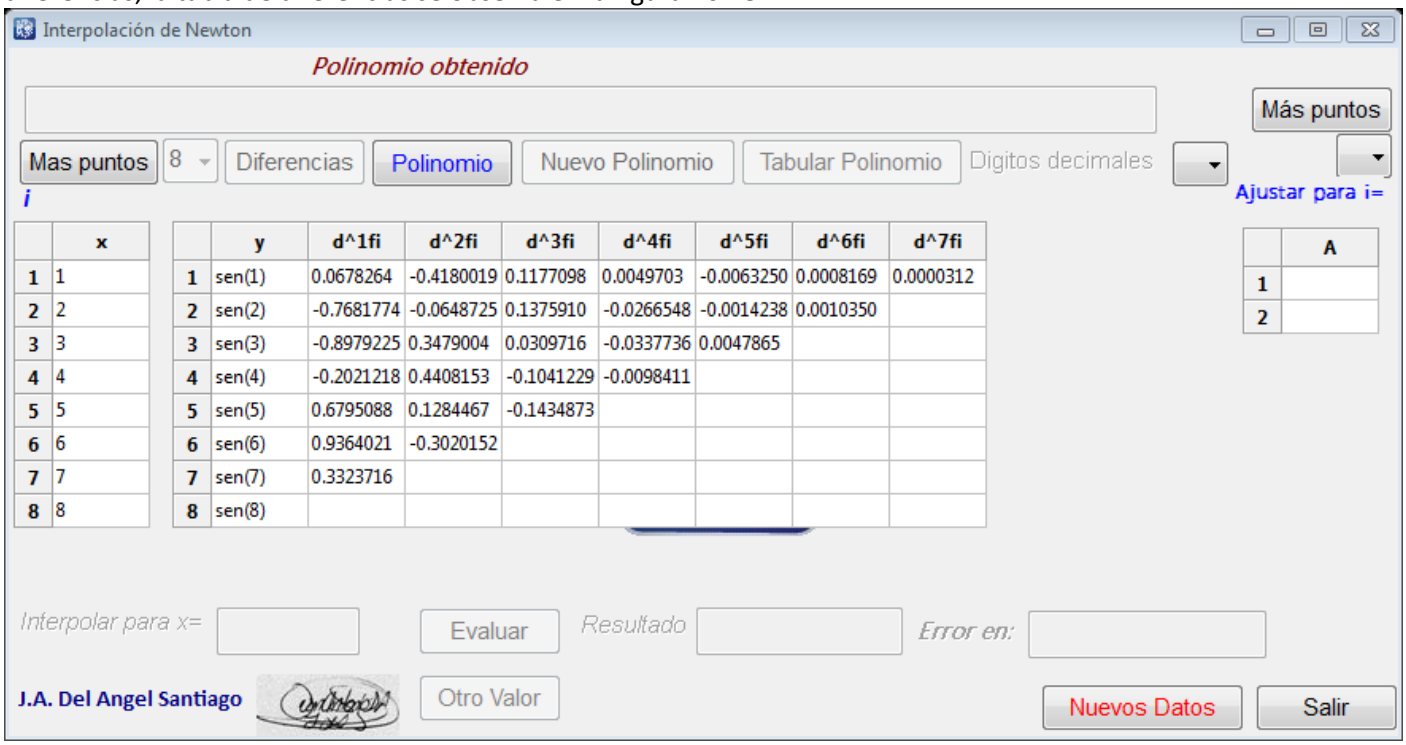


Figura 10.18 Tabla de diferencias divididas

Una vez que se ha obtenido la tabla de diferencias divididas se procede a insertar los puntos de ajuste. Para el primer caso se debe seleccionar el valor 8 de la lista de puntos de ajuste. En la figura 10.19 se observa lo que sucede si se intenta seleccionar un número mayor al número de puntos.

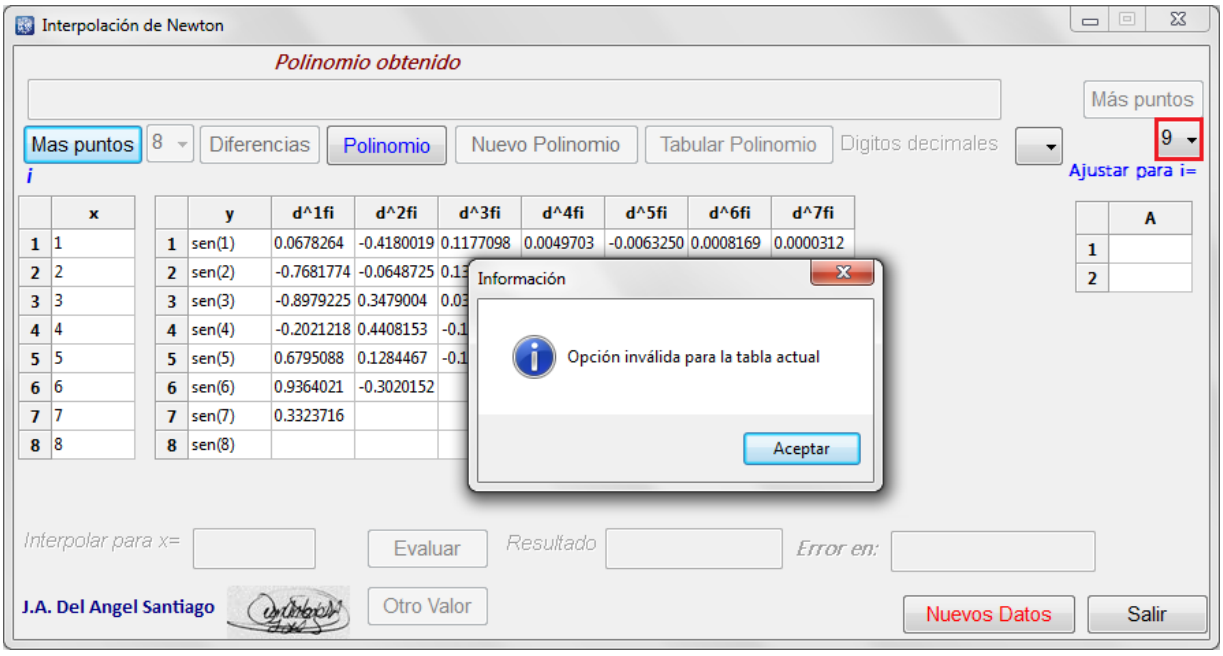


Figura 10.19 Opción inválida para la tabla

Ahora se insertaran los valores de i excepto i=8 y se pulsa Polinomio para observar el comportamiento de la figura 10.20.

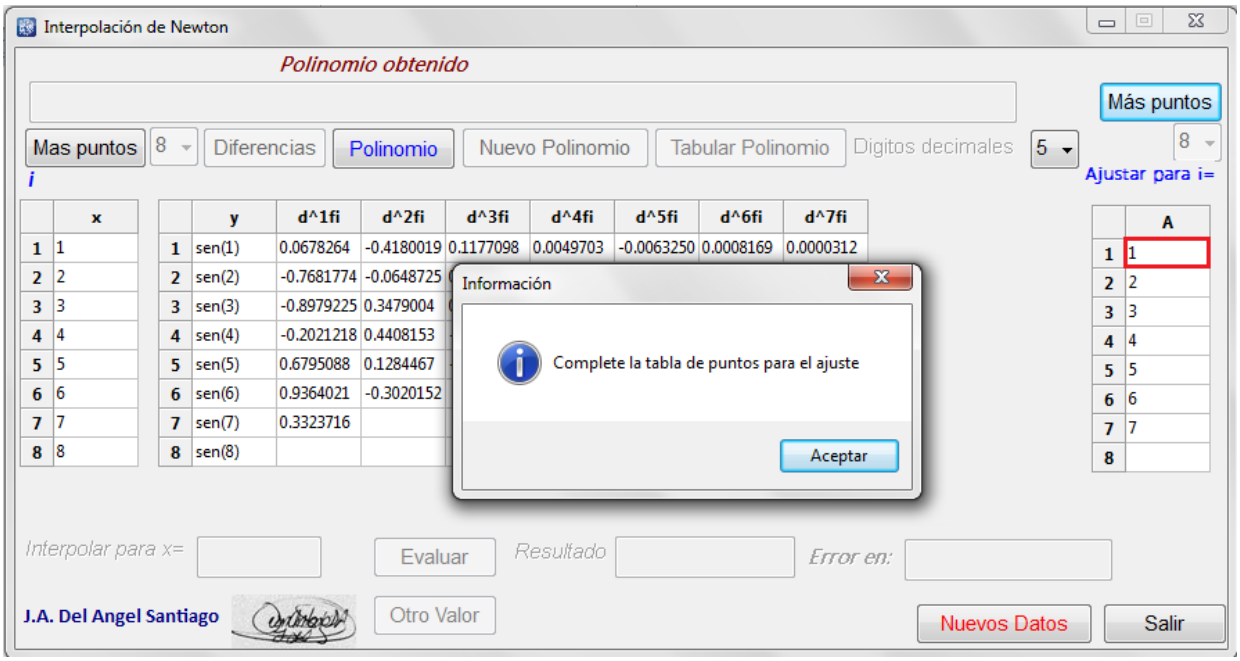


Figura 10.20 Tabla de ajuste incompleta.

Ahora se insertan todo los valores de i para el ajuste y se pulsa el botón polinomio. Además se inserta el valor a interpolar, se pulsa Evaluar y se realiza una tabulación desde 1 hasta 8 con un incremento de 1 para observar si el polinomio pasa por los puntos tabulados, esto se observa en la figura 10.21. A continuación se pulsa Nuevo Polinomio.

Para el segundo caso se selecciona 4 de la lista de puntos de ajuste y se insertan intencionalmente los valores de i=4, 6, 7, 8. Se observa en la figura 10.22 que en esta ocasión se muestra un mensaje que indica a la usuario que los valores insertados de i deben se consecutivos. En la figura 10.23 se muestra lo que sucede si se insertan los valores de i=5, 6,7, 9. Para esta situación se muestra un mensaje que indica que se verifiquen los valores de i debido a que se está intentando ajustar el polinomio un valor que no es permitido para el número de puntos que se han insertado, además se muestra el mensaje que indica que los valores deben ser consecutivos. Finalmente se insertan los valores

correctos, se pulsa Polinomio, se inserta el valor a interpolar y se pulsa Evaluar además de que se realiza una tabulación desde 5 hasta 8 con un incremento de 1/2, esto se aprecia en la figura 10.24.

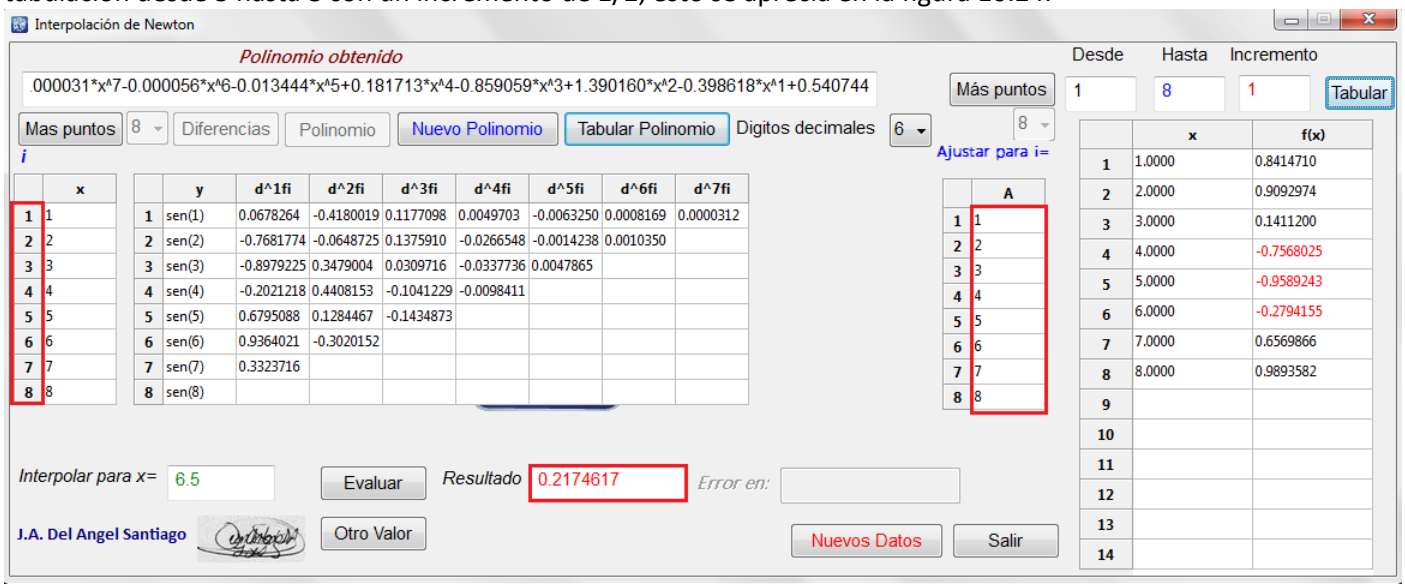


Figura 10.21 Resultados para el primer caso.

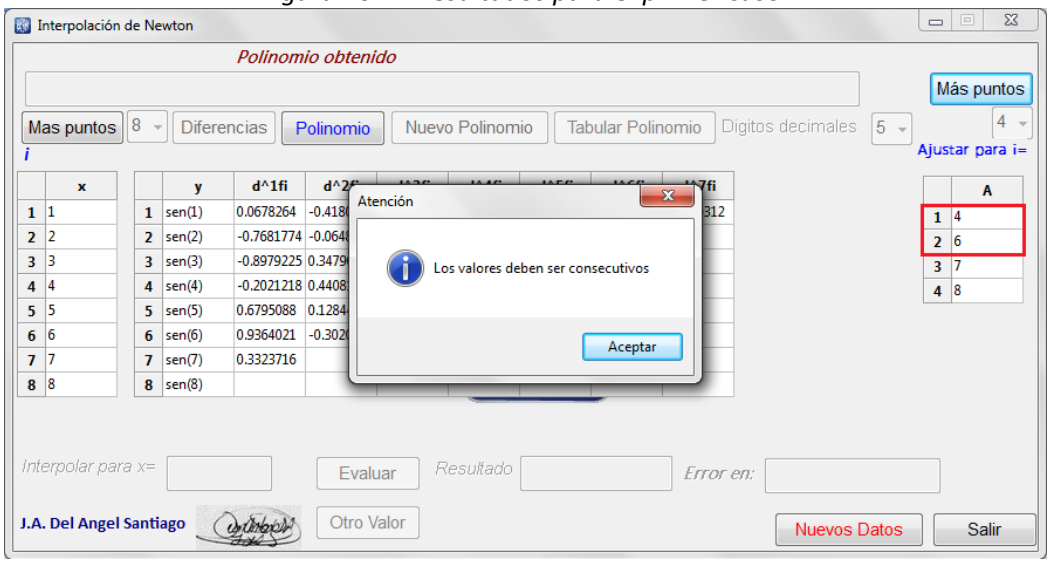


Figura 10.22 Los valores deben ser consecutivos

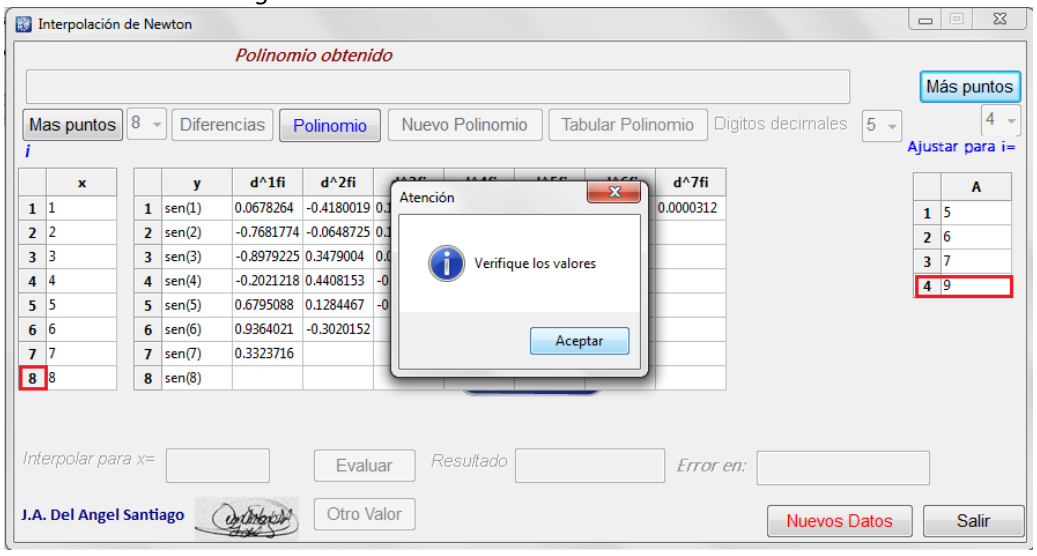


Figura 10.23 Verifique los valores

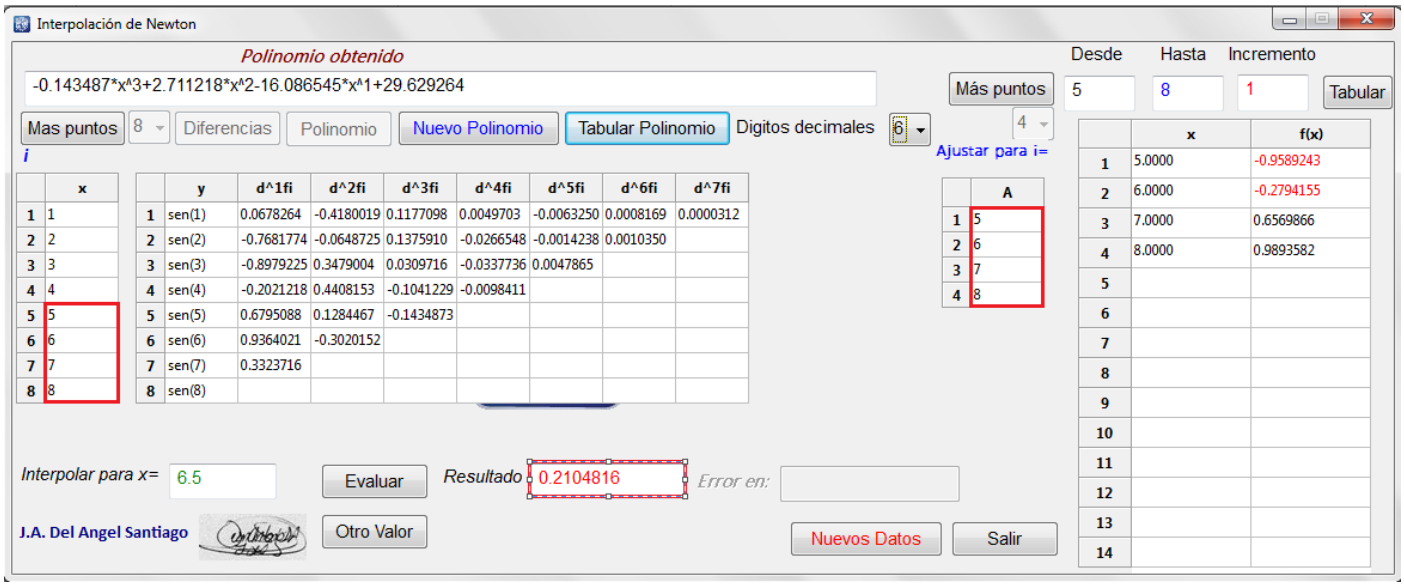


Figura 10.24 Resultados para el segundo caso.

En el primer caso tenemos que $p(6.5) = 0.2174617$ y en el segundo caso $p(6.5) = 0.2104816$. Sustituyendo en la función seno tenemos $sen(6.5) = 0.2151199$.
Para el primer caso se tiene un error de $error = abs(0.2151199 - 0.2174617) = 0.0023$
Para el segundo caso se tiene un error de $error = abs(0.2151199 - 0.2174617) = 0.0046$

10.6 Interpolación Spline.

Considere los puntos de la tabla 10.4. Se deben obtener los polinomios cúbicos de spline con las condiciones de frontera $y''(-1) = 0 \quad y''(3) = 0$. Además se debe interpolar para x=1.5.

i	0	1	2	3
f(xi)	1	0.5	3.5	5
xi	-1	1	2	3

Tabla 10.4

El primer caso es Natural por lo que se selecciona la opción Natural de los botones circulares, se selecciona el valor 4 de la lista de selección de puntos, se insertan los valores de los puntos y se pulsa calcular. Si se omite algún valor de la tabla de puntos se mostrara el mensaje de la figura 10.25 y se indicará la posición de la tabla en que se ha omitido un valor o se ha cometido algún error de sintaxis, esto ocurre en todas las tablas de los programas en los que se omita algún valor en el llenado de las tablas. Si no se ha cometido error alguno se muestra el primer polinomio que se observa en la figura 10.26.

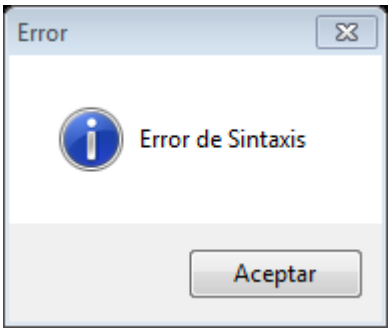
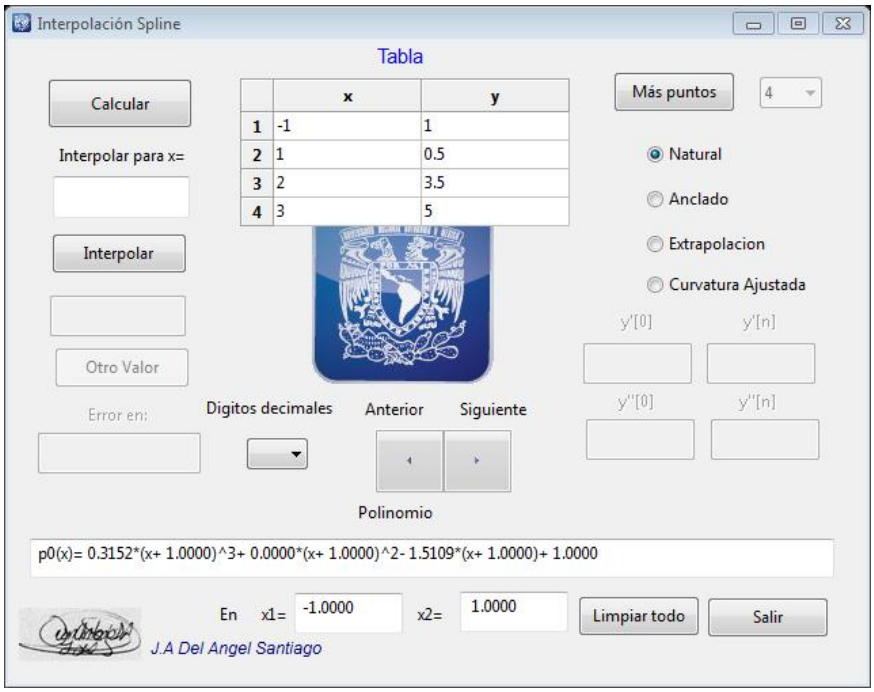


Figura 10.25 Mensaje de error de sintaxis



	x	y
1	-1	1
2	1	0.5
3	2	3.5
4	3	5

Figura 10.26 Primer polinomio obtenido.

Pulsando dos veces el botón Siguiente se observan los dos polinomios restantes que se observan en la figura 10.27.

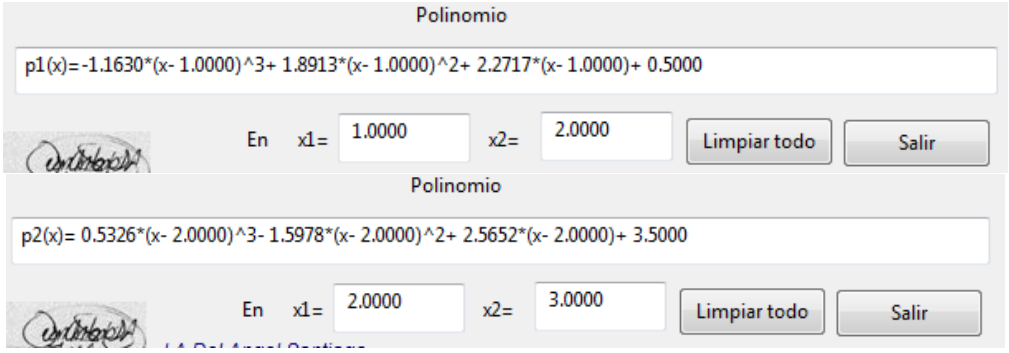
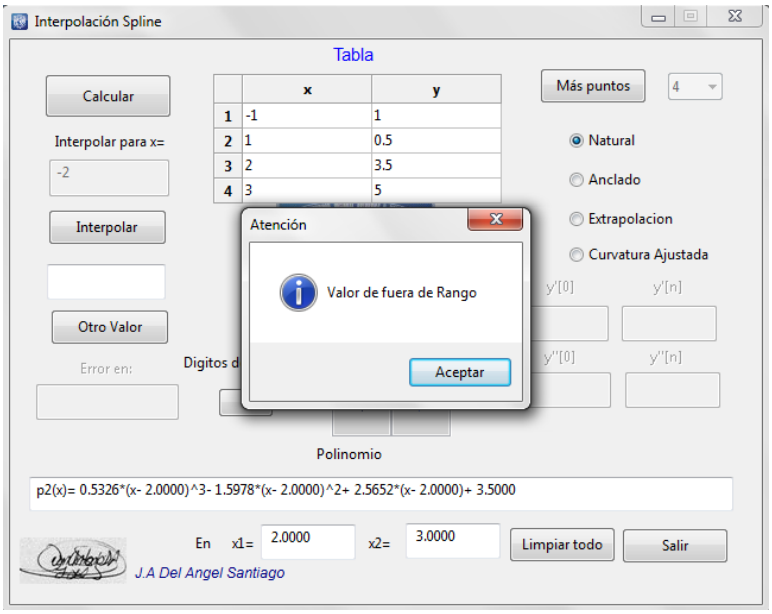


Figura 10.27 Polinomios p1(x) y p2(x)

Ahora se inserta el valor a interpolar, observemos en la figura 10.28 lo que sucede si insertamos un valor fuera del intervalo:



	x	y
1	-1	1
2	1	0.5
3	2	3.5
4	3	5

Figura 10.28 Valor fuera de rango

Insertando el valor de x=1.5 se obtiene el resultado mostrado en la figura 10.29.

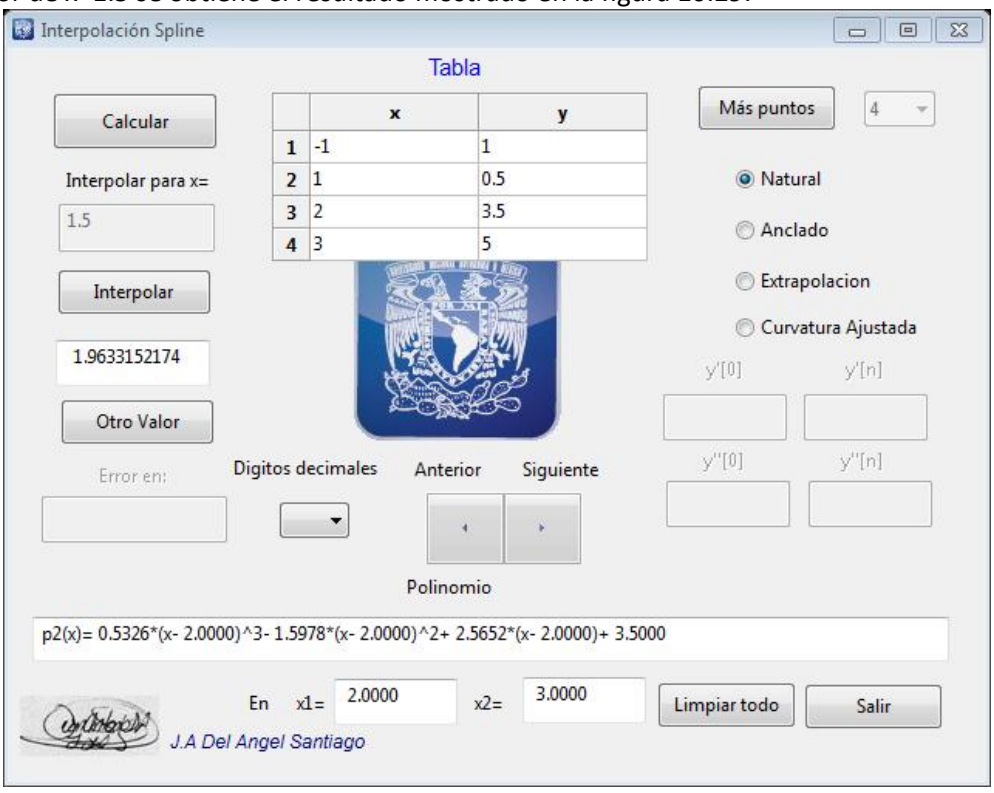


Figura 10.29 Resultado de la interpolación

De tal manera que los resultados son:
p0(x)= 0.3152*(x+ 1.0000)^3+ 0.0000*(x+ 1.0000)^2- 1.5109*(x+ 1.0000)+ 1.0000 en [-1,1]
p1(x)=-1.1630*(x- 1.0000)^3+ 1.8913*(x- 1.0000)^2+ 2.2717*(x- 1.0000)+ 0.5000 en [1,2]
p2(x)= 0.5326*(x- 2.0000)^3- 1.5978*(x- 2.0000)^2+ 2.5652*(x- 2.0000)+ 3.5000 en [2,3]
p1(1.5]=1.9633152174

Si se muestra el último Polinomio y se pulsa una vez más siguiente se notificará al usuario que se trata del primer polinomio. Por otra parte si se muestra el primer polinomio y se pulsa Anterior se notificará al usuario que se trata del primer polinomio. Estos mensajes se aprecian en la figura 10.30.

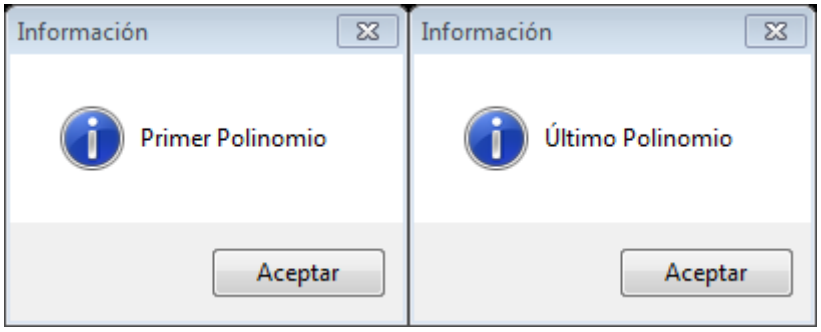


Figura 10.31

En caso de que se necesite usar las opciones Anclado o Curvatura solo se insertan los valores requeridos en las cajas de texto que se habiliten según el caso que se utilice. De la misma manera se revisa la sintaxis de lo que se inserte y se verifica el llenado de las cajas de texto.

10.7 Integración Numérica

Considere la integral definida:

$$\int_0^{\frac{\pi}{2}} sen(x)dx$$

Para realizar la integración se debe tabular desde 0 hasta π/2 (recuerde que π se inserta como 2*asn(1) por lo que π/2 se inserta como asn(1)), se inserta un incremento de 0.1,se pulsa el botón Tabular como se observa en la figura 10.32.

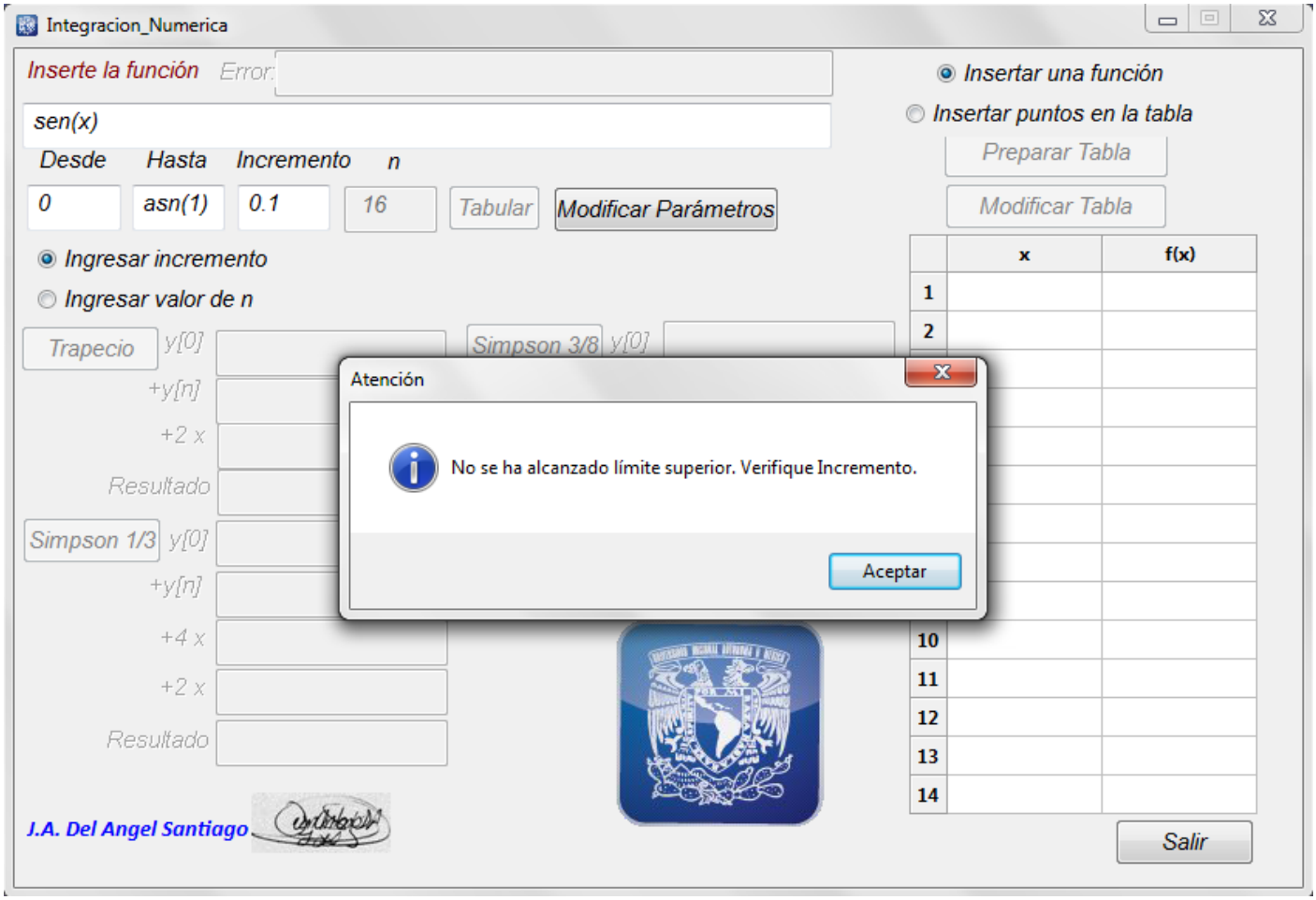


Figura 10.32 No se alcanza el valor exacto del límite superior

Esto ocurre debido a que con incremento de 0.1 no se puede llegar exactamente al valor de π. Se pueden dar distintos casos dependiendo del valor de n insertado:

1. Con valores de n que no son múltiplos de 2 o 3 solo se habilitará el método Trapezoidal.
2. Con valores de n que son múltiplos de 2 se habilita además del método Trapezoidal el método Simpson 1/3.
3. Con valores de n múltiplos de 3 se habilita el método Trapezoidal y Simpson 3/8.
4. Para valores de n múltiplos de 2 y de 3 se habilitan los tres métodos.

Para obtener una buena aproximación de la integral se inserta n=300 para observar además los resultados proporcionados por cada uno de los métodos, esto se observa en la figura 10.33. El resultado de la integral es:

$$\int_0^{\frac{\pi}{2}} sen(x)dx = 1$$

Se puede observar que con la regla trapezoidal se obtiene un error de: 0.000002285 mientras que con ambas reglas de Simpson se obtiene el valor exacto de la integral.

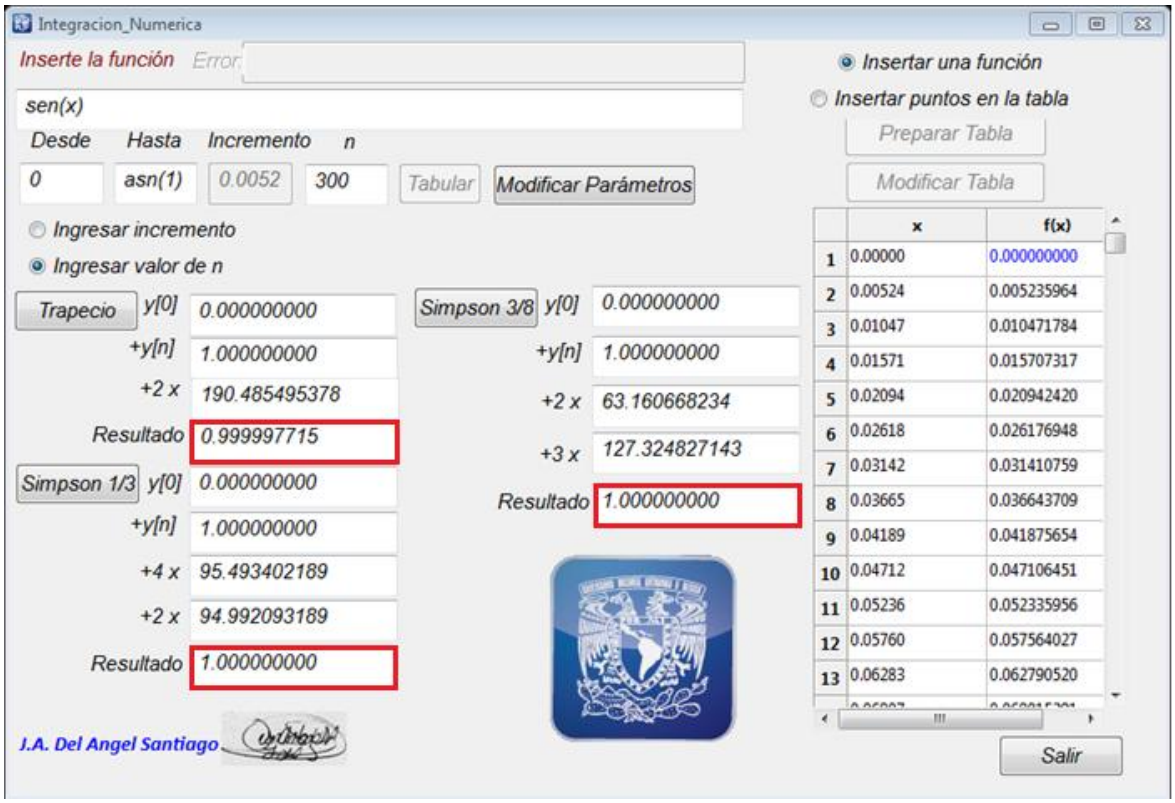


Figura 10.33 Resultado de la integral con n=300.

Cuando se pretenda integrar un intervalo que contiene errores matemáticos la integración no será posible como se aprecia en la figura 10.34, observe además que este caso se inserta el parámetro incremento en lugar de n.

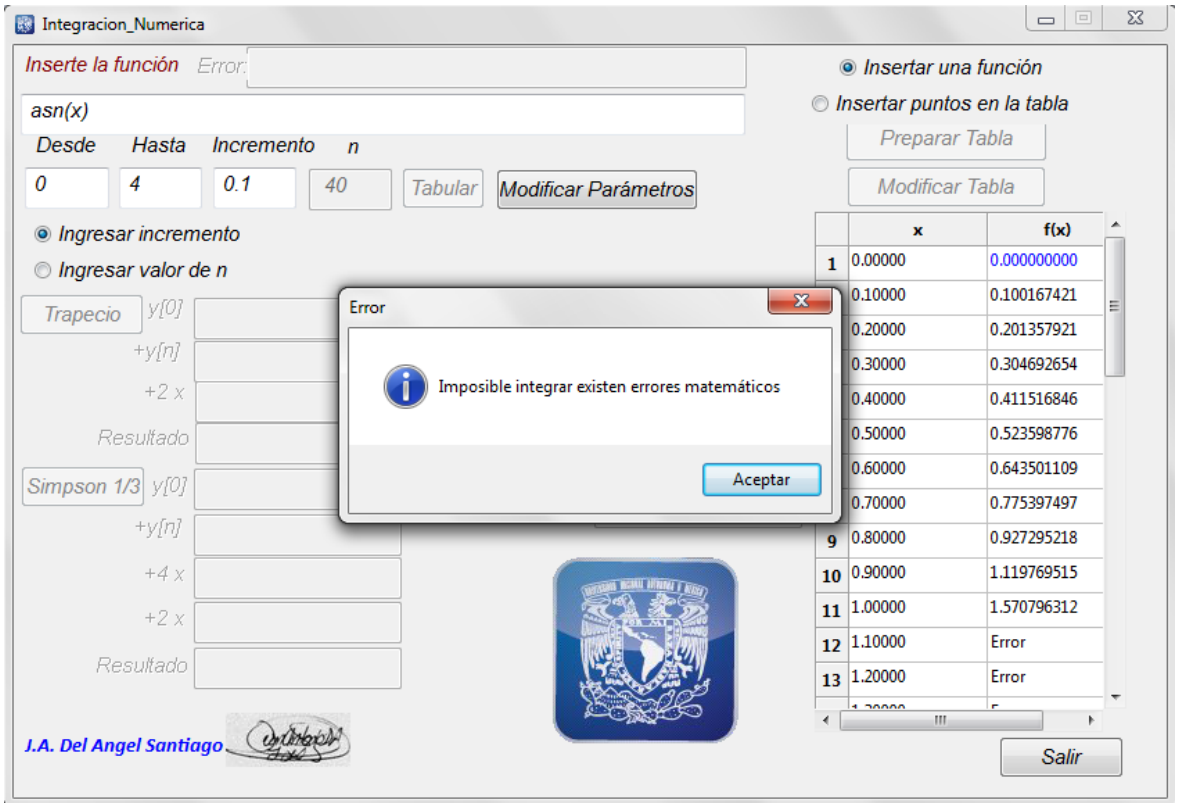


Figura 10.34 Imposible integrar

Ahora considere e la tabla 10.5. Se debe obtener el área de la función definida en forma tabular.

x	0	1	2	3	4	5	6	7	8
F(x)	0	1	4	9	16	25	36	49	64

Tabla 10.5

Se debe seleccionar la opción *Insertar puntos en la tabla,* se prepara la tabla desde 0 hasta 8 con un incremento de uno y se pulsa preparar tabla como se observa en la figura 10.35.

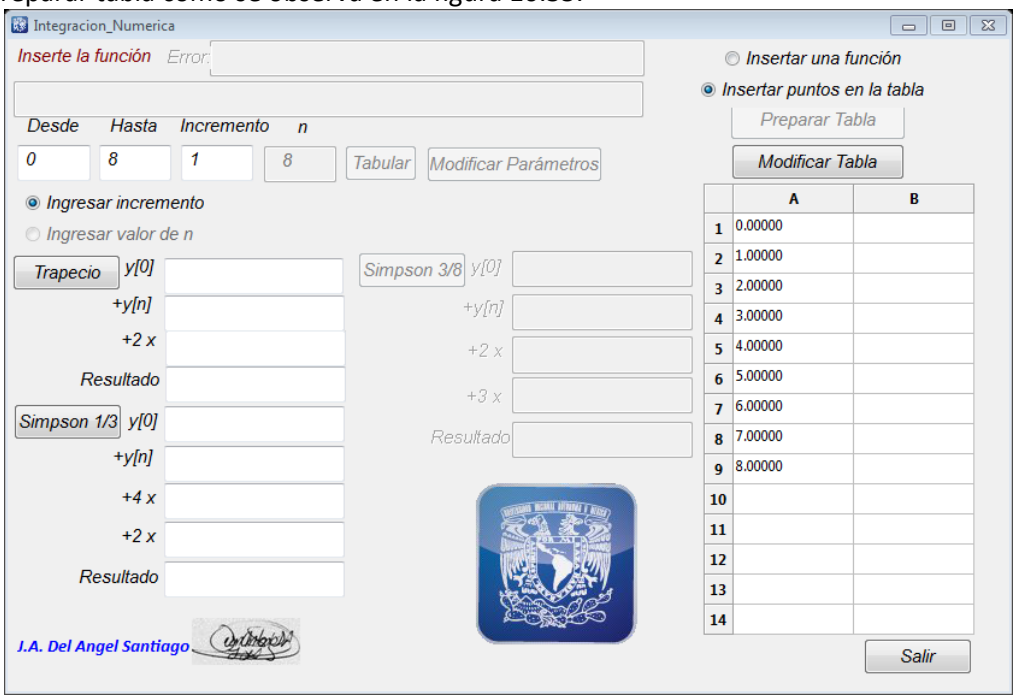


Figura 10.35 Tabla preparada para insertar valores de f(x).

Se debe notar que n es múltiplo de 2 por lo que se habilitan los métodos trapezoidal y Simpson 1/3. Se insertan los valores de f(x) y se obtienen los resultados de la figura 10.36.

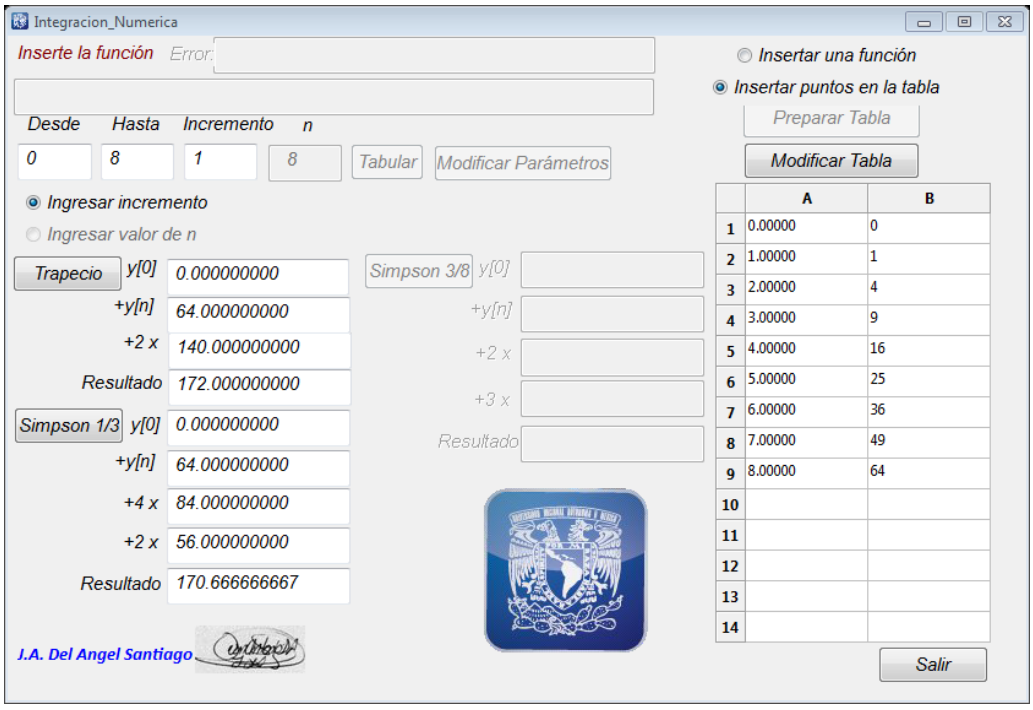


Figura 10.36 Resultados de la integración

10.8 Runge Kutta de cuarto orden.

Considere la ecuación diferencial con la condición inicial dada para $0 < x < 1$:

$$\frac{dy}{dx} = \cos(2x) + sen(3x) \qquad y'(0) = 1$$

Se insertan los valores dados y la ecuación con N=100 para obtener una buena aproximación de la raíz como se muestra en la figura 10.37.

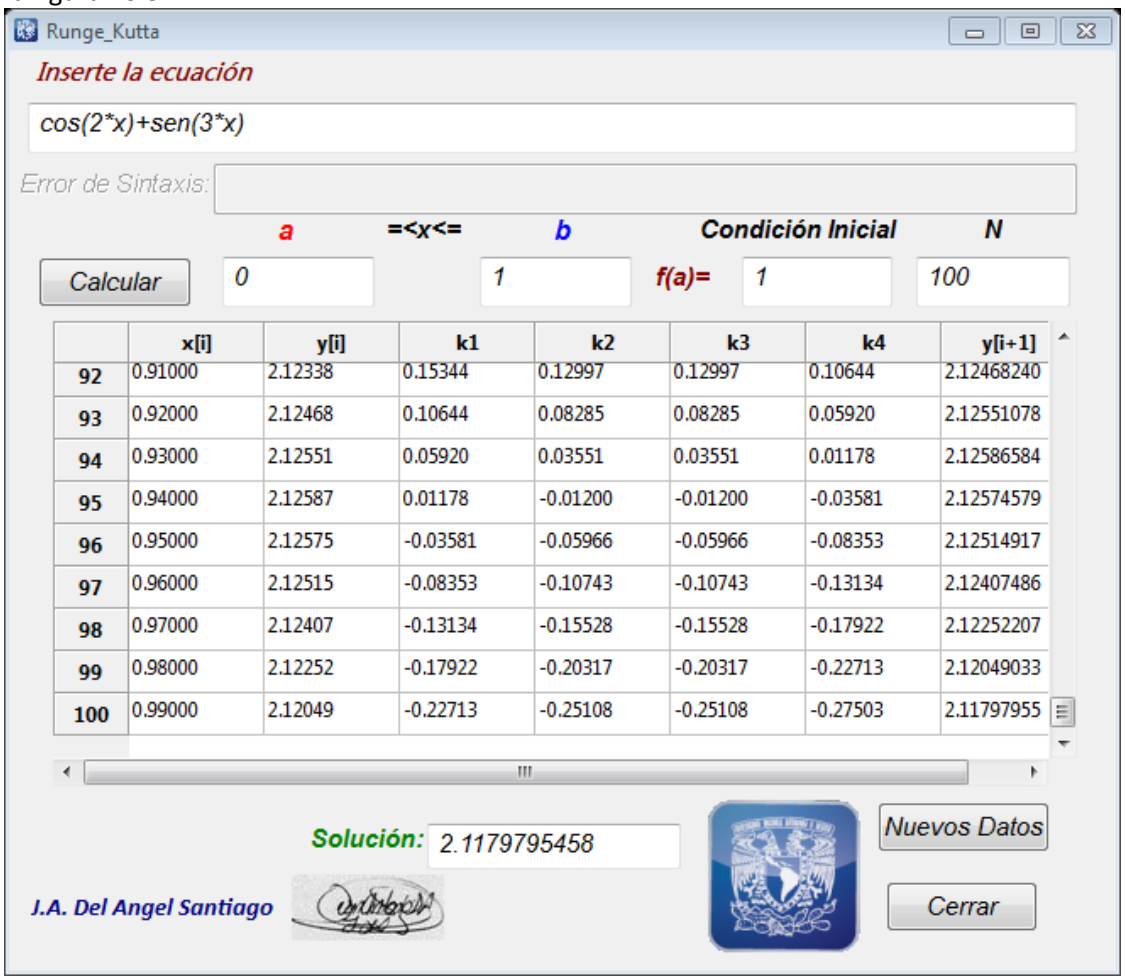


Figura 10.37 solución de la ecuación diferencial.

Si se compara esta solución con la obtenida por el programa Maple puede comprobarse que no existe diferencia alguna con la solución presentada en la figura 10.37.

Las literales que se deben usar en la ecuación solo pueden ser (x) y (y) de tal manera que si se tiene una ecuación como:

$$\frac{dy}{dt} = te^{3t} - 2y$$

Debe ser insertada como:

$$\frac{dy}{dx} = xe^{3x} - 2y$$

Es decir (x) es la variable independiente y (y) la variable dependiente.

11. APLICACIONES Y RESULTADOS DE LOS PROGRAMAS.

En este capítulo se presentan diferentes casos de estudio en los que se aplica cada uno de los programas, como se mencionó en el capítulo anterior se omiten explicaciones detalladas y las capturas de pantalla mostradas están enfocadas a mostrar los resultados para cada caso de estudio.

11.1 Bisección.

1. La velocidad v de un paracaidista está dada por:

$$v = \frac{gm}{c}\left(1 - e^{-\left(\frac{c}{m}\right)t}\right) \dots 11.1$$

Donde $g = 9.81\ m/s^2$. Para el paracaidista con un coeficiente de arrastre $c = 14kg/s$, calcule la masa m de tal manera que la velocidad sea de $v = 32\ m/s$ en $t = 7s$.

Solución:

- Sustituyendo los valores e igualando a cero queda:

$$35 = \frac{9.81m}{14}\left(1 - e^{-\left(\frac{14}{m}\right)7}\right) \dots 11.2 \qquad \frac{9.81m}{14}\left(1 - e^{-\left(\frac{14}{m}\right)7}\right) - 35 = 0 \dots 11.3$$

Se inserta la expresión 11.3 *(con la literal x como variable independiente)* y se presiona Analizar raíces, se tabula de -100 a 100 y se presiona buscar raíces. Se encuentran dos raíces como se observa en la figura 11.1.

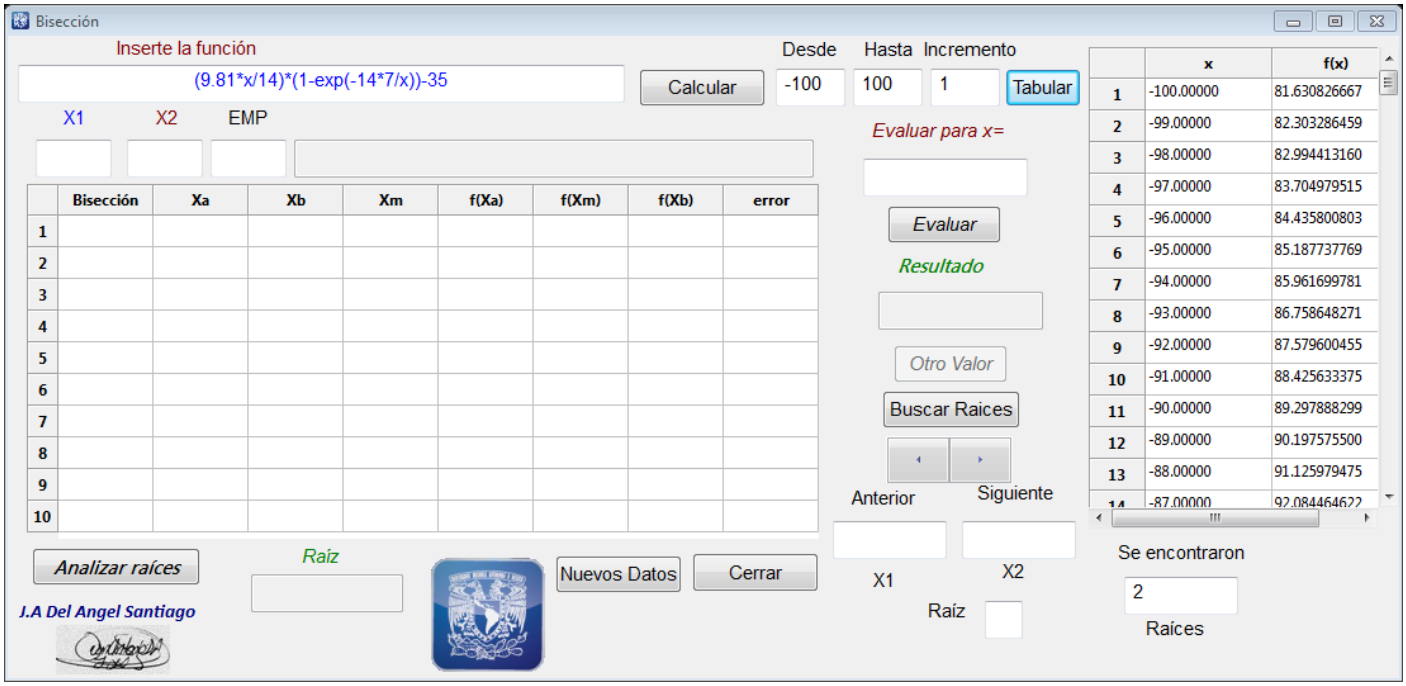


Figura 11.1

- Se pulsa dos veces siguientes y se obtienen los intervalos de las raíces como se aprecia en la figura 11.2.

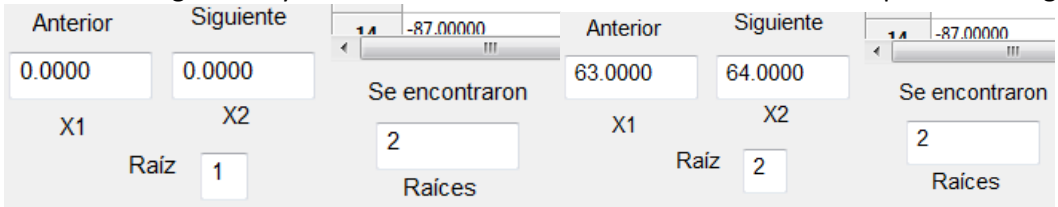


Figura 11.2 Intervalos de las raíces

Se observa que hay una raíz exacta en x=0 pero no será considerada debido a que se desea calcular la masa del paracaidista. Por lo tanto se inserta el intervalo de la segunda raíz con $EMP = 1 \times 10^{-10}$ y se presiona Calcular. Los resultados obtenidos se observan en la figura 11.3.

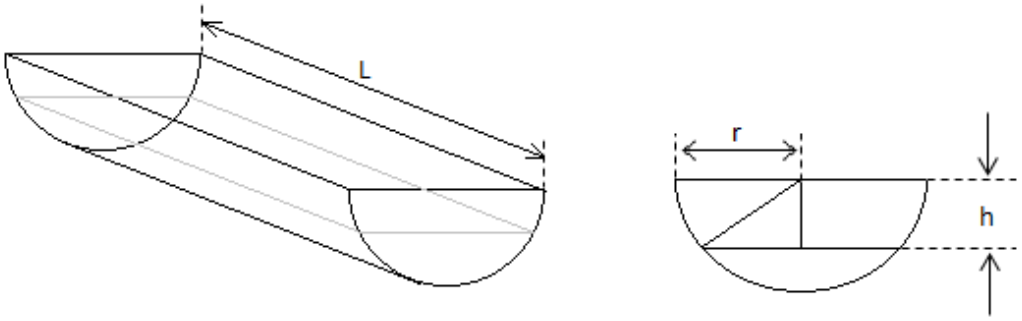
Figura 11.3 Resultados obtenidos

Por lo tanto la masa del paracaidista es $m = 63.5378\ kg$

2. Un depósito de longitud L y tiene una sección transversal en forma de semicírculo con radio r. Cuando se llena de agua hasta una distancia h de la parte superior, el volumen V de agua es:

$$V = L\left[0.5\pi r^2 - r^2 arcsen\left(\frac{h}{r}\right) - h(r^2 - h^2)^{\frac{1}{2}}\right] \ldots 11.4$$

Suponga que $L = 10\ ft, r = 1\ ft\ \ y\ que\ V = 12.4\ ft^3$. Determine la profundidad del agua en el abrevadero.



Solución:
Sustituyendo los valores en la expresión 11.4 se obtiene:

$$12.4 = 10\left[0.5\pi 1^2 - 1^2 arcsen\left(\frac{h}{1}\right) - h(1^2 - h^2)^{\frac{1}{2}}\right] \ldots 11.5$$

Igualando a cero:

$$10\left[0.5\pi - arcsen(h) - h(1 - h^2)^{\frac{1}{2}}\right] - 12.4 = 0 \ldots 11.6$$

El valor exacto de π se inserta como 2*asn(1) por lo que $0.5\pi = 0.5\big(2asn(1)\big) = asn(1)$ y queda:

$$10\left[asn(1) - asn(h) - h(1 - h^2)^{\frac{1}{2}}\right] - 12.4 \ldots 11.7$$

Donde asn es la sintaxis de arcsen para el evaluador de expresiones algebraicas.

- Se inserta la expresión 11.7 y se tabula Desde -3 hasta 3 con un incremento de 0.1. A continuación se pulsa buscar raíces y se oprime el botón siguiente para obtener el intervalo de la raíz. Se inserta el intervalo obtenido con $EMP = 1 \times 11^{-10}$ y se pulsa Calcular como se aprecia en la figura 11.4 para obtener el valor de$h$. Recuerde que la variable independiente debe ser insertada como (x).

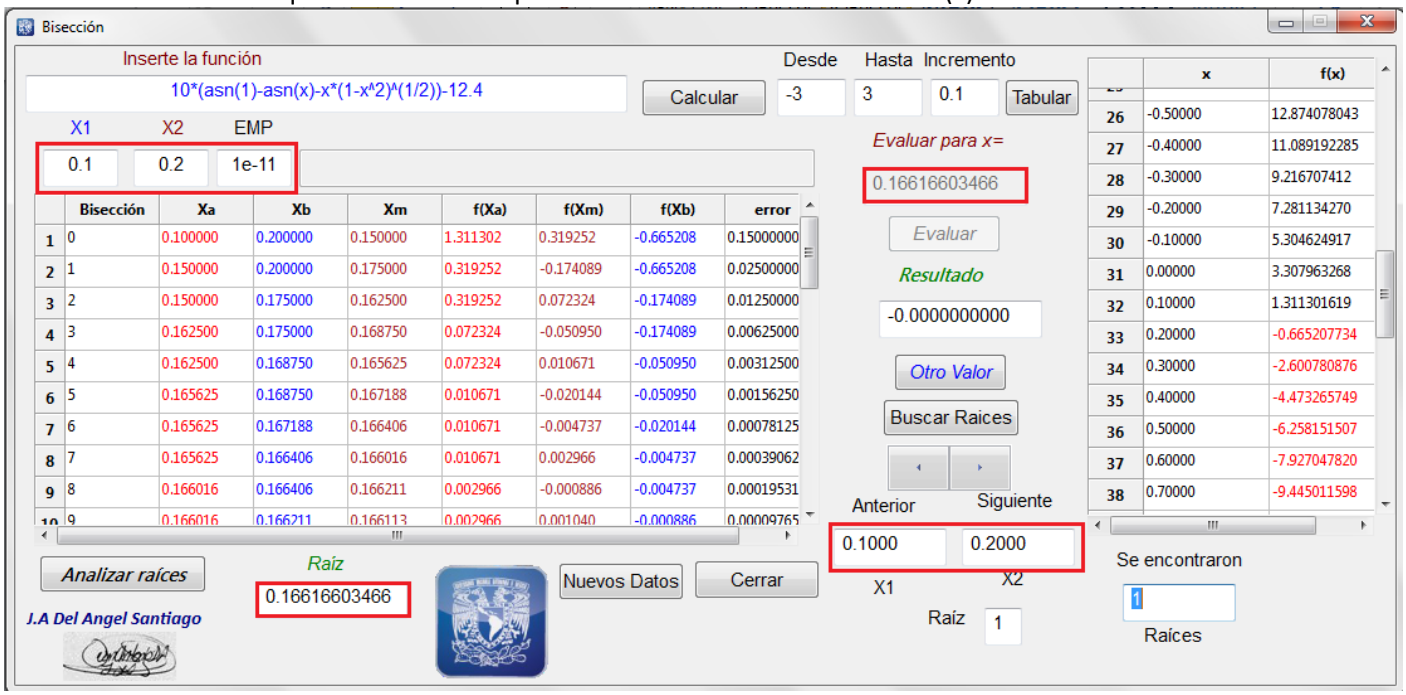


Figura 11.4 Resultados obtenidos.

El resultado es $h = 0.16616603466$. La profundidad es:

$$1 - h = 0.83383\ ft$$

11. Determinar las raíces de la función:

$$f(x) = senh(x) - x^3 \ldots 11.8$$

Solución:

Como puede observarse en la tabla 10.1 $senh(x)$ no es una expresión soportada por el evaluador de expresiones. En estos casos se pueden utilizar identidades que definen al seno y al cosen hiperbólico:

seno hiperbólico $$senh(x) = \frac{e^x - e^{-x}}{2}$$

coseno hiperbólico $$\cosh(x) = \frac{e^x + e^{-x}}{2}$$

Sustituyendo la identidad para el seno hiperbólico en 11.8 se obtiene:

$$f(x) = \frac{e^x - e^{-x}}{2} - x^3 \ldots 11.9$$

- Se tabula desde -10 hasta 10 con un incremento de 0.1 y se pulsa Tabular. Posteriormente se pulsa 4 veces Siguiente para obtener los intervalos mostrados en la figura 11.5.

- Se inserta cada intervalo con $EMP = 1 \times 10^{-11}$ y se calcula cada una de las raíces mostradas en la figura 11.6.

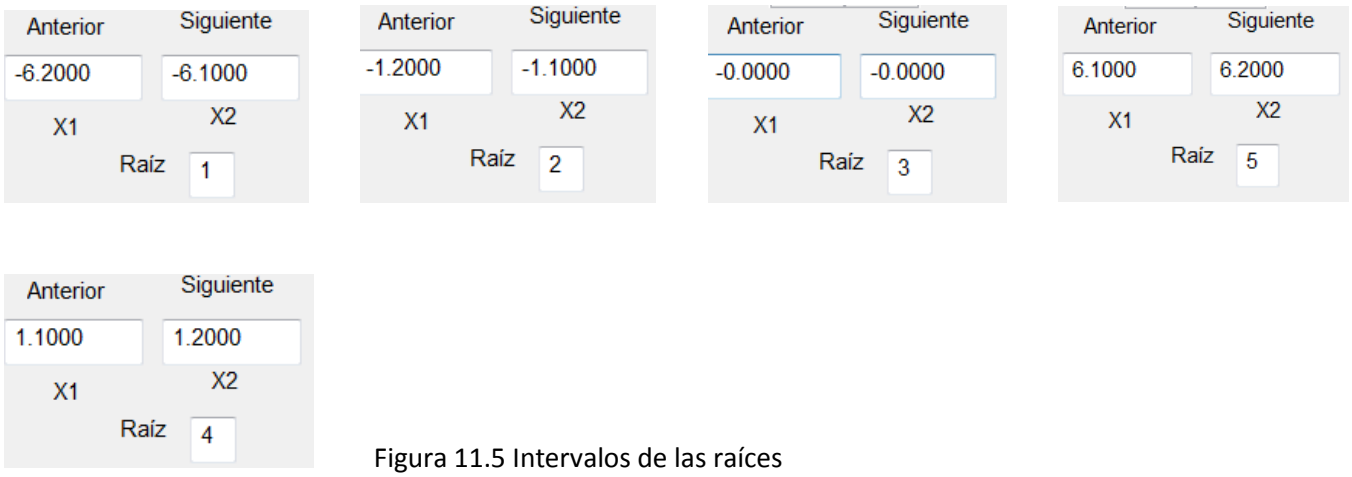


Figura 11.5 Intervalos de las raíces

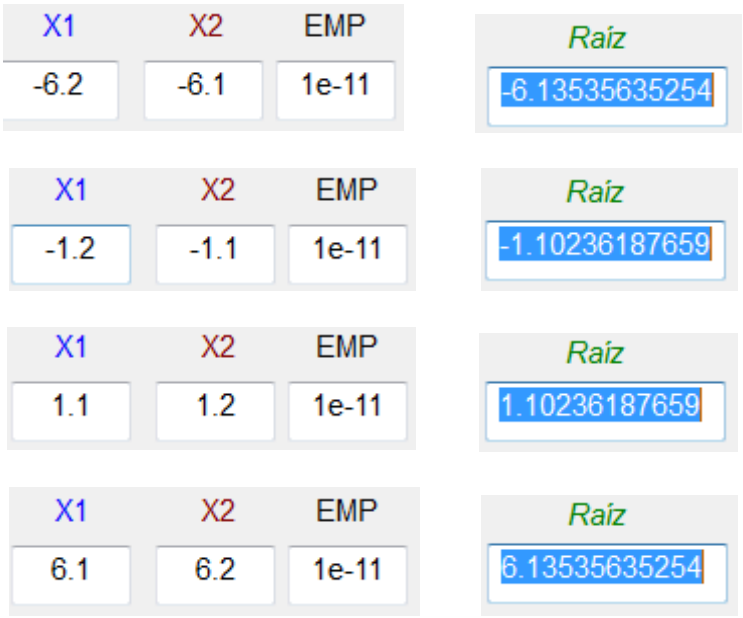


Figura 11.6 Raíces Calculadas

Se observa en la figura 11.5 que la raíz 3 es exacta: $x_3 = 0$

Por lo tanto las raíces son:

$$x_1 = -6.13535635254 \quad x_2 = -1.10236187659 \quad \boldsymbol{x_3 = 0} \quad x_4 = 1.10236187659 \quad x_5 = 6.13535635254$$

11.2 Newton – Raphson.

12. Suponga que el fenómeno de la transmisión de calor en un cierto material obedece en forma aproximada al modelo:

$$T = T_0 + \frac{q}{k}\left(\beta\left(\frac{\alpha T}{\pi}\right)^{\frac{1}{2}} e^{-\frac{x^2}{4\alpha t}}\right) \ldots 11.10$$

Calcule el tiempo requerido para que la temperatura a la distancia x alcance un valor dado. Use la siguiente información:

$$T_0 = 25^oC \qquad q = 300\frac{BTU}{h}ft^2 \quad k = 1\frac{BTU}{h}\,ft^2\,°\text{F} \quad \alpha = 0.04\frac{ft^2}{h} \quad T = 120\,°\text{F} \quad \beta = 2\frac{°\text{F}ft}{h^{\frac{1}{2}}}°\text{C}^{\frac{1}{2}} \quad x = 1ft$$

Solución:
Sustituyendo los valores en 11.10 se obtiene:

$$120 = 25 + \frac{300}{1}\left(2\left(\frac{0.04 \times 120}{\pi}\right)^{\frac{1}{2}} e^{-\frac{1^2}{4\times 0.04\times t}}\right) \ldots 11.10$$

Realizando operaciones:

$$120 = 25 + 741.6464679e^{-\frac{1}{0.16t}} \ldots 11.11$$

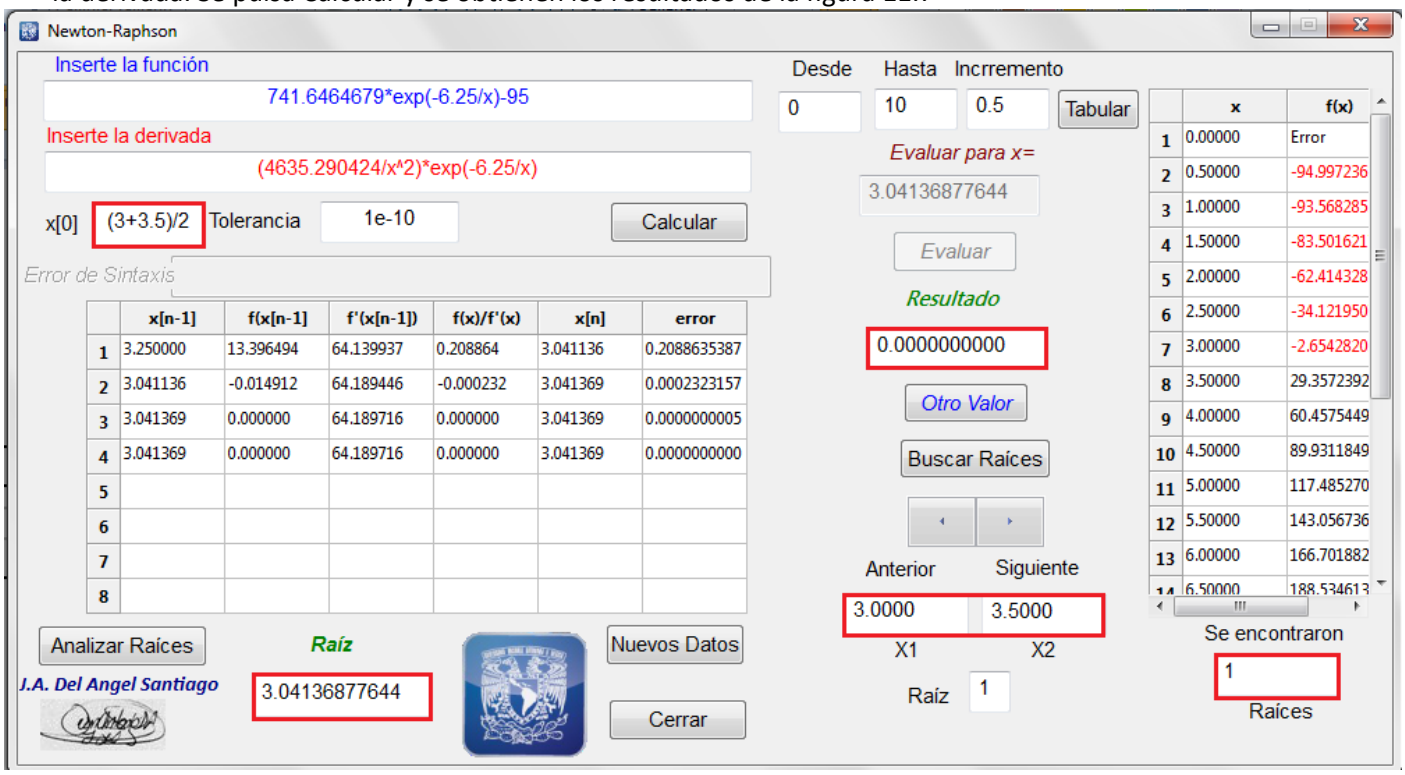
Igualando a cero:

$$741.6464679e^{-6.25t} - 95 = 0 \ldots 11.12$$

Derivado se obtiene:

$$\frac{4635.290424e^{-6.25t}}{t^2} = 0 \ldots 11.13$$

- Primero se inserta la ecuación 11.12 y se tabula desde 1 hasta 10 con incremento de 0.1, se pulsa Buscar raíces y se pulsa *Siguiente* para observar el intervalo de la raíz. En x[0] se inserta la mitad del intervalo: $x_0 = (3 + 3.5)/2$ y una tolerancia de 1×10^{-10}, además se debe insertar la expresión 11.13 en la caja de texto que corresponde a la derivada. Se pulsa Calcular y se obtienen los resultados de la figura 11.7



	x[n-1]	f(x[n-1])	f'(x[n-1])	f(x)/f'(x)	x[n]	error
1	3.250000	13.396494	64.139937	0.208864	3.041136	0.2088635387
2	3.041136	-0.014912	64.189446	-0.000232	3.041369	0.0002323157
3	3.041369	0.000000	64.189716	0.000000	3.041369	0.0000000005
4	3.041369	0.000000	64.189716	0.000000	3.041369	0.0000000000
5						
6						
7						
8						

	x	f(x)
1	0.00000	Error
2	0.50000	-94.997236
3	1.00000	-93.568285
4	1.50000	-83.501621
5	2.00000	-62.414328
6	2.50000	-34.121950
7	3.00000	-2.6542820
8	3.50000	29.3572392
9	4.00000	60.4575449
10	4.50000	89.9311849
11	5.00000	117.485270
12	5.50000	143.056736
13	6.00000	166.701882
14	6.50000	188.534613

Figura 11.7 Resultados obtenidos

Por lo tanto el resultado es $t = 3.041\ hrs.$

13. La velocidad de un paracaidista está dada por:

$$v = \frac{gm}{c}\left(1 - e^{-\left(\frac{c}{m}\right)t}\right) \ldots 11.1$$

Donde $g = \frac{980cm}{seg^2}$ $\quad m = 75000gr \quad v = 3600\ cm/seg$

Calcular el coeficiente de rozamiento c cuando $t = 6\ seg$.

Sustituyendo valores en la expresión se obtiene:

$$3600 = \frac{980 \times 75000}{c}\left(1 - e^{-\left(\frac{c}{75000}\right)\times 6}\right) \ldots 11.14$$

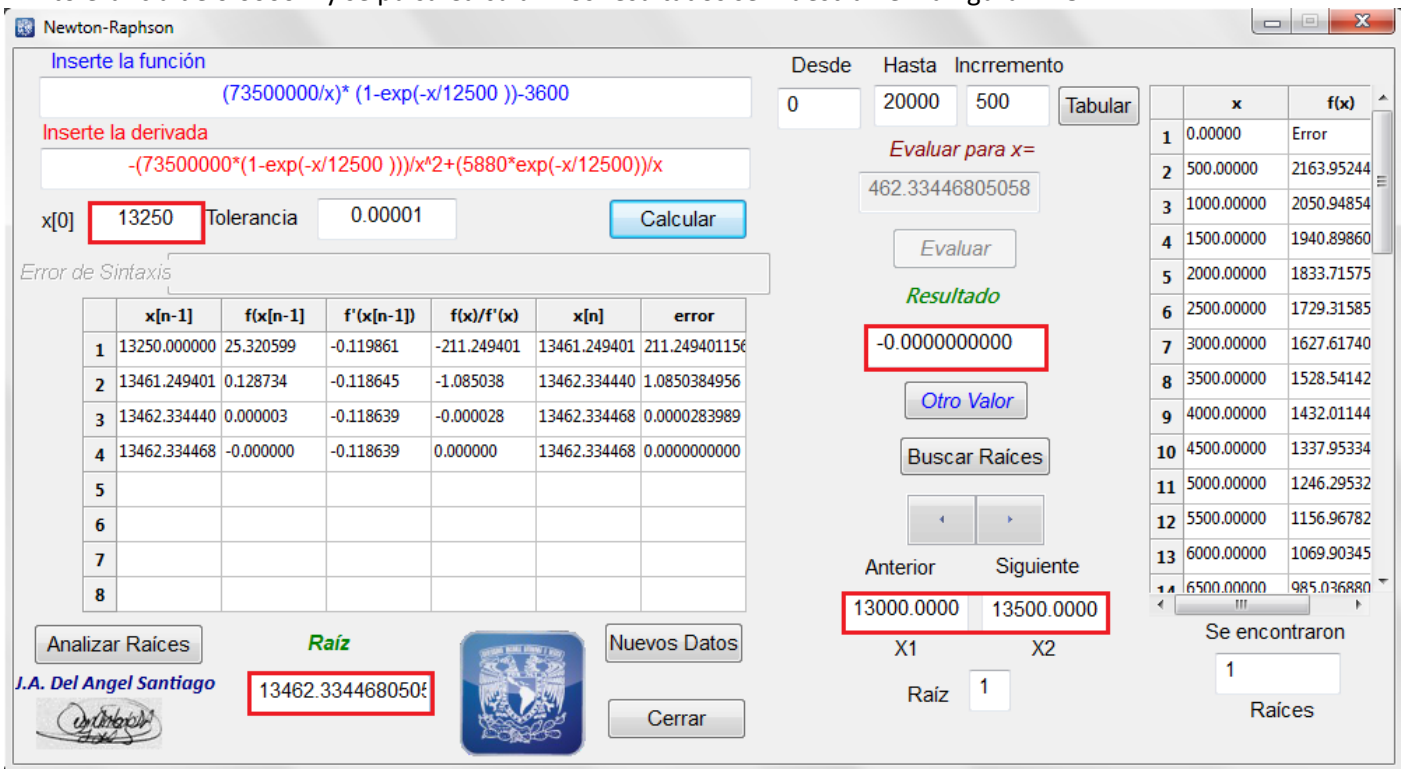
Realizando operaciones e igualando a cero:

$$\frac{73500000}{c}\left(1 - e^{-\left(\frac{c}{12500}\right)}\right) - 3600 = 0 \ldots 11.15$$

Derivando:

$$-\frac{73500000\left(1 - e^{\left(-\frac{c}{12500}\right)}\right)}{c^2} + \frac{5880e^{-\frac{c}{12500}}}{c} = 0 \ldots 11.16$$

- Se inserta la expresión 11.15 y se tabula de 0 a 20,000 con un incremento de 500. Se presiona Buscar Raíces y a continuación Siguiente para observar el intervalo de la raíz. Se inserta la mitad del intervalo en x[0], una tolerancia de 0.00001 y se pulsa Calcular. Los resultados se muestran en la figura 11.8.



	x[n-1]	f(x[n-1]	f'(x[n-1])	f(x)/f'(x)	x[n]	error
1	13250.000000	25.320599	-0.119861	-211.249401	13461.249401	211.249401156
2	13461.249401	0.128734	-0.118645	-1.085038	13462.334440	1.0850384956
3	13462.334440	0.000003	-0.118639	-0.000028	13462.334468	0.0000283989
4	13462.334468	-0.000000	-0.118639	0.000000	13462.334468	0.0000000000
5						
6						
7						
8						

	x	f(x)
1	0.00000	Error
2	500.00000	2163.95244
3	1000.00000	2050.94854
4	1500.00000	1940.89860
5	2000.00000	1833.71575
6	2500.00000	1729.31585
7	3000.00000	1627.61740
8	3500.00000	1528.54142
9	4000.00000	1432.01144
10	4500.00000	1337.95334
11	5000.00000	1246.29532
12	5500.00000	1156.96782
13	6000.00000	1069.90345
14	6500.00000	985.036880

Figura 11.8 Resultados obtenidos.

Por lo tanto el coeficiente es $c = 13462.334468$.

14. Un objeto de peso $w = 4\ lb$, se suelta con una velocidad cero en $t = 0\ (seg)$. Encuentra resistencia del aire con un coeficiente de fricción de $r = 0.1\ \left(lb \cdot \frac{seg}{pie}\right)$, por lo que la distancia s en pies que ha caído después de t segundos es:

$$s = v\left[t - v\left(\frac{1 - e^{\left(-\frac{gt}{v}\right)}}{g}\right)\right] \ldots 11.17$$

Donde $v = \frac{w}{r} = \frac{4}{0.1} = 40$ y $g = 32.2\ pies/seg^2$

Determine:

a) ¿Cuánto tiempo le toma al objeto caer 128 pies?
b) ¿Cuánto tiempo le toma al objeto caer 256 pies?

Solución a)

Se sustituyen valores en 11.17 :

$$128 = 40\left[t - 40\left(\frac{1 - e^{\left(-\frac{32.2t}{40}\right)}}{32.2}\right)\right] \ldots 11.17$$

Realizando operaciones e igualando a cero:

$$40t - \frac{1600}{32.2}\left(1 - e^{(-0.805t)}\right) - 128 = 0 \ldots 11.18$$

Derivando:

$$40 - 40e^{-0.805t} = 0 \ldots 11.19$$

- Se inserta la función y se tabula desde 0 hasta 10 con un incremento de 0.2. Se pulsa Buscar Raíces y se presiona Siguiente para obtener el intervalo de la raíz. Se inserta la mitad del intervalo en x[0], una tolerancia de 0.00001 y se pulsa Calcular. Los resultados se muestran en la figura 11.9.

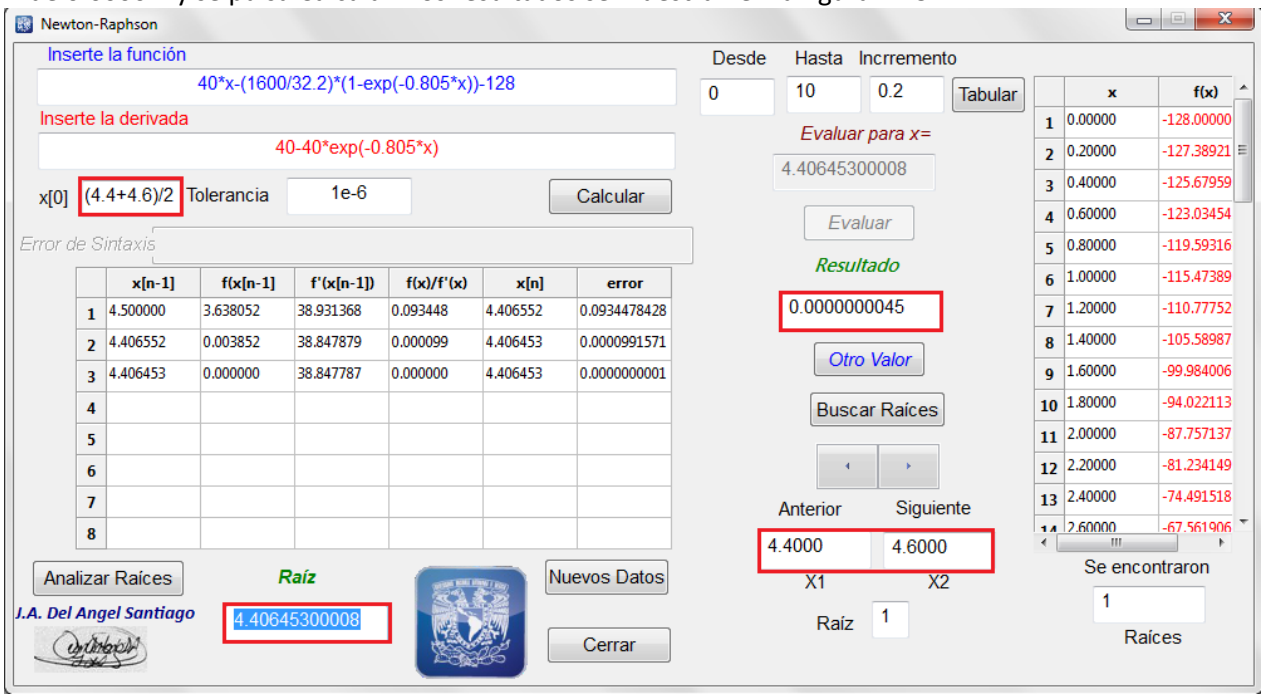


Figura 11.9 Resultados para el inciso a)

Figura 11.10 Resultados para el inciso b)

Solución b)
Se inserta la expresión 11.19 con la misma derivada y se siguen los mismos pasos que para el inciso a)

$$40t - \frac{1600}{32.2}\left(1 - e^{(-0.805t)}\right) - 256 = 0 \ldots 11.20$$

La solución se observa en la figura 11.10.
Por lo tanto tenemos:

$$a)\ t = 4.406453\ s \qquad b)\ t = 7.639585\ s$$

11.3 Descomposición LU

15. Corrientes y voltajes en circuito eléctricos.

Un problema común dentro de la ingeniería eléctrica involucra la determinación de corrientes y voltajes en varios puntos en circuitos de resistores. Estos problemas se resuelven usando las leyes para corrientes y voltajes de Kirchhoff. La regla para la corriente (o nodo) establece que la suma algebraica de todas las corrientes que entran a un nodo debe ser cero (véase figura 11.11a).

$$\sum i = 0 \ldots 11.21$$

Donde todas las corrientes que entran al nodo son de signo positivo. La regla para el voltaje (o malla) especifica que la suma algebraica de las diferencias de potencial (es decir, cambios en el voltaje) en cualquier ciclo debe ser igual a cero. Para un circuito de resistores esto se expresa como:

$$\sum \xi - \sum iR = 0 \ldots 11.22$$

Donde ξ es la fem (fuerza electromotriz) de las fuentes de voltaje, y R es la resistencia de cualquier resistor en la malla. Observe que el segundo término se deriva de la ley de Ohm (véase figura 11.11b), la cual establece que la caída de voltaje a través de un resistor ideal es igual al producto de la corriente y la resistencia. La ley de Kirchhoff para el voltaje es una expresión de la conservación de la energía.

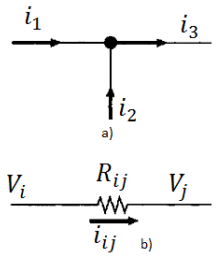


Figura 11.11 Representación de a)Las leyes de Kirchhoff y b)La ley de Ohm

Solución. La aplicación de estas reglas resulta en un sistema de ecuaciones algebraicas lineales, ya que son varios los ciclos o mallas que forman un circuito. Por ejemplo, considere el circuito mostrado en la figura 11.12.

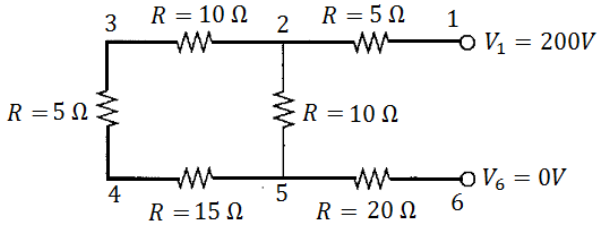


Figura 11.12 Un circuito de resistores que habrá de ser resuelto usando ecuaciones algebraicas lineales simultáneas.

Las corrientes asociadas con este circuito son desconocidas tanto en magnitud como en dirección. Esto no presenta gran dificultad, ya que simplemente se supone una dirección para cada corriente. Si la solución resultante a partir de las leyes de Kirchhoff es negativa, entonces la dirección supuesta fue incorrecta. Por ejemplo, en la figura 11.13 se muestran las direcciones supuestas de las corrientes.

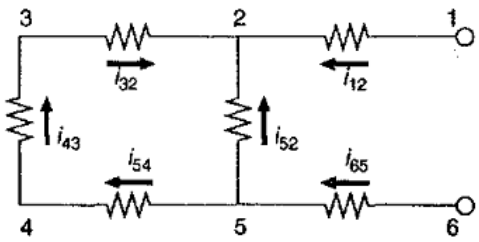


Figura 11.13 Corrientes supuestas.

Dadas estas suposiciones, la regla de la corriente de Kirchhoff se aplica a cada nodo para obtener:

$$i_{12} + i_{52} + i_{32} = 0$$
$$i_{65} - i_{52} - i_{54} = 0$$
$$i_{43} - i_{32} = 0$$
$$i_{54} - i_{43} = 0$$

La aplicación de la regla del voltaje en cada una de las mallas da:

$$-i_{54}R_{54} - i_{43}R_{43} - i_{32}R_{32} + i_{52}R_{52} = 0$$
$$-i_{65}R_{65} - i_{52}R_{52} - i_{12}R_{12} - 200 = 0$$

Si se sustituyen las resistencias de la figura 11.13 y se pasan las constantes al lado derecho:

$$-15i_{54} - 5i_{43} - 10i_{32} + 10i_{52} = 0$$
$$-20i_{65} - 10i_{52} - 5i_{12} = 200$$

Por tanto, el problema se reduce a la resolución del conjunto de seis ecuaciones con seis corrientes como incógnitas:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & -1 \\ 0 & 10 & -10 & 0 & -15 & -5 \\ 5 & -10 & 0 & -20 & 0 & 0 \end{bmatrix} \begin{bmatrix} i_{1,2} \\ i_{5,2} \\ i_{3,2} \\ i_{6,5} \\ i_{5,4} \\ i_{4,3} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 200 \end{bmatrix}$$

Se observa que la matriz tiene ceros en la diagonal principal lo cual da lugar a indeterminaciones al aplicar el método de LU, por lo que es necesario intercambiar los renglones 2 y 5 además de los renglones 4 y 6.

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 10 & -10 & 0 & -15 & -5 \\ 0 & 0 & -1 & 0 & 0 & 1 \\ 5 & -10 & 0 & -20 & 0 & 0 \\ 0 & -1 & 0 & 1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \begin{bmatrix} i_{1,2} \\ i_{5,4} \\ i_{3,2} \\ i_{4,3} \\ i_{5,2} \\ i_{6,5} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 200 \\ 0 \\ 0 \end{bmatrix}$$

Una vez que se tiene el sistema de esta forma se insertan los coeficientes de la matriz y los términos independientes en el programa de Descomposición LU seleccionando previamente el orden de 6x6 para obtener la solución del sistema que se muestra en la figura 11.14.

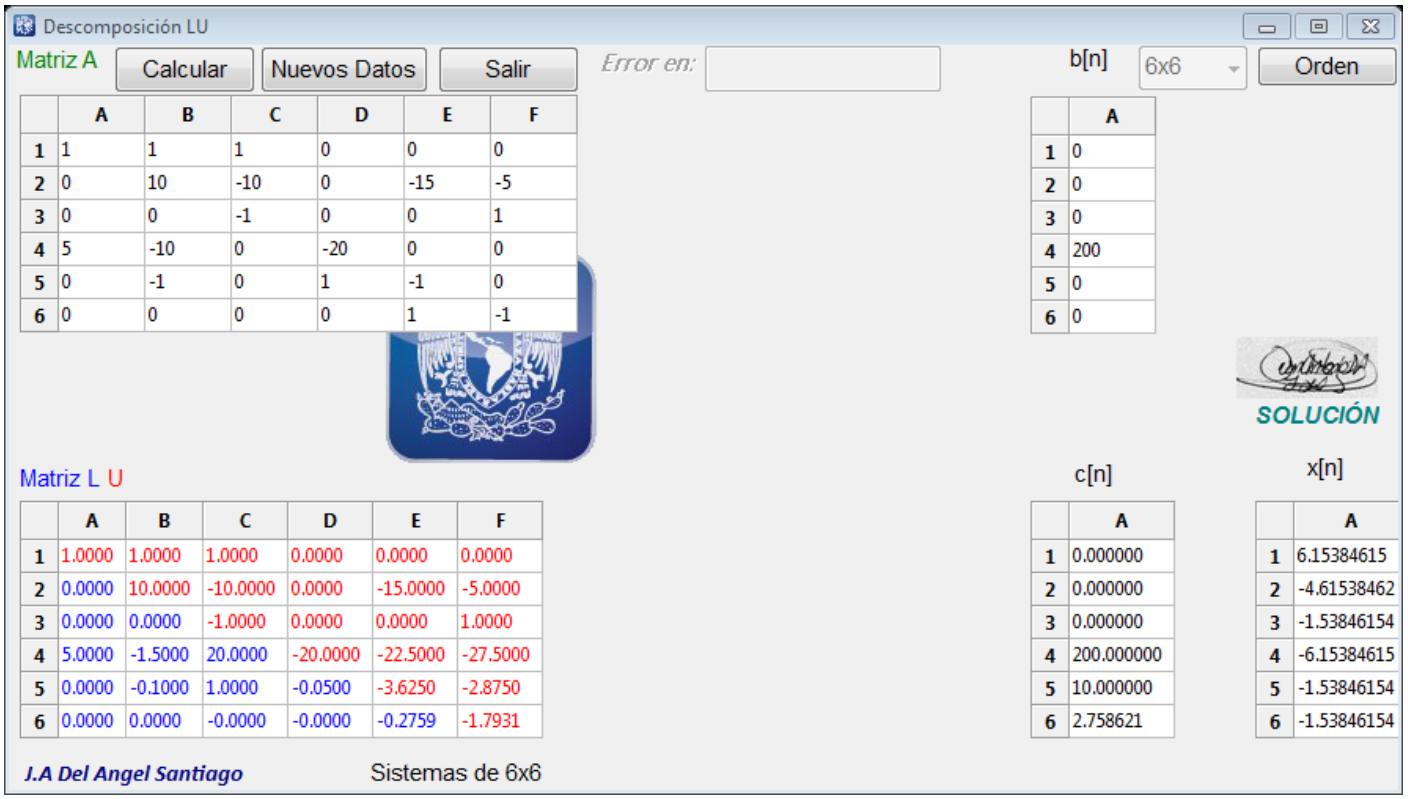


Figura 11.14 Solución del sistema de ecuaciones

Por lo tanto la solución del sistema es:

$$i_{12} = 6.15384 \quad i_{52} = -4.61538 \quad i_{32} = -1.53846 \quad i_{65} = -6.15384 \quad i_{54} = i_{43} = -1.53846$$

Así, con una adecuada interpretación de signos en el resultado, las corrientes y voltajes en el circuito se muestran en la figura11.15.

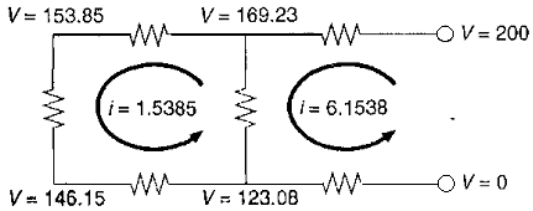


Figura 11.15 Solución obtenida para las corrientes y voltajes.

8. Determinar las corrientes y voltajes del circuito de la figura 11.26.

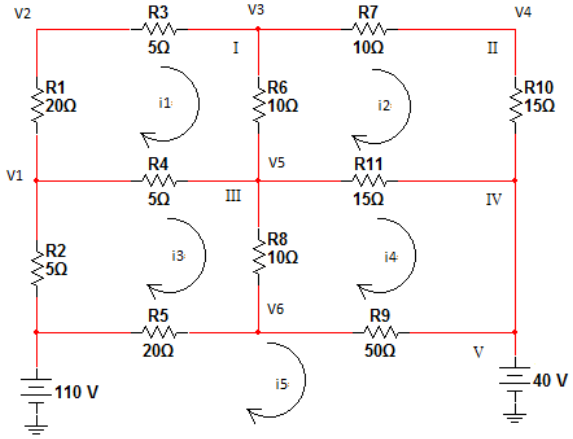


Figura 11.26

Solución:

Malla I

$$(20+5+10+5)i_1 - 10i_2 - 5i_3 = 0$$

Malla II

$$-10i_1 + (10+10+15+15)i_2 - 15i_4 = 0$$

Malla III

$$-5i_1 + (5+5+10+20)i_3 - 10i_4 - 20i_5 = 0$$

Malla IV

$$-15i_2 - 10i_3 + (10+15+50)i_4 - 50i_5 = 0$$

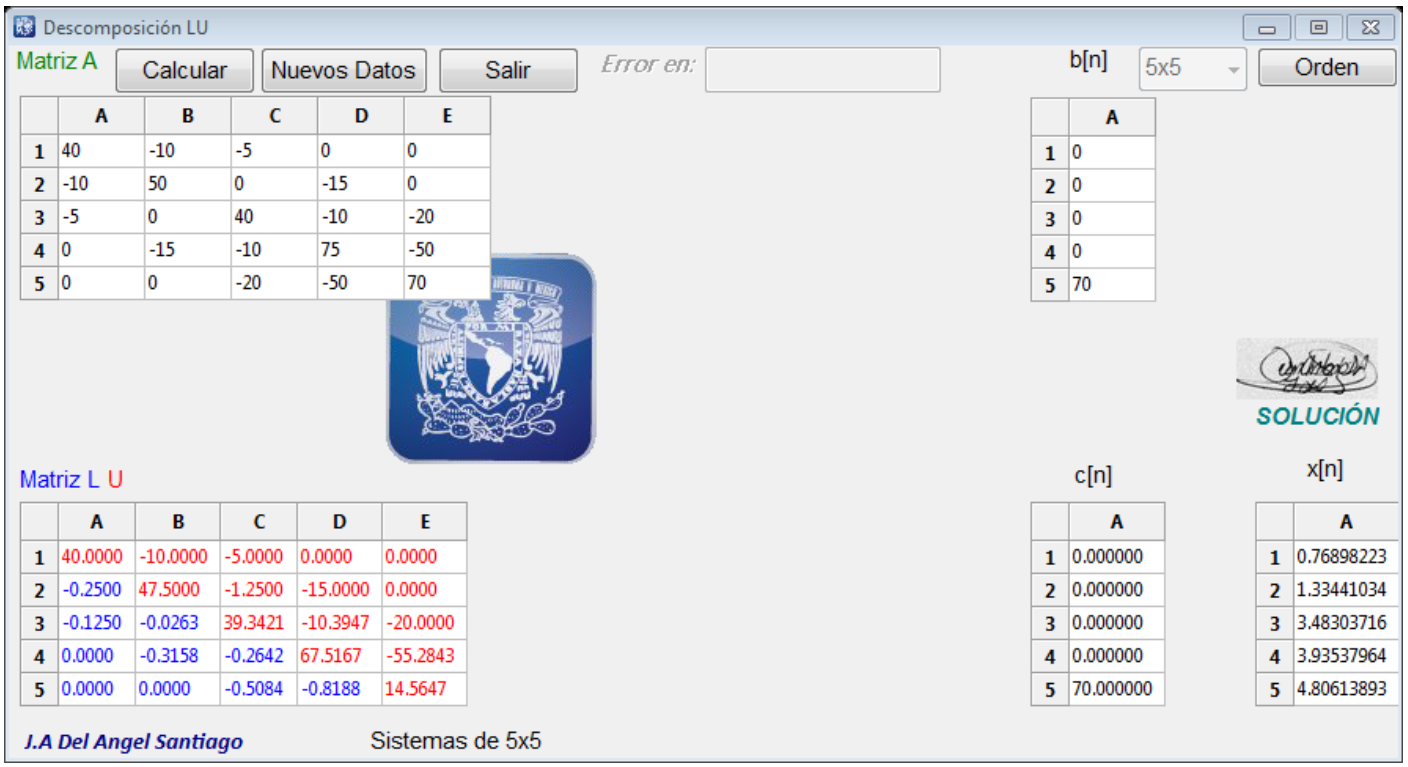
Malla V

$$-20i_3 - 50i_4 + (20+50)i_5 + 40 - 110 = 0$$

Simplificando y ordenando:

$$\begin{bmatrix} 40 & -10 & -5 & 0 & 0 \\ -10 & 50 & 0 & -15 & 0 \\ -5 & 0 & 40 & -10 & -20 \\ 0 & -15 & -10 & 75 & -50 \\ 0 & 0 & -20 & -50 & 70 \end{bmatrix} \begin{bmatrix} i1 \\ i2 \\ i3 \\ i4 \\ i5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 70 \end{bmatrix}$$

Se debe selecciona el orden de 5x5 de la lista de selección, se insertan los datos y se pulsa calcular.



	A	B	C	D	E
1	40	-10	-5	0	0
2	-10	50	0	-15	0
3	-5	0	40	-10	-20
4	0	-15	-10	75	-50
5	0	0	-20	-50	70

	A
1	0
2	0
3	0
4	0
5	70

	A	B	C	D	E
1	40.0000	-10.0000	-5.0000	0.0000	0.0000
2	-0.2500	47.5000	-1.2500	-15.0000	0.0000
3	-0.1250	-0.0263	39.3421	-10.3947	-20.0000
4	0.0000	-0.3158	-0.2642	67.5167	-55.2843
5	0.0000	0.0000	-0.5084	-0.8188	14.5647

	A
1	0.000000
2	0.000000
3	0.000000
4	0.000000
5	70.000000

	A
1	0.76898223
2	1.33441034
3	3.48303716
4	3.93537964
5	4.80613893

Por lo tanto las corrientes son:

$$i_1 = 0.7689A \quad i_2 = 1.3344A \quad i_3 = 3.483A \quad i_4 = 3.9353A \quad i_5 = 4.8061A$$

Cálculo de los voltajes:

$$V1 = 110 - 3.483(5) = 92.5848V$$
$$V2 = 92.585 - 0.7689(20) = 77.2051V$$
$$V3 = 77.205 - 0.7869(5) = 75.3602V$$
$$V4 = 75.3602 - 1.3344(10) = 60.0161V$$
$$V5 = 92.5848 - (3.483 - 0.7689)(5) = 79.0145V$$
$$V6 = 110 - (4.8061 - 3.483)(20) = 83.5379V$$

En la figura 11.27 se muestra el circuito con las corrientes y los voltajes encontrados.

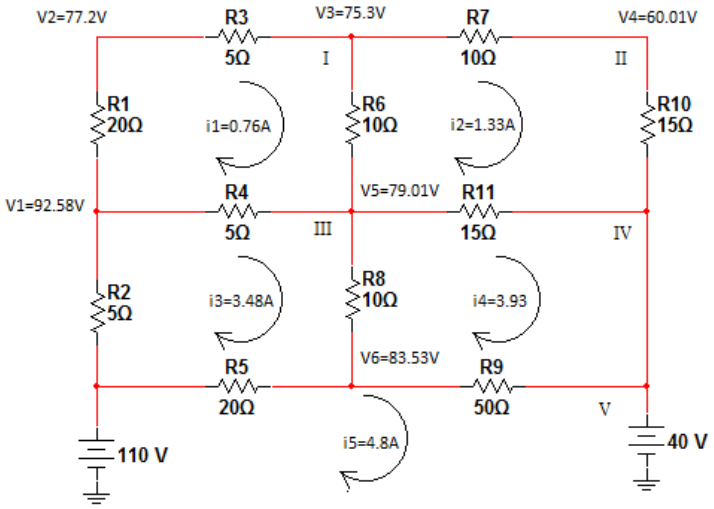
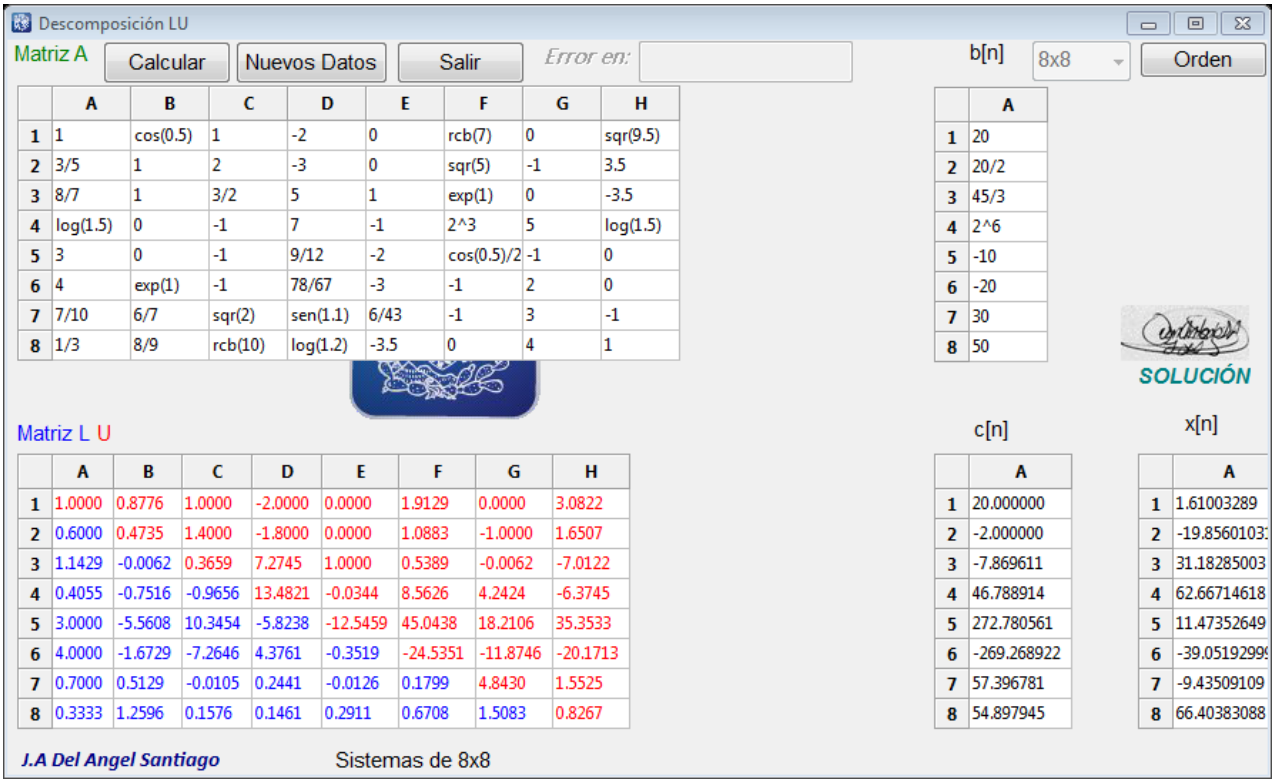


Figura 11.27

9. Resolver el sistema siguiente sistema de ecuaciones:

$$\begin{bmatrix} 1 & \cos(0.5) & 1 & -2 & 0 & \sqrt[3]{7} & 0 & \sqrt{9.5} \\ \frac{3}{5} & 1 & 2 & -3 & 0 & \sqrt{5} & -1 & 3.5 \\ \frac{8}{7} & 1 & \frac{3}{2} & 5 & 1 & & 0 & -3.5 \\ \ln(1.5) & 0 & -1 & 7 & -1 & 2^3 & 5 & \ln(1.5) \\ 3 & 0 & -1 & \frac{9}{12} & -2 & \frac{\cos(0.5)}{2} & -1 & 0 \\ 4 & e^1 & -1 & \frac{78}{67} & -3 & -1 & 2 & 0 \\ \frac{7}{10} & \frac{6}{7} & \sqrt{2} & sen(1.1) & \frac{6}{43} & -1 & 3 & -1 \\ \frac{1}{3} & \frac{8}{9} & \sqrt[3]{10} & \ln(1.2) & -3.5 & 0 & 4 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \\ x_6 \\ x_7 \\ x_8 \end{bmatrix} = \begin{bmatrix} 20 \\ \frac{20}{2} \\ \frac{45}{3} \\ 2^6 \\ -10 \\ -20 \\ 30 \\ 50 \end{bmatrix}$$

Solución:



Es necesario seleccionar el orden de 8x8 e insertar los datos con la sintaxis correcta. La solución es:

$$x_1 = 1.610003 \quad x_2 = -19.85601 \quad x_3 = 31.18285 \quad x_4 = 62.666714 \quad x_5 = 11.473526$$
$$x_6 = -39.051929 \quad x_7 = -9.435091 \quad x_8 = 66.40383$$

11.4 Interpolación de Lagrange.

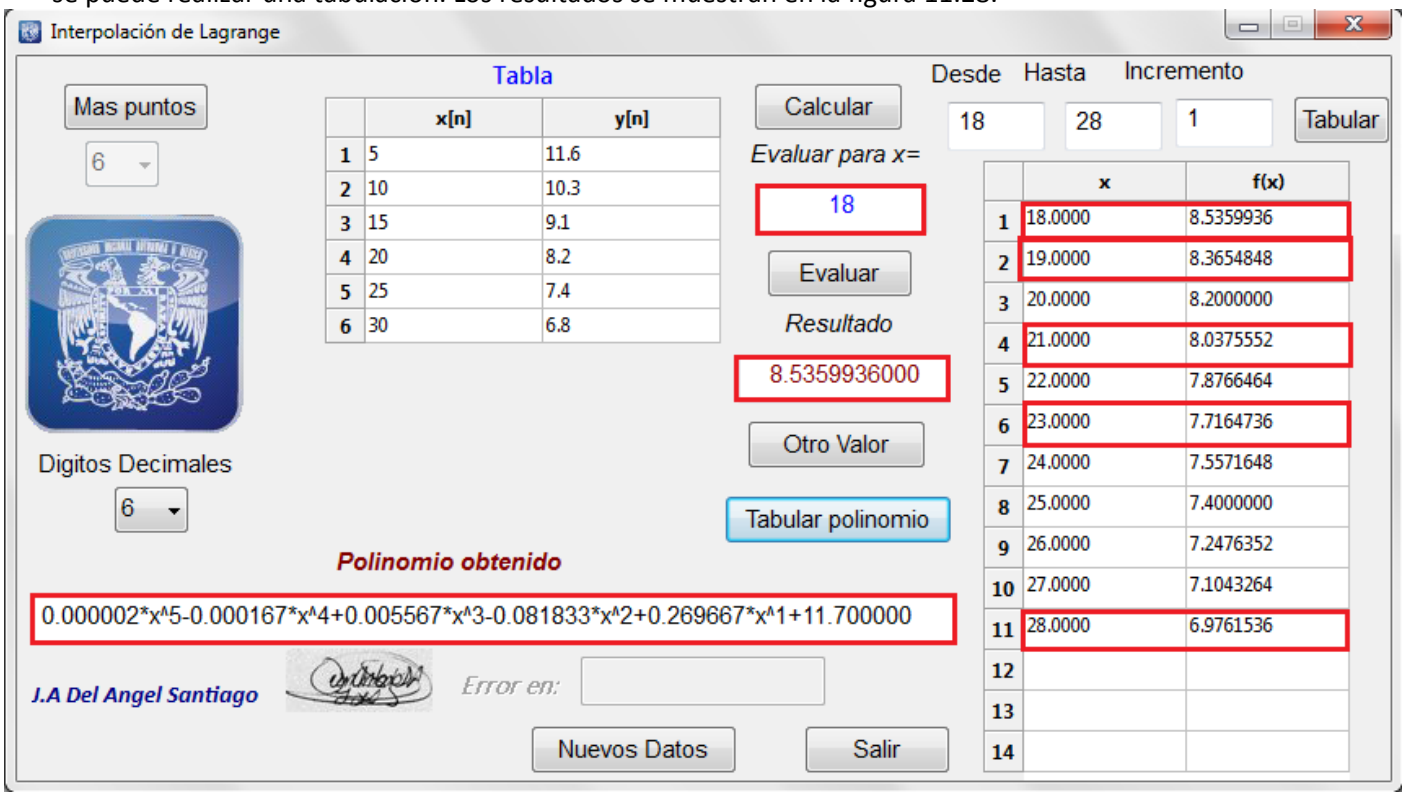
10. La concentración saturada de oxígeno disuelto en agua como una función de la temperatura y de la concentración de cloro se lista en la tabla 11.1. Obtenga el polinomio de interpolación para estimar el nivel de oxígeno disuelto para T=18 ,19, 21,23 y 28 ℃ con cloro = 10 000 mg/L.

T °C	Cloro=0mg/L	Cloro=10000mg/L	Cloro=20000mg/L
5	12.8	11.6	10.5
10	11.3	10.3	9.2
15	10	9.1	8.2
20	9	8.2	7.4
25	8.2	7.4	6.7
30	7.4	6.8	6.1

Tabla 11.1 Oxígeno disuelto (mg/L) para la concentración establecida de cloro y Temperatura.

Solución:

- Se debe seleccionar 6 puntos de la lista de selección e insertar los puntos. Para conocer los valores oxigeno se puede realizar una tabulación. Los resultados se muestran en la figura 11.28.



11. Se sospecha que grandes cantidades de tanino en las hojas de robles maduros, inhibe el crecimiento de la polilla de invierno, larva que daña extremadamente estos árboles en ciertos años. La tabla 11.2 muestra el peso promedio en *mg* de dos muestras de larvas en los primeros 28 días de haber nacido. La primera muestra de las hojas jóvenes de un roble, mientras que la segunda muestra fue tomada de las hojas maduras del mismo árbol.

a) Use la interpolación de Lagrange para aproximar la curva del peso promedio para cada muestra.
b) Encuentre una aproximación al valor máximo del peso promedio de cada muestra determinando el máximo del polinomio de interpolación.

Dia	0	6	10	13	17	20	28
Muestra 1	6.67	17.33	42.67	37.33	30.1	29.31	28.74
Muestra 2	6.67	16.11	18.89	15	10.56	9.44	8.89

Tabla 1.2

Solución:

Figura 11.28

En la figura 11.8 se muestran los polinomios obtenidos con 5 decimales, seleccionando 10 dígitos decimales se obtienen los polinomios:

Muestra 1:

$$P_6(x) = 0.0000409458x^6 - 0.0036716794x^5 + 0.1269023634x^4 - 2.0946390777x^3 + 16.1427243596x^2 - 42.6434808862x^1 + 6.6700000000$$

Para encontrar el valor máximo se deriva el polinomio y se encuentran sus raíces. Las raíces son:

$$x_1 = 1.95584 \;\; x_2 = 10.18854 \;\; x_3 = 25.96364$$

Evaluando en la segunda derivada:

$$P_6''(x_1) = 12.99 \quad P''_6(x_2) = -2.11 \quad P''_6(x_3) = 5.47$$

Por lo que existe un máximo en $x_2 = 10.18854$. Sustituyendo en $P_6(x)$ se obtiene $P_6(10.18854) = \mathbf{42.7084\ mg}$.

Muestra 2:

$$P_6(x) = 0.0000083616x^6 - 0.0007525462x^5 + 0.0258412834x^4 - 0.4137986507x^3 + 2.9128091018x^2 - 5.6782069578x^1 + 6.6700000000$$

Las raíces de la primera derivada son:

$$x_1 = 1.29558 \;\; x_2 = 8.76945 \;\; x_3 = 19.44577$$

Evaluando en la segunda derivada:

$$P_6''(x_1) = 3.097 \quad P''_6(x_2) = -0.766 \quad P''_6(x_3) = 1.027$$

Por lo que existe un máximo en $x_2 = 8.76945$. Sustituyendo en $P_6(x)$ se obtiene $P_6(10.18854) = \mathbf{19.4157mg}$.

11.5 Interpolación de Newton.

12. Dada una función $y = f(x)$ en forma tabular a menudo se desea encontrar un valor de x correspondiente a un valor dado de y, este proceso es llamado interpolación inversa, se lleva a cabo en la forma ya vista, pero intercambiando los papeles de x y y. Dada la información de la tabla 11.3 donde y es la amplitud de la oscilación de un péndulo largo en cm y x es el tiempo medido en minutos desde que empezó la oscilación.

Puntos	0	1	2	3	4	5	6
y	0	2.5	5	7.5	10	12.5	15
x	10	4.97	2.47	1.22	0.61	0.3	0.14

Tabla 11.3

Encuentre:

- El polinomio de aproximación de Newton de segundo grado que pasa por los puntos (1), (2) y (3).
- El polinomio de tercer grado que pasa por los puntos (0), (1), (2) y (3).
- El polinomio de quinto grado que pasa por los puntos 1 al 6.
- En todos los casos encontrar el valor de x correspondiente a y=2 cm.

Solución:


Interpolación de Newton
Polinomio obtenido
0.1000*x^2-1.7500*x^1+8.7200
Más puntos
Mas puntos 7 Diferencias Polinomio Nuevo Polinomio Tabular Polinomio Digitos decimales 3
Ajustar para i=
i
x: 0, 2.5, 5, 7.5, 10, 12.5, 15
y: 10, 4.97, 2.47, 1.22, 0.61, 0.3, 0.14
d^1fi: -2.0120000, -1.0000000, -0.5000000, -0.2440000, -0.1240000, -0.0640000
d^2fi: 0.2024000, 0.1000000, 0.0512000, 0.0240000, 0.0120000
d^3fi: -0.0136533, -0.0065067, -0.0036267, -0.0016000
d^4fi: 0.0007147, 0.0002880, 0.0002027
d^5fi: -0.0000341, -0.0000068
d^6fi: 0.0000018
A: 2, 3, 4
Interpolar para x= 2
Evaluar
Resultado 5.6200000
Error en:
J.A. Del Angel Santiago
Otro Valor
Nuevos Datos
Salir


Interpolación de Newton
Polinomio obtenido
-0.0137*x^3+0.3048*x^2-2.6887*x^1+10.0000
Más puntos
Mas puntos 7 Diferencias Polinomio Nuevo Polinomio Tabular Polinomio Digitos decimales 4
Ajustar para i=
i
x: 0, 2.5, 5, 7.5, 10, 12.5, 15
y: 10, 4.97, 2.47, 1.22, 0.61, 0.3, 0.14
d^1fi: -2.0120000, -1.0000000, -0.5000000, -0.2440000, -0.1240000, -0.0640000
d^2fi: 0.2024000, 0.1000000, 0.0512000, 0.0240000, 0.0120000
d^3fi: -0.0136533, -0.0065067, -0.0036267, -0.0016000
d^4fi: 0.0007147, 0.0002880, 0.0002027
d^5fi: -0.0000341, -0.0000068
d^6fi: 0.0000018
A: 1, 2, 3, 4
Interpolar para x= 2
Evaluar
Resultado 5.7326400
Error en:
J.A. Del Angel Santiago
Otro Valor
Nuevos Datos
Salir


Interpolación de Newton
Polinomio obtenido
-0.00000683*x^5+0.00054400*x^4-0.01733333*x^3+0.28460000*x^2-2.49540000*x^1+9.68000000
Más puntos
Mas puntos 7 Diferencias Polinomio Nuevo Polinomio Tabular Polinomio Digitos decimales 8
Ajustar para i=
6
i
x: 0, 2.5, 5, 7.5, 10, 12.5, 15
y: 10, 4.97, 2.47, 1.22, 0.61, 0.3, 0.14
d^1fi: -2.0120000, -1.0000000, -0.5000000, -0.2440000, -0.1240000, -0.0640000
d^2fi: 0.2024000, 0.1000000, 0.0512000, 0.0240000, 0.0120000
d^3fi: -0.0136533, -0.0065067, -0.0036267, -0.0016000
d^4fi: 0.0007147, 0.0002880, 0.0002027
d^5fi: -0.0000341, -0.0000058
d^6fi: 0.0000018
A: 2, 3, 4, 5, 6, 7
Interpolar para x= 2
Evaluar
Resultado 5.6974189
Error en:
J.A. Del Angel Santiago
Otro Valor
Nuevos Datos
Salir

Los resultados son:

$$P_2(y) = 0.1000 * y^2 - 1.7500y^1 + 8.7200 \qquad P_2(2) = 5.62.$$

$$P_3(y) = -0.0137y^3 + 0.3048y^2 - 2.6887y^1 + 10.0000 \qquad P_3(2) = 5.73264$$

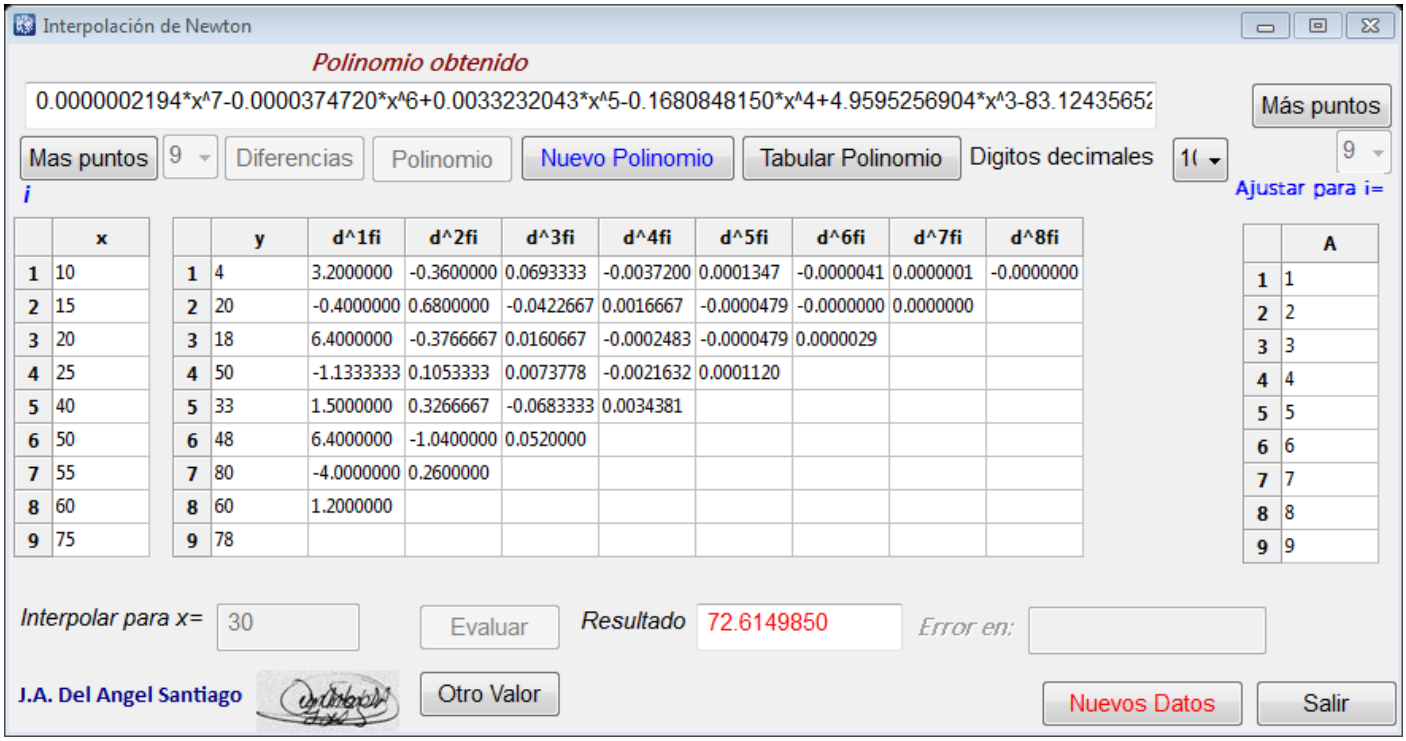
$$P_5(y) = -0.0000068y^5 + 0.0005440y^4 - 0.0173333y^3 + 0.2846y^2 - 2.4954y^1 + 9.68 \qquad P_5(2) = 5.6974189$$

13. Se sabe que la resistencia a la tensión de un plástico aumenta como una función del tiempo cuando se calienta. Se dispone de los siguientes datos:

Tiempo	10	15	20	25	40	50	55	60	75
R a la tensión	4	20	18	50	33	48	80	60	78

Ajuste un polinomio a los datos y utilice el polinomio para determinar la resistencia a la tensión en un tiempo de 30 minutos.

Solución:



El polinomio es:

$$P_8(x) = 0.0000002194x^7 - 0.0000374720x^6 + 0.0033232043x^5 - 0.1680848150x^4 + 4.9595256904x^3 - 83.1243565250x^2 + 724.0374347874x^1 - 2499.6080586081$$

$$\boldsymbol{P_8(30) = 72.6149850}$$

11.6 Interpolación Spline

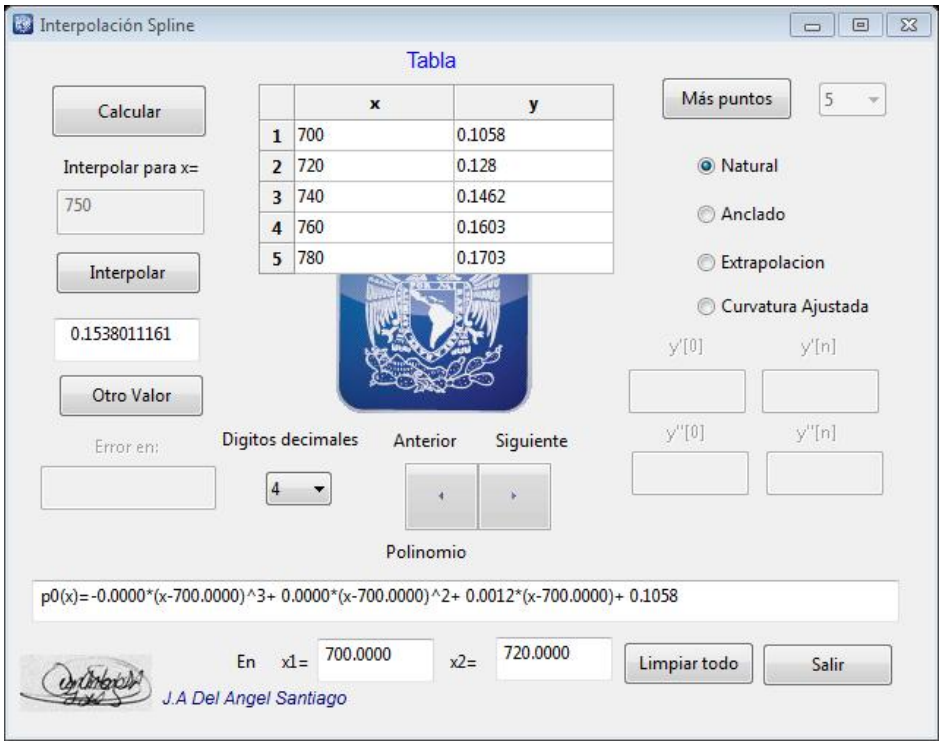
14. El volumen específico de un vapor sobrecalentado es enlistado en tablas de vapor para diferentes temperaturas. Por ejemplo, a una presión de 2 950 $lb/pulg^2$ absolutas:

T °F	700	720	740	760	780
V	0.1058	0.128	0.1462	0.1603	0.1703

a) Determine V para T= 750 °F usando un polinomio de Newton o Lagrange que se ajuste a todos los puntos.
b) Usando Interpolación Spline Natural.

Solución a)

Solución b)



Por lo tanto usando interpolación de Lagrange el valor calculado es $V = 0.15376$.
Usando interpolación Spline natural el valor calculado es: $V = 0.15380$.

15. Dados los puntos:

0	2	4	6	8	10	12	14	16	18
sen(0)	sen(2)	sen(4)	sen(6)	sen(8)	sen(10)	sen(12)	sen(14)	sen(16)	sen(18)

a) Grafique el polinomio de interpolación de Lagrange en la misma grafica que sen(x).

Solución:

El polinomio obtenido es:

$$P_7(x) = -0.0000446847x^7 + 0.0020145254x^6 - 0.0344150724x^5 + 0.2738320465x^4 - 0.9814241866x^3 + 1.1200115106x^2 + 0.4387022300x^1$$

b) Grafique los polinomios spline cúbicos en la misma grafica que sen(x).

Solución:

Los polinomios obtenidos son:

$$p0(x) = -0.1016291x + 0.8611652x$$

$$p1(x) = 0.1862210(x-2)^3 - 0.6097748(x-2)^2 - 0.3583843(x-2) + 0.9092974 \quad [0,2]$$

$$p2(x) = -0.0533942(x-4)^3 + 0.5075510(x-4)^2 - 0.5628318(x-4) - 0.7568025 \quad [2,4]$$

$$p3(x) = -0.1416567(x-6)^3 + 0.1871859(x-6)^2 + 0.8266420(x-6) - 0.2794155 \quad [4,6]$$

$$p4(x) = 0.1708286(x-8)^3 - 0.6627545(x-8)^2 - 0.1244952(x-8) + 0.9893582 \quad [6,8]$$

$$p5(x) = 0.0012149(x-10)^3 + 0.3622171(x-10)^2 - 0.7255698(x-10) - 0.5440211 \quad [8,10]$$

$$p6(x) = -0.1783252(x-12)^3 + 0.3695066(x-12)^2 + 0.7378777(x-12) - 0.5365729 \quad [10,12]$$

$$p7(x) = 0.1714081(x-14)^3 - 0.7004446(x-14)^2 + 0.0760017(x-14) + 0.9906074 \quad [14,16]$$

$$p8(x) = -0.0546673(x-16)^3 + 0.3280037(x-16)^2 - 0.6688802(x-16) - 0.2879033 \quad [16,18]$$

En la figura 11.29 se observa la gráfica de seno(x) y del polinomio de Lagrange y en la figura 11.30 se observa la gráfica de los polinomios Spline cúbicos y seno(x). Como se menciona en el capítulo 9 la interpolación polinomial no es una técnica adecuada para representar globalmente una función en un intervalo largo y esto se aprecia en la figura 11.29 debido a que el comportamiento del polinomio es muy diferente al de la función seno al principio y al final del intervalo mientras que solo el intervalo [6,14] se comporta de una manera aceptable. Por otra parte en la figura 11.30 se observa como los polinomios Spline se comportan de una manera más parecida a la función seno(x) en todo el intervalo.

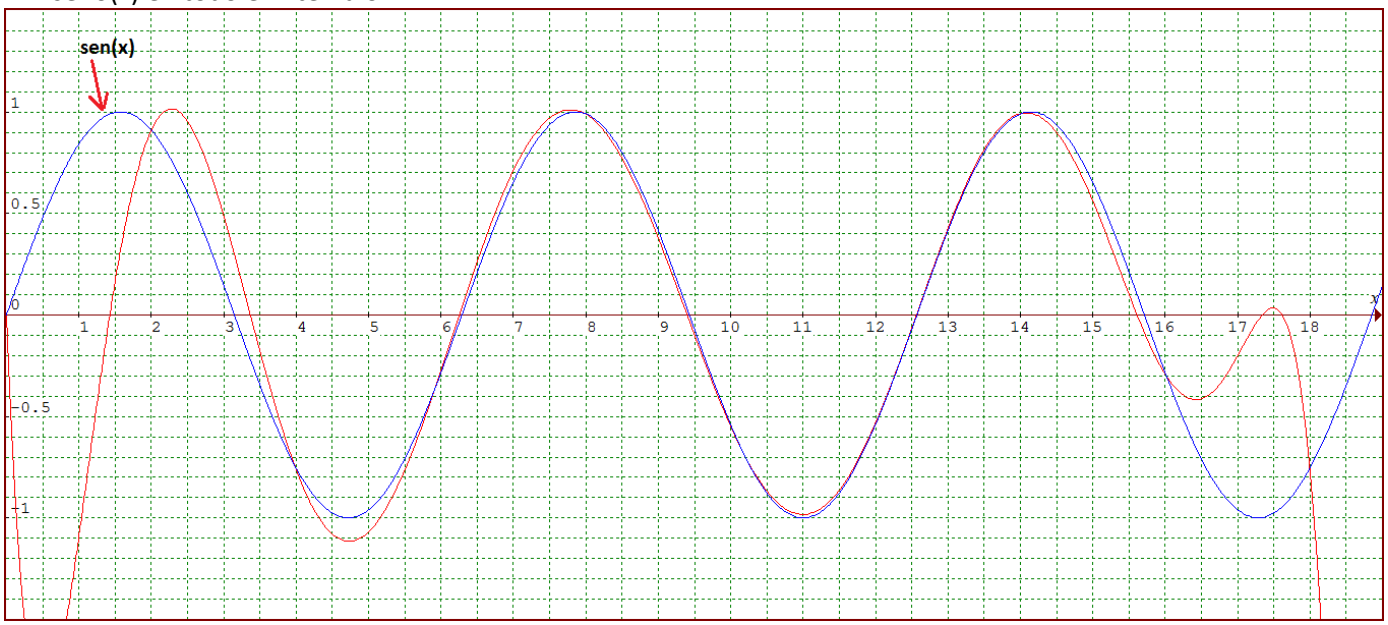


Figura 11.29 Grafica de seno(x) y el polinomio de Lagrange

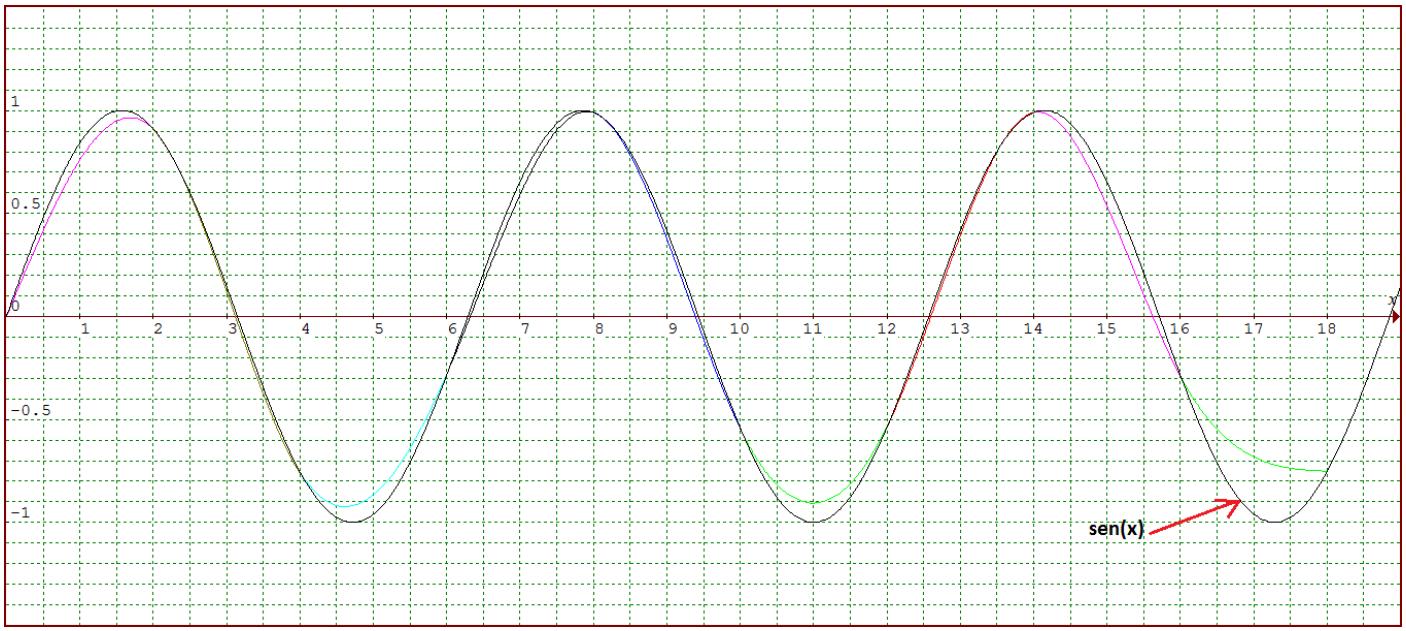


Figura 11.30 Gráfica de seno(x) y polinomios Spline cúbicos

16. Ajustar una cúbica Spline para los siguientes datos:

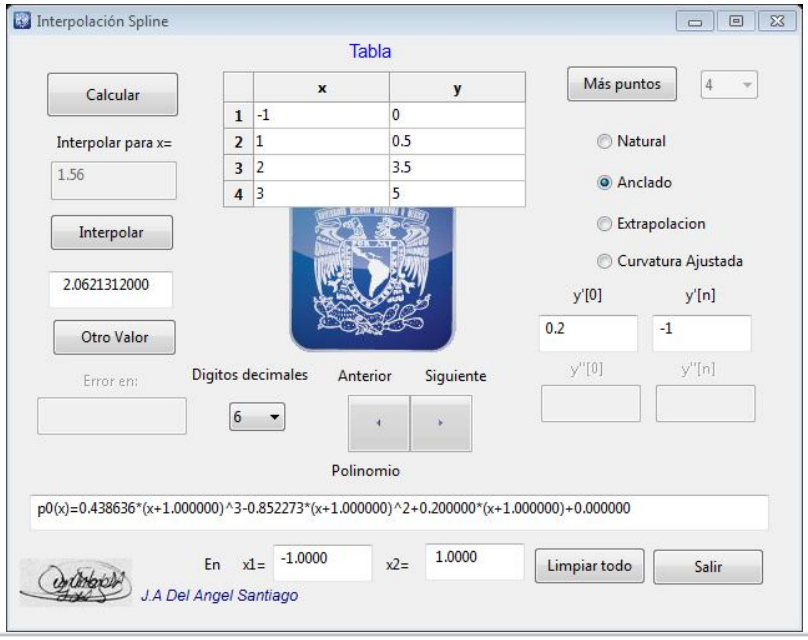
x	-1	1	2	3
f(x)	0	0.5	3.5	5

Con las condiciones inicial y final:

$$y(0) = 0.2 \quad y(1) = -1$$

Además interpole para x=1.56.

Solución: Se selecciona 4 de la lista de selección de puntos y se insertan los datos de los puntos, se selecciona el caso Anclado y se insertan las condiciones. A continuación se pulsa Calcular y se pulsa Siguiente 2 veces para obtener los tres polinomios.



p0(x)=0.438636*(x+1.000000)^3-0.852273*(x+1.000000)^2+0.200000*(x+1.000000)+0.000000

p1(x)=-0.834091*(x-1.000000)^3+1.779545*(x-1.000000)^2+2.054545*(x-1.000000)+0.500000

p2(x)=-0.888636*(x-2.000000)^3-0.722727*(x-2.000000)^2+3.111364*(x-2.000000)+3.500000

$$p1(1.56) = 2.0621312$$

11.7 Integración Numérica.

17. El valor promedio de una corriente eléctrica oscilatoria en un periodo puede ser cero. Por ejemplo, suponga que la corriente es descrita por una senoidal simple: $i(t) = sen(2\pi)/T$, donde T es el periodo. El valor promedio de esta función se puede determinar por la siguiente ecuación:

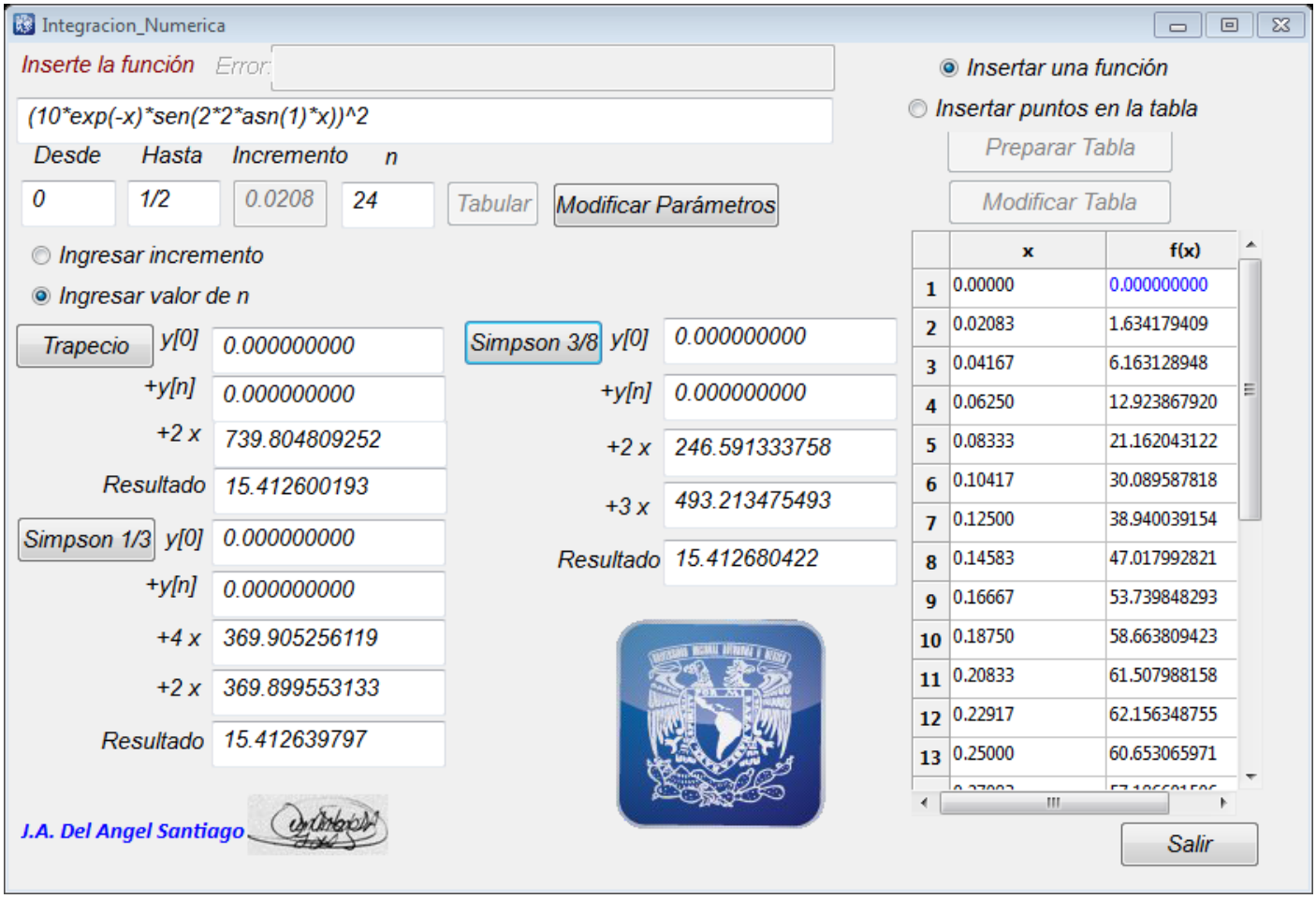
$$i = \frac{\int_0^T sen\left(\frac{2\pi}{T}\right)}{T-0} = \frac{-\cos(2\pi)+\cos(0)}{T}$$

A pesar del hecho de que el resultado neto es cero, tal corriente es capaz de realizar trabajo y generar calor. Por tanto, los ingenieros eléctricos a menudo caracterizan esa corriente por:

$$I_{RMS} = \sqrt{\frac{1}{T}\int_0^T i^2(t)dt}$$

Donde $i(t)$ = corriente instantánea. Calcule la I_{RMS} de la forma de onda de la figura 11.31 Utilizando la regla de Simpson 1/3, Simpson 3/8 y Trapezoidal para n=60. Calcule el error para cada caso comparando con el valor verdadero de 3.92588946.

Solución:



- Trapezoidal $I_{RMS} = \sqrt{15.412600193} = 3.925888459 \quad Error = 1.001 \times 10^{-6}$
- Simpson 1/3 $I_{RMS} = \sqrt{15.412600193} = 3.925888459 \quad Error = -4.043 \times 10^{-6}$
- Simpson 3/8 $I_{RMS} = \sqrt{15.412680422} = 3.925888459 \quad Error = -9.217 \times 10^{-6}$

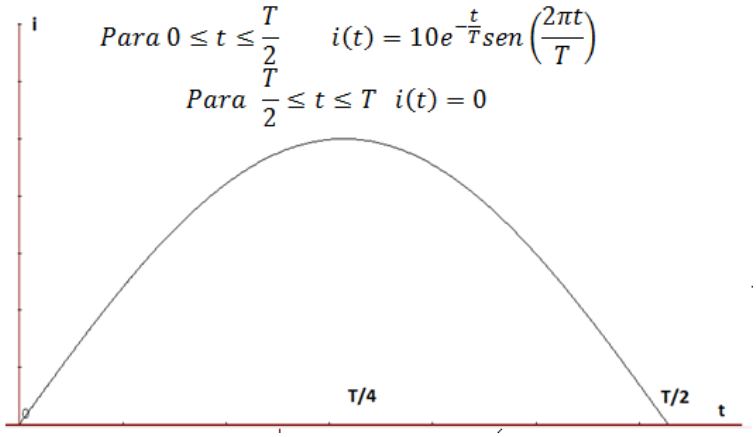
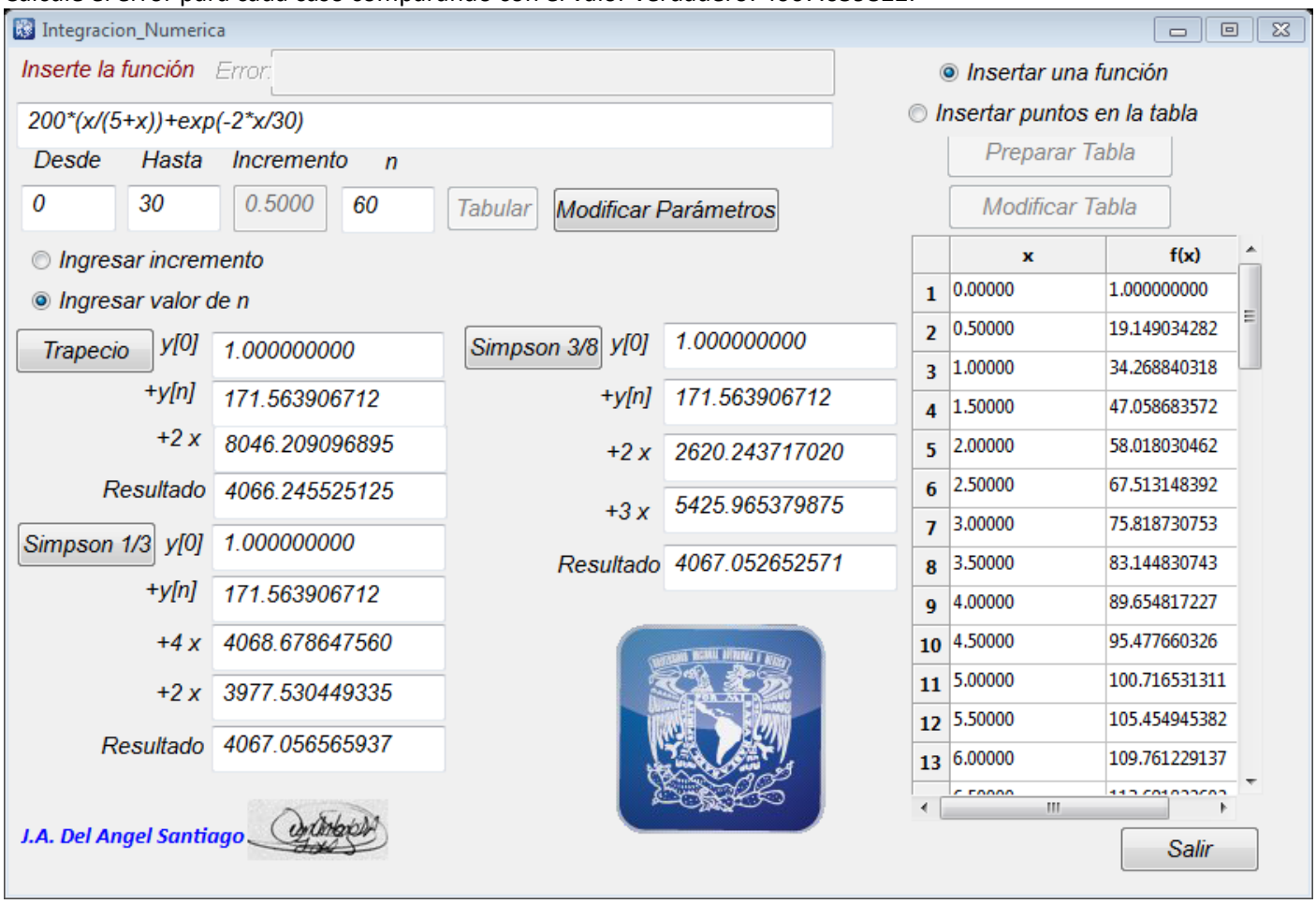


Figura 11.31

18. Realice la integral con n=60:

$$\int_0^{30} 200\left(\frac{z}{5+z}\right) e^{-2z/30}\, dz$$

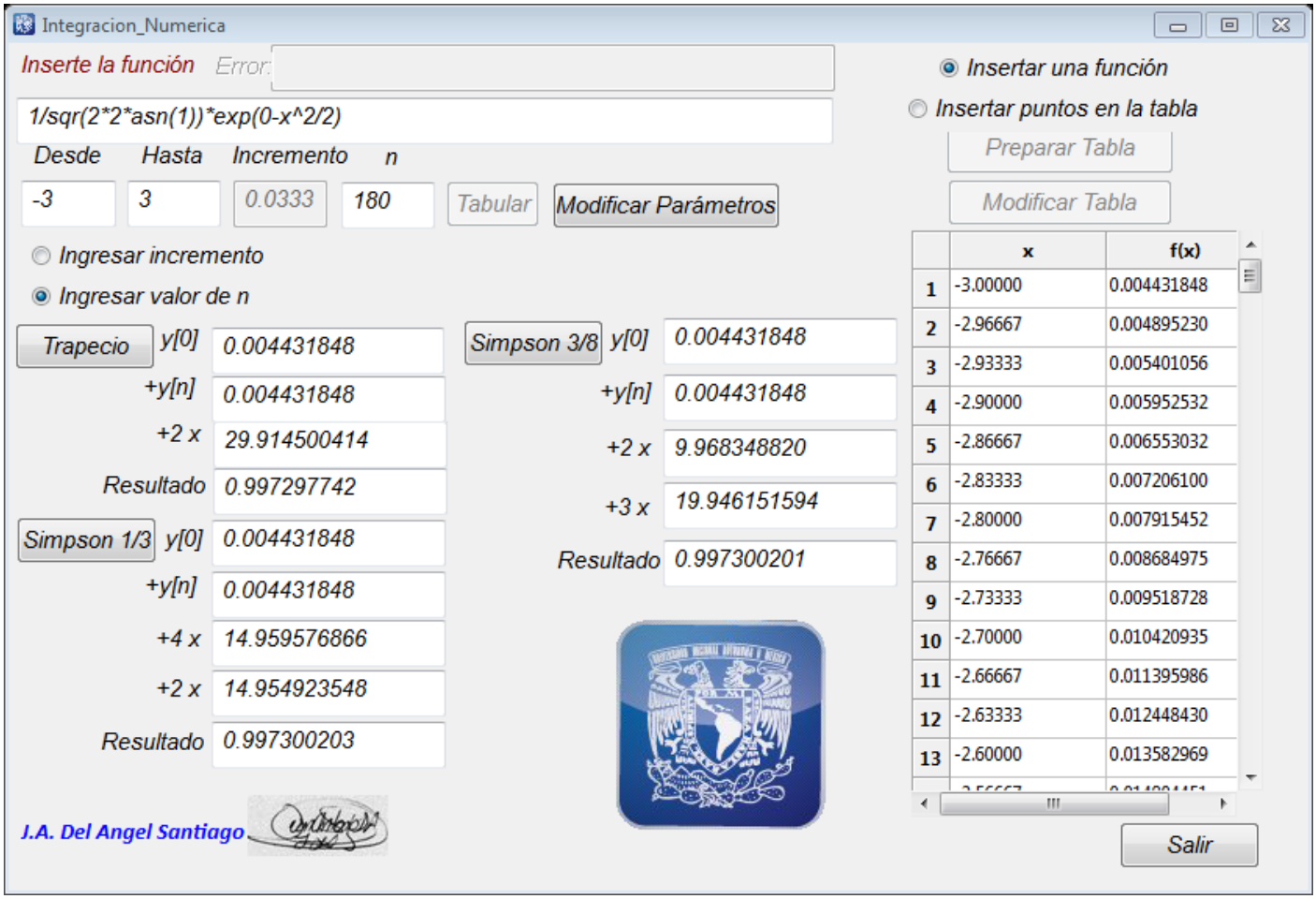
Calcule el error para cada caso comparando con el valor verdadero: 4067.059822.



Trapezoidal: $Error = 4067.059822 - 4066.245525125 = 0.814297$
Simpson 1/3: $Error = 4067.059822 - 4067.056565937 = 0.003256$
Simpson 3/8: $Error = 4067.059822 - 4067.052652571 = 0.007169$

19. Realice la integral con n=180 y obtenga el error comparando con el valor verdadero: 0.9973204

$$\int_{-3}^{3} \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$



$$Trapezoidal \quad Error = 0.9973204 - 0.997297742 = 2.4619 \times 10^{-6}$$

$$Simpson\frac{1}{3} \quad Error = 0.9973204 - 0.997300203 = 9 \times 10^{-10}$$

$$Simpson\frac{3}{8} \quad Error = 0.9973204 - 0.997300201 = 2.9 \times 10^{-9}$$

11.9 Runge Kutta de Cuarto Orden.

20. Resolver la ecuación diferencial:

$$\frac{dy}{dt} = te^{3t} - 2y \qquad para \quad 0 \le t \le 1 \quad con \quad y(0) = 0$$

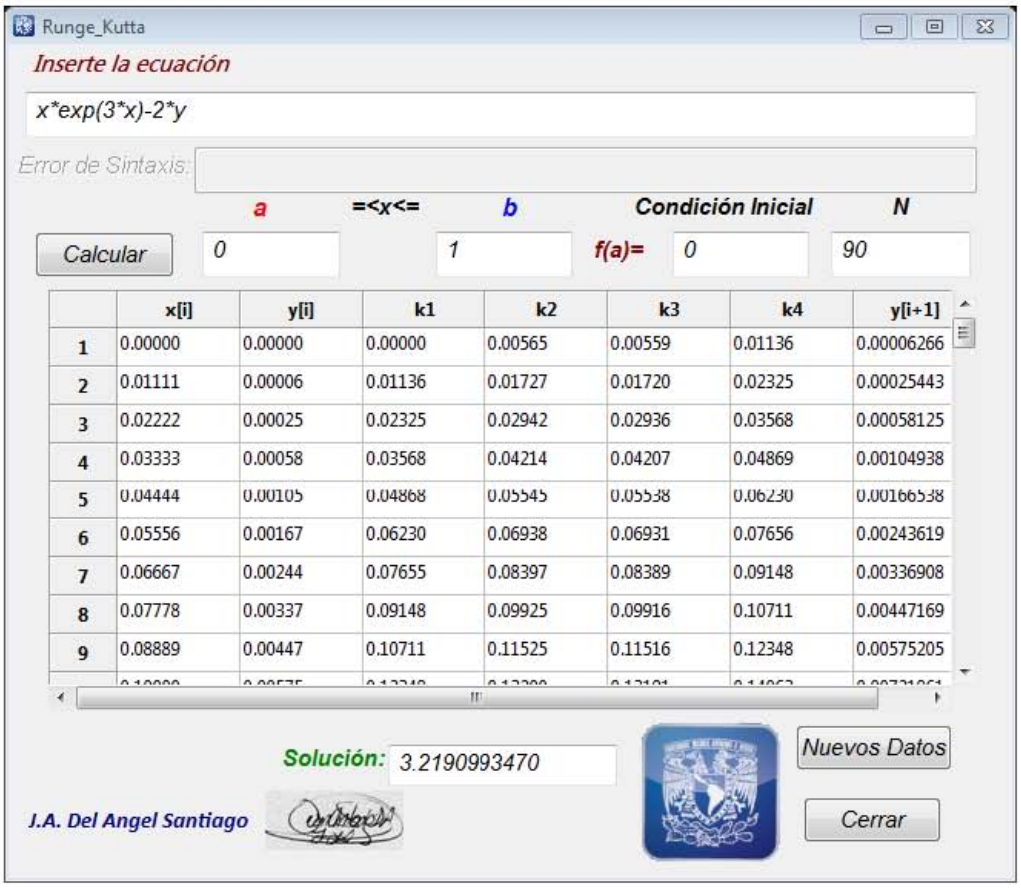
21. Resolver la ecuación diferencial.

$$\frac{dy}{dt} = 1 + (t - y)^2 \quad para \quad 2 \le t \le 3 \quad con \quad y(2) = 1$$

22. Resolver la ecuación diferencial:

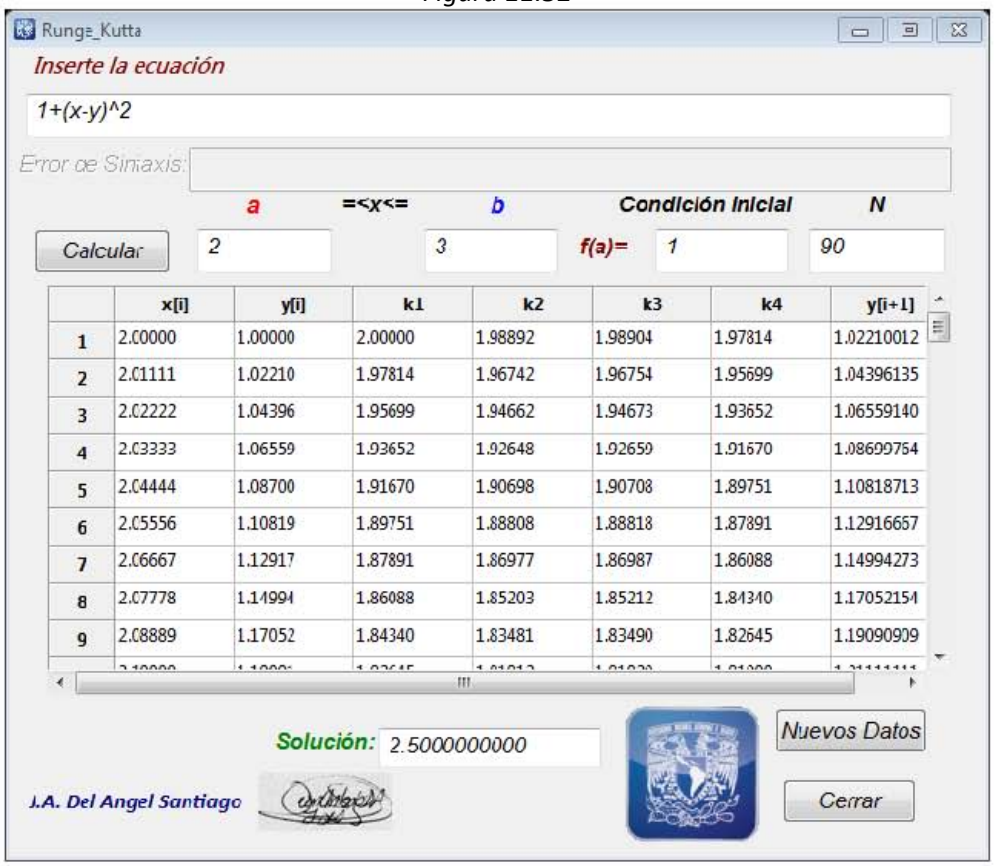
$$\frac{dy}{dt} = ysen^2(t) \quad para \quad 0 \le t \le 3 \quad con\, y(0) = 1$$

Los resultados se muestran en las figuras 11.32, 11.33 y 11.34.



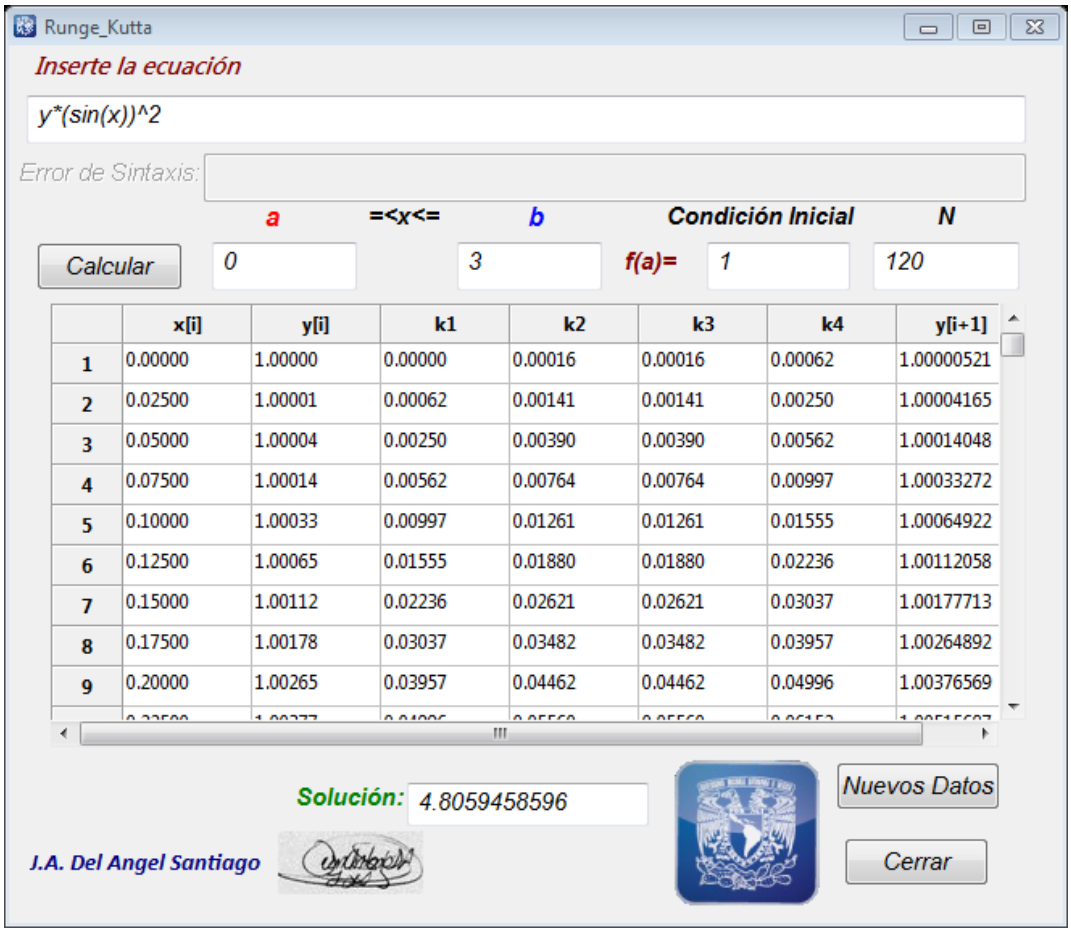
	x[i]	y[i]	k1	k2	k3	k4	y[i+1]
1	0.00000	0.00000	0.00000	0.00565	0.00559	0.01136	0.00006266
2	0.01111	0.00006	0.01136	0.01727	0.01720	0.02325	0.00025443
3	0.02222	0.00025	0.02325	0.02942	0.02936	0.03568	0.00058125
4	0.03333	0.00058	0.03568	0.04214	0.04207	0.04869	0.00104938
5	0.04444	0.00105	0.04868	0.05545	0.05538	0.06230	0.00166538
6	0.05556	0.00167	0.06230	0.06938	0.06931	0.07656	0.00243619
7	0.06667	0.00244	0.07655	0.08397	0.08389	0.09148	0.00336908
8	0.07778	0.00337	0.09148	0.09925	0.09916	0.10711	0.00447169
9	0.08889	0.00447	0.10711	0.11525	0.11516	0.12348	0.00575205

Figura 11.32



	x[i]	y[i]	k1	k2	k3	k4	y[i+1]
1	2.00000	1.00000	2.00000	1.98892	1.98904	1.97814	1.02210012
2	2.01111	1.02210	1.97814	1.96742	1.96754	1.95699	1.04396135
3	2.02222	1.04396	1.95699	1.94662	1.94673	1.93652	1.06559140
4	2.03333	1.06559	1.93652	1.92648	1.92659	1.91670	1.08699754
5	2.04444	1.08700	1.91670	1.90698	1.90708	1.89751	1.10818713
6	2.05556	1.10819	1.89751	1.88808	1.88818	1.87891	1.12916657
7	2.06667	1.12917	1.87891	1.86977	1.86987	1.86088	1.14994273
8	2.07778	1.14994	1.86088	1.85203	1.85212	1.84340	1.17052154
9	2.08889	1.17052	1.84340	1.83481	1.83490	1.82645	1.19090909

Figura 11.33



	x[i]	y[i]	k1	k2	k3	k4	y[i+1]
1	0.00000	1.00000	0.00000	0.00016	0.00016	0.00062	1.00000521
2	0.02500	1.00001	0.00062	0.00141	0.00141	0.00250	1.00004165
3	0.05000	1.00004	0.00250	0.00390	0.00390	0.00562	1.00014048
4	0.07500	1.00014	0.00562	0.00764	0.00764	0.00997	1.00033272
5	0.10000	1.00033	0.00997	0.01261	0.01261	0.01555	1.00064922
6	0.12500	1.00065	0.01555	0.01880	0.01880	0.02236	1.00112058
7	0.15000	1.00112	0.02236	0.02621	0.02621	0.03037	1.00177713
8	0.17500	1.00178	0.03037	0.03482	0.03482	0.03957	1.00264892
9	0.20000	1.00265	0.03957	0.04462	0.04462	0.04996	1.00376569

Figura 11.34

Comparando los resultados de este programa con los de Maple el error es para cada caso:

$$Error = 3.219099319 - 3.2190993470 = -2.8 \times 10^{-8}$$
$$Error = 2.5 - 2.5 = 0$$
$$Error = 4.805945874 - 4.8059458596 = 1.4 \times 10^{-8}$$

12. Tiempo de ejecución de los programas.

En una computadora portátil con un procesador AMD(E-300) APU de 1.3 GHz y 2 GB de memoria RAM se pudo observar lo siguiente en cuánto al tiempo que toman los programas para mostrar resultados.

- Al Analizador de Raíces le toman alrededor de 2 segundos tabular 1000 datos, la demora se debe a que para cada valor sustituido hace uso de dos de los métodos del evaluador de expresiones algebraicas.

A excepción de la tabulación mencionada las siguientes acciones toman alrededor de medio segundo en ser ejecutadas.

- Búsqueda de Raíces en una tabulación de 1000 valores.
- Cálculo una raíz con un error mínimo permitido de $EMP = 1 \times 10^{-11}$ con Bisección o Newton Raphson.
- Resolver un sistema de ecuaciones de 10x10 con descomposición LU
- Obtención de un polinomio de orden nueve con Interpolación de Lagrange o de Newton.
- Cálculo de cada Spline cúbico con Interpolación Spline.
- Integración con cada regla incluida en el programa de Integración Numérica.
- Solución de una ecuación diferencial ordinaria con N=100 en el programa Runge Kutta de cuarto orden.

En una computadora con características superiores a las mencionadas la velocidad mencionada se eleva.

CONCLUSIONES.

El estudio de los métodos numéricos hace evidente la necesidad de programas que realicen cientos de iteraciones en poco tiempo para poder obtener buenas aproximaciones de soluciones de problemas que se presentan en diferentes áreas de la ingeniería. Los programas obtenidos se apegan a esta necesidad además de que muestran los resultados parciales de cada método, de tal manera que los estudiantes de distintas carreras que cursan la asignatura de métodos numéricos puedan comprobar los resultados en cada paso de la aplicación de los métodos.

Durante el desarrollo de cada uno de los programas encontré la necesidad de herramientas como el multiplicador de polinomios implementado en interpolación de Lagrange y Newton que no existen como tales en el lenguaje C++ pero que me di a la tarea de desarrollar y explicar detalladamente su funcionamiento para que los estudiantes que necesiten de ellas puedan consultar el código y aplicarlo a sus programas.

Por otra parte las habilidades de programación obtenidas en la creación los programas complementan mi formación como Ingeniero Mecánico Electricista del módulo de sistemas digitales debido a que existen microcontroladores que manejan el lenguaje de programación C como herramienta para su programación. Las técnicas de programación aquí aprendidas ayudan además a desarrollar una mayor habilidad de creación e implementación de algoritmos independientemente del lenguaje de programación.

Las aplicaciones con interfaz gráfica pueden ser de gran utilidad para dar solución a distintas situaciones que se puedan presentar en la ingeniería no solo en el caso de los métodos numéricos. Por esta razón se ha detallado la elaboración de cada programa con la intención de que los estudiantes con conocimientos básicos de programación puedan aprender a desarrollar este tipo de aplicaciones que no usan gran cantidad de componentes pero pueden llegar a ser de gran utilidad.

Es importante mencionar que la primera etapa del desarrollo de los programas fue realizada en Maple 14 que maneja un lenguaje de programación orientado a modelos matemáticos y facilita la programación de los métodos numéricos tratados en esta tesis. Después de comprobar la funcionalidad de cada método numérico en Maple se implementó y adaptó el código a WxDevC++, entorno que fue elegido debido a que es software libre al igual que el Evaluador de expresiones algebraicas que forma una parte muy importante de cada uno de los programas creados.

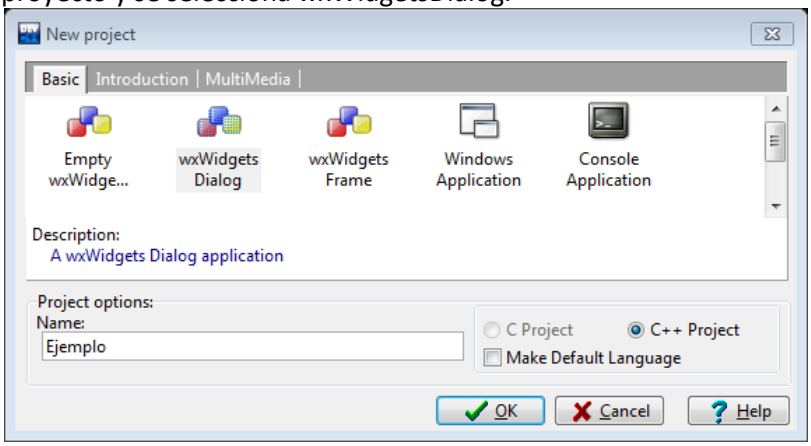
En cada programa se hizo especial énfasis en lo errores que el usuario puede cometer al usar los programas con la finalidad de que los programas sean robustos ante errores de sintaxis y errores que dependen de la naturaleza de cada método además con el objetivo de que los programas sean confiables. La revisión de sintaxis hace que el código de los programas aumente considerablemente de tamaño pero es necesario para garantizar la funcionalidad de los programas para que además puedan insertarse expresiones matemáticas como parámetros y no perder dígitos decimales.

Al desarrollar el capítulo de aplicación de los programas pude constatar que los programas son fáciles de utilizar y proporcionan muy buenas aproximaciones a la solución de los problemas debido a que cada solución obtenida fue comparada con la solución obtenida con Maple 14. Por lo anterior se cumplió el objetivo de que los programas tengan tanto un uso didáctico como un uso preciso. Finalmente cabe mencionar que el objetivo principal de incluir los apéndices es que los estudiantes interesados en reproducir los programas o desarrollar sus propios programas no tengan alguna dificultad en hacerlo.

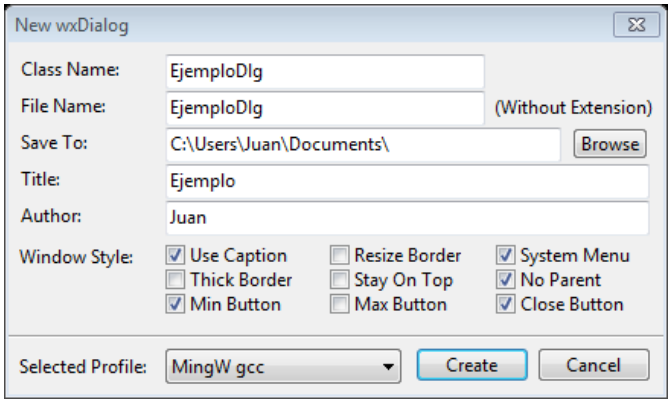
APÉNDICE A. Crear una aplicación con ventana en WxDevC++ y renombrar componentes.

Para crear una aplicación con ventana se procede de la siguiente manera.

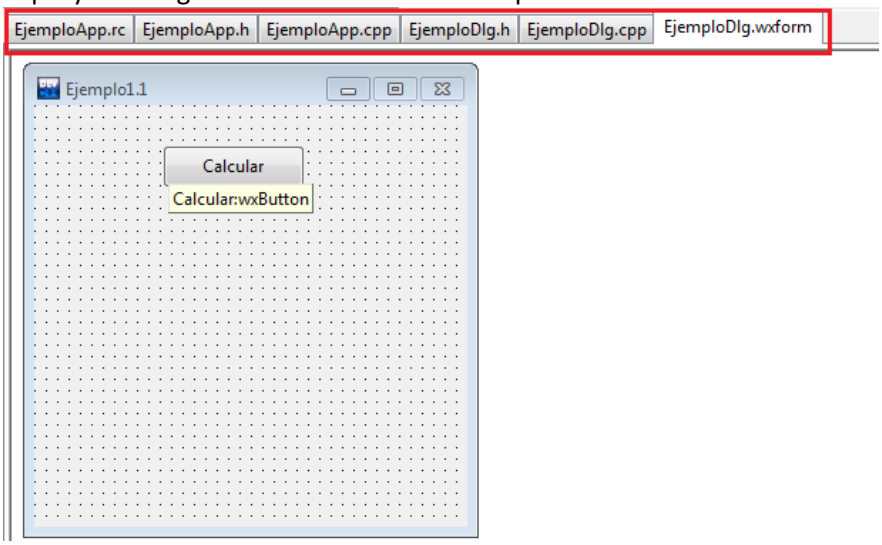
1. Se ejecuta WxDevC++.
2. Se pulsa en nuevo proyecto y se selecciona wxWidgetsDialog.



3. Se abre una ventana para seleccionar la ubicación donde se guardará el proyecto.
4. A continuación se muestran algunas opciones. Se observa que la casilla de verificación Min Button esta activada, lo que quiere decir que cuando se cree nuestra ventana contará con el botón Minimizar en la esquina superior derecha. Los programas explicados en esta tesis han sido creados sin modificar las opciones que se muestran, solo se pulsa *Create.*

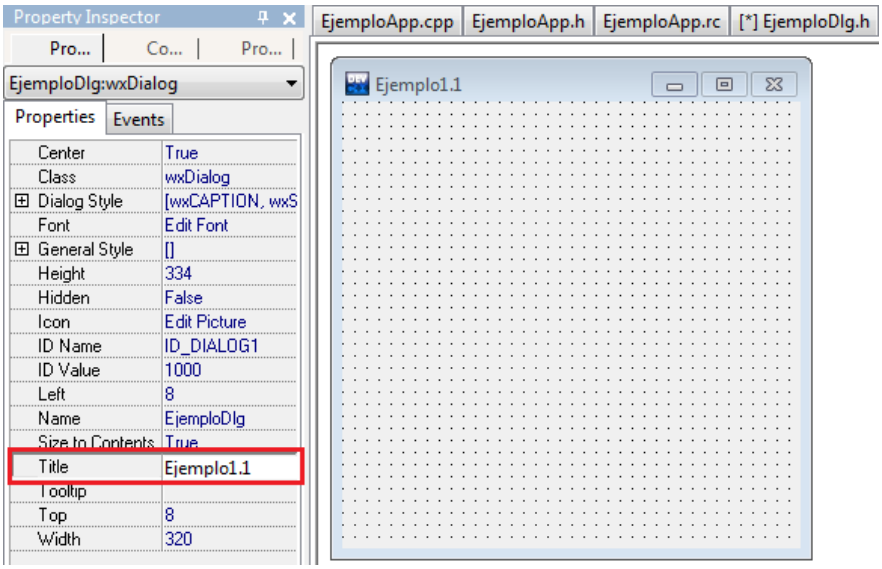


5. Cuando se crea el proyecto se generan 6 archivos como se puede observar:

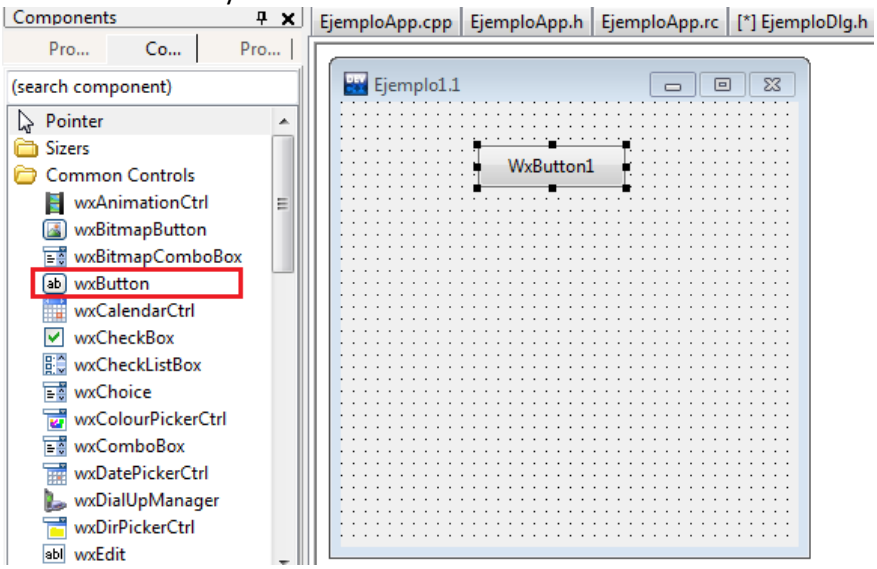


Ejemplo.wxform corresponde al diseño y EjemploDlg.cpp corresponde al código del diseño.

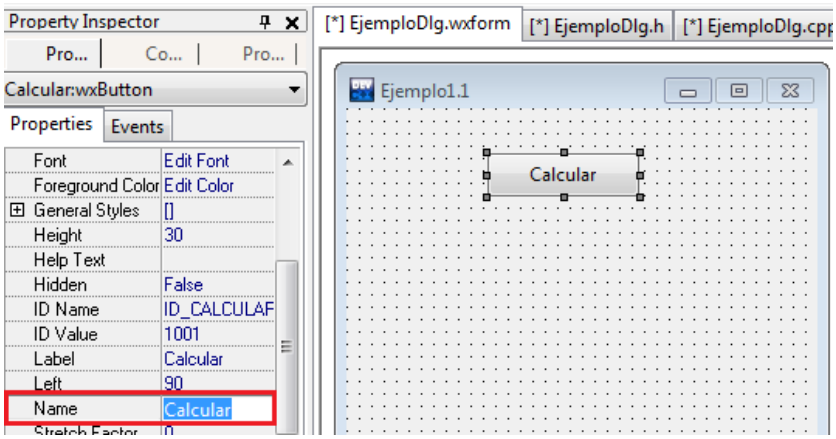
6. Una vez creado el proyecto se puede cambiar el nombre del programa y otras propiedades como como se indica.



7. Pulsando en la pestaña *Components* se muestra la lista de componentes que pueden ser utilizados. En este caso se ha seleccionado wxButton y se ha colocado en el diseño.



8. Para renombrar el componente se da click sobre el componente y a continuación en la pestaña *Property Inspector* se busca la propiedad **Name.** En este caso se ha colocado el nombre *Calcular* y a continuación se presiona enter.



9. Para insertar el código que ejecutará el botón se debe dar doble click sobre el componente. En este caso al dar doble click sobre un componente wxButton se crea automáticamente código del *evento Click* donde insertaremos nuestro código y automáticamente nos coloca sobre las siguientes líneas de código.

```
/*
 * CalcularClick
 */
void EjemploDlg::CalcularClick(wxCommandEvent& event)
{
    // insert your code here
}
```

En la línea nos indica donde insertar nuestro código. De esta manera se ha insertado el código presentado al final de cada capítulo

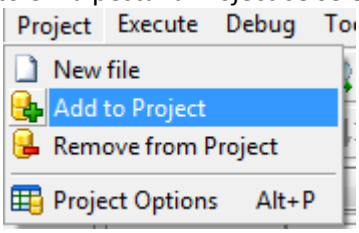
Se ha resaltado **CalcularClick** ya que si no se cambiará el nombre del componente aparecerían las líneas:

```
/*
 * CalcularClick
 */
void EjemploDlg::WxButton1Click(wxCommandEvent& event)
{
    // insert your code here
}
```

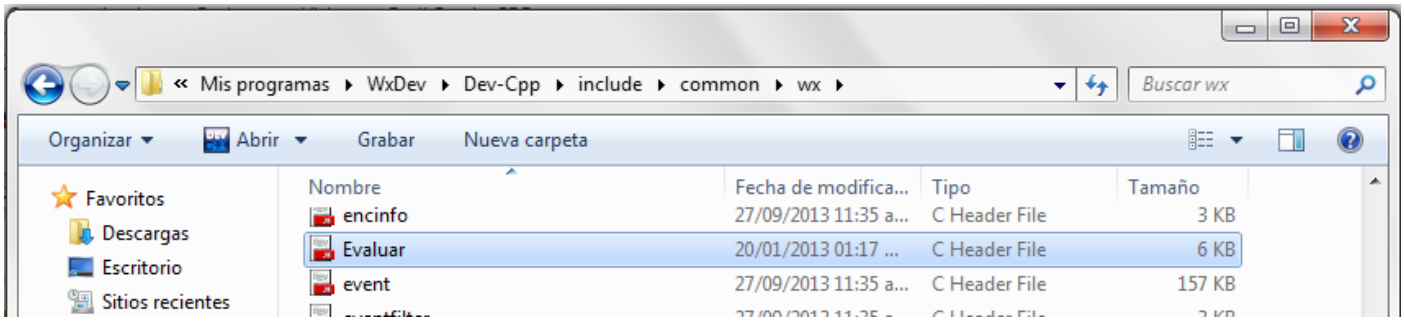
APÉNDICE B. Uso del evaluador de expresiones algebraicas.

Una parte muy importante de los programas es el evaluador de expresiones algebraicas. Para utilizarlo en una aplicación con ventana de WxDevC++ se deben tener en cuenta varios aspectos.

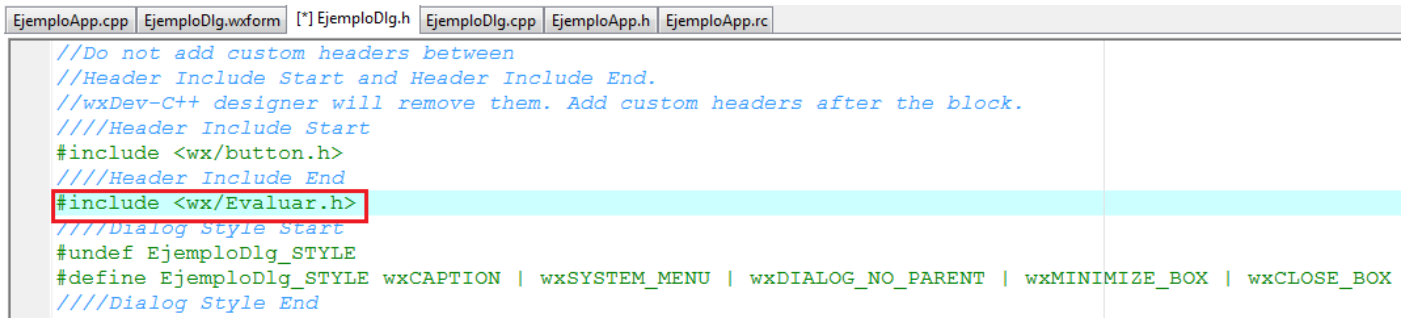
1. Se descarga de http://darwin.50webs.com/Espanol/Evaluador.htm.
2. Una vez que se ha creado el proyecto en la pestaña *Project* se selecciona:



3. En la ventana que aparece debemos ir a la carpeta EvaluadorCPP y añadir los archivos:
4. Es importante copiar los archivos *Evalua, Pieza_Ejecuta y Pieza_Simple* de tipo *C header File* del evaluador en la carpeta donde se ha instalado WxDevC++ que corresponde a los archivos con extensión *.h.* Siguiendo la ruta: Dev-Cpp -> include -> common ->wx.



5. Siguiendo con el programa que hemos creado en el apéndice A. Nos colocamos en archivo EjemploDlg.h e insertamos la siguiente línea de código en el lugar indicado:



6. Después de seguir estos pasos podemos comenzar a usar el evaluador de expresiones como se ha mostrado en el código de cada programa.

APÉNDICE C.1 Código del Analizador de Raíces.

TABULAR

```
void AnalizadorDlg::TabularBClick(wxCommandEvent& event)
{
    //Se ocultan los siguientes componentes para verificar
    //si se cumplen las condiciones necesarias para mostrarlos
    SiguienteST->Enable(false);     AnteriorST->Enable(false);
    X1ST->Enable(false);            X2ST->Enable(false);
    EncontrarST->Enable(false);     RaicesST->Enable(false);
    RaizST->Enable(false);          SintaxisST->Enable(false);
    RaicesB->Enable(false);         X1CT->Enable(false);
    X2CT->Enable(false);            NraizCT->Enable(false);
    RactualCT->Enable(false);       SintaxisCT->Enable(false);
    DesplazarSB->Enable(false);
    //Se limpian las siguientes cajas de texto
    X1CT->Clear();     X2CT->Clear();     NraizCT->Clear();     RactualCT->Clear();
    TablaG->ClearGrid();    SintaxisCT->Clear();
    // verifsin :  verifica la sintaxis de la función
    //verifsinp1,verifsinp2 y verifsinp3 verifican las sintaxis
    //de los parámetros
    int verifsin=0,verifsinp1=0,verifsinp2=0,verifsinp3=0;
    //tamano: tamaño de la expresión matemática
    int tamano=200;
    //expresion: contiene la expresión original tecleada
    char expresion[tamano];
    //Transformado: contiene a la expresion transformada después de aplicar
    //el método Transforma Expresión
    //ExprNegativos: contiene los menos unarios (0-1)
    //Mensaje: contiene al mensaje adecuado segun el código de error
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    //Se crea el objeto evaluadorExpresiones de la clase Evaluar
    Evaluar evaluadorExpresiones;
    //Se inicializan las variables Transformado,ExprNegativos,Transformado
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    //Si no se ha tecleado una función y se ha oprimido tabular muestra el
    //mensaje de error "Inserte una función"
    if (FuncionCT->IsEmpty()) FuncionDM->ShowModal();
    else
    {
        //Se ha tecleado la función y se copia en expresion
        strcpy(expresion, FuncionCT->GetLineText(0));
        //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        //Se guarda en verifsin el código de error en caso de haberlo
        //si es correcta la sintaxis verifsin=0
        verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        if(verifsin)
        {
            //Si la sintaxis no es correcta se muestra el mensaje
            SintaxisDM->ShowModal();
            //Se copia en la variable mensaje el texto del mensaje
            //de acuerdo al código de error contenido en la variable
            //verifsin
            evaluadorExpresiones.MensajeSintaxis(verifsin, Mensaje);
            //Se habilita la caja de texto correspondiente y se muestra el
            //mensaje
            SintaxisCT->Enable(true);
            SintaxisCT->WriteText(Mensaje);
            SintaxisST->Enable(true);
        }
    }
    //Si la caja de texto esta vacia muestra el mensaje de error
    if (DesdeCT->IsEmpty()) DesdeDM->ShowModal();
    //Si no esta vacia
    else
    {
        //copia el paramtro desde en expresion
```

```
    strcpy(expresion, DesdeCT->GetLineText(0));
    //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    //Se guarda en verifsinp1 el código de error en caso de haberlo
    //si es correcta la sintaxis verifsinp1=0
    verifsinp1 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}
//Se verifica sintaxis de la misma manera
if (HastaCT->IsEmpty()) HastaDM->ShowModal();
else
{
    strcpy(expresion, HastaCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    //Se guarda en verifsinp2 el código de error
    verifsinp2 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}
//Se verifica sintaxis
if (IncCT->IsEmpty()) IncDM->ShowModal();
else
{
    strcpy(expresion, IncCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    //Se guarda el código de error en verifsinp3
    verifsinp3 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}

//verifpar es la variable de verificación de sintaxis correcta de los
//3 parámetros si no ha habido error la suma debe ser cero
int verifpar=verifsinp1+verifsinp2+verifsinp3;
//Si es distinta de cero muestra mensaje de error
if(verifpar) SintaxisPDM->ShowModal();
//validator es la variable que indica si todas las cajas de texto
//han sido llenadas si es asi de la multiplicacion debe resultar 1
int validator=!FuncionCT->IsEmpty()*!DesdeCT->IsEmpty()*!HastaCT->IsEmpty()
*!IncCT->IsEmpty();
    //Comienza la tabulación
//Esto solo se ejecuta si no hay errores de sintaxis y si no hay cajas
//de texto vacías
if (verifsin==0&&verifpar==0&&validator)
{
    //filas contiene el numero actual de filas de la tabla
    //j es un contador
    //del es el numero de filas a borrar
    //iter es el numero de iteraciones que se van realizando
    //i es el valor que se le da a la variable independiente
    //a y b se usan para convertir los numeros a caracteres y mostrarlos
    //en cajas de texto como resultados
    //valor es la variable que contiene el reultado de la evaluación de la
    //función
    int filas,j,del,iter;
    double i,dsd,sta,inc;
    char a[3],b[3];
    double valor;
    ////char valor1[10];
    //double m;
    ///char d[3];
    //Se obtiene el número de filas de la tabla
    filas=TablaG->GetNumberRows();
    //si el numero de filas es mayor a 14
    if (filas>14)
    {
        //al numero de filas se le restan 14 para que solo deje las 14
        //predeterminadas y se ajuste a la nueva tabulación
        del=filas-14;
        //Se borran las sobrantes
        TablaG->DeleteRows(11,del);
    }
    //se vuelven a texto negro todas las celdas
```

```
for(j=0;j<filas;j++) TablaG->SetCellTextColour(j,1,"black");
//Se incializan los arreglos
//for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
//= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
//Se copia en expresion el parámetro
strcpy(expresion, DesdeCT->GetLineText(0));
//Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
//Transforma la expresión para aceptar los menos unarios agregando (0-1)#
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
//Calcula en caso de que el parametro sea dado como una expresion matematica
dsd=evaluadorExpresiones.Calcular();
//Se calcula el parametro hasta
//for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
//= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, HastaCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
sta=evaluadorExpresiones.Calcular();
//Se calcula el parametro incremento
//for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
//= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, IncCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
inc=evaluadorExpresiones.Calcular();
//Se hace i igual a dsd y se incializa iter=0
i=dsd;
iter=0;
//Se analiza la función
//for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
//= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, FuncionCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
//Se deja preparado todo para comenzar a darle valores a la variable
//independiente
do
{
    //Si son mas de 14 iteraciones cada iteracion se inserta 1 fila
    if(iter>13) TablaG->InsertRows(iter,1);
    //Se le da a x el valor de i
    evaluadorExpresiones.ValorVariable('x', i);
    //Se calcula el valor
    valor = evaluadorExpresiones.Calcular();
    //Existe error matemático?
    if (_isnan(valor) || !_finite(valor))
    //Si existe y escribe en la celda "Error"
    TablaG->SetCellValue(iter,1,"Error");
    //No existe error
    else
    {
    if (valor<0)
    //Si el resultado es menor de cero
    {
        //Se convierte el número a una cadena de caracteres
        sprintf(a,"%*.*f",3,8,valor);
        //Se escribe el numero en la celda
        TablaG->SetCellValue(iter,1,a);
        //El texto es de color rojo para los negativos
        TablaG->SetCellTextColour(iter,1,"red");
    }

    if(valor>0)
```

```
            {//Si el resultado es mayor de cero
                //Se convierte el número a una cadena de caracteres
                sprintf(a,"%*.*f",3,8,valor);
                //Se escribe el numero en la celda
                TablaG->SetCellValue(iter,1,a);
                //El texto es negro para los positivos
                TablaG->SetCellTextColour(iter,1,"black");
            }

            if(valor==0.0)
            {//Si el resultado es cero
                sprintf(a,"%*.*f",3,8,valor);
                TablaG->SetCellValue(iter,1,a);
                //El texto es azul para cero
                TablaG->SetCellTextColour(iter,1,"blue");
            }
            }
            //Convierte a cadena de caracteres el valor de x
            sprintf(b,"%*.*f",3,5,i);
            //Se escribe en la primera columna
            TablaG->SetCellValue(iter,0,b);
            //Se suma el incremento para el siguiente calculo
            i=i+inc;
            //Se suma 1 a las iteraciones
            iter=iter+1;
            //Se detiene cuando se alcanza el limite superior del intervalo
        }while (i<=sta);
          if(iter<13)
        {
        //Si el numero de iteraciones no es suficiente para que se inserten
        //filas se deshabilitan los siguientes componentes
        X1CT->Enable(false);            X2CT->Enable(false);
        NraizCT->Enable(false);         RactualCT->Enable(false);
        DesplazarSB->Enable(false);     SiguienteST->Enable(false);
        AnteriorST->Enable(false);      X1ST->Enable(false);
        X2ST->Enable(false);            EncontrarST->Enable(false);
        RaicesST->Enable(false);        RaizST->Enable(false);
        RaicesB->Enable(false);
        }
        //Si es mayor a 13 se activa el boton de busqueda de raices
        else RaicesB->Enable(true);
    }
}
```

LIMPIAR

```
void AnalizadorDlg::LimpiarBClick(wxCommandEvent& event)
{
    //filas contiene el numero de filas de la tabla
    //i es un contador
    //del es el numero de filas a borrar
    int filas,i,del;
    //Se obtiene el número de filas
    filas=TablaG->GetNumberRows();
    //Si son mas de 14 se borran las sobrantes
    if (filas>14)
    {
        del=filas-14;
        TablaG->DeleteRows(14,del);
    }
    //Se vuelve el texto de las celdas a color negro
    for(i=0;i<filas;i++) TablaG->SetCellTextColour(i,1,"black");
    //Se limpian y deshabilitan los siguientes componentes
    TablaG->ClearGrid();    FuncionCT->Clear();    DesdeCT->Clear();
    HastaCT->Clear();       IncCT->Clear();        XvalCT->Clear();
    ResultCT->Clear();      X1CT->Clear();
    X2CT->Clear();           NraizCT->Clear();              RactualCT->Clear();
    SintaxisCT->Clear();     SiguienteST->Enable(false);    AnteriorST->Enable(false);
    X1ST->Enable(false);     X2ST->Enable(false);           RaicesB->Enable(true);
    X1CT->Enable(false);     X2CT->Enable(false);           DesplazarSB->Enable(false);
```

```
    EncontrarST->Enable(false);    RaicesST->Enable(false);    RaizST->Enable(false);
    RactualCT->Enable(false);       NraizCT->Enable(false);    RaicesB->Enable(false);
    SintaxisCT->Clear();
    SintaxisCT->Enable(false);    SintaxisST->Enable(false);
}
```

CALCULAR

```
void AnalizadorDlg::CalcularBClick(wxCommandEvent& event)
{
       //Se deshabilitan los siguientes componentes
       SintaxisCT->Enable(false);
       SintaxisST->Enable(false);
       //Se limpian los siguientes componentes
       ResultCT->Clear();
       SintaxisCT->Clear();
    int tamano=200,verifsin,verifsinp;
    //expresion: contiene la expresión original tecleada
    char expresion[tamano],b[3];
    //Transformado: contiene a la expresion transformada después de aplicar
    //el método Transforma Expresión
    //ExprNegativos: contiene los menos unarios (0-1)
    //Mensaje: contiene al mensaje adecuado segun el código de error
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    //Se crea el objeto evaluadorExpresiones de la clase Evaluar
    Evaluar evaluadorExpresiones;
    //Se inicializan las variables Transformado,ExprNegativos,Transformado
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    //Si no se ha tecleado una función y se ha oprimido calcular muestra el
    //mensaje de error "Inserte una función"
    if (FuncionCT->IsEmpty()) FuncionDM->ShowModal();
    else
    {
        //Se ha tecleado la función y se copia en expresion
        strcpy(expresion, FuncionCT->GetLineText(0));
        //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        //Se guarda en verifsin el código de error en caso de haberlo
        //si es correcta la sintaxis verifsin=0
        verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);

        if(verifsin)
        {
            //Si la sintaxis no es correcta se muestra el mensaje
            SintaxisDM->ShowModal();
            //Se copia en la variable mensaje el texto del mensaje
            //de acuerdo al código de error contenido en la variable
            //verifsin
            evaluadorExpresiones.MensajeSintaxis(verifsin, Mensaje);
            //Se habilita la caja de texto correspondiente y se muestra el
            //mensaje
            SintaxisCT->Enable(true);
            SintaxisCT->WriteText(Mensaje);
            SintaxisST->Enable(true);
        }
    }
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]= '\0';
    ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    if (XvalCT->IsEmpty()) XvalDM->ShowModal();
    //Si no esta vacia
    else
    {
        //copia el paramtro desde en expresion
        strcpy(expresion, XvalCT->GetLineText(0));
        //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        //Se guarda en verifsinp1 el código de error en caso de haberlo
        //si es correcta la sintaxis verifsin=0
```

```
        verifsinp = evaluadorExpresiones.EvaluaSintaxis(Transformado);
    }
    if(verifsinp) SintaxisPDM->ShowModal();

    if(!(FuncionCT->IsEmpty())&&!(XvalCT->IsEmpty())&&verifsin==0&&verifsinp==0)
    {//Si se cumplen las condiciones calcula
    //m contiene al valor del calculo
    double m;
    //se obtiene el valor de la variable independiente
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]= '\0';
    ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, XvalCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
    evaluadorExpresiones.Analizar(ExprNegativos);
    m = evaluadorExpresiones.Calcular();

       //se calcula el valor de la funcion
    double valor;
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa] = '\0';
    ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, FuncionCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
    evaluadorExpresiones.Analizar(ExprNegativos);
    evaluadorExpresiones.ValorVariable('x', m);
    valor=evaluadorExpresiones.Calcular();
    //Si hay error matemático se escribe la palabra error
    if (_isnan(valor) || !_finite(valor)) ResultCT->WriteText("Error");
    else //No hay fallo matemático, se muestra el valor
    {
        sprintf(b,"%*.*f",3,8,valor);
        ResultCT->WriteText(b);
    }
    }
}
```

BUSCAR RAICES

```
void AnalizadorDlg::RaicesBClick(wxCommandEvent& event)
{
    char fail[20],root[3];
    //filas almacena el numero de filas de la tabla
    int filas;
    //i es un contador
    int i;
    //iter almacena el numero de iteraciones
    //vant almacena el valor anterior de f(x)
    //vact almacena el valor actual de f(x)
    //condicion almacena la multiplicacion del valor actual
    //por el valor anterior
    int iter,raiz;
    float condicion;
    float vact;
    float vant;
    //Se limpia el numero de raices anterior
    NraizCT->Clear();
    //Se obtiene el numero de filas
    filas=TablaG->GetNumberRows();
    //Se inicializan las variables
    vant=0; raiz=0;
    //Se limpian las cajas de texto:
    X1CT->Clear();
    X2CT->Clear();
    NraizCT->Clear();
    RactualCT->Clear();
    //Se busca el numero de raices en el intervalo
    for(i=0;i<=(filas-1);i++)
       {
        //Ya que se ha escrito error en algunas celdas se copia en fail
```

```
        //el valor de cada celda y se verifica si el caracter 'E' aparece
        //en la primera posicion
        strcpy(fail,TablaG->GetCellValue(i,1));
        //Si no hay error se procede a obtener el valor de la celda
        if (fail[0]!='E')
        {
            vact=atof(TablaG->GetCellValue(i,1));
            //Si es exactamente cero hay una raiz
            if(vact==0)
            {
                raiz=raiz+1;
            }
            //Se multiplica el valor actual por el anterior para
            //verificar el signo del resultado
            condicion=vant*vact;
            if((condicion<0)&&(vact!=0))
            {
                //Si el signo es negativo y el valor actual no es cero
                //Se ha encontrado una raiz
                raiz=raiz+1;

            }
            //Se hace el valor anterior igual al actual para la siguiente
            //iteracion
            vant=vact;
        }
    }
    if(raiz>0)
    {
        //Si se ha encontrado al menos una raiz se muestra el numero de raices
        //se convierte ese numero a una cadena de caracteres
        sprintf(root,"%d",raiz);
        //se escribe en la caja de texto 9
        NraizCT->WriteText(root);
        //Se habilitan los siguientes componentes

        X1CT->Enable(true);
        X2CT->Enable(true);
        NraizCT->Enable(true);
        RactualCT->Enable(true);
        SiguienteST->Enable(true);
        AnteriorST->Enable(true);
        X1ST->Enable(true);
        X2ST->Enable(true);
        EncontrarST->Show(true);
        RaicesST->Enable(true);
        RaizST->Enable(true);
        DesplazarSB->Enable(true);
    }
    if(raiz<=0)
    {
        //Si no se han encontrado raices se deshabilitan los siguientes
        //componentes
        X1CT->Enable(false);
        X2CT->Enable(false);
        DesplazarSB->Enable(false);
        SiguienteST->Enable(false);
        AnteriorST->Enable(false);
        X1ST->Enable(false);
        X2ST->Enable(false);
        EncontrarST->Enable(false);
        RaicesST->Enable(false);
        RaizST->Enable(false);
        RactualCT->Enable(false);
        NraizCT->Enable(false);
        NoraizDM->ShowModal();
    }
}
```

DESPLAZAR

```
void AnalizadorDlg::DesplazarSBUpdated(wxSpinEvent& event)
{
       //cadenas de caracteres para mostrar los resultados
       char max[3],min[3],root[3],sp[3],fail[20];
       //arreglo que el almacena el limite superior de cada raiz
      float mayor[200];
       //arreglo que el almacena el limite inferior de cada raiz
       float menor[200];
       //filas almacena el numero de filas de la tabla
       int filas;
       //contador
       int i;
       //valores de x actual y anterior
       float xval,xant;
       //iter almacena el numero de iteraciones
       //index es el indice para los arreglos que almacenan los limites superior
       // e inferior de cada raiz
       //raiz es un contador que se incrementa cuando se encuentra una raiz
       //svalue es el numero que se obtiene del boton de navegación
       //spin?????????????
       int iter,index,raiz,svalue,spin;
       //condicion almacena la condición para encontrar raices
       float condicion;
       //vact es el valor actual de la función
       //vant es el valor anterior de la función
       float vact;/*valor actual*/
      float vant;/*valor anterior*/
      //Se limpian los siguientes componentes
      RactualCT->Clear();
      X1CT->Clear();
      X2CT->Clear();
      //Se obtiene el numero de filas
       filas=TablaG->GetNumberRows();
       //se inicializan los valores de la siguiente manera
       vant=0; xant=0; index=1; raiz=0;
       //En este bucle se encuentran las raices
    for(i=0;i<=(filas-1);i++)
       {
         //se copia el valor de f(x) de la celda actual
         strcpy(fail,TablaG->GetCellValue(i,1));
         //se verifica que no exista error matemático
         if (fail[0]!='E')
         {//if del err0r
         //se obtiene el valor de f(x)
         vact=atof(TablaG->GetCellValue(i,1));
         //se obtiene el valor de x
         xval=atof(TablaG->GetCellValue(i,0));
         //if(vact==0) vact=0.000001;
         //si el valor de f(x) es cero existe una raiz
         if(vact==0.0)
         {
             //se almacena en mayor y menor el mismo valor de x ya que es
             //la manera de indicar que la raiz es exacta
             mayor[index]=xval;
             menor[index]=xval;
             //se incrementan los siguientes contadores
             index=index+1;
             raiz=raiz+1;
         }
         //se evalua la condicion para encontrar una raiz
         condicion=vant*vact;
         //Si el resultado es negativo existe una raiz
         if((condicion<0)&&(vact!=0))
         {
             //se almacenan en el arreglo mayor los limites superiores de las
             //raíces encontradas
             mayor[index]=xval;
```

```
            //se almcaenan en el arreglo menor los límites inferiores de las
            //raíces encontradas
            menor[index]=xant;
            //se incrementan los siguientes contadores
            index=index+1;
            raiz=raiz+1;
        }
        //se hacen los valores anteriores igual a los actuales para la
        //siguiente iteración
        vant=vact;
        xant=xval;
        }//if del error
    }

    menor[0]=menor[1];
    mayor[0]=mayor[1];
    //Se establece un rango para el botón de navegación
    DesplazarSB->SetRange(0,200);
    //Se obtiene el valor actual del botón de navegación
    svalue=DesplazarSB->GetValue();
    //Si el valor es menor de cero se fuerza a que sea 1 para que este dentro
    //del rango de los índices de los arreglos
    if(svalue<=0)
    {
        DesplazarSB->SetValue(1);
        //Se muestra el mensaje correspondiente
        PraizDM->ShowModal();
    }

    if(svalue>raiz)
    {
        //Si es mayor que el número de raíces se fuerza al valor de las raíces
        //que se hayan encontrado para tener control sobre el índice de los
        //arreglos
        DesplazarSB->SetValue(raiz);
        //Se convierte a caracteres el intervalo de la última raíz para
        //mostrarlo
        sprintf(min,"%*.*f",3,4,menor[raiz]);
        X1CT->WriteText(min);
        sprintf(max,"%*.*f",3,4,mayor[raiz]);
        X2CT->WriteText(max);
        //Solo es permitido mostrar el valor máximo de raíces
        spin=DesplazarSB->GetValue();
        //Se muestra el numero máximo de raíces
        sprintf(sp,"%d",spin);
        //Se escribe el texto en la caja de texto correspondiente
        RactualCT->WriteText(sp);
        //Se muestra el mensaje
        UraizDM->ShowModal();
    }
    else
    {
    //Si el valor del botón de navegación no ésta en los extremos
    spin=DesplazarSB->GetValue();
    //Se muestran los valores de los intervalos que están almacenados en los
    //arreglos usados para tal fin usando como índice el valor del
    //SpinButton
    sprintf(sp,"%d",spin);
    RactualCT->WriteText(sp);
    sprintf(min,"%*.*f",3,4,menor[svalue]);
    X1CT->WriteText(min);
    sprintf(max,"%*.*f",3,4,mayor[svalue]);
    X2CT->WriteText(max);
    }

}
```

APÉNDICE C.2 CÓDIGO DE BISECCIÓN

CALCULAR

```
void BISECCIONDlg::CalcBClick(wxCommandEvent& event)
{//principal
    // insert your code here
    //variables de verificación de sintaxis
    int verifsin=0,verifsinp1=0,verifsinp2=0;
    //Se limpian los componentes:
    TablaResG->ClearGrid();
    SintaxisCT->Clear();
    RaizresultCT->Clear();
    //Se deshabilitan los componentes:
    SintaxisCT->Enable(false);
    SintaxisST->Enable(false);
    //tamano: tamaño de la expresión matemática
    int tamano=200;
    //expresion: almacena la expresión original tecleada
    char expresion[tamano];
    //Transformado: almacena a la expresion transformada después de aplicar
    //el método Transforma Expresión
    //ExprNegativos: almacena los menos unarios (0-1)
    //Mensaje: almacena al mensaje adecuado segun el código de error
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    //Se crea el objeto evaluadorExpresiones de la clase Evaluar
    Evaluar evaluadorExpresiones;

    //Si no se ha tecleado una función y se ha oprimido calcular se muestra el
    //mensaje de error "Inserte una función"
    if (FuncionCT->IsEmpty()) FuncionDM->ShowModal();
    else
    {
       //Se inicializan las variables Transformado,ExprNegativos,Transformado
       for(intinicializa=0;inicializa<tamano;inicializa++){Transformado[inicializa]='\0';
       ExprNegativos[inicializa]= '\0'; Mensaje[inicializa] = '\0'; }
       //Se ha tecleado la función y se copia en expresion
       strcpy(expresion, FuncionCT->GetLineText(0));
       //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
       evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
       //Se guarda en verifsin el código de error en caso de haberlo
       //si es correcta la sintaxis verifsin=0
       verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);
       if(verifsin)
       {
            //Si la sintaxis no es correcta se muestra el mensaje
            SintaxisDM->ShowModal();
            //Se copia en la variable mensaje el texto del mensaje
            //de acuerdo al código de error contenido en la variable
            //verifsin
            evaluadorExpresiones.MensajeSintaxis(verifsin, Mensaje);
            //Se habilitan los componentes necesarios y se muestra el
            //mensaje
            SintaxisST->Enable(true);
            SintaxisCT->Enable(true);
            SintaxisCT->WriteText(Mensaje);
       }
    }

   //Si la caja de texto esta vacia muestra el mensaje de error
    if (X1BisCT->IsEmpty()) X1DM->ShowModal();
    //Si no esta vacia
    else
    {
        //Se inicializan las variables Transformado,ExprNegativos,Transformado
        for (int inicializa=0;inicializa<tamano;inicializa++){Transformado[inicializa]='\0';
        ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
        //copia el parametro X1 en expresion
        strcpy(expresion, X1BisCT->GetLineText(0));
        //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
```

```
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    //Se guarda en verifsinp1 el código de error en caso de haberlo
    //si es correcta la sintaxis verifsinp1=0
    verifsinp1 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}
//Si la caja de texto esta vacia muestra el mensaje de error
if (X2BisCT->IsEmpty()) X2DM->ShowModal();
else
{
    //Se inicializan las variables Transformado,ExprNegativos,Transformado
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    //copia el parametro X2 en expresion
    strcpy(expresion, X2BisCT->GetLineText(0));
    //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    //Se guarda en verifsinp2 el código de error en caso de haberlo
    //si es correcta la sintaxis verifsinp2=0
    verifsinp2 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}
//Si no se ha introducido el EMP se muestra el mensaje
if (EmpCT->IsEmpty()) EmpDM->ShowModal();
//verifpar es la variable de verificación de sintaxis correcta de los
//2 parámetros si no ha habido error la suma debe ser cero
int verifpar=verifsinp1+verifsinp2;
//Si es distinta de cero muestra mensaje de error
if(verifpar) SintaxisPDM->ShowModal();
//validator es la variable que indica si todas las cajas de texto
//han sido llenadas si es asi de la multiplicacion debe resultar 1
  int validator=!FuncionCT->IsEmpty()*!X1BisCT->IsEmpty()*!X2BisCT->IsEmpty()
*!EmpCT->IsEmpty();
//Comienza el Metodo de biseccion
//Esto solo se ejecuta si no hay errores de sintaxis y si no hay cajas
//de texto vacías
if(verifsin==0&&verifpar==0&&validator)
{//if principal
int iter;//almacena el número de iteraciones
//filas almacena el numero de filas de la tabla y del almacena el número de
//filas que serán eliminadas
int filas,del;
double ini;//valor de X1 tecleado por el usuario
double fin;//valor de X2 tecleado por el usuario
double xa;//valor anterior de x
double xm;//valor de x a la mitad del intervalo
double emp;//erro mínimo permitido
double ea;//error absoluto
//fx1 almacena el valor de f(ini), fx2 almacena el valor de f(fin)
//fx3 almacena el valor de f(xm)
//Este algoritmo necesita evaluar dos condiciones para observar entre
//que valores ocurre al cambio de signo por lo que con1 y cond2 almacenan
//tales valores, raiz almacena el valor de la raiz calculada
double fx1,fx2,fx3,cond1,cond2,raiz;
//Cadenas de carácteres para mostrar resultados
char bis[3],ma[3],mb[3],mm[3],error[3],f1[3],f2[3],f3[3];
//Se obtiene el numero de filas
filas=TablaResG->GetNumberRows();
if (filas>10)
{
    //al numero de filas se le restan 10 para que solo deje las 10
    //predeterminadas y se ajuste a la nueva tabulación
    del=filas-10;
    //Se borran las sobrantes
    TablaResG->DeleteRows(10,del);    }

//Se obtiene el parametro X1 y se guarda en ini
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, X1BisCT->GetLineText(0));
```

```
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
ini=evaluadorExpresiones.Calcular();
//Se obtiene el parámetro X2 y se guarda en fin
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, X2BisCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
fin=evaluadorExpresiones.Calcular();
//Se obtiene el parámetro emp
emp=atof(EmpCT->GetLineText(0));
//Se inicializa x anterior=0
xa=0;
//Se obtiene la primera xm (punto medio entre X1 y X2 para calcular ea
//xm=(ini+fin)/2.0;
//Se calcula el error absoluto
//ea=xa-xm;
//si la cantidad es negativa se multiplica por -1 para el valor absoluto
//if(ea<0) ea=ea*(-1.0);
//Se inicializa el numero de iteraciones
iter=0;
//Se analiza la expresión y se deja lista para emezar a sustituir valores
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, FuncionCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
//Comienza el bucle que desarrolla las iteraciones del método Bisección
do
{//do
    //Se establece color rojo para las columnas 1 y 4
    TablaResG->SetCellTextColour(iter,1,"red");
    TablaResG->SetCellTextColour(iter,4,"red");
    //Se establece color azul para las columnas 2 y 6
    TablaResG->SetCellTextColour(iter,2,"blue");
    TablaResG->SetCellTextColour(iter,6,"blue");
    //Se establece color cafe para las columnas 3 y 5
    TablaResG->SetCellTextColour(iter,3,"brown");
    TablaResG->SetCellTextColour(iter,5,"brown");

    //Si el numero de iteraciones es mayor que 9 entonces cada nueva
    //iteracion se inserta una fila
    if(iter>9) TablaResG->InsertRows(iter,1);
    //Se calcula el valor de  xm
    xm=(ini+fin)/2.0;
    //Se calcula el error
    ea=xa-xm;
    //Si el reultado es negativo se obtiene el valor absoluto
    if(ea<0) ea=ea*(-1.0);
    //Se evalua f(ini)
    evaluadorExpresiones.ValorVariable('x', ini);
    fx1 = evaluadorExpresiones.Calcular();
    //Se evalua f(fin)
    evaluadorExpresiones.ValorVariable('x', fin);
    fx2 = evaluadorExpresiones.Calcular();
    //Se evalua f(xm)
    evaluadorExpresiones.ValorVariable('x', xm);
    fx3 = evaluadorExpresiones.Calcular();

    //Se convierten los valores a cadenas de carácteres para ser mostrados
    //Se muestra el numero de iteracione en la columna 0
    sprintf(bis,"%d",iter);
    TablaResG->SetCellValue(iter,0,bis);
    //Se muestra el valor del limite inferior del intervalo
```

```
        sprintf(ma,"%*.*f",3,6,ini);
        TablaResG->SetCellValue(iter,1,ma);
        //Se muestra el valor del límite superior del intervalo
        sprintf(mb,"%*.*f",3,6,fin);
        TablaResG->SetCellValue(iter,2,mb);
        //Se muestra el valor del punto medio del intervalo
        sprintf(mm,"%*.*f",3,6,xm);
        TablaResG->SetCellValue(iter,3,mm);
        //Se muestra el valor de f(ini)
        sprintf(f1,"%*.*f",3,6,fx1);
        TablaResG->SetCellValue(iter,4,f1);
        //Se muestra el valor de f(xm)
        sprintf(f3,"%*.*f",3,6,fx3);
        TablaResG->SetCellValue(iter,5,f3);
        //Se muestra el valor de f(fin)
        sprintf(f2,"%*.*f",3,6,fx2);
        TablaResG->SetCellValue(iter,6,f2);
        //Se muestra el valor del error absoluto
        sprintf(error,"%*.*f",3,9,ea);
        TablaResG->SetCellValue(iter,7,error);
        //Se evaluan las condicions para asignar el nuevo intervalo
        cond1=fx1*fx3;
        cond2=fx2*fx3;
        //Si la condicion 1 ha resultado negativa el nuevo intervalo corresponde
        //a [ini,xm]
        if(cond1<0) fin=xm;
        //Si la condicion 1 ha resultado negativa el nuevo intervalo corresponde
        //a [xm,fin]
        if(cond2<0) ini=xm;
        //Se hace x anterior igual a la xm calculada para calcular el error
        //en la siguiente iteración
        xa=xm;
        //Se incrementa en uno el número de iteraciones
        iter=iter+1;
    //El proceso se repite hasta que se alcanza el valor del emp
    }while(ea>emp);//fin de do
    //Cuando el proceso termina se toma el último valor de xm y se muestra
    //como la raíz encontrada
    sprintf(mm,"%*.*f",3,11,xm);
    RaizresultCT->WriteText(mm);
    RaizresultCT->Enable(true);
    }//if principal
}//principal
```

ANALIZAR RAICES

```
void BISECCIONDlg::AnalizarBClick(wxCommandEvent& event)
{
       AnteriorST->Show(true);    SiguienteST->Show(true);  X1ST->Show(true);
       X2ST->Show(true);          RaizST->Show(true);       EncontrarST->Show(true);
       RaicesST->Show(true);      EvaluarxST->Show(true);   ResultST->Show(true);
       DesdeST->Show(true);       HastaST->Show(true);      IncST->Show(true);
       DesdeCT->Show(true);       HastaCT->Show(true);      IncCT->Show(true);
       X1CT->Show(true);         X2CT->Show(true);          RactualCT->Show(true);
       NraizCT->Show(true);       XvalCT->Show(true);       ResultCT->Show(true);
       TablaG->Show(true);        TabularB->Show(true);     RaicesB->Show(true);
       CalcularB->Show(true);     OtroB->Show(true);  DesplazarSB->Show(true);
       Fit();

}
```

NUEVOS DATOS

```
*/
void BISECCIONDlg::NuevosBClick(wxCommandEvent& event)
{
    RaizresultCT->Clear();
    RaizresultCT->Enable(false);
    int iter;
    int filas,del;
```

```
    filas=TablaResG->GetNumberRows();
    if (filas>10)
    {
        del=filas-10;
        TablaResG->DeleteRows(10,del);
    }

    AnteriorST->Show(false);    SiguienteST->Show(false);    X1ST->Show(false);
    X2ST->Show(false);          RaizST->Show(false);         EncontrarST->Show(false);
    RaicesST->Show(false);      EvaluarxST->Show(false);     ResultST->Show(false);
    DesdeST->Show(false);       HastaST->Show(false);        IncST->Show(false);
    DesdeCT->Show(false);       HastaCT->Show(false);        IncCT->Show(false);
    X1CT->Show(false);          X2CT->Show(false);           RactualCT->Show(false);
    NraizCT->Show(false);       XvalCT->Show(false);         ResultCT->Show(false);
    TablaG->Show(false);        TabularB->Show(false);       RaicesB->Show(false);
    CalcularB->Show(false);     OtroB->Show(false);          DesplazarSB->Show(false);
    FuncionCT->Clear();         X1BisCT->Clear();            X2BisCT->Clear();
    EmpCT->Clear();             DesdeCT->Clear();            HastaCT->Clear();
    IncCT->Clear();             X1CT->Clear();               X2CT->Clear();
    RactualCT->Clear();         NraizCT->Clear();            XvalCT->Clear();
    ResultCT->Clear();          SintaxisCT->Clear();         TablaResG->ClearGrid();
    TablaG->ClearGrid();        X1CT->Enable(false);         X2CT->Enable(false);
    RactualCT->Enable(false);   NraizCT->Enable(false);      ResultCT->Enable(false);
    SintaxisCT->Enable(false);  AnteriorST->Enable(false);   SiguienteST->Enable(false);
    X1ST->Enable(false);        X2ST->Enable(false);         RaizST->Enable(false);
    EncontrarST->Enable(false); RaicesST->Enable(false);     SintaxisST->Enable(false);
    DesplazarSB->Enable(false); RaicesB->Enable(false);      OtroB->Enable(false);
    CalcularB->Enable(true);    XvalCT->Enable(true);        Fit();
}
```

APÉNDICE C.3 CÓDIGO DE NEWTON – RAPHSON

CALCULAR

```
void Newton_RaphsonDlg::CalcBClick(wxCommandEvent& event)
{//Principal
       // insert your code

    //Se limpian y deshabilitan los siguientes componentes.
   TablaResG->ClearGrid();
   SintaxisCT->Clear();
   SintaxisCT->Enable(false);
   RaizresultCT->Clear();
   RaizresultCT->Enable(false);
   SintaxisST->Enable(false);
   //Variables de verificación de sintaxis
   int verifsin=0,verifsin1=0,verifsinp1=0;
   //tamano: tamaño de la expresión matemática
   int tamano=200;
   //expresion: contiene la expresión original tecleada
   char expresion[tamano];
   //Transformado: contiene a la expresion transformada después de aplicar
   //el método Transforma Expresión
   //ExprNegativos: contiene los menos unarios (0-1)
   //Mensaje: contiene al mensaje adecuado segun el código de error
   char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
   //Se crea el objeto evaluadorExpresiones de la clase Evaluar
   Evaluar evaluadorExpresiones;
   //Si no se ha tecleado una función y se ha oprimido tabular muestra el
   //mensaje de error "Inserte una función"
   if (FuncionCT->IsEmpty()) FuncionDM->ShowModal();
   else
    {//else1
       //Se inicializan las cadenas Transformado,ExprNegativos,Transformado
       for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
       = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
       //Se ha tecleado la función y se copia en expresion
       strcpy(expresion, FuncionCT->GetLineText(0));
       //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
       evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
       //Se guarda en verifsin el código de error en caso de haberlo
       //si es correcta la sintaxis verifsin=0
       verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);

        if(verifsin)
        {
            //Si la sintaxis no es correcta se muestra el mensaje
            SintaxisDM->ShowModal();
            //Se copia en la variable mensaje el texto del mensaje
            //de acuerdo al código de error contenido en la variable
            //verifsin
            evaluadorExpresiones.MensajeSintaxis(verifsin, Mensaje);
            //Se habilita la caja de texto y se muestra el mensaje
            SintaxisST->Enable(true);
            SintaxisCT->Enable(true);
            SintaxisCT->WriteText(Mensaje);
        }
        //Una vez revisada la sintaxis de la función se revisa la sintaxis
        //de la derivada
        else
        {//else 2
            //Si la caja de texto esta vacía se muestra el mensaje
            if (DerivadaCT->IsEmpty()) DerivadaDM->ShowModal();
            else
            {//else3
                //Se limpian los arreglos
                for (int inicializa=0; inicializa<tamano; inicializa++) {Transformado[inicializa]
                = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
                //Se copia la expresion
                strcpy(expresion, DerivadaCT->GetLineText(0));
```

```
            //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
            evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
            //Se obtiene el código de error
            verifsin1 = evaluadorExpresiones.EvaluaSintaxis(Transformado);

            if(verifsin1)
            {
                //Si ha ocurrido error de sintaxis se muestra
                SintaxisDDM->ShowModal();
                SintaxisST->Enable(true);
                SintaxisCT->Enable(true);
                evaluadorExpresiones.MensajeSintaxis(verifsin1, Mensaje);
                SintaxisCT->WriteText(Mensaje);
            }
        }//else3
    }//else2

}//else1
  //Se verifica la sintaxis del parámetro x[0]
if (X0CT->IsEmpty()) X0DM->ShowModal();
else
{
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, X0CT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    verifsinp1 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}

//Se notifica si se ha omitido la Tolerancia
if (TolCT->IsEmpty()) TolDM->ShowModal();
//Se notifica si hay error de sintaxis en el valor de x[0]
if(verifsinp1) SintaxisPDM->ShowModal();
//Se verifica que no se hayan omitido cajas de texto
int validator=!FuncionCT->IsEmpty()*!DerivadaCT->IsEmpty()*!X0CT->IsEmpty()
*!TolCT->IsEmpty();
//Si las condiciones son adecuadas comienza el método Newton Raphson
if(validator&&verifsin==0&&verifsin1==0&&verifsinp1==0)
{
double x[100];//arreglo que almacena los valores de xn
  double tol;//tolerancia al error
int filas,del;//filas de la tabla y filas a borrar
int iter;//número de iteraciones
double fe;//valor obtenido de la función
double fpe;//valor obtenido de la derivada
double div;//valor de fe/fpe
double ea;//error
//cadenas de caracteres para mostrar resultados
char c1[3],c2[3],c3[3],c4[3],c5[3],c6[3];
//Se obtiene el número de filas
filas=TablaResG->GetNumberRows();
//Si son más de 8 filas se eliminan las excedentes
if (filas>8)
{
    //Se obtiene el numero de filas a eliminar
    del=filas-8;
    TablaResG->DeleteRows(8,del);
}

//Se obtiene el valor de x[0]
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, X0CT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
x[0]=evaluadorExpresiones.Calcular();
//Se obtiene el valor de la tolerancia
```

```
tol=atof(TolCT->GetLineText(0));
//Se inicializa el número de iteraciones
iter=0;

//Comienza el bucle del Método de Newton Raphson
do
{//do
    //Si el numero de iteraciones es mayor que 7 se inserta una fila
    //cada nueva iteracion
    if(iter>7) TablaResG->InsertRows(iter,1);

    //Se analiza la función dada
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, FuncionCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
    evaluadorExpresiones.Analizar(ExprNegativos);
    //Se sustituye el valor de x[iter]
    evaluadorExpresiones.ValorVariable('x', x[iter]);
    //Se obtiene el valor de la función
    fe=evaluadorExpresiones.Calcular();

    //Se analiza la derivada
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, DerivadaCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
    evaluadorExpresiones.Analizar(ExprNegativos);
    //Se sustituye el valor de x[iter]
    evaluadorExpresiones.ValorVariable('x', x[iter]);
    //Se calcula el valor de la derivada
    fpe=evaluadorExpresiones.Calcular();

    //Se realiza la división siguiente
    div=fe/fpe;
    //ESTA ES LA EXPRESIÓN DEL MÉTODO DE NEWTON RAPHSON
     x[iter+1]=x[iter]-div;
    //Se calcula el error
    ea=x[iter+1]-x[iter];
    //Se obtiene el valor absoluto
    if(ea<0) ea=ea*(-1.0);
    //Se muestra el valor de x[iter]
    sprintf(c1,"%*.*f",3,6,x[iter]);
     TablaResG->SetCellValue(iter,0,c1);
     //Se muestra el valor de la función en ese punto
     sprintf(c2,"%*.*f",3,6,fe);
     TablaResG->SetCellValue(iter,1,c2);
     //Se muestra el valor de la derivada en ese punto
     sprintf(c3,"%*.*f",3,6,fpe);
     TablaResG->SetCellValue(iter,2,c3);
     //Se muestra el valor de fe/fpe
     sprintf(c4,"%*.*f",3,6,div);
     TablaResG->SetCellValue(iter,3,c4);
     //Se muestra el valor de x[iter+1]
     sprintf(c5,"%*.*f",3,6,x[iter+1]);
     TablaResG->SetCellValue(iter,4,c5);
     //Se muestra el valor del error
     sprintf(c6,"%*.*f",3,6,ea);
     TablaResG->SetCellValue(iter,5,c6);
     //Se incrementa en uno el número de iteraciones
       iter=iter+1;

       //Se repite mientras sea verdad que ea>tol

}while(ea>tol);
//Se obtiene el último valor de x y se muestra en una caja de texto
```

```
      sprintf(c5,"%*.*f",3,11,x[iter-1]);
      RaizresultCT->WriteText(c5);
      RaizresultCT->Enable(true);
  }//do
}//principal
```

NUEVOS DATOS

```
void Newton_RaphsonDlg::NuevosBClick(wxCommandEvent& event)
{
      // insert your code here
    int filas,del;
    filas=TablaResG->GetNumberRows();
    if (filas>8)
    {
        del=filas-8;
        TablaResG->DeleteRows(8,del);
    }

    TablaResG->ClearGrid();
    FuncionCT->Clear();
    DerivadaCT->Clear();
    X0CT->Clear();
    TolCT->Clear();
    SintaxisST->Enable(false);
    SintaxisCT->Clear();
    SintaxisCT->Enable(false);
    RaizresultCT->Clear();
    RaizresultCT->Enable(false);
    //Este botón además ejecuta el código mostrado en el botón Nuevos Datos del programa
    //Bisección. Algunas lineas son idénticas a las del dicho código solo hay que eliminarlas.
    //Se ajusta el tamaño de la ventana según los componentes que se muestren
    Fit();
}
```

APENDICE C.4 CÓDIGO DE DESCOMPOSICIÓN LU

Orden

```
void LU1Dlg::OrdenLDSelected(wxCommandEvent& event )
{//principal
       //sel almacena la selección de la lista desplegable
       //ins almacena el numero de filas y columnas a insertar
       int sel,ins;
       sel=OrdenLD->GetCurrentSelection();
       //La selección de la lista comienza en cero por lo que hay que
       //sumar 1 para obtener las filas y columnas a insertar
       ins=sel+1;
       //Se insertan filas en la tabla 1 a partir de la fila 2.
       CoefG->InsertRows(2,ins);
       //Se insertan columnas en la tabla 1 a partir de la fila 2.
       CoefG->InsertCols(2,ins);
       //Se insertan filas en la tabla 2 a partir de la fila 2.
       LuG->InsertRows(2,ins);
       LuG->InsertCols(2,ins);
       //Se insertan filas en la tabla 3 a partir de la fila 2.
       IndG->InsertRows(2,ins);
       //Se insertan filas en la tabla 4 a partir de la fila 2.
       MatCG->InsertRows(2,ins);
       //Se insertan filas en la tabla 5 a partir de la fila 2.
       SolucionG->InsertRows(2,ins);
       CoefG->ClearGrid();
       LuG->ClearGrid();
       IndG->ClearGrid();
       MatCG->ClearGrid();
       SolucionG->ClearGrid();
       //Se deshabilita la lista de selección
        OrdenLD->Enable(false);
        //Se habilita el botón que habilita a la lista desplegable
        OrdenB->Enable(true);
        //Se limpian todas las tablas
       Fit();
}//principal
```

MAS PUNTOS

```
void LU1Dlg::OrdenBClick(wxCommandEvent& event)
{
    // insert your code here
    //lim almacena el numero de filas de la tabla filas=columnas
    int lim,erase;
    char a[3];
    float num;
    num=0.0000;
    2x2ST->Show(true);
    //Se habilita la lista desplegable
    OrdenLD->Enable(true);
    //Se deshabilita este botón
    OrdenB->Enable(false);
    //Se obtiene el orden de la matriz
    lim=CoefG->GetNumberRows();
    //Se calcula el numero de filas y columnas a eliminar
    erase=lim-2;
    //Si el orden es mayor a 2x2 se reestablece a 2x2 por defecto
    if(lim>2)
    {
        //Se eliminan filas y columnas a partir de la posición 2
        //para cada tabla en el caso de las tablas 3 4 y 5 solo
        //se eliminan filas porque son de una sola columna
        CoefG->DeleteRows(2,erase);         CoefG->DeleteCols(2,erase);
        LuG->DeleteRows(2,erase);           LuG->DeleteCols(2,erase);
        IndG->DeleteRows(2,erase);          MatCG->DeleteRows(2,erase);
        SolucionG->DeleteRows(2,erase);
    }
    CoefG->ClearGrid();     LuG->ClearGrid();     IndG->ClearGrid();     MatCG->ClearGrid();
    SolucionG->ClearGrid();
```

```
    //Se ajusta el  tamaño de la ventana segun el tamaño de las tablas
    Fit();
}
```

NUEVOS DATOS

```
void LU1Dlg::NuevosBClick(wxCommandEvent& event)
{
       // insert your code here
       //Se limpian todas las tablas
       CoefG->ClearGrid();
       LuG->ClearGrid();
       IndG->ClearGrid();
       MatCG->ClearGrid();
       SolucionG->ClearGrid();
       //Se limpia y deshabilita la caja de texto que muestra errores
       SintaxisCT->Clear();
       SintaxisCT->Enable(false);
       ErrorST->Enable(false);
}
```

CALCULAR

```
*/
void LU1Dlg::CalcularBClick(wxCommandEvent& event)
{//principal
    // insert your code here
    //Se limpia y deshabilita la caja de texto que muestra errores
    SintaxisCT->Clear();
    SintaxisCT->Enable(false);
    ErrorST->Enable(false);
    //Se limpian las tablas que muestran resultados
    LuG->ClearGrid();
    MatCG->ClearGrid();
    SolucionG->ClearGrid();
    int verifsin;//almacena el código de error de verificación de sintaxis
    int sintax[101];//almacena los códigos de error de cada celda
    //se inicializa el arreglo
    for(int i=0;i<=100;i++) sintax[i]=0;
    //indice es el indice del arreglo sintax
    //sumas almacena la suma del arreglo de sintax de la tabla 1
    //sumas1 almacena la suma del arreglo de sintax para la tabla 3
    int indice=0,sumas=0,sumas1=0;
    //tamano de las expresiones de las celdas
    int tamano=30;
    //expresion: contiene la expresión original tecleada
    //eror contiene en cadena de caracteres la coordenada de error
    char expresion[tamano],eror[3];
    //Transformado: contiene a la expresion transformada después de aplicar
    //el método Transforma Expresión
    //ExprNegativos: contiene los menos unarios (0-1)
    //Mensaje: contiene al mensaje adecuado segun el código de error
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    //Se crea el objeto evaluadorExpresiones de la clase Evaluar
    Evaluar evaluadorExpresiones;
    //Se obtiene el órden del sistema
    int ord1=CoefG->GetNumberRows();
    //Este bucle anidado verifica la sintaxis de cada celda de la tabla 1
    //El for externo tiene el control de las filas
    for(int i=0;i<=ord1-1;i++)
    {//for1
        //El for interno tiene el control de las columnas
        //Este bucle se realiza y cuando termina se incrementa en 1 el bucle
        //externo
        for(int j=0;j<=ord1-1;j++)
        {//for2

            //Se limpian las variables interpretacion de expresiones
            for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
           = '\0'; ExprNegativos[inicializa] = '\0'; }
```

```
        //Se copia la expresion correspondiente a la fila i y la columna j
        strcpy(expresion, CoefG->GetCellValue(i,j));
        //Quita espacios, tabuladores, encierra en paréntesis, vuelve a minúsculas
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        //Se obtiene el código de error
        verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        //Se almace cada código de error en el arreglo
        sintax[indice]=verifsin;
        //Si la celda actual contiene error se muestra su coordenada
        if(verifsin)

        {
            //Se habilita la caja de texto que muestra las coordenadas de error
            SintaxisCT->Enable(true);
           ErrorST->Enable(true);
            //Se inicia con parentesis
            SintaxisCT->WriteText("(");
            //Se convierte a cadena de carácteres el numero de fila
            sprintf(eror,"%d",i+1);
            //Se escribe en la caja de Texto
            SintaxisCT->WriteText(eror);
            //Sigue una coma
            SintaxisCT->WriteText(",");
            //Se convierte a cadena de caracteres el numero de columna
            sprintf(eror,"%d",j+1);
            //Se escribe en la caja de texto
            SintaxisCT->WriteText(eror);
            //Ya que la coordenada de error esta completa se cierra el
            //paréntesis
            SintaxisCT->WriteText(")");
            //Se escribe un espacio para la siguiente coordenada en caso
            //de haberla
            SintaxisCT->WriteText(" ");


        }
        //Se incrementa el indice del arreglo sintax en 1
        indice=indice+1;
    }//for2
}//for1


for(int i=0; i<=100;i++)
{
    //Si algun elemento del arreglo es distinto de cero la variable
    //sumas se incrementa en uno
    if(sintax[i]!=0) sumas=sumas+1;
}

//Si ha ocurrido error la variable sumas es distinta de cero y se muestra
//el mensaje de error correspondiente
if(sumas) SintaxisCDM->ShowModal();
//Si no hay error en la matriz de coeficiente se procede a verificar la
//sintaxis de los términos independientes
else
{//else1
    //se limpia el arreglo y se vuelven a inicializar indice y sumas
    for(int i=0;i<=101;i++) sintax[i]=0;
    indice=0;
    verifsin=0;

    //En este caso la tabla de terminos independientes solo cuenta con una
    //columna por lo que solo se necesita de un ciclo for
    for(int j=0;j<=ord1-1;j++)
    {//for3
        //se incrementa el índice del arreglo
        indice=indice+1;
        //se limpian arreglos
```

```
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; }
        //Se copia cada termino independiente
        strcpy(expresion, IndG->GetCellValue(j,0));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        //Se obtiene el código de error
        verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        //Se almacena cada código de error
        sintax[indice]=verifsin;

        if(verifsin)
        //Si ha detectado error de sintaxis se indica la fila
        {
            //Se habilita la caja de texto para mostrar errores
            SintaxisCT->Enable(true);
           ErrorST->Enable(true);
            //Se abre un paréntesis
            SintaxisCT->WriteText("(");
            //Se convierte a caracter el numero de fila
            sprintf(eror,"%d",j+1);
            //se muestra el numero de fila
            SintaxisCT->WriteText(eror);
            //Se cierra el parentesis
            SintaxisCT->WriteText(")");
            //Se escribe un espacio
            SintaxisCT->WriteText(" ");


        }
    }//for3

    for(int i=0; i<=100;i++)
    {
        //Si algun elemento del arreglo es distinto de cero la variable
        //sumas1 se incrementa en uno
        if(sintax[i]!=0) sumas1=sumas1+1;
    }
    //Si ha ocurrido error la variable sumas es distinta de cero y se muestra
    //el mensaje de error correspondiente
    if(sumas1) SintaxisIDM->ShowModal();

}//else1
//Una vez que se ha revisado la sintaxis comienza el método
if(!sumas&&!sumas1)
{//control de la ejecución
//variables para el control de los ciclos
int i,j,ord,k,r,c,lim,n,lims,limi,index;
//almacena el valor obtenido de cada celda
double valor;
//cadena de caracteres para mostrar resultados
char a[3];
//se obtiene el orden del sistema
ord=CoefG->GetNumberRows();
//matriz A
double  A[ord+1][ord];
//matriz B,b de terminos independientes
double B[ord][ord],b[ord],test;
//matriz x y c de resultados de la eliminación hacia atras y hacia adelante
//respectivamente
double x[ord],c1[ord];
//variables para las sumatorias
double suma,suma1,suma2,cond;
B[2][2]=1.0;//***
for(i=0;i<=ord-1;i++)//for4
{//recorre cada una de las filas
    for(j=0;j<=ord-1;j++)
    {//for5
        //recorre todos los elementos de cada fila
```

```
        //Se limpian arreglos
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; }
        //Se copia la expresión
        strcpy(expresion, CoefG->GetCellValue(i,j));
        //Se transforma la expresión
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
        //Se agregan menos unarios
        evaluadorExpresiones.Analizar(ExprNegativos);
        //Se calcula el valor de la expresión
        valor = evaluadorExpresiones.Calcular();
        //Se almacena en la matriz A cada uno de los valores de la tabla 1
        A[i][j]=valor;
    }//for5
}//for4

for(n=0;n<=ord-1;n++)//for6
{//El for mas externo indica el numero de fila y columna que se esta
    //calculando
    r=n; c=n; lim=n-1; i=c+1;
    for(j=n;j<=ord-1;j++)//for7
    {//este ciclo calcula todos los elementos de cada fila
        suma1=0.0; suma2=0.0;
        //este ciclo realiza las sumatorias indicadas en las expresiones 1 y 2
        for(k=0;k<=lim;k++)
        {
            //Se realizan las sumatorias
            suma1=suma1+B[r][k]*B[k][j];
            suma2=suma2+B[i][k]*B[k][c];
        }
        //Se establece color rojo para ls renglones de U
        LuG->SetCellTextColour(r,j,"red");
        //Se calculan valores de la fila de U indicada por n
        B[r][j]=A[r][j]-suma1;
        //Se establece color rojo para las columnas de L
        LuG->SetCellTextColour(i,c,"blue");
        //Se calculan valores de la columna de U indicada por n
        //si no se apunta a una posicón fuera del arreglo
        if(i<=ord-1) B[i][c]=(A[i][c]-suma2)/B[c][c];
        //Se incrementa i en uno
        i=i+1;
    }//for6
}//for7
//Este ciclo obtiene los valores de los terminos independientes y los
//almacena en el arreglo b
for(i=0;i<=ord-1;i++)
{
    //se limpian arreglos
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; }
    //se copia la expresión
    strcpy(expresion, IndG->GetCellValue(i,0));
    //Se transforma la expresión
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
    //se agregan menos unarios
    evaluadorExpresiones.Analizar(ExprNegativos);
    //se calcula el valor de la expresión
    valor = evaluadorExpresiones.Calcular();
    //se almacena cada termino independiente en el arreglo b
    b[i]=valor;
}
//se aplica la expresión 3
c1[0]=b[0];
//este ciclo indica que elemento de c1 se esta calculando
for(i=1;i<=ord-1;i++)
{
```

```
        suma=0.0; lim=i-1;
        //este ciclo realiza la sumatoria
        for(k=0;k<=lim;k++)
        {
            suma=suma+B[i][k]*c1[k];
        }
        //se aplica la expresión 4 para obtener el valor de cada término de c1
        c1[i]=b[i]-suma;
    }

    index=ord-1;
    //Se aplica la expresión 5
    x[index]=c1[index]/B[index][index];
    lims=index-1;
    //este ciclo indica que elemento de x se esta calculando
    for(i=lims;i>=0;i--)
    {
        limi=i+1;
        suma=0.0;
        //este ciclo realiza la sumatoria
        for(k=index;k>=limi;k--)
        {
            suma=suma+B[i][k]*x[k];
        }
        //se aplica la expresión 6 para obtner los demás elementos de la
        //solución
        x[i]=(c1[i]-suma)/B[i][i];
    }

    //se muestran en la tabla 2 las matrices L y U almacenadas en el arreglo B
    for(i=0;i<=ord-1;i++)
       {
        for(j=0;j<=ord-1;j++)
        {
            sprintf(a,"%*.*f",3,4,B[i][j]);
            LuG->SetCellValue(i,j,a);
        }
    }
    //se muestra en la tabla 4 el resultado de la eliminación hacia adelante
    for(i=0;i<=ord-1;i++)
    {
        sprintf(a,"%*.*f",3,6,c1[i]);
        MatCG->SetCellValue(i,0,a);
    }
     //se muestra en la tabla 5 el resultado de la eliminación hacia atrás
    for(i=0;i<=ord-1;i++)
    {
        sprintf(a,"%*.*f",3,8,x[i]);
        SolucionG->SetCellValue(i,0,a);
    }

    }//control de la ejecución
}//principal
```

APÉNDICE C.5 CÓDIGO DE LAGRANGE

LISTA DE SELECCIÓN DE PUNTOS

```
void LagrangeDlg::PuntosLDSelected(wxCommandEvent& event )
{
    //sel almacena la selección de la lista desplegable
    //ins almacena el numero de filas y columnas a insertar
    int sel,ins;
    //Se obtiene la selección de la lista
    sel=PuntosLD->GetSelection();
    //La selección de la lista comienza en cero por lo que hay que
    //sumar 1 para obtener las filas a insertar
    ins=sel+1;
    //Se insertan filas en la tabla 1 a partir de la fila 2.
    TablaResG->InsertRows(2,ins);
    //Se limpia la tabla 1
    TablaResG->ClearGrid();
    //Se habilita el botón que habilita a la lista desplegable
    MasPuntosB->Enable(true);
    //Se deshabilita la lista de selección
    PuntosLD->Enable(false);
}
```

MAS PUNTOS

```
void LagrangeDlg::MasPuntosBClick(wxCommandEvent& event)
{
    // insert your code here
    //Se deshabilitan los componentes:
    XvalCT->Enable(false);     ResultCT->Enable(false);     SintaxisCT->Enable(false);
    ErrorST->Enable(false);    CalcularB->Enable(false);    OtroB->Enable(false);
    TabPolB->Enable(false);    EvaluarST->Enable(false);    ResultST->Enable(false);
    //Se ocultan los componentes para tabular
    DesdeCT->Show(false);    HastaCT->Show(false);    IncCT->Show(false);
    DesdeST->Show(false);    HastaST->Show(false);    IncST->Show(false);
    TablaG->Show(false);    TabularB->Show(false);
    //Se limpian los componentes:
    PolinomioCT->Clear();    DesdeCT->Clear();    HastaCT->Clear();
    IncCT->Clear();          SintaxisCT->Clear(); TablaG->ClearGrid();
    //Se ajusta el tamaño de la ventana
    Fit();
    int lim,erase;
    lim=TablaResG->GetNumberRows();
    erase=lim-2;
    if(lim>2)
    {
        TablaResG->DeleteRows(2,erase);
    }
    //Se habilita la lista desplegable
    PuntosLD->Enable(true);
    //Se deshabilita este botón
    MasPuntosB->Enable(false);
    //Se deshabilita la caja de texto que muestra la función
    PolinomioCT->Enable(false);
    //Se limpia la tabla 1
    TablaResG->ClearGrid();
    TablaResG->Fit();
}
```

CALCULAR

```
void LagrangeDlg::CalcBClick(wxCommandEvent& event)
{//principal
      // insert your code here
      //Se deshabilita la lista de selección de digitos decimales, si no hay errores
      //de sintaxis se habilitará
      DigitosLD->Enable(false);
      //Se limpia y deshabilita la caja de texto donde se muestra el polinomio
      PolinomioCT->Clear();
      PolinomioCT->Enable(false);
      //Limpia y desahbilita la caja de texto donde se muestran los errores
      SintaxisCT->Clear();
      SintaxisCT->Enable(false);
```

```
   ErrorST->Enable(false);
   //almacena el código de error
   int verifsin;
   //arreglo que almacena los códigos de error
   int sintax[21];
   //se limpia el arreglo sintax
   for(int i=0;i<=21;i++) sintax[i]=0;
   //indice es el indice del arreglo sintax
   //sumas almacena la suma del arreglo de sintax de la tabla 1
    int indice=0,sumas=0;
   //tamaño de las expresiones introducidas en las celdas de la tabla
   int tamano=30;
   //expresion: contiene la expresión original tecleada
   //eror contiene en cadena de caracteres la coordenada de error
    char expresion[tamano],eror[3];
    //Transformado: contiene a la expresion transformada después de aplicar
    //el método Transforma Expresión
    //ExprNegativos: contiene los menos unarios (0-1)
    //Mensaje: contiene al mensaje adecuado segun el código de error
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    //Se crea el objeto evaluadorExpresiones de la clase Evaluar
    Evaluar evaluadorExpresiones;
    //ord1 almacena el numero de filas de la tabla
    //ord  almacena el numero de filas menos uno
   int ord1,ord,i,j,k,c,n,d,p,de,de1;
   ord1=TablaResG->GetNumberRows();
   ord=ord1-1;
   //cadenas de caracteres para mostrar resultados
   char a1[3],b[3];

   float A[1][ord1];
    double x[ord],y[ord],m1[1],m2[ord],suma1[ord],coci,suma,a,valor;
   //Inicia la matriz de multiplicacion
   for(i=0;i<=1;i++)
  {
    for(j=0;j<=ord1;j++)
    {
        A[i][j]=0;
    }
  }
//
//Se obtienen los datos de los puntos
//El for externo tiene el control de las 2 columnas
for(int i=0;i<=1;i++)
{//for1
    //El for interno recorre todas las filas de la columna indicada por i
    for(int j=0;j<=ord;j++)
    {//for2
        //Se limpian los arreglos para el evaluador de expresiones
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; }
        //Se copia la expresión de la celda j,i
        strcpy(expresion, TablaResG->GetCellValue(j,i));
        //Se transforma la expresión
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        //Se obtiene el código de error
        verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        //Se almacena cada código de error en el arreglo sintax
        sintax[indice]=verifsin;
        //Si ha ocurrido error se muestran los valores j,i
        if(verifsin)

        {
           //Se habilita la caja de texto donde se muestran las posiciones
            SintaxisCT->Enable(true);
             ErrorST->Enable(true);
            //Se escribe un paréntesis
            SintaxisCT->WriteText("(");
```

```
            //Se convierte a cadena de carácteres el valor de j
            sprintf(eror,"%d",j+1);
            //Se escribe en la caja de texto
            SintaxisCT->WriteText(eror);
            //Se escribe una coma
            SintaxisCT->WriteText(",");
            //Se convierte a cadena de caracteres el valor de i
            sprintf(eror,"%d",i+1);
            //Se escribe en la caja de texto
            SintaxisCT->WriteText(eror);
            //se cierra el paréntesis
            SintaxisCT->WriteText(")");
            //se escribe une espacio
            SintaxisCT->WriteText(" ");
        }
        //El indice del arreglo sintax se incrementa en uno
        indice=indice+1;
    }//for2
}//for1
//Se obtiene la sumatoria del arreglo sintax
for(int i=0; i<=21;i++)
{
    if(sintax[i]!=0) sumas=sumas+1;
}
//Si la suma es distinta de cero ha ocurrido error y se muestra el mensaje
if(sumas) SintaxisDM->ShowModal();
//Si no ha ocurrido ningún error de sintaxis comienza el método
if(!sumas)
{//control de la ejecucion
//Se muestra la lista que muestra el número de digitos decimales
DigitosLD->Enable(true);
//Se obtienen los valores de los puntos
for(i=0;i<=ord;i++)
{//for3
    //Se limpian los arreglos
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    //Se copian las expresiones de la primera columna
    strcpy(expresion, TablaResG->GetCellValue(i,0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
     evaluadorExpresiones.Analizar(ExprNegativos);
     //Se obtiene el valor de las expresiones de la primera columna
     valor = evaluadorExpresiones.Calcular();
    //Se almacenan en el arreglo x
    x[i]=valor;
    //Se obtienen los valores de ls segunda columna
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, TablaResG->GetCellValue(i,1));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
    evaluadorExpresiones.Analizar(ExprNegativos);
    valor = evaluadorExpresiones.Calcular();
    //Los valores se almacenan en el arreglo y
    y[i]=valor;
}//for3
//
//POLINOMIO DE LAGRANGE
//Se inicializa el arreglo que almacena los coeficientes de cada sumando
for(k=0;k<=ord;k++) suma1[k]=0;
//Este ciclo indica que sumando se esta calculando
for(k=0;k<=ord;k++)
{//for4
    //El coeficiente de x simpre es uno en cada termino
    m1[1]=1.0;
    //Para cada sumando se reinicia el multiplicando m2
    for(c=0;c<=ord;c++) m2[c]=0;
```

```
    //Se inicializa el cociente y el primer elemento del multiplicando m2
    m2[0]=1.0; coci=1.0;
    //este ciclo toma cada uno de los términos que se van multiplicando
    for(n=0;n<=ord;n++)
    {
        //si n es distinto de k toma el siguiente término
        if(n!=k)
        {
            //se calcula paso a paso el valor del dividendo
            coci=coci*(x[k]-x[n]);
            //Se coloca el primer del elemento del arreglo para formar
            //cada uno de los terminos (x-1)=[1 -1] por ejemplo
            m1[0]=-x[n];
            //el for externo toma los elementos del arreglo m1
            for(i=0;i<=1;i++)
            {
                //el for interno multiplica cada elemento del arreglo
                //por todos los elementos del arreglo m2
                for(j=0;j<=ord;j++)
                {
                    //j+1 se refiere a que cuando i=1 se multiplica por
                    //el coeficiente de x y debe aumentar en 1 el grado
                    //de cada término por lo que se recorren los elementos
                    //una posición a la izquierda
                    A[i][j+i]=m1[i]*m2[j];
                }
            }
            //El for externo recorre las columnas de la matriz A
            for(j=0;j<=ord;j++)
            {
                //Se suma cada columna y cada iteración se resetea
                //el valor de suma
                suma=0;
                //se suman los dos elementos de cada columna
                for(i=0;i<=1;i++)
                {
                    suma=suma+A[i][j];
                }
                //cada suma se almacena en la misma posición de columna
                //indicada por j en el arreglo m2 para la siguiente
                //multiplicación
                m2[j]=suma;
            }
        }
    }
    //Al final de los ciclos anteriores en el arreglo m2 quedan almacenados
    //los coeficientes de las multiplicaciones de los terminos de cada
    //sumando y se aplica el siguiente ciclo para obtener los coeficientes
    //de cada sumando y sumarlos en cada iteración hasta obtener los
    //coeficientes finales del polinomio
    for(c=0;c<=ord;c++) suma1[c]=suma1[c]+(m2[c]*y[k])/coci;
}//for4
//Acaba la obtencion de los coeficiente del polinomio
//Se procede a mostrar el polinomio
//Se obtiene la seleccion de la lista desplegable 2
de=DigitosLD->GetSelection();
//Se suma 1 para obtener el número de dígitos decimales
de1=de+1;
//Si no se ha realizado alguna selección de la lista 2 se establecen
//4 dígitos decimales por defecto
if(de==-1) de1=4;
//Este ciclo escribe el polinomio obtenido
for(i=ord;i>=1;i--)
{
        //Cada coeficiente de sumas1 se convierte a cadena de caracteres
        sprintf(a1,"%*.*f*x^",3,de1,suma1[i]);
        //Se escribe cada coeficiente y se escribe '*x^'
        PolinomioCT->WriteText(a1);
```

```
            //Se convierte a cadena de caracteres el exponente
            sprintf(a1,"%d",i);
            //Se escribe el exponente
            PolinomioCT->WriteText(a1);
            //Si el siguiente coeficiente es positivo se escribe '+'
            if(suma1[i-1]>=0) PolinomioCT->WriteText("+");
    }
    //Se escribe el término independiente
    sprintf(a1,"%*.*f",3,de1,suma1[0]);
    PolinomioCT->WriteText(a1);
    //Se habilitan los siguientes componentes
    PolinomioCT->Enable(true);
    XvalCT->Enable(true);
    EvaluarST->Enable(true);
    ResultST->Enable(true);
    CalcularB->Enable(true);
    OtroB->Enable(true);
    TabPolB->Enable(true);
    }//control de la ejecución
}//principal
```

NUEVOS DATOS

```
void LagrangeDlg::NuevosBClick(wxCommandEvent& event)
{
   // insert your code here
   TablaResG->ClearGrid();    PolinomioCT->Clear();         DigitosLD->Enable(false);
   SintaxisCT->Clear();       SintaxisCT->Enable(false);    ErrorST->Enable(false);
   XvalCT->Clear();           ResultCT->Clear();            PolinomioCT->Enable(false);
   XvalCT->Enable(false);     ResultCT->Enable(false);      EvaluarST->Enable(false);
   ResultST->Enable(false);   CalcularB->Enable(false);     OtroB->Enable(false);
   TabPolB->Enable(false);    DesdeCT->Show(false);         HastaCT->Show(false);
   IncCT->Show(false);        DesdeST->Show(false);         HastaST->Show(false);
   IncST->Show(false);        TablaG->Show(false);          TabularB->Show(false);
   DesdeCT->Clear();          HastaCT->Clear();             IncCT->Clear();
   TablaG->ClearGrid();
   Fit();

}
```

OTRO VALOR

```
void LagrangeDlg::OtroBClick(wxCommandEvent& event)
{
       // insert your code here
       XvalCT->Clear();          ResultCT->Clear();        ResultCT->Enable(false);
       XvalCT->Enable(true);     CalcularB->Enable(true);
       OtroB->Enable(false);
}
```

TABULAR POLINOMIO

```
void LagrangeDlg::TabPolBClick(wxCommandEvent& event)
{
       // insert your code here
       DesdeCT->Show(true);      HastaCT->Show(true);      IncCT->Show(true);
       DesdeST->Show(true);      HastaST->Show(true);      IncST->Show(true);
       TablaG->Show(true);       TabularB->Show(true);
       Fit();
}
```

TABULAR

```
void LagrangeDlg::TabularBClick(wxCommandEvent& event)
{
    //Se copia aqui la parte del código del botón calcular hasta donde son obtenidos los
    //coeficientes del arreglo suma
    int verifsinp1=0,verifsinp2=0,verifsinp3=0;
    int tamano=200;
    char expresion[tamano];
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    Evaluar evaluadorExpresiones;
    TablaG->ClearGrid();
    if (DesdeCT->IsEmpty()) DesdeDM->ShowModal();
    else
    {
```

```
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
     = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, DesdeCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    verifsinp1 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}
if (HastaCT->IsEmpty()) HastaDM->ShowModal();
else
{
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
     = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, HastaCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    verifsinp2 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}
if (IncCT->IsEmpty()) IncDM->ShowModal();
else
{
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, IncCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    verifsinp3 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}
int verifpar=verifsinp1+verifsinp2+verifsinp3;
if(verifpar) SintaxisDM->ShowModal();
int validator=!DesdeCT->IsEmpty()*!HastaCT->IsEmpty()
*!IncCT->IsEmpty();
if(validator&&verifpar==0)
{//if1
int filas,j,del;
char d[3];
filas=TablaG->GetNumberRows();
sprintf(d,"%d",filas);
if (filas>14)
{
    del=filas-14;
    TablaG->DeleteRows(11,del);
}
for(j=0;j<filas;j++) TablaG->SetCellTextColour(j,1,"black");
TablaG->SetColLabelValue(0,"x");
TablaG->SetColLabelValue(1,"f(x)");
int iter;
double i1,dsd,sta,inc;
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, DesdeCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
dsd=evaluadorExpresiones.Calcular();
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, HastaCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
sta=evaluadorExpresiones.Calcular();
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, IncCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
inc=evaluadorExpresiones.Calcular();
i1=dsd;
iter=0;
do
```

```
    {
        if(iter>13) TablaG->InsertRows(iter,1);

         for(i=1;i<=ord;i++)
        {
            suma2[i]=suma1[i]*pow(i1,i);
        }
        suma=0;
        suma2[0]=suma1[0];
        for(i=0;i<=ord;i++)
        {
            suma=suma+suma2[i];
        }
       if (suma<0)
        {
            sprintf(a1,"%*.*f",3,7,suma);
            TablaG->SetCellValue(iter,1,a1);
            TablaG->SetCellTextColour(iter,1,"red");
        }
        else if(suma>0)
        {
            sprintf(a1,"%*.*f",3,7,suma);
            TablaG->SetCellValue(iter,1,a1);
            TablaG->SetCellTextColour(iter,1,"black");
        }
        else if(suma==0.0)
        {
            sprintf(a1,"%*.*f",3,7,suma);
            TablaG->SetCellValue(iter,1,a1);
            TablaG->SetCellTextColour(iter,1,"blue");
        }
        sprintf(b,"%*.*f",3,4,i1);
        TablaG->SetCellValue(iter,0,b);
       i1=i1+inc;
       iter=iter+1;
      }   while (i1<=sta);
    }//if1
}
```

EVALUAR

```
void LagrangeDlg::CalcularBClick(wxCommandEvent& event)
{
    //Se copia aqui la parte del código del botón calcular hasta donde son obtenidos los
    //coeficientes del arreglo suma
    int tamano=30,verifsinp1=0;
    char expresion[tamano],eror[3];
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    Evaluar evaluadorExpresiones;

    //Si no se ha insertado un valor a interpolar se muestra mensaje
    if(XvalCT->IsEmpty())  XvalDM->ShowModal();

        else
        {
            //Se verifica la sintaxis del valor insertado
            for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
             = '\0'; ExprNegativos[inicializa] = '\0';  }
            strcpy(expresion, XvalCT->GetLineText(0));
            evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
            verifsinp1 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        }
        //Si hay error de sintaxis se muestra el mensaje correspondiente
        if(verifsinp1) SintaxisDM->ShowModal();
        //Variable de validación de caja vacía
        int validator=!XvalCT->IsEmpty();
        //Si no hay error de sintaxis y la caja de texto ha sido llenada
        if(validator&&verifsinp1==0)
        {//control de la ejecución
```

```
        //Se habilitan los siguientes componentes
        ResultCT->Enable(true);
           XvalCT->Enable(true);
           OtroB->Enable(true);
           //Se deshabilita el siguiente componente
           CalcularB->Enable(false);

        //Se obtiene el valor a interpolar
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
        strcpy(expresion, XvalCT->GetLineText(0));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
           evaluadorExpresiones.Analizar(ExprNegativos);
           double m = evaluadorExpresiones.Calcular();

           //Se sustituye el valor en el polinomio
        for(i=1;i<=ord;i++)
        {
            suma2[i]=suma1[i]*pow(m,i);
        }
        suma=0;
        suma2[0]=suma1[0];
        //Se suman los resultados del arreglo suma2
        for(i=0;i<=ord;i++)
        {
            suma=suma+suma2[i];
        }
        //Se convierte a cadena de caracteres el resultado
        sprintf(b,"%*.*f",3,10,suma);
        //Se muestra el resultado en la caja de texto
        ResultCT->WriteText(b);
        }//control de la ejecucción

}
```

APÉNDICE C.6 CÓDIGO DE INTERPOLACIÓN DE NEWTON

Mas puntos

```
void Interpolacion_NewtonDlg::MasPuntosBClick(wxCommandEvent& event)
{
    int lim,erase;
    PuntosLD->Enable(true);
    MasPuntosB->Enable(false);
    lim=DiferenciasG->GetNumberRows();
    erase=lim-2;
    if(lim>2)
    {
        DiferenciasG->DeleteRows(2,erase);
        AjusteG->DeleteRows(2,erase);
        DiferenciasG->DeleteCols(2,erase);
    }
    DiferenciasG->ClearGrid();
    AjusteG->ClearGrid();
    Fit();
}
```

DIFERENCIAS

```
void Interpolacion_NewtonDlg::DiferenciasBClick(wxCommandEvent& event)
{//principal
    // insert your code here
    //Se limpia la caja de texto donde se muestrab los errores y se deshabilita
    SintaxisCT->Clear();
    SintaxisCT->Enable(false);
    SintaxisST->Enable(false);
    //Se deshabilita la lista de ajuste de puntos
    DigitosLD->Enable(false);
    //Variable de verificación de sintaxis
    int verifsin;
    //Arreglo que almacena los códigos de error
    int sintax[11];
    //Se limpia el arreglo sintax
    for(int i=0;i<=11;i++) sintax[i]=0;
    //indice almacena el índice del arreglo sintax
    //sumas almacena la sumatoria de los códigos de error de la tabla 1
    int indice=0,sumas=0,sumas1=0;
    //tamaño de las expresiones insertadas en las celdas
    int tamano=30;
    //expresion: contiene la expresión original tecleada
    //eror contiene en cadena de caracteres la coordenada de error
    char expresion[tamano],eror[3];
    //Transformado: contiene a la expresion transformada después de aplicar
    //el método Transforma Expresión
    //ExprNegativos: contiene los menos unarios (0-1)
    //Mensaje: contiene al mensaje adecuado segun el código de error
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    //Se crea el objeto evaluadorExpresiones de la clase Evaluar
    Evaluar evaluadorExpresiones;
    //se obtiene el numero de filas de la tabla 1
    int ord1=DiferenciasG->GetNumberRows();
    int ord=ord1-1;
        //recorre todos los elementos de la tabla 1 columna 0
        for(int j=0;j<=ord;j++)
        {//for1
            //Se limpian los arreglos para el evaluador de expresiones
            for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
            = '\0'; ExprNegativos[inicializa] = '\0'; }
            //Se copia la expresión de la celda j,0
            strcpy(expresion, DiferenciasG->GetCellValue(j,0));
            //Se transforma la expresión
            evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
            //Se obtiene el código de error
            verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);
            //Se almacena cada código de error en el arreglo sintax
            sintax[indice]=verifsin;
            //Si ha ocurrido error se muestran los valores j,0
```

```
        if(verifsin)

        {   //Se habilita la caja de texto donde se muestran las posiciones
            SintaxisCT->Enable(true);
               SintaxisST->Enable(true);
               //Se escribe parentesis que abre
            SintaxisCT->WriteText("(");
            sprintf(eror,"%d",j);
            //Se escribe el valor de j+1
            SintaxisCT->WriteText(eror);
            //Ya que la columna es fija se escribe ,0)
            SintaxisCT->WriteText(",0)");
            //Se escribe un espacio
            SintaxisCT->WriteText(" ");


        }
        //Se incrmenta en uno el índice
        indice=indice+1;
    }//for1
//Sumatoria del arreglo sintax
for(int i=0; i<=11;i++)
{
    if(sintax[i]!=0) sumas=sumas+1;
}
 //Si la suma es distinta de cero ha ocurrido error y se muestra el mensaje
if(sumas) SintaxisXDM->ShowModal();
//Si no ha ocurrido error se procede a verificar la sintaxis de la tabla 3
else
{
    //Se resetea el arreglo sintax
    for(int i=0;i<=11;i++) sintax[i]=0;
    //Se resetea la variable sintax
    indice=0;
    //Se inicia verifsin=0;
    verifsin=0;
    //Este ciclo recorre todos los elementos de la tabla
    for(int j=0;j<=ord;j++)
    {//for2

        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; }
        strcpy(expresion, AjusteG->GetCellValue(j,0));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        //Se almacenan los códigos de error
        sintax[indice]=verifsin;
        //Si hay error se muestra la posición del error
        if(verifsin)

        {//if1
            SintaxisCT->Enable(true);
               SintaxisST->Enable(true);
            SintaxisCT->WriteText("(");
            sprintf(eror,"%d",j);
            SintaxisCT->WriteText(eror);
            SintaxisCT->WriteText(")");
            SintaxisCT->WriteText(" ");
        }//if1
        //Se incrementa indice
        indice=indice+1;
    }//for2
    //Sumatoria del arreglo
    for(int i=0; i<=11;i++)
    {
        if(sintax[i]!=0) sumas1=sumas1+1;
    }
    //Si ha ocurrido error se muestra el mensaje de error
```

```
    if(sumas1) SintaxisYDM->ShowModal();

}
   //Si no hay errores de sintaxis se obtienen las diferencias divididas
   if(!sumas&&!sumas1)
   {//control de la ejecucion
    //Se habilita el botón nuevos datos
   NewDataB->Enable(true);
   //Se deshabilita el botón diferencias
   DiferenciasB->Enable(false);
   //Se habilita la lista de puntos para el ajuste
   DigitosLD->Enable(true);
   //ord1 almacena el numero de filas de las tablas
    //ord es el numero filas menos uno
    //i y j son contadores
   int i,j,lim;
   //para convertir a cadenas de caracteres
   char a[3];
   //Se obtiene el numero de filas
   //ord1=DiferenciasG->GetNumberRows();
   //Se calcula ord
   //ord=ord1-1;
   //C es una matriz cuadrada que almacena en la columna 0 los valores de y
   //y en el resto de la matriz las diferencias divididas
   //x es el arreglo que almacena los valores de x
   double C[ord][ord],x[ord];
   //este ciclo obtiene los valores de x, y
   for(i=0;i<=ord;i++)
   {
     //Se limpian los arreglos
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    //Se obtiene un valor de y
    strcpy(expresion, DiferenciasG->GetCellValue(i,0));
    //se transforma la expresión
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
    //se agregan los menos unarios
   evaluadorExpresiones.Analizar(ExprNegativos);
    //se calcula el valor insertado en la celda
   double valor = evaluadorExpresiones.Calcular();
    //se almacena en la primera columna de C
    C[i][0]=valor;
    //Se limpian los arreglos
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    //Se copia en expresión cada valor de x
    strcpy(expresion, AjusteG->GetCellValue(i,0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
   evaluadorExpresiones.Analizar(ExprNegativos);
   valor = evaluadorExpresiones.Calcular();
   //Se almacenan el arreglo x los valores obtenidos
    x[i]=valor;
   }
 //se limpia la tabla cada que se calculan las diferencias
 //comenzando por la segunda columna 1 ya que en la columna cero estan los
 //valores de y
  for(j=1;j<=ord;j++)
  {
   for(i=0;i<=ord;i++)
   {
       DiferenciasG->SetCellValue(i,j," ");
   }
  }
 //
  //Se obtienen las diferencias divididas
  for(j=0;j<=ord-1;j++)
```

```
       {
        //lim sirve para limitar hasta donde se hacen los calculos
        //genera la tabla escalonada
        lim=ord-j-1;
        //este ciclo calcula los valores de cada columna
        for(i=0;i<=lim;i++)
        {
            C[i][j+1]=(C[i+1][j]-C[i][j])/(x[i+j+1]-x[i]);
            sprintf(a,"%*.*f",3,7,C[i][j+1]);
            DiferenciasG->SetCellValue(i,j+1,a);
        }
    }
    PolinomioB->Enable(true);
    }
}
```

NUEVOS DATOS

```
void Interpolacion_NewtonDlg::NewDataBClick(wxCommandEvent& event)
{
      DigitosST->Enable(false); DigitosLD->Enable(false); DiferenciasB->Enable(true);
      NewDataB->Enable(false);  DiferenciasG->ClearGrid(); PolinomioCT->Enable(false);
      PolinomioCT->Clear();     XvalCT->Clear();           ResultCT->Clear();
      XvalG->ClearGrid();       AjusteG->ClearGrid();      InterpolarST->Enable(false);
       ResultST->Enable(false);  EvaluarB->Enable(false);  OtroValB->Enable(false);
       XvalCT->Enable(false);    ResultCT->Enable(false);  DesdeST->Show(false);
      HastaST->Show(false);     IncST->Show(false);        DesdeCT->Show(false);
      HastaCT->Show(false);     IncCT->Show(false);        DesdeCT->Clear();
      HastaCT->Clear();         IncCT->Clear();            NpolinomioB->Enable(false);
      TabularB->Show(false);    TablaG->Show(false);       TablaG->ClearGrid();
      PolinomioB->Enable(false);TabPolB->Enable(false);    Fit();
}
```

Mas puntos (ajuste)

```
void Interpolacion_NewtonDlg::AjusteBClick(wxCommandEvent& event)
{
      int lim,erase;
      AjusteLD->Enable(true);
      AjusteB->Enable(false);
      lim=XvalG->GetNumberRows();
      erase=lim-2;
      if(lim>2)
      {
       XvalG->DeleteRows(2,erase);
       }
    XvalG->ClearGrid();
    Fit();
}
```

POLINOMIO

```
*/
void Interpolacion_NewtonDlg::PolinomioBClick(wxCommandEvent& event)
{//principal
    int row,row1,control,ini,ini1,end,lim,prev,next,con,i,warn,dec;
    double r1;
    char b[3];
    //tamaño de las expresiones insertadas en las celdas
    int tamano=30;
    //expresion: contiene la expresión original tecleada
    //eror contiene en cadena de caracteres la coordenada de error
    char expresion[tamano],eror[3];
    //Transformado: contiene a la expresion transformada después de aplicar
    //el método Transforma Expresión
    //ExprNegativos: contiene los menos unarios (0-1)
    //Mensaje: contiene al mensaje adecuado segun el código de error
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    //Se crea el objeto evaluadorExpresiones de la clase Evaluar
    Evaluar evaluadorExpresiones;
    //almacena el valor de las expresiones insertadas en las celdas
    double valor;
    //Se obtiene el numero de filas de la tabla 2
    row=XvalG->GetNumberRows();
```

```
//Se calcula row1
row1=row-1;
//
double suma1[row1];
//se inicializa control
control=0;
//ini almacena el primer valor de la tabla
ini=atoi(XvalG->GetCellValue(0,0));
//se calcula ini1
ini1=ini-1;
//end almacena el último valor de la tabla
end=atoi(XvalG->GetCellValue(row1,0));
//lim almacena el numero de filas de la tabla1
lim=DiferenciasG->GetNumberRows();
//Si el ultimo numero de los puntos de ajuste es mayor que el numero
//de puntos insertados se muestra el mensaje correspondiente
if(end>lim) VerifiqueDM->ShowModal();
//tabla completa?
//este ciclo verificqa que se haya llenado la tabla de puntos para el
//ajuste
for(i=0;i<=row1;i++)
{
    //si se ha insertado un valor se incrementa control en 1
    if(XvalG->GetCellValue(i,0)!="") control=control+1;
}
//si se ha omitido algun valor se muestra mensaje correspondiente
if(control!=row) CompleteTablaDM->ShowModal();
//valores consecutivos?
//Se ha llenado la tabla de ajuste?
if(control==row)
{//if1
//se inicializa warn
warn=0;
//este ciclo recorre todos los valores de ajuste
for(i=0;i<=row1-1;i++)
{
    //se obtiene un valor y su consecutivo
    prev=atoi(XvalG->GetCellValue(i,0));
    next=atoi(XvalG->GetCellValue(i+1,0));
    //los valores son restados
    con=next-prev;
    //si la diferencia entre los valores es diferente de 1
    //warn se incrementa en uno
    if(con!=1) warn=warn+1;
}
//Si warn es distinto de cero se muestra el mensaje correspondiente
if(warn>0) ConsecutivosDM->ShowModal();
}//if1

//Si se ha llenado la tabla de ajuste, el último valor no excede el
//número de puntos y los valores son consecutivos se procede
if(control==row&&end<=lim&&warn==0)
{//control de la ejecución
    //Se habilita el botón Polinomio
    PolinomioB->Enable(false);
    //Se habilita el botón Nuevo Polinomio
    NpolinomioB->Enable(true);
    //Se habilita la caja donde se muestra el polinomio
    InterpolarST->Enable(true);
    ResultST->Enable(true);
    //Se habilita el botón Evaluar
    EvaluarB->Enable(true);
    //Se habilita la caja donde se inserta el valor a interpolar
    XvalCT->Enable(true);
    PolinomioCT->Enable(true);
    //Se limpian los componentes:
    PolinomioCT->Clear();
    WxEdit7->Clear();
```

```
    DigitosST->Enable(true);
    //Se habilita la lista de selección de dígitos
       DigitosLD->Enable(true);

   //Método de Interpolación de Newton
   int ord1,ord,i,j,k,c,n,ope;
   //Se obtiene el numero de filas de la tabla de ajuste
   ord1=XvalG->GetNumberRows();
   //se calcula ord
   ord=ord1-1;
   //la matriz A sirve para multiplicacion de terminos como en el caso de
   //Lagrange
   //los arreglos x e y almacenan los valores de los puntos insertados
   //m1 y m2 son dos multiplicandos
   //suma1 almacena los coeficientes de cada sumando
   //suma almacena la suma de cada columna de A en las multiplicaiones
   //el arreglo C almacena las diferencias dividas
   double A[1][ord1],x[ord],y[ord],m1[1],m2[ord],suma1[ord],suma,C[ord][ord];
double r[ord];
char a[3];
Se ininicia la matriz de multiplicacion
for(i=0;i<=1;i++)
{
    for(j=0;j<=ord1;j++)
    {
        A[i][j]=0;
    }
}

//Se obtienen los valores de los coeficientes de x comenzando por el
//primer valor indicado en la tabla de ajuste y adquiriendo ini1+ord
//valores de x
for(i=ini1;i<=ini1+ord;i++)
{
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, AjusteG->GetCellValue(i,0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
       evaluadorExpresiones.Analizar(ExprNegativos);
       valor = evaluadorExpresiones.Calcular();
    x[i-ini1]=valor;
}
//Se obtienen los valores de los coeficientes de y comenzando por el
//primer valor indicado en la tabla de ajuste y adquiriendo ini1+ord
//valores de y
for(i=ini1;i<=ini1+ord;i++)
{
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, DiferenciasG->GetCellValue(i,0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
       evaluadorExpresiones.Analizar(ExprNegativos);
       valor = evaluadorExpresiones.Calcular();
    C[i-ini1][0]=valor;
}
// Se genera una matriz de diferencias divididas
for(j=0;j<=ord-1;j++)
   {
    lim=ord-j;
    for(i=0;i<=lim;i++)
    {

        C[i][j+1]=(C[i+1][j]-C[i][j])/(x[i+j+1]-x[i]);
    }
}
//
```

```
    //Se obtienen los valores de interes para el cálculo del polinomio que
    //son los de la primera fila de la matriz de diferencias divididas
    for(i=0;i<=ord;i++)
    {
        r[i]=C[0][i];
    }



    //6 Obtención del polinomio interpolador de Newton
    //Se inicializa el arreglo suma1
    for(k=0;k<=ord;k++) suma1[k]=0;
//El ciclo externo indica el sumando que se esta trabajando
    for(k=0;k<=ord;k++)
    {//for externo
        //El segundo elemento del multiplicando 1 es igual a 1
        m1[1]=1.0;
        //Se limpia el arreglo del multiplicando 2
        for(c=0;c<=ord;c++) m2[c]=0;
        //Se inicializa el elemento 0 de m2
        m2[0]=1.0;
        //indica cuantos terminos serán multiplicados
        ope=k-1;
        //cuando k=0 solo se coloca el primer valor del renglon de C
        if(ope==-1) suma1[0]=r[0];
        //Cuando k>0 se realiza multiplicación de terminos
        else
        {//else
        //este ciclo indica el termino que se toma
        for(n=0;n<=ope;n++)
        {
                //para el multiplicando se hace el elemento cero igual
                //al valor de x indicado por el índice n
                m1[0]=-x[n];
                //Estos ciclos multiplican m1*m2 elemento a elemento
                for(i=0;i<=1;i++)
                {
                    for(j=0;j<=ord;j++)
                    {
                        A[i][j+i]=m1[i]*m2[j];
                    }
                }
                //Estos ciclos suman las columnas de A
                for(j=0;j<=ord;j++)
                {
                    suma=0;
                    for(i=0;i<=1;i++)
                    {
                        suma=suma+A[i][j];
                    }
                    //Se almacena la suma de cada columna en el arreglo m2
                    //para la siguiente multiplicación
                    m2[j]=suma;
                }

        }
        //Se multiplica cada coeficiente por cada elemento del 1er renglon
        //de las diferencias divididas
        for(c=0;c<=ord;c++) suma1[c]=suma1[c]+m2[c]*r[k];
        }//else
   }//for externo
   //se procede a mostrar en forma de polinomio
   //Si no se ha realizado ninguna selección de dígitos se establece 4 por
   //defecto
   if(DigitosLD->GetSelection()==-1) dec=4;
   //De lo contrario se obtiene el número de dígitos decimales
   else dec=DigitosLD->GetSelection()+2;
```

```
    for(i=ord;i>=1;i--)
    {
        if(suma1[i]<0) r1=-1.0*suma1[i];//***
        else r1=suma1[i];//***
        //if(suma1[0]==0.0) suma1[0]==0.000001;
        if(r1>0.0000001)//***
        {
            //Cada coeficiente de sumas1 se convierte a cadena de caracteres
            sprintf(b,"%*.*f*x^",3,dec,suma1[i]);
            //Se escribe cada coeficiente y se escribe '*x^'
            PolinomioCT->WriteText(b);
            //Se convierte a cadena de caracteres el exponente
            sprintf(b,"%d",i);
            //Se escribe el exponente
            PolinomioCT->WriteText(b);
            //Si el siguiente coeficiente es positivo se escribe '+'
            if(suma1[i-1]>=0) PolinomioCT->WriteText("+");

        }
    }
    //Se escribe el término independiente
    sprintf(b,"%*.*f",3,dec,suma1[0]);
    PolinomioCT->WriteText(b);
    //Se habilita el botón Tabular Polinomio
    TabPolB->Enable(true);
    }//control de la ejecución
}//principal
```

EVALUAR

```
void Interpolacion_NewtonDlg::EvaluarBClick(wxCommandEvent& event)
{
       // Se utiliza el mismo codigo que obtiene el arreglo suma1
        //Se va a evaluar el polinomio
        int tamano=200,verifsinp1=0;
        char expresion[tamano],b[3];
        char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
        Evaluar evaluadorExpresiones;
        if(XvalCT->IsEmpty())  XvalDM->ShowModal();
          else
        {
            for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
             = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
            strcpy(expresion, XvalCT->GetLineText(0));
            evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
            verifsinp1 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        }
       if(verifsinp1) SintaxisDM->ShowModal();
       int validator=!XvalCT->IsEmpty();
       if(validator&&verifsinp1==0)
       {//control de la ejecución
       ResultCT->Enable(true);
        OtroValB->Enable(true);
        EvaluarB->Enable(false);
        XvalCT->Enable(false);
       for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
       = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
       strcpy(expresion, XvalCT->GetLineText(0));
       evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
       if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
       evaluadorExpresiones.Analizar(ExprNegativos);
       double m = evaluadorExpresiones.Calcular();
        for(i=1;i<=ord;i++)
        {
            suma2[i]=suma1[i]*pow(m,i);
        }
        suma=0;
        suma2[0]=suma1[0];
        for(i=0;i<=ord;i++)
```

```
        {
            suma=suma+suma2[i];
        }
        sprintf(b,"%*.*f",3,7,suma);
        ResultCT->WriteText(b);
        }//control de la ejecución
}//Principal



/*Otro Valor
void Interpolacion_NewtonDlg::OtroValBClick(wxCommandEvent& event)
{
      // insert your code here
      XvalCT->Clear();    ResultCT->Clear();  ResultCT->Enable(false);
      ResultCT->Clear();  XvalCT->Enable(true);     EvaluarB->Enable(true);
      OtroValB->Enable(false);
}
Nuevo Polinomio
void Interpolacion_NewtonDlg::NpolinomioBClick0(wxCommandEvent& event)
{
    DigitosST->Enable(false);    DigitosLD->Enable(false);    NpolinomioB->Enable(false);
    PolinomioB->Enable(true);    PolinomioCT->Enable(false);  PolinomioCT->Clear();
    XvalCT->Clear();             ResultCT->Clear();           InterpolarST->Enable(false);
    ResultST->Enable(false);     EvaluarB->Enable(false);     OtroValB->Enable(false);
    XvalCT->Enable(false);       ResultCT->Enable(false);     DesdeST->Show(false);
    HastaST->Show(false);        IncST->Show(false);          DesdeCT->Show(false);
    HastaCT->Show(false);        IncCT->Show(false);          DesdeCT->Clear();
    HastaCT->Clear();            IncCT->Clear();              TabularB->Show(false);
    TablaG->Show(false);         TablaG->ClearGrid();         TabPolB->Enable(false);
    Fit();
}
TABULAR Polinomio
void Interpolacion_NewtonDlg::TabPolBClick(wxCommandEvent& event)
{
      DesdeST->Show(true);      HastaST->Show(true);      IncST->Show(true);
      DesdeCT->Show(true);      HastaCT->Show(true);      IncCT->Show(true);
      TabularB->Show(true);     TablaG->Show(true);       Fit();
}
TABULAR
void Interpolacion_NewtonDlg::TabularBClick0(wxCommandEvent& event)
{//principal
    //Utiliza el mismo código que obtiene el arreglo suma1
    //Tabulación
    int verifsin,verifsinp1,verifsinp2,verifsinp3;
    TablaG->ClearGrid();
    if (DesdeCT->IsEmpty()) DesdeDM->ShowModal();
    else
    {
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
         = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
        strcpy(expresion, DesdeCT->GetLineText(0));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        verifsinp1 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
    }
    if (HastaCT->IsEmpty()) HastaDM->ShowModal();
    else
    {
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
         = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
        strcpy(expresion, HastaCT->GetLineText(0));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        verifsinp2 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
    }
    if (IncCT->IsEmpty()) IncDM->ShowModal();
    else
    {
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
```

```
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, IncCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    verifsinp3 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}
int verifpar=verifsinp1+verifsinp2+verifsinp3;
if(verifpar) SintaxisDM->ShowModal();
int validator=!DesdeCT->IsEmpty()*!HastaCT->IsEmpty()
*!IncCT->IsEmpty();

if(validator&&verifpar==0)
{//if tabla
int filas,j,del;
char d[3];
filas=TablaG->GetNumberRows();
sprintf(d,"%d",filas);
if (filas>14)
{
    del=filas-14;
    TablaG->DeleteRows(11,del);
}
for(j=0;j<filas;j++) TablaG->SetCellTextColour(j,1,"black");
TablaG->SetColLabelValue(0,"x");
TablaG->SetColLabelValue(1,"f(x)");
int iter;
double i1,dsd,sta,inc;
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, DesdeCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
dsd=evaluadorExpresiones.Calcular();
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, HastaCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
sta=evaluadorExpresiones.Calcular();
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, IncCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
inc=evaluadorExpresiones.Calcular();
i1=dsd;
iter=0;
do
{
    if(iter>13) TablaG->InsertRows(iter,1);

      //Evaluador propio :)
      for(i=1;i<=ord;i++)
    {
        suma2[i]=suma1[i]*pow(i1,i);
    }
    suma=0;
    suma2[0]=suma1[0];
    for(i=0;i<=ord;i++)
    {
        suma=suma+suma2[i];
    }
      //Evaluador propio :)
      if (suma<0)
    {
        sprintf(a,"%*.*f",3,7,suma);
```

```
            TablaG->SetCellValue(iter,1,a);
            TablaG->SetCellTextColour(iter,1,"red");
         }
         else if(suma>0)
         {
            sprintf(a,"%*.*f",3,7,suma);
            TablaG->SetCellValue(iter,1,a);
            TablaG->SetCellTextColour(iter,1,"black");

         }
         else if(suma==0.0)
         {
            sprintf(a,"%*.*f",3,7,suma);
            TablaG->SetCellValue(iter,1,a);
            TablaG->SetCellTextColour(iter,1,"blue");
         }

         sprintf(b,"%*.*f",3,4,i1);
         TablaG->SetCellValue(iter,0,b);
           i1=i1+inc;
           iter=iter+1;

       }   while (i1<=sta+0.0000001);
    }//if tabla
}//principal

/*LISTA DE SELECCIÓN 1
void Interpolacion_NewtonDlg::PuntosLDSelected(wxCommandEvent& event )
{
       // insert your code here
       float num,i;
       char a[3];
       int sel,ins;
       num=0.0000;
       sel=PuntosLD->GetCurrentSelection();
       ins=sel+1;
       DiferenciasG->InsertRows(2,ins);
       DiferenciasG->InsertCols(2,ins);
       AjusteG->InsertRows(2,ins);

       for(i=0;i<=ins+1;i++)
       {
         sprintf(a,"%*.*f",3,7,num);
         DiferenciasG->SetCellValue(0,i,a);
    }
    PuntosLD->Enable(false);    MasPuntosB->Enable(true);    DiferenciasG->ClearGrid();
    AjusteG->ClearGrid();
}

LISTA DE SELECCIÓN 2
void Interpolacion_NewtonDlg::AjusteLDSelected(wxCommandEvent& event )
{
    float num,i;
       char a[3];
       int sel,ins,val;
       val=DiferenciasG->GetNumberRows();
       sel=AjusteLD->GetSelection()+3;
       if(sel>val) OpcionInvalidaDM->ShowModal();
       if(sel<=val)
       {
         sel=AjusteLD->GetCurrentSelection();
         ins=sel+1;
         XvalG->InsertRows(2,ins);
        AjusteLD->Enable(false);
        AjusteB->Enable(true);
        XvalG->ClearGrid();
      }
}
```

APÉNDICE C.7 CÓDIGO DE SPLINE

CALCULAR

```
*/
void SplineDlg::CalcularBClick(wxCommandEvent& event)
{
    // insert your code here
    //Se deshabilitan los componentes que muestran error
    DesplazarSB->SetValue(0);     SpinST->Enable(true);    DigitosLD->Enable(false);
    DigitosST->Enable(true);      DesplazarSB->Enable(false); PolinomioCT->Enable(false);
    PolinomioST->Enable(false);   X1CT->Enable(false);       X2CT->Enable(false);
    X1ST->Enable(false);          X2ST->Enable(false);       ErrorCT->Enable(false);
    ErrorCT->Clear();             ErrorST->Enable(false);    PolinomioCT->Clear();
    X1CT->Clear();                X2CT->Clear();
    //almacena el código de error
    int verifsin,n;
    char w[200];
    bool ypd=true;
    bool ysd=true;
    bool verifpar=true;
    bool verifpar1=true;
    int verifp1=0,verifp2=0;
    int dig;
    //arreglo que almacena los códigos de error
    int sintax[21];
    //se limpia el arreglo sintax
    for(int i=0;i<=21;i++) sintax[i]=0;
    //indice es el indice del arreglo sintax
    //sumas almacena la suma del arreglo de sintax de la tabla 1
    int indice=0,sumas=0;
    //tamaño de las expresiones introducidas en las celdas de la tabla
    int tamano=30;
    //expresion: contiene la expresión original tecleada
    //eror contiene en cadena de caracteres la coordenada de error
    char expresion[tamano],eror[3];
    //Transformado: contiene a la expresion transformada después de aplicar
    //el método Transforma Expresión
    //ExprNegativos: contiene los menos unarios (0-1)
    //Mensaje: contiene al mensaje adecuado segun el código de error
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    //Se crea el objeto evaluadorExpresiones de la clase Evaluar
    Evaluar evaluadorExpresiones;
    //ord1 almacena el numero de filas de la tabla
    //ord  almacena el numero de filas menos uno
    int i,j;
    int ord1,ord;
    ord1=TablaG->GetNumberRows();
    ord=ord1-1;
    double x[ord],y[ord];
    //Se obtienen los datos de los puntos
    //El for externo tiene el control de las 2 columnas
    for(int i=0;i<=1;i++)
    {//for1
        //El for interno recorre todas las filas de la columna indicada por i
        for(int j=0;j<=ord;j++)
        {//for2
            //Se limpian los arreglos para el evaluador de expresiones
            for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
            = '\0'; ExprNegativos[inicializa] = '\0'; }
            //Se copia la expresión de la celda j,i
            strcpy(expresion, TablaG->GetCellValue(j,i));
            //Se transforma la expresión
            evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
            //Se obtiene el código de error
            verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);
            //Se almacena cada código de error en el arreglo sintax
            sintax[indice]=verifsin;
            //Si ha ocurrido error se muestran los valores j,i
            if(verifsin)
```

```
        {   //Se habilita la caja de texto donde se muestran las posiciones
            ErrorCT->Enable(true);
               ErrorST->Enable(true);
            //Se escribe un paréntesis
            ErrorCT->WriteText("(");
            //Se convierte a cadena de carácteres el valor de j
            sprintf(eror,"%d",j+1);
            //Se escribe en la caja de texto
            ErrorCT->WriteText(eror);
            //Se escribe una coma
            ErrorCT->WriteText(",");
            //Se convierte a cadena de caracteres el valor de i
            sprintf(eror,"%d",i+1);
            //Se escribe en la caja de texto
            ErrorCT->WriteText(eror);
            //se cierra el paréntesis
            ErrorCT->WriteText(")");
            //se escribe une espacio
            ErrorCT->WriteText(" ");
        }
        //El indice del arreglo sintax se incrementa en uno
        indice=indice+1;
    }//for2
}//for1
//Se obtiene la sumatoria del arreglo sintax
for(int i=0; i<=21;i++)
{
    if(sintax[i]!=0) sumas=sumas+1;
}
//Si la suma es distinta de cero ha ocurrido error y se muestra el mensaje
if(sumas) SintaxisDM->ShowModal();
//Si no ha ocurrido ningún error de sintaxis comienza el método
if(AncladoRB->GetValue())
{

    if((Ypd0CT->IsEmpty())||(YpdnCT->IsEmpty()))
    {
        ypd=false;
        PerivaDM->ShowModal();
    }
    else
    {
    //Se limpian los arreglos para el evaluador de expresiones
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; }
    //Se copia la expresión de la celda j,i
    strcpy(expresion, Ypd0CT->GetLineText(0));
    //Se transforma la expresión
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    //Se obtiene el código de error
    verifp1 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
    //Se limpian los arreglos para el evaluador de expresiones
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; }
    //Se copia la expresión de la celda j,i
    strcpy(expresion, YpdnCT->GetLineText(0));
    //Se transforma la expresión
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    //Se obtiene el código de error
    verifp2 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
    if(verifp1!=0||verifp2!=0)
    {
        verifpar=false;
        ErrorparDM->ShowModal();
    }
    }//else

}
```

```
    if(CurvaturaRB->GetValue())
    {

        if((YsdnCT->IsEmpty())||(ysdnCT->IsEmpty()))
        {
            ysd=false;
            SerivaDMDM->ShowModal();
        }
        else
        {
        //Se limpian los arreglos para el evaluador de expresiones
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; }
        //Se copia la expresión de la celda j,i
        strcpy(expresion, YsdnCT->GetLineText(0));
        //Se transforma la expresión
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        //Se obtiene el código de error
        verifp1 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        //Se limpian los arreglos para el evaluador de expresiones
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; }
        //Se copia la expresión de la celda j,i
        strcpy(expresion, ysdnCT->GetLineText(0));
        //Se transforma la expresión
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        //Se obtiene el código de error
        verifp2 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        if(verifp1!=0||verifp2!=0)
        {
            verifpar1=false;
            ErrorparDM->ShowModal();
        }
        }//else
    }
if(!sumas&&ypd&&verifpar&&verifpar1&&ysd)
{//control de la ejecución
    DigitosLD->Enable(true);
    DesplazarSB->Enable(true);
    PolinomioCT->Enable(true);
    PolinomioST->Enable(true);
    X1CT->Enable(true);
    X2CT->Enable(true);
    X1ST->Enable(true);
    X2ST->Enable(true);
    //Se obtienen los valores de los puntos
    for(i=0;i<=ord;i++)
    {//for3
        //Se limpian los arreglos
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
        //Se copian las expresiones de la primera columna
        strcpy(expresion, TablaG->GetCellValue(i,0));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
       evaluadorExpresiones.Analizar(ExprNegativos);
       //Se obtiene el valor de las expresiones de la primera columna
       ouble valor = evaluadorExpresiones.Calcular();
       //Se almacenan en el arreglo x
        x[i]=valor;
        //Se obtienen los valores de ls segunda columna
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
        strcpy(expresion, TablaG->GetCellValue(i,1));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
         evaluadorExpresiones.Analizar(ExprNegativos);
```

```
     valor = evaluadorExpresiones.Calcular();
    //Los valores se almacenan en el arreglo y
     y[i]=valor;
}//for3
//SE OBTIENEN LOS POLINOMIOS
//variables para obtener los polinomios
n=ord;
char opt,s[5];
float h[20],A[20],B[20],C[20],D[20],a[20],b[20],c[20],d[20];
float xg,yd0,ydn,temp,yc;
for(i=0;i<=n;i++) h[i]=x[i]-x[i-1];
for(i=1;i<n;i++)
{
    A[i]=h[i];
    B[i]=2*(h[i]+h[i+1]);
    C[i]=h[i+1];
    D[i]=6*((y[i+1]-y[i])/h[i+1]-(y[i]-y[i-1])/h[i]);
}
if(NaturalRB->GetValue())
{
    temp=TriDiag(A,B,C,D,n-1);
    M[0]=0; M[n]=0;
}
if(AncladoRB->GetValue())
{
    //Se limpian los arreglos
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    //Se copian las expresiones de la primera columna
    strcpy(expresion, Ypd0CT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
     evaluadorExpresiones.Analizar(ExprNegativos);
    //Se obtiene el valor de yd0
    yd0 = evaluadorExpresiones.Calcular();
    //Se limpian los arreglos
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    //Se copian las expresiones de la primera columna
    strcpy(expresion, YpdnCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
     evaluadorExpresiones.Analizar(ExprNegativos);
    //Se obtiene el valor de ydn
    ydn = evaluadorExpresiones.Calcular();
    D[0]=6*((y[1]-y[0])/h[1]-yd0)/h[1];
    D[n]=6*(ydn-(y[n]-y[n-1])/h[n])/h[n];
    for(i=n+1;i>=1;i--) D[i]=D[i-1];
    A[n+1]=1; B[n+1]=2;
    for(i=n;i>=2;i--)
    {
        A[i]=A[i-1]; B[i]=B[i-1]; C[i]=C[i-1];
    }
    B[1]=2; C[1]=1;
    temp=TriDiag(A,B,C,D,n+1);
    for(i=0;i<=n;i++) M[i]=M[i+1];
}
if(ExtrapolacionRB->GetValue())
{
    B[1]=A[1]+B[1]+A[1]*h[1]/h[2];
    C[1]=C[1]-A[1]*h[1]/h[2];
    A[n-1]=A[n-1]-C[n-1]*h[n]/h[n-1];
    B[n-1]=B[n-1]+C[n-1]+C[n-1]*h[n]/h[n-1];
    temp=TriDiag(A,B,C,D,n-1);
    M[0]=M[1]-h[1]*(M[2]-M[1])/h[2];
    M[n]=M[n-1]+h[n]*(M[n-1]-M[n-2])/h[n-1];
}
if(CurvaturaRB->GetValue())
```

```
    {
        //Se limpian los arreglos
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
        //Se copian las expresiones de la primera columna
        strcpy(expresion, YsdnCT->GetLineText(0));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
         evaluadorExpresiones.Analizar(ExprNegativos);
       //Se obtiene el valor yd0
       yd0 = evaluadorExpresiones.Calcular();
       //Se limpian los arreglos
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
        //Se copian las expresiones de la primera columna
        strcpy(expresion, ysdnCT->GetLineText(0));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
       evaluadorExpresiones.Analizar(ExprNegativos);
       //Se obtiene el valor de ydn
        ydn = evaluadorExpresiones.Calcular();
        D[1]=D[1]-A[1]*yd0;
        D[n-1]=D[n-1]-C[n-1]*ydn;
        temp=TriDiag(A,B,C,D,n-1);
       M[0]=yd0; M[n]=ydn;
    }
    if(fail==true) SintaxisDM->ShowModal();
    else
    {//el método no ha fallado

        for(i=0;i<=n;i++)
        {
            a[i]=(M[i+1]-M[i])/(6*h[i+1]); b[i]=M[i]/2;
            c[i]=(y[i+1]-y[i])/h[i+1]-(2*h[i+1]*M[i]+h[i+1]*M[i+1])/6;
            d[i]=y[i];
        }
        s[1]=(x[0]>0) ? '-':'+';
        s[2]=(b[0]<0) ? '-':'+';
        s[3]=(c[0]<0) ? '-':'+';
        s[4]=(d[0]<0) ? '-':'+';
        temp=fabs(x[0]);
        dig=DigitosLD->GetSelection();
        if(dig==-1) dig=3;
        dig=dig+1;
       sprintf(w,"p0(x)=%7.*f*(x%c%7.*f)^3%c%7.*f*(x%c%7.*f)^2%c%7.*f*(x%c%7.*f)%c%7.*f\n",dig,
       a[0],s[1],dig,temp,s[2],dig,fabs(b[0]),s[1],dig,temp,s[3],dig,fabs(c[0]),s[1],dig,temp,
       s[4],dig,fabs(d[0]));
        PolinomioCT->WriteText(w);
        sprintf(w,"%7.4f",x[0]);
        X1CT->WriteText(w);
        sprintf(w,"%7.4f",x[1]);
        X2CT->WriteText(w);
    }
}//control de la ejecución
}
```

MAS PUNTOS LISTA

```
void SplineDlg::MasPuntosLDSelected(wxCommandEvent& event )
{
    // insert your code here
    //sel almacena la selección de la lista desplegable
    //ins almacena el numero de filas y columnas a insertar
    int sel,ins;
    //Se obtiene la selección de la lista
    sel=MasPuntosLD->GetSelection();
    //La selección de la lista comienza en cero por lo que hay que
    //sumar 1 para obtener las filas a insertar
    ins=sel+1;
```

```
    //Se insertan filas en la tabla 1 a partir de la fila 2.
    TablaG->InsertRows(2,ins);
    //Se limpia la tabla 1
    TablaG->ClearGrid();
    //Se habilita el botón que habilita a la lista desplegable
    PuntosB->Enable(true);
    //Se deshabilita la lista de selección
    MasPuntosLD->Enable(false);
}
```

MAS PUNTOS

```
void SplineDlg::PuntosBClick(wxCommandEvent& event)
{
    // insert your code here
    int lim,erase;
    lim=TablaG->GetNumberRows();
    erase=lim-2;
    if(lim>2)
    {
        TablaG->DeleteRows(2,erase);
    }
    TablaG->ClearGrid();     PuntosB->Enable(false);    MasPuntosLD->Enable(true);
    YsdnCT->Enable(false);   ysdnCT->Enable(false);     YsdnCT->Clear();
    ysdnCT->Clear();         Ysd0ST->Enable(false);     YsdnST->Enable(false);
    Ypd0CT->Enable(false);   YpdnCT->Enable(false);     Ypd0CT->Clear();
    YpdnCT->Clear();         Ypd0ST->Enable(false);     YpdnST->Enable(false);
    DesplazarSB->Enable(false);    PolinomioCT->Enable(false);    PolinomioST->Enable(false);
    X1CT->Enable(false);     X2CT->Enable(false);       X1ST->Enable(false);
    X2ST->Enable(false);     ErrorCT->Enable(false);    ErrorCT->Clear();
    ErrorST->Enable(false);  PolinomioCT->Clear();      X1CT->Clear();
    X2CT->Clear();           DigitosLD->Enable(false);  SpinST->Enable(false);
    DigitosST->Enable(false);
}
```

DESPLAZAR

```
void SplineDlg::DesplazarSBUpdated(wxSpinEvent& event)
{
         //Utiliza el mismo código hasta antes de mostrar el polinomio y luego se inserta lo
         //siguiente
        int spin=DesplazarSB->GetValue();
        if(spin<=0)
        {
            DesplazarSB->SetValue(0);
            PrimeroDM->ShowModal();
        }
        if(spin>=n)
        {
        DesplazarSB->SetValue(n-1);
        UltimoDM->ShowModal();
        spin=n-1;
        }
        s[1]=(x[spin]>0) ? '-':'+';
        s[2]=(b[spin]<0) ? '-':'+';
        s[3]=(c[spin]<0) ? '-':'+';
        s[4]=(d[spin]<0) ? '-':'+';
        temp=fabs(x[spin]);
        dig=DigitosLD->GetSelection();
        if(dig==-1) dig=3;
        dig=dig+1;
        sprintf(w,"p%d(x)=%7.*f*(x%c%7.*f)^3%c%7.*f*(x%c%7.*f)^2%c%7.*f*(x%c%7.*f)%c%7.*f\n",
        spin,dig,a[spin],s[1],dig,temp,s[2],dig,fabs(b[spin]),s[1],dig,temp,s[3],dig,
       fabs(c[spin]),s[1],dig,temp,s[4],dig,fabs(d[spin]));
        PolinomioCT->WriteText(w);
        sprintf(w,"%7.4f",x[spin]);
        X1CT->WriteText(w);
        sprintf(w,"%7.4f",x[spin+1]);
        X2CT->WriteText(w);

    }
}
```

INTERPOLAR

```
void SplineDlg::InterpolarBClick(wxCommandEvent& event)
{
    //Utiliza el mismo código hasta antes de mostrar el polinomio
    if(XvalCT->IsEmpty()) XvalDM->ShowModal();
    else
    {
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; }
        //Se copia la expresión de la celda j,i
        strcpy(expresion, XvalCT->GetLineText(0));
        //Se transforma la expresión
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        //Se obtiene el código de error
        verifp = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        if(verifp) ErrorparDM->ShowModal();
    }

if(!sumas&&!verifp)
{//control de la ejecución


    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; }
    strcpy(expresion, XvalCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
    evaluadorExpresiones.Analizar(ExprNegativos);
    //Se obtiene el valor de las expresiones de la primera columna
    xg=evaluadorExpresiones.Calcular();
       if((xg<x[0])||(xg>x[n]))
    {
        RangoDM->ShowModal();
    }
    else
    {
    for(int i=0;i<=n-1;i++)
    {
        if(xg<=x[i+1]&&xg>=x[i])
        {
            temp=xg-x[i];
            yc=a[i]*temp*temp*temp+b[i]*temp*temp+c[i]*temp+d[i];
            char w[3];
            sprintf(w,"  %7.10f",yc);
            ResultCT->WriteText(w);
        }
    }

    }
}
}
```

Otro valor

```
void SplineDlg::OtroValBClick(wxCommandEvent& event)
{
      // insert your code here
      OtroValB->Enable(false);
      InterpolarB->Enable(true);
      XvalCT->Clear();
      XvalCT->Enable(true);
      ResultCT->Enable(false);
      ResultCT->Clear();


}
```

Anclado Lista

```
void SplineDlg::AncladoRBClick(wxCommandEvent& event)
{
      // insert your code here
```

```
    Ypd0CT->Enable(true);    YpdnCT->Enable(true);    Ypd0ST->Enable(true);
    YpdnST->Enable(true);    YsdnCT->Enable(false);    ysdnCT->Enable(false);
    Ysd0ST->Enable(false);    YsdnST->Enable(false);
}
```

Curvatura Lista

```
void SplineDlg::CurvaturaRBClick(wxCommandEvent& event)
{
      // insert your code here
    YsdnCT->Enable(true);    YsdnCT->Enable(true);    Ysd0ST->Enable(true);
    YsdnST->Enable(true);    Ypd0CT->Enable(false);    YpdnCT->Enable(false);
    Ypd0ST->Enable(false);   YpdnST->Enable(false);
}
```

NATURAL LISTA

```
void SplineDlg::NaturalRBClick(wxCommandEvent& event)
{
      // insert your code here
    YsdnCT->Enable(false);    ysdnCT->Enable(false);    Ysd0ST->Enable(false);
    YsdnST->Enable(false);    Ypd0CT->Enable(false);    YpdnCT->Enable(false);
    Ypd0ST->Enable(false);    YpdnST->Enable(false);
}
```

EXTRAPOLACION LISTA

```
void SplineDlg::ExtrapolacionRBClick(wxCommandEvent& event)
{
      // insert your code here
    YsdnCT->Enable(false);    ysdnCT->Enable(false);    Ysd0ST->Enable(false);
    YsdnST->Enable(false);    Ypd0CT->Enable(false);    YpdnCT->Enable(false);
    Ypd0ST->Enable(false);    YpdnST->Enable(false);
}
```

LIMPIAR

```
void SplineDlg::LimpiarBClick(wxCommandEvent& event)
{
      // insert your code here
   YsdnCT->Enable(false);    YsdnCT->Enable(false);    YsdnCT->Clear();
    ysdnCT->Clear();         Ysd0ST->Enable(false);    YsdnST->Enable(false);
    Ypd0CT->Enable(false);   YpdnCT->Enable(false);    Ypd0CT->Clear();
    YpdnCT->Clear();         Ypd0ST->Enable(false);    YpdnST->Enable(false);
    DesplazarSB->Enable(false);    PolinomioCT->Enable(false);    PolinomioST->Enable(false);
    X1CT->Enable(false);     X2CT->Enable(false);      X1ST->Enable(false);
    X2ST->Enable(false);     ErrorCT->Enable(false);   ErrorCT->Clear();
      ErrorST->Enable(false);   PolinomioCT->Clear();     X1CT->Clear();
      X2CT->Clear();            TablaG->ClearGrid();      XvalCT->Clear();
      XvalCT->Enable(true);     OtroValB->Enable(false);  ResultCT->Clear();
      ResultCT->Enable(false);  DigitosLD->Enable(false); SpinST->Enable(false);

}
```

APÉNDICE C.8 CÓDIGO DE INTEGRACIÓN NUMÉRICA.

TABULAR

```
*/
void Integracion_NumericaDlg::TabularBClick(wxCommandEvent& event)
{
    // insert your code here
    //Se limpia la tabla
    TablaG->ClearGrid();
    //Se deshabilita el botón trapezoidal
    TrapezoidalB->Enable(false);
    //Se deshabilitan los componentes que muestran resultados de trapezoidal
    TansCT->Enable(false);    Ty0CT->Enable(false);    TynCT->Enable(false);
    TsumaCT->Enable(false);   Ty0ST->Enable(false);    TynST->Enable(false);
    TsumaST->Enable(false);   TansST->Enable(false);
    //Se deshabilitan los componentes de error de sintaxis
    SintaxisCT->Enable(false);     SintaxisCT->Clear();    SintaxisST->Enable(false);
    //Si se selecciona Ingresar incremento se limpia la caja de texto:
    if(IncrementoRB->GetValue()) NCT->Clear();
    //Si se selecciona Ingresar n se limpia la caja de texto:
    if(IngresarnRB->GetValue()) IncCT->Clear();
    //Variables de verificación de sintaxis
    int verifsin=0,verifsinp1=0,verifsinp2=0,verifsinp3=0;
    //tamano es el tamano de la expresión
    int tamano=200,tamano1=20,validator=0,question=0;
    //Almacena la expresión insertada
    char expresion[tamano];
    //Transformado: contiene a la expresion transformada después de aplicar
    //el método Transforma Expresión
    //ExprNegativos: contiene los menos unarios (0-1)
    //Mensaje: contiene al mensaje adecuado segun el código de error
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    //Se crea el objeto evaluadorExpresiones de la clase Evaluar
    Evaluar evaluadorExpresiones;
    //Se verifica si se ha insertado una función
    if (FuncionCT->IsEmpty()) FuncionDM->ShowModal();
    else
    {
        //Se obtiene el código de error
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
        strcpy(expresion, FuncionCT->GetLineText(0));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);

        if(verifsin)
        {
            //Si ha ocurrido error se muestra en la caja de texto 20
            SintaxisFDM->ShowModal();
            SintaxisST->Enable(true);
            SintaxisCT->Enable(true);
            evaluadorExpresiones.MensajeSintaxis(verifsin, Mensaje);
            SintaxisCT->WriteText(Mensaje);
        }
    }
    //Se verifica si se ha insertado el parámetro Desde
    if (DesdeCT->IsEmpty()) DesdeDM->ShowModal();
    else
    {
        for (int inicializa=0; inicializa<tamano1; inicializa++) { Transformado[inicializa]
         = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
        strcpy(expresion, DesdeCT->GetLineText(0));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        //Se obtiene el código de error
        verifsinp1 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
    }
    //Se verifica si se ha insertado el parámetro Hasta
    if (HastaCT->IsEmpty()) HastaDM->ShowModal();
    else
```

```
{
    for (int inicializa=0; inicializa<tamano1; inicializa++) { Transformado[inicializa]
     = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, HastaCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    //Se obtiene el código de error
    verifsinp2 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}
//Si la selección es insertar Incremento se verifica si se ha insertado
if (IncCT->IsEmpty()&&IncrementoRB->GetValue()) IncDM->ShowModal();
if (!IncCT->IsEmpty()&&IncrementoRB->GetValue())
{
    for (int inicializa=0; inicializa<tamano1; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, IncCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    //se obtiene el código de error
    verifsinp3 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}
//Si la selección es insertar el valor de n se verifica si se ha insertado
if((NCT->IsEmpty())&&IngresarnRB->GetValue())
//y se muestra el mensaje de error
InsNDM->ShowModal();
//Se suman los códigos de error de los parámetros
int verifpar=verifsinp1+verifsinp2+verifsinp3;
//Si verifpar es distinto de cero se muestra mensaje de error
if(verifpar) SintaxisPDM->ShowModal();
//Se calcula el valor de validator dependiendo del RadioButton seleccionado
if(IncrementoRB->GetValue()) validator=!FuncionCT->IsEmpty()*!DesdeCT->IsEmpty()*
!HastaCT->IsEmpty()*!IncCT->IsEmpty();
//Se calcula el valor de validator dependiendo del RadioButton seleccionado
if(IngresarnRB->GetValue()) validator=!FuncionCT->IsEmpty()*!DesdeCT->IsEmpty()*
!HastaCT->IsEmpty()*!NCT->IsEmpty();
//Si no hay errores de sintaxis y no se han omitido cajas de texto comienza
//la tabulación
if(validator&&verifsin==0&&verifpar==0)
{//control de la ejecución
//Se deshabilita el botón tabular
TabularB->Enable(false);
//Se habilita el botón Modificar Parámetros
ModParB->Enable(true);
//filas almacena el número de filas de la tabla
//j es un contador
//del es el número de filas a eliminar
int filas,j,del,nn;
double n,valor,dif;
char d[3];
//se obtiene el número de filas
filas=TablaG->GetNumberRows();
//Se borran las celdas excedentes del número predeterminado
if (filas>14)
{
    del=filas-14;
    TablaG->DeleteRows(11,del);
}
//Se reestable el color del texto predeterminado
for(j=0;j<filas;j++) TablaG->SetCellTextColour(j,1,"black");
//Etiquetas de las columnas
TablaG->SetColLabelValue(0,"x");
TablaG->SetColLabelValue(1,"f(x)");
int iter,resc,resc1;
double i,dsd,sta,inc,res,res1,veri1,verif,verif1,resc2;
char a[3],b[3],c[3];
//Se obtiene el valor de el parámetro desde
for (int inicializa=0; inicializa<tamano1; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, DesdeCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
```

```
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
dsd=evaluadorExpresiones.Calcular();
//Se obtiene el valor del parámetro hasta
for (int inicializa=0; inicializa<tamano1; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, HastaCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
sta=evaluadorExpresiones.Calcular();
//Si se ha seleccionado la opción de insertar incremento se obtiene el
//valor del incremento
   if(IncrementoRB->GetValue())
{
    for (int inicializa=0; inicializa<tamano1; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, IncCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
    evaluadorExpresiones.Analizar(ExprNegativos);
    inc=evaluadorExpresiones.Calcular();
    //Se calcula el valor de n
    n=(sta-dsd)/inc;
    sprintf(a,"%*.*f",3,0,n);
    //Se muestra el valor de n
    NCT->WriteText(a);
}
//Si se ha seleccionado Ingresar el valor de n se obtiene el valor
if(IngresarnRB->GetValue())
{
    n=atof(NCT->GetLineText(0));
    //Se calcula el incremento
    inc=(sta-dsd)/n;
    sprintf(a,"%*.*f",3,4,inc);
    //Se muestra el valor del incremento
    IncCT->WriteText(a);
}
//en este punto con cualquiera de las dos opciones ya se cuenta con el
//valor de n y se obtiene de la caja de texto
n=atoi(NCT->GetLineText(0));
//Variable de verificación para verificar el límite superior del intervalo
verif1=n*inc;
//Verif almacena el último valor de x
verif=dsd+(verif1);
//Se resta el parámetro hasta
dif=verif-sta;
//Se obtiene el valor absoluto
if(dif<0) dif=-1.0*dif;
//Si no se ha alcanzado el limite superior se muestra el mensaje
//if(dif!=0&&IncrementoRB->GetValue()) LimSupDM->ShowModal();
//Si se ha alcanzado el límite superior comienza la tabulación
if(dif==0);
{//ifdif
i=dsd;
iter=0;
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0';  }
strcpy(expresion, FuncionCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);

do
    {
        if(iter>13) TablaG->InsertRows(iter,1);
        evaluadorExpresiones.ValorVariable('x', i);
        valor = evaluadorExpresiones.Calcular();
```

```
        if (_isnan(valor) || !_finite(valor))
        {
            //Si han ocurrido un error matemático la variable cuestión
            //se incrementa en uno
            TablaG->SetCellValue(iter,1,"Error");
            question=question+1;
        }
        else
        {
        if (valor<0)
        {
            sprintf(a,"%*.*f",3,9,valor);
            TablaG->SetCellValue(iter,1,a);
            TablaG->SetCellTextColour(iter,1,"red");
        }
        if(valor>0)
        {
            sprintf(a,"%*.*f",3,9,valor);
            TablaG->SetCellValue(iter,1,a);
            TablaG->SetCellTextColour(iter,1,"black");
        }

        if(valor==0.0)
        {
            sprintf(a,"%*.*f",3,9,valor);
            TablaG->SetCellValue(iter,1,a);
            TablaG->SetCellTextColour(iter,1,"blue");
        }
        }
        sprintf(b,"%*.*f",3,5,i);
        TablaG->SetCellValue(iter,0,b);
        i=i+inc;
        iter=iter+1;

    }while (i<=sta);
}//ifdif
//Si han ocurrido errores matemáticos no se puede integrar y se muestra
//mensaje de error
if(question) ImposibleIntDM->ShowModal();
//Si no hay mensajes de error y se ha alcanzado el límite superior
//se verifica que método de integración se habilita
//if(!question&&dif==0)
{//mostrar o no los botones de integración
//Se habilitan los componentes para integración trapezoidal
TrapezoidalB->Enable(true);    TansCT->Enable(true);    Ty0CT->Enable(true);
TynCT->Enable(true);           TsumaCT->Enable(true);   Ty0ST->Enable(true);
TynST->Enable(true);           TsumaST->Enable(true);   TansST->Enable(true);
//res es una variable de tipo double
res=n/2.0;
//resc es una variable entera
resc=n/2;
//Si ambas son iguales estamos seguros de que el número de puntos es
//múltiplo de 2 y se habilitan los componentes para integración
//Simpson 1/3
if(res==resc)
{
    Simpson13B->Enable(true);        S13ansCT->Enable(true);        S13y0CT->Enable(true);
    S13ynCT->Enable(true);           S13siCT->Enable(true);         S13spCT->Enable(true);
    S13y0ST->Enable(true);           S13ynST->Enable(true);         S13spST->Enable(true);
    S13siST->Enable(true);           S13ansST->Enable(true);
}
else
//Si no es múltiplo de 2 se deshabilitan los componentes
{
    Simpson13B->Enable(false);     S13ansCT->Enable(false);        S13y0CT->Enable(false);
    S13ynCT->Enable(false);        S13siCT->Enable(false);         S13spCT->Enable(false);
    S13y0ST->Enable(false);        S13ynST->Enable(false);         S13spST->Enable(false);
    S13siST->Enable(false);        S13ansST->Enable(false);
```

```
    }

    //De la misma manera se verifica si el número n es multiplo de 3
    res1=n/3.0;
    resc1=n/3;
    //Si es mpultiplo de 3 se habilitan los componentes para integración
    //Simpson 3/8
    if(res1==resc1)
    {
        Simpson38B->Enable(true);        S38ansCT->Enable(true);        S38y0CT->Enable(true);
        S38ynCT->Enable(true);           S38s3CT->Enable(true);         S38resCT->Enable(true);
        S38y0ST->Enable(true);           S38ynST->Enable(true);         S38s3ST->Enable(true);
        S38resST->Enable(true);          S38ansST->Enable(true);
    }
    else
    //Si no es así se deshabilitan los componentes
    {
        Simpson38B->Enable(false);     S38ansCT->Enable(false);       S38y0CT->Enable(false);
        S38ynCT->Enable(false);        S38s3CT->Enable(false);        S38resCT->Enable(false);
        S38y0ST->Enable(false);        S38ynST->Enable(false);        S38s3ST->Enable(false);
        S38resST->Enable(false);        S38ansST->Enable(false);
    }
    }//mosttrar o no los botones de integración*/
    }//control de la ejecución
}
```

TRAPEZOIDAL

```
void Integracion_NumericaDlg::TrapezoidalBClick(wxCommandEvent& event)
{
    // insert your code here
    //Se limpian los componentes que muestran resultados de éste método
    TansCT->Clear();    Ty0CT->Clear();    TynCT->Clear();    TsumaCT->Clear();
    int n,n1,i;
    //Se obtiene el valor de n
    n=atoi(NCT->GetLineText(0));
    double dsd,dsd1,sta,sta1,inc,y[n],suma,inte;
    char b[3];
    int tamano=200;
    char expresion[tamano];
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    //Se crea el objeto evaluadorExpresiones de la clase Evaluar
    Evaluar evaluadorExpresiones;
    //Se obtiene el valor del parámetro desde
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, DesdeCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
    evaluadorExpresiones.Analizar(ExprNegativos);
    dsd=evaluadorExpresiones.Calcular();
     //Se obtiene el valor del parámetro Hasta
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, HastaCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
    evaluadorExpresiones.Analizar(ExprNegativos);
    sta=evaluadorExpresiones.Calcular();
    //Se calcula el incremento
    inc=(sta-dsd)/n;
    //Límite para no tomar el valor de y[n]
    n1=n-1;
    //Este código se ejecuta si se ha insertado una función
    if(InsFuncionRB->GetValue())
    {//Se prepara todo para comenzar a darle valores a la variable independiente
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, FuncionCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
```

```
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
double valor;

   for(i=0;i<=n;i=i++)
   {
    //Se almacenan todas las ordenadas en el arreglo y[n]
    evaluadorExpresiones.ValorVariable('x', dsd);
    valor = evaluadorExpresiones.Calcular();
   y[i]=valor;
    //Se incrementa dsd para el siguiente valor de x
    dsd=dsd+inc;

   }
//Se inicializa la variable que almacena la sumatoria
suma=0;
//Se obtiene la sumatoria de los elementos sin incluir y[0] ni y[n]
for(i=1;i<=n1;i++)
{
    suma=suma+y[i];
}
//Se obtiene el valor de la integral
inte=0.5*inc*(y[0]+y[n]+2*suma);
//Se muestra el resultado de la integral
sprintf(b,"%*.*f",3,9,inte);
TansCT->WriteText(b);
//Se muestra el valor de y[0]
sprintf(b,"%*.*f",3,9,y[0]);
Ty0CT->WriteText(b);
//Se muestra el valor de y[n]
sprintf(b,"%*.*f",3,9,y[n]);
TynCT->WriteText(b);
//Se muestra el valor de la sumatoria
sprintf(b,"%*.*f",3,9,suma);
TsumaCT->WriteText(b);
}
//Si se han insertado puntos en la tabla se ejecuta este código
if(InsPuntosRB->GetValue())
{
//arreglo para almacenar los códigos de error
int sintax[n];
//se inicializa el arreglo
for(i=0;i<=n;i++) sintax[i]=0;
//indice es el indice del arreglo sintax
//sumas almacena la sumatoria de los códigos de error
//verifsin almacena cada código de error
int indice=0,sumas=0,verifsin=0;
//este ciclo recorre todas las filas de la columana de f(x)
for(int j=0;j<=n;j++)
    {
        //se limpian los arreglos del evaluador
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; }
        //se copia cada expresion de la columna de f(x)
        strcpy(expresion, TablaG->GetCellValue(j,1));
        //se transforma la expresión
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        //se obtiene el código de error
        verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        //se almacena el código de error en el arreglo sintax
        sintax[indice]=verifsin;
        //se incrementa el indice del arreglo
        indice=indice+1;
    }
    //se obtiene la sumatoria del arreglo sintax
    for(int m=0; m<=n;m++)
    {
        if(sintax[m]!=0) sumas=sumas+1;
```

```
        }
        //si la sumatoria no es cero se muestra mensaje de error
        if(sumas) SintaxisPDM->ShowModal();

    //si no hay errores de sintaxis se ejecuta el método
    if(!sumas)
    {//control de la ejecución
    for(i=0;i<=n;i++)
        //se obtienen los valores de las ordenadas y se almcenana en el arreglo
        //y
        {
            for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
        strcpy(expresion, TablaG->GetCellValue(i,1));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
         evaluadorExpresiones.Analizar(ExprNegativos);
        double valor = evaluadorExpresiones.Calcular();
        y[i]=valor;

        }
    //Se inicializa la variable que almacena la sumatoria
    suma=0;
    //Se obtiene la sumatoria de los elementos sin incluir y[0] ni y[n]
    for(i=1;i<=n1;i++)
    {
        suma=suma+y[i];
    }

    //Se obtiene el valor de la integral
    inte=0.5*inc*(y[0]+y[n]+2*suma);
    //Se muestra el resultado de la integral
    sprintf(b,"%*.*f",3,9,inte);
    TansCT->WriteText(b);
    //Se muestra el valor de y[0]
    sprintf(b,"%*.*f",3,9,y[0]);
    Ty0CT->WriteText(b);
    //Se muestra el valor de y[n]
    sprintf(b,"%*.*f",3,9,y[n]);
    TynCT->WriteText(b);
    //Se muestra el valor de la sumatoria
    sprintf(b,"%*.*f",3,9,suma);
    TsumaCT->WriteText(b);
    }//control de la ejecucion
    }
}
Simpson 13
*/
void Integracion_NumericaDlg::Simpson13BClick(wxCommandEvent& event)
{
       // insert your code here
    //Se limpian los componentes que muestran reultados de Simpson 1/3
    S13ansCT->Clear();    S13y0CT->Clear();    S13ynCT->Clear();    S13siCT->Clear();
    S13spCT->Clear();
    //n el el número de divisiones del intervalo
    int n;
    int n1,i;
    //Se obtiene el valor de n
    n=atoi(NCT->GetLineText(0));
    double dsd,dsd1,sta,sta1,inc,y[n],suma,inte,suma1;
    char b[3];
    //tamaño de las expresiones
    int tamano=200;
    //cadenas usadas para el evaluador
    char expresion[tamano];
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    Evaluar evaluadorExpresiones;
    //Se obtiene el valor del parámetro desde
```

```
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, DesdeCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
dsd=evaluadorExpresiones.Calcular();
//Se obtiene el valor del parámetro Hasta
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, HastaCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
sta=evaluadorExpresiones.Calcular();
//Se calcula el valor del incremento
inc=(sta-dsd)/n;
//Límite para no tomar el valor de y[n]
n1=n-1;
//Si se ha seleccionado insertar una función se ejecuta este código
if(InsFuncionRB->GetValue())
{
//Se prepara todo para dar valores a 'x'
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, FuncionCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
double valor;
for(i=0;i<=n;i=i++)
{
    //Se almacenan todas las ordenadas en el arreglo y[n]
    evaluadorExpresiones.ValorVariable('x', dsd);
    valor = evaluadorExpresiones.Calcular();
   y[i]=valor;
   //Se incrementa dsd para el siguiente valor de x
   dsd=dsd+inc;
}
//Se inicializa la variable que almacena la sumatoria
suma=0;
//Se obtiene la sumatoria de las ordenadas con indice impar
for(i=1;i<=n1;i=i+2)
{
    suma=suma+y[i];
}
//Se obtiene la sumatoria de las ordenadas con indice par
suma1=0;
for(i=2;i<=n1;i=i+2)
{
    suma1=suma1+y[i];
}
//Se obtiene el valor de la integral
inte=(inc/3)*(y[0]+y[n]+4*suma+2*suma1);
//Se muestra el resultado de la integral
sprintf(b,"%*.*f",3,9,inte);
S13ansCT->WriteText(b);
//Se muestra el valor de y[0]
sprintf(b,"%*.*f",3,9,y[0]);
S13y0CT->WriteText(b);
//Se muestra el valor de y[n]
sprintf(b,"%*.*f",3,9,y[n]);
S13ynCT->WriteText(b);
//Se muestra la sumatoria de las ordenadas con indice impar
sprintf(b,"%*.*f",3,9,suma);
S13siCT->WriteText(b);
//Se muestra la sumatoria de las ordenadas con índice par
sprintf(b,"%*.*f",3,9,suma1);
```

```
S13spCT->WriteText(b);
}
//Si se han insertado puntos en la tabla se ejecuta este código
if(InsPuntosRB->GetValue())
{
//arreglo para almacenar los códigos de error
int sintax[n];
//se inicializa el arreglo
   for(i=0;i<=n;i++) sintax[i]=0;
   //indice es el indice del arreglo sintax
   //sumas almacena la sumatoria de los códigos de error
   //verifsin almacena cada código de error
   int indice=0,sumas=0,verifsin=0;
    //este ciclo recorre todas las filas de la columana de f(x)
for(int j=0;j<=n;j++)
{
        //se limpian los arreglos del evaluador
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; }
        //se copia cada expresion de la columna de f(x)
        strcpy(expresion, TablaG->GetCellValue(j,1));
        //se transforma la expresión
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        //se obtiene el código de error
        verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        //se almacena el código de error en el arreglo sintax
        sintax[indice]=verifsin;
        //se incrementa el indice del arreglo
        indice=indice+1;
}
//se obtiene la sumatoria del arreglo sintax
for(int m=0; m<=n;m++)
{
        if(sintax[m]!=0) sumas=sumas+1;
}
//si la sumatoria no es cero se muestra mensaje de error
if(sumas) SintaxisPDM->ShowModal();
//si no hay errores de sintaxis se ejecuta el método
if(!sumas)
{//control de la ejecución
for(i=0;i<=n;i++)
{
//se obtienen los valores de las ordenadas y se almcenana en el arreglo y
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, TablaG->GetCellValue(i,1));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
double valor = evaluadorExpresiones.Calcular();
y[i]=valor;
}
//Se inicializa la variable que almacena la sumatoria
suma=0;
//Se obtiene la sumatoria de las ordenadas con indice impar
for(i=1;i<=n1;i=i+2)
{
    suma=suma+y[i];
}

suma1=0;
//Se obtiene la sumatoria de las ordenadas con indice par
for(i=2;i<=n1;i=i+2)
{
    suma1=suma1+y[i];
}
//Se obtiene el valor de la integral
inte=(inc/3)*(y[0]+y[n]+4*suma+2*suma1);
```

```
    //Se muestra el resultado de la integral
    sprintf(b,"%*.*f",3,9,inte);
    S13ansCT->WriteText(b);
    //Se muestra el valor de y[0]
    sprintf(b,"%*.*f",3,9,y[0]);
    S13y0CT->WriteText(b);
    //Se muestra el valor de y[n]
    sprintf(b,"%*.*f",3,9,y[n]);
    S13ynCT->WriteText(b);
    //Se muestra la sumatoria de las ordenadas con índice impar
    sprintf(b,"%*.*f",3,9,suma);
    S13siCT->WriteText(b);
    //Se muestra la sumatoria de las ordenadas con índice par
    sprintf(b,"%*.*f",3,9,suma1);
    S13spCT->WriteText(b);
    }//control de la ejecucion
    }//if InsPuntosRB
}
```

Simpson 3/8

```
void Integracion_NumericaDlg::Simpson38BClick(wxCommandEvent& event)
{
       // insert your code here
    //Se limpian los componentes que muestra información de Simpson 3/8
    S38ansCT->Clear();    S38y0CT->Clear();    S38ynCT->Clear();
    S38s3CT->Clear();    S38resCT->Clear();
    //n el el número de divisiones del intervalo
    //n1 es número de divisiones menos uno
    //i es un contador
    int n,n1,i,veri;
    //Se obtiene el valor de n
    n=atoi(NCT->GetLineText(0));
    double dsd,dsd1,sta,sta1,inc,y[n],suma,inte,suma1;
    char b[3];
    //tamaño de las expresiones
    int tamano=200;
    //cadenas usadas para el evaluador
    char expresion[tamano];
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    //Se crea el objeto evaluadorExpresiones de la clase Evaluar
    Evaluar evaluadorExpresiones;
    //Se obtiene el valor del parámetro desde
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, DesdeCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
    evaluadorExpresiones.Analizar(ExprNegativos);
    dsd=evaluadorExpresiones.Calcular();
    //Se obtiene el valor del parámetro Hasta
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, HastaCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
    evaluadorExpresiones.Analizar(ExprNegativos);
    sta=evaluadorExpresiones.Calcular();
    //Se calcula el valor del incremento
    inc=(sta-dsd)/n;
    //Límite para no tomar el valor de y[n]
    n1=n-1;
    //Si se ha seleccionado insertar una función se ejecuta este código
    if(InsFuncionRB->GetValue())
    {//if InsFuncionRB
    //Se prepara todo para dar valores a 'x'
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, FuncionCT->GetLineText(0));
```

```
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
double valor;

   for(i=0;i<=n;i=i++)
   {
    //Se almacenan todas las ordenadas en el arreglo y[n]
    evaluadorExpresiones.ValorVariable('x', dsd);
    valor = evaluadorExpresiones.Calcular();
   y[i]=valor;
    //Se incrementa dsd para el siguiente valor de x
    dsd=dsd+inc;
    }
    //Se inicializa la variable suma
    suma=0;
    //Se obtiene la sumatoria de las ordenadas con índice múltiplo de 3
for(i=3;i<=n1;i=i+3)
{
    suma=suma+y[i];
}

//Se obtiene la sumatoria del resto de las ordenadas sin incluir y[0],y[n]
suma1=0;
veri=0;
for(i=1;i<=n1;i++)
{
    veri=veri+1;
    if(veri!=3)
    {
        suma1=suma1+y[i];
    }
    if(veri==3) veri=0;
}
//Se obtiene el valor de la integral
inte=(3.0/8.0)*inc*(y[0]+y[n]+2*suma+3*suma1);
//Se muestra el resultado de la integral
sprintf(b,"%*.*f",3,9,inte);
S38ansCT->WriteText(b);
//Se muestra el valor de y[0]
sprintf(b,"%*.*f",3,9,y[0]);
S38y0CT->WriteText(b);
//Se muestra el valor de y[n]
sprintf(b,"%*.*f",3,9,y[n]);
S38ynCT->WriteText(b);
//Se muestra la sumatoria de las ordenadas con índice múltiplo de 3
sprintf(b,"%*.*f",3,9,suma);
S38s3CT->WriteText(b);
//Se muestra la sumatoria del resto de las ordenadas
sprintf(b,"%*.*f",3,9,suma1);
S38resCT->WriteText(b);
}//if InsFuncion
//Éste código se ejecuta si se han insertado puntos en la tabla
if(InsPuntosRB->GetValue())
{//if InsPuntos
//arreglo para almacenar los códigos de error
int sintax[n];
//se inicializa el arreglo
for(i=0;i<=n;i++) sintax[i]=0;
//indice es el indice del arreglo sintax
//sumas almacena la sumatoria de los códigos de error
//verifsin almacena cada código de error
int indice=0,sumas=0,verifsin=0;
//este ciclo recorre todas las filas de la columana de f(x)
for(int j=0;j<=n;j++)
    {
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; }
```

```
        strcpy(expresion, TablaG->GetCellValue(j,1));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        //Se obtiene el código de error
        verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        //Se almacena el código de error
        sintax[indice]=verifsin;
        //Se incrementa el índice
        indice=indice+1;
    }
    //se obtiene la sumatoria del arreglo sintax
    for(int m=0; m<=n;m++)
    {
        if(sintax[m]!=0) sumas=sumas+1;
    }
//si la sumatoria no es cero se muestra mensaje de error
if(sumas) SintaxisPDM->ShowModal();
//si no hay errores de sintaxis se ejecuta el método
if(!sumas)
{//control de la ejecución
for(i=0;i<=n;i++)
    {
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, TablaG->GetCellValue(i,1));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
       evaluadorExpresiones.Analizar(ExprNegativos);
       double valor = evaluadorExpresiones.Calcular();
       //Se obtienen los valores de los puntos
    y[i]=valor;

    }
//Se inicializa la variable suma
suma=0;
//Se obtiene la sumatoria de las ordenadas con índice múltiplo de 3
for(i=3;i<=n1;i=i+3)
{
    suma=suma+y[i];
}
//Se obtiene la sumatoria del resto de las ordenadas sin incluir y[0],y[n]
suma1=0;
veri=0;
for(i=1;i<=n1;i++)
{
    veri=veri+1;
    if(veri!=3)
    {
        suma1=suma1+y[i];
    }
    if(veri==3) veri=0;
}
//Se obtiene el valor de la integral
inte=(3.0/8.0)*inc*(y[0]+y[n]+2*suma+3*suma1);
//Se muestra el resultado de la integral
sprintf(b,"%*.*f",3,9,inte);
S38ansCT->WriteText(b);
//Se muestra el valor de y[0]
sprintf(b,"%*.*f",3,9,y[0]);
S38y0CT->WriteText(b);
//Se muestra el resultado de y[n]
sprintf(b,"%*.*f",3,9,y[n]);
S38ynCT->WriteText(b);
//Se muestra la sumatoria de las ordenadas con índice múltiplo de 3
sprintf(b,"%*.*f",3,9,suma);
S38s3CT->WriteText(b);
//Se muestra la sumatoria del resto de las ordenadas
sprintf(b,"%*.*f",3,9,suma1);
S38resCT->WriteText(b);
```

```
    }//control de la ejecucion
    }//If InsPuntosRB
}
```

Ingresar Incremento

```
void Integracion_NumericaDlg::IncrementoRBClick(wxCommandEvent& event)
{
       // insert your code here
       //Habilita la caja de texto donde se inserta el incremento
       IncCT->Enable(true);
       //Deshabilita la caja de texto donde se inserta el valor de n
       NCT->Enable(false);

}
```

Ingresar N

```
void Integracion_NumericaDlg::IngresarnRBClick(wxCommandEvent& event)
{
       // insert your code here
       //Deshabilita la caja de texto donde se inserta el incremento
       IncCT->Enable(false);
       //Habilita la caja de texto donde se inserta el valor de n
       NCT->Enable(true);
       //IngresarnRB->SetValue(false);
       //IncrementoRB->SetValue(true);
       IngresarnRB->SetValue(true);
       IncrementoRB->SetValue(false);
}
```

Insertar puntos en la tabla

```
void Integracion_NumericaDlg::InsPuntosRBClick(wxCommandEvent& event)
{
    // insert your code here
    //Se deshabilita RadioButton2
    IngresarnRB->Enable(false);
    InsPuntosRB->SetValue(true);
    InsFuncionRB->SetValue(false);
    //Se limpia la tabla
    TablaG->ClearGrid();
    //Se deshabilita la tabla
    TablaG->Enable(false);
    //Se deshabilita la caja de texto de la función
    FuncionCT->Enable(false);
    //Se deshabilita el botón tabular
    TabularB->Enable(false);
    //Se deshabilita el botón modificar parámetros
    ModParB->Enable(false);
    //Se habilita el botón Preparar tabla
    PrepararB->Enable(true);
    //Se fuerza a que se inserte el valor del incremento
    IncrementoRB->SetValue(true);
    IngresarnRB->SetValue(false);
    //Se habilita la caja de texto para insertar el incremento
    IncCT->Enable(true);
    //Se deshabilita la caja para insertar el valor de n
    NCT->Enable(false);

    //Se deshabilitan las cajas de texto
    TansCT->Clear();    S13ansCT->Clear();    S38ansCT->Clear();    NCT->Clear();
    Ty0CT->Clear();     TynCT->Clear();       TsumaCT->Clear();     S13y0CT->Clear();
    S13ynCT->Clear();   S13siCT->Clear();     S13spCT->Clear();     S38y0CT->Clear();
    S38ynCT->Clear();   S38s3CT->Clear();     S38resCT->Clear();    TansCT->Enable(false);
    S13ansCT->Enable(false);   S38ansCT->Enable(false);   Ty0CT->Enable(false);
    TynCT->Enable(false);      TsumaCT->Enable(false);    S13y0CT->Enable(false);
    S13ynCT->Enable(false);    S13siCT->Enable(false);    S13spCT->Enable(false);
    S38y0CT->Enable(false);    S38ynCT->Enable(false);    S38s3CT->Enable(false);
    S38resCT->Enable(false);
    //Se deshabilitan los textos estáticos:
    Ty0ST->Enable(false);    TynST->Enable(false);    TsumaST->Enable(false);
    TansST->Enable(false);   S13y0ST->Enable(false);  S13ynST->Enable(false);
```

```
    S13spST->Enable(false);  S13siST->Enable(false);  S13ansST->Enable(false);
    S38y0ST->Enable(false);  S38ynST->Enable(false);  S38s3ST->Enable(false);
    S38resST->Enable(false); S38ansST->Enable(false);
    //Se deshabilitan los botones de los 3 métodos de integración
    TrapezoidalB->Enable(false);    Simpson13B->Enable(false);    Simpson38B->Enable(false);
}
```

Insertar función

```
void Integracion_NumericaDlg::InsFuncionRBClick(wxCommandEvent& event)
{
    // insert your code here
    //Se habilita el RadioButton de insertar el valor de n
    IngresarnRB->Enable(true);
    InsFuncionRB->SetValue(true);
    InsPuntosRB->SetValue(false);
    //Se habilita la caja de texto de insertar función
    FuncionCT->Enable(true);
    //Se habilita el botón tabular
    TabularB->Enable(true);
    //Se habilita el botón modificar parámetros
    ModParB->Enable(true);
    //Se deshabilita el botón Modificar tabla
    ModTabB->Enable(false);
    //Se deshabilita el botón preparar tabla
    PrepararB->Enable(false);
    //Se establece insertar incremento
    IncrementoRB->SetValue(true);
    //Se habilita la tabla
    TablaG->Enable(true);
    //Se limpian las cajas de texto:
    TansCT->Clear();     S13ansCT->Clear();    S38ansCT->Clear();    NCT->Clear();
    Ty0CT->Clear();      TynCT->Clear();       TsumaCT->Clear();     S13y0CT->Clear();
    S13ynCT->Clear();    S13siCT->Clear();     S13spCT->Clear();     S38y0CT->Clear();
    S38ynCT->Clear();    S38s3CT->Clear();     S38resCT->Clear();
    //Se deshabilitan las cajas de texto:
    TansCT->Enable(false);    S13ansCT->Enable(false);     S38ansCT->Enable(false);
    Ty0CT->Enable(false);     TynCT->Enable(false);        TsumaCT->Enable(false);
    S13y0CT->Enable(false);   S13ynCT->Enable(false);      S13siCT->Enable(false);
    S13spCT->Enable(false);   S38y0CT->Enable(false);      S38ynCT->Enable(false);
    S38s3CT->Enable(false);   S38resCT->Enable(false);
    //Se deshabilitan los textos estáticos
    Ty0ST->Enable(false);    TynST->Enable(false);    TsumaST->Enable(false);
    TansST->Enable(false);   S13y0ST->Enable(false);  S13ynST->Enable(false);
    S13spST->Enable(false);  S13siST->Enable(false);  S13ansST->Enable(false);
    S38y0ST->Enable(false);  S38ynST->Enable(false);  S38s3ST->Enable(false);
    S38resST->Enable(false); S38ansST->Enable(false);
   //Se deshabilitan los botones:
    TrapezoidalB->Enable(false);    Simpson13B->Enable(false);    Simpson38B->Enable(false);
    ModParB->Enable(false);
}
```

Modificar parámetros

```
*/
void Integracion_NumericaDlg::ModParBClick(wxCommandEvent& event)
{
    // insert your code here
    InsFuncionRB->SetValue(true);    ModParB->Enable(false);    TabularB->Enable(true);
    TrapezoidalB->Enable(false);     Simpson13B->Enable(false); Simpson38B->Enable(false);
    TansCT->Enable(false);           S13ansCT->Enable(false);    S38ansCT->Enable(false);
    NCT->Clear();                    TansCT->Clear();            S13ansCT->Clear();
    S38ansCT->Clear();               TrapezoidalB->Enable(false);TansCT->Enable(false);
    Ty0CT->Enable(false);            TynCT->Enable(false);       TsumaCT->Enable(false);
    TansCT->Clear();                 Ty0CT->Clear();             TynCT->Clear();
    TsumaCT->Clear();                Ty0ST->Enable(false);       TynST->Enable(false);
    TsumaST->Enable(false);          TansST->Enable(false);      Simpson13B->Enable(false);
    S13ansCT->Enable(false);         S13y0CT->Enable(false);     S13ynCT->Enable(false);
    S13siCT->Enable(false);          S13spCT->Enable(false);     S13ansCT->Clear();
    S13y0CT->Clear();                S13ynCT->Clear();           S13siCT->Clear();
    S13spCT->Clear();                S13y0ST->Enable(false);     S13ynST->Enable(false);
```

```
    S13spST->Enable(false);          S13siST->Enable(false);     S13ansST->Enable(false);
    Simpson38B->Enable(false);       S38ansCT->Enable(false);    S38y0CT->Enable(false);
    S38ynCT->Enable(false);          S38s3CT->Enable(false);     S38resCT->Enable(false);
    S38ansCT->Enable(false);         S38y0CT->Clear();           S38ynCT->Clear();
    S38s3CT->Clear();                S38resCT->Clear();          S38y0ST->Enable(false);
    S38ynST->Enable(false);          S38s3ST->Enable(false);     S38resST->Enable(false);
    S38ansST->Enable(false);
}
```

Modificar Tabla

```
void Integracion_NumericaDlg::ModTabBClick(wxCommandEvent& event)
{
    // insert your code here
    NCT->Clear();    TansCT->Clear();    S13ansCT->Clear();    S38ansCT->Clear();
    TrapezoidalB->Enable(false);    Simpson13B->Enable(false);    Simpson38B->Enable(false);
    TansCT->Enable(false);    S13ansCT->Enable(false);    S38ansCT->Enable(false);
    IngresarnRB->Enable(false);    InsPuntosRB->SetValue(true);
    InsFuncionRB->SetValue(false);    ModTabB->Enable(false);    PrepararB->Enable(true);
    TablaG->Enable(false);    TablaG->ClearGrid();    FuncionCT->Enable(false);
    TabularB->Enable(false);    IncrementoRB->SetValue(true);    IngresarnRB->SetValue(false);
    IncCT->Enable(true);    NCT->Enable(false);    PrepararB->Enable(true);
    ModTabB->Enable(false);    InsFuncionRB->SetValue(false);    InsPuntosRB->SetValue(true);
    ModParB->Enable(false);    IngresarnRB->Enable(false);    TrapezoidalB->Enable(false);
    TansCT->Enable(false);    Ty0CT->Enable(false);    TynCT->Enable(false);
    TsumaCT->Enable(false);    TansCT->Clear();    Ty0CT->Clear();    TynCT->Clear();
    TsumaCT->Clear();    Ty0ST->Enable(false);    TynST->Enable(false);
    TsumaST->Enable(false);    TansST->Enable(false);    Simpson13B->Enable(false);
    S13ansCT->Enable(false);    S13y0CT->Enable(false);    S13ynCT->Enable(false);
    S13siCT->Enable(false);    S13spCT->Enable(false);    S13ansCT->Clear();
    S13y0CT->Clear();    S13ynCT->Clear();    S13siCT->Clear();    S13spCT->Clear();
    S13y0ST->Enable(false);    S13ynST->Enable(false);    S13spST->Enable(false);
    S13siST->Enable(false);    S13ansST->Enable(false);    Simpson38B->Enable(false);
    S38ansCT->Enable(false);    S38y0CT->Enable(false);    S38ynCT->Enable(false);
    S38s3CT->Enable(false);    S38resCT->Enable(false);    S38ansCT->Enable(false);
    S38y0CT->Clear();    S38ynCT->Clear();    S38s3CT->Clear();    S38resCT->Clear();
    S38y0ST->Enable(false);    S38ynST->Enable(false);    S38s3ST->Enable(false);
    S38resST->Enable(false);    S38ansST->Enable(false);
}
```

Prepara Tabla

```
void Integracion_NumericaDlg::ModTabBClick(wxCommandEvent& event)
{
    // insert your code here
    NCT->Clear();    TansCT->Clear();    S13ansCT->Clear();    S38ansCT->Clear();
    TrapezoidalB->Enable(false);    Simpson13B->Enable(false);    Simpson38B->Enable(false);
    TansCT->Enable(false);    S13ansCT->Enable(false);    S38ansCT->Enable(false);
    IngresarnRB->Enable(false);    InsPuntosRB->SetValue(true);
    InsFuncionRB->SetValue(false);    ModTabB->Enable(false);    PrepararB->Enable(true);
    TablaG->Enable(false);    TablaG->ClearGrid();    FuncionCT->Enable(false);
    TabularB->Enable(false);    IncrementoRB->SetValue(true);    IngresarnRB->SetValue(false);
    IncCT->Enable(true);    NCT->Enable(false);    PrepararB->Enable(true);
    ModTabB->Enable(false);    InsFuncionRB->SetValue(false);    InsPuntosRB->SetValue(true);
    ModParB->Enable(false);    IngresarnRB->Enable(false);    TrapezoidalB->Enable(false);
    TansCT->Enable(false);    Ty0CT->Enable(false);    TynCT->Enable(false);
    TsumaCT->Enable(false);    TansCT->Clear();    Ty0CT->Clear();    TynCT->Clear();
    TsumaCT->Clear();    Ty0ST->Enable(false);    TynST->Enable(false);
    TsumaST->Enable(false);    TansST->Enable(false);    Simpson13B->Enable(false);
    S13ansCT->Enable(false);    S13y0CT->Enable(false);    S13ynCT->Enable(false);
    S13siCT->Enable(false);    S13spCT->Enable(false);    S13ansCT->Clear();
    S13y0CT->Clear();    S13ynCT->Clear();    S13siCT->Clear();    S13spCT->Clear();
    S13y0ST->Enable(false);    S13ynST->Enable(false);    S13spST->Enable(false);
    S13siST->Enable(false);    S13ansST->Enable(false);    Simpson38B->Enable(false);
    S38ansCT->Enable(false);    S38y0CT->Enable(false);    S38ynCT->Enable(false);
    S38s3CT->Enable(false);    S38resCT->Enable(false);    S38ansCT->Enable(false);
    S38y0CT->Clear();    S38ynCT->Clear();    S38s3CT->Clear();    S38resCT->Clear();
    S38y0ST->Enable(false);    S38ynST->Enable(false);    S38s3ST->Enable(false);
    S38resST->Enable(false);    S38ansST->Enable(false);
}
```

APÉNDICE C.9 CÓDIGO DE RUNGE KUTTA

CALCULAR

```
*/
void Runge_KuttaDlg::CalcularBClick(wxCommandEvent& event)
{
    // insert your code here
    //Se limpia la tabla
    TablaG->ClearGrid();
    //Se deshabilitan los componentes que muestran error de sintaxis
    SintaxisST->Enable(false);
    SintaxisCT->Enable(false);
    SintaxisCT->Clear();
    //Se limpia y deshabilita la caja de texto que muestra el resultado
    ResultCT->Clear();
    ResultCT->Enable(false);
    //Variables de verificación de sintaxis
    int verifsin=0,verifsinp1=0,verifsinp2=0,verifsinp3=0,verifsinp4=0;
    //Tamaño de las expresiones
    int tamano=200;
    //cadena que almacena la expresión
    char expresion[tamano];
    //Transformado: contiene a la expresion transformada después de aplicar
    //el método Transforma Expresión
    //ExprNegativos: contiene los menos unarios (0-1)
    //Mensaje: contiene al mensaje adecuado segun el código de error
    char Transformado[tamano], ExprNegativos[tamano], Mensaje[tamano];
    //Se crea el objeto evaluadorExpresiones de la clase Evaluar
    Evaluar evaluadorExpresiones;
    //Si no se ha insertado la función se muestra el mensaje de error
    if (EcuacionCT->IsEmpty()) EcuacionDM->ShowModal();
    else
    {
        //Si se ha insertado se obtiene el código de error
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
        = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
        strcpy(expresion, EcuacionCT->GetLineText(0));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        verifsin = evaluadorExpresiones.EvaluaSintaxis(Transformado);
        //Si el código es distinto de cero se muestra el error de sintaxis
        if(verifsin)
        {
            SintaxisEDM->ShowModal();
            SintaxisST->Enable(true);
            SintaxisCT->Enable(true);
            evaluadorExpresiones.MensajeSintaxis(verifsin, Mensaje);
            SintaxisCT->WriteText(Mensaje);
        }
    }

    //Se verifica si se ha insertado el valor de a
    if (AvalorCT->IsEmpty()) AvalorDM->ShowModal();
    else
    {

        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
         = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
        strcpy(expresion, AvalorCT->GetLineText(0));
        evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
        //Si se ha insertado se obtiene el código de error
        verifsinp1 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
    }
    //Se  verifica si se ha insertado el valor de b
    if (BvalorCT->IsEmpty()) BvalorDM->ShowModal();
    else
    {
        for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
         = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
        strcpy(expresion, BvalorCT->GetLineText(0));
```

```
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    //Si se ha insertado se obtiene el código de error
    verifsinp2 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}
//Se  verifica si se ha insertado el valor de N
if (NvalorCT->IsEmpty()) NvalorDM->ShowModal();
else
{
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, NvalorCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    //Si se ha insertado se obtiene el código de error
    verifsinp3 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}
//Se  verifica si se ha insertado el valor de la condición inicial
if (IcsvalorCT->IsEmpty()) IcsvalorDM->ShowModal();
else
{
    for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
    = '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
    strcpy(expresion, IcsvalorCT->GetLineText(0));
    evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
    //Si se ha insertado se obtiene el código de error
    verifsinp4 = evaluadorExpresiones.EvaluaSintaxis(Transformado);
}
//Se suman los códigos de error
int verifpar=verifsinp1+verifsinp2+verifsinp3+verifsinp4;
//Si la suma no es cero se muestra el mensaje de error
if(verifpar) SintaxisPDM->ShowModal();
//Verifica si no se ha omitido ninguna caja de texto
int validator=!EcuacionCT->IsEmpty()*!AvalorCT->IsEmpty()*!BvalorCT->IsEmpty()
*!NvalorCT->IsEmpty()*!IcsvalorCT->IsEmpty();
//Si no hay errores de sintaxis ni cajas de texto vacías se ejecuta el
//código del método
if(validator&&verifsin==0&&verifpar==0)
{//control de la ejecucion
//Se habilita la caja de texto que muestra el resultado
ResultCT->Enable(true);
//Cadenas para mostrar resultados
char b[3];
char a[3];
//xa es el límite inferior del intervalo
//xb es el límite superior del intervalo
//h es el valor de(xa-xb)/N
//m es el valor que se utiliza para sustituir en x
//n es el valor que se utiliza para sustituir en y
//los valores de k son los mostrados en la expresión
double xa,xb,h,m,n,k1,k2,k3,k4;
//i es un contador
//N es el valor de iteraciones
//filas almacena el número de filas de la tabla
//del es el número de filas a eliminar
int i,N,filas,del;
//Se obtiene el numero de filas
filas=TablaG->GetNumberRows();
//Si hay más de 10 filas se eliminan las excedentes
if (filas>10)
{
    del=filas-10;
    TablaG->DeleteRows(10,del);
}
//Se obtiene el valor del límite inferior del intervalo
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, AvalorCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
```

```
evaluadorExpresiones.Analizar(ExprNegativos);
xa=evaluadorExpresiones.Calcular();
//Se obtiene el valor del límite superior del intervalo
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, BvalorCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
xb=evaluadorExpresiones.Calcular();
//Se obtiene el valor de N
N=atoi(NvalorCT->GetLineText(0));
//arreglos que almacenana los valores de x y y en cada iteración
double y[N],x[N];
//Se calcula el valor de h
h=(xb-xa)/N;
//Se almacenan en el arreglo todos los valores que tomará x
for(i=0;i<=N;i++)
{
    x[i]=xa;
    xa=x[i]+h;
}
//Se obtiene el valor de y[0]
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, IcsvalorCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
y[0]=evaluadorExpresiones.Calcular();

//Se deja todo listo para dat valores a la ecuación
for (int inicializa=0; inicializa<tamano; inicializa++) { Transformado[inicializa]
= '\0'; ExprNegativos[inicializa] = '\0'; Mensaje[inicializa] = '\0'; }
strcpy(expresion, EcuacionCT->GetLineText(0));
evaluadorExpresiones.TransformaExpresion(Transformado, expresion);
if (evaluadorExpresiones.ArreglaNegativos(ExprNegativos, Transformado) != -1)
evaluadorExpresiones.Analizar(ExprNegativos);
//Este ciclo realiza todas las iteraciones del método
for(i=0;i<=N-1;i++)
{
    //Si i es mayor que 9 se inserta una fila en cada iteración
    if(i>9) TablaG->InsertRows(i,1);
    //Se obtienen los valors a sustituir para obtener k1 y se muestran en la
    //tabla
    m=x[i]; n=y[i];
    sprintf(a,"%*.*f",3,5,x[i]);
    TablaG->SetCellValue(i,0,a);
    sprintf(a,"%*.*f",3,5,y[i]);
    TablaG->SetCellValue(i,1,a);
    //Se sutituyen los valores en la ecuación
    evaluadorExpresiones.ValorVariable('x', m);
    evaluadorExpresiones.ValorVariable('y', n);
    //Con esos valores se calcula k1
    k1 = evaluadorExpresiones.Calcular();
    //Se muestra el valor de k1 en la tabla
    sprintf(a,"%*.*f",3,5,k1);
    TablaG->SetCellValue(i,2,a);

    //Se obtienen los valore a sustituir para obtener k2
    m=x[i]+h/2.0; n=y[i]+(h*k1)/2.0;
    evaluadorExpresiones.ValorVariable('x', m);
    evaluadorExpresiones.ValorVariable('y', n);
    //Se sustituyen los valores para calcular k2
    k2 = evaluadorExpresiones.Calcular();
    //Se muestra el valor de k2
    sprintf(a,"%*.*f",3,5,k2);
    TablaG->SetCellValue(i,3,a);
```

```
        //Se obtienen los valore a sustituir para obtener k3
        m=x[i]+h/2.0; n=y[i]+(h*k2)/2.0;
        evaluadorExpresiones.ValorVariable('x', m);
        evaluadorExpresiones.ValorVariable('y', n);
        //Se sustituyen los valores para calcular k3
        k3 = evaluadorExpresiones.Calcular();
        //Se muestra el valor de k3
        sprintf(a,"%*.*f",3,5,k3);
        TablaG->SetCellValue(i,4,a);
        //Se obtienen los valores a sustituir para obtener k4
       m=x[i]+h; n=y[i]+(h*k3);
        evaluadorExpresiones.ValorVariable('x', m);
        evaluadorExpresiones.ValorVariable('y', n);
       //Se sustituyen los valores para calcular k4
       k4 = evaluadorExpresiones.Calcular();
       //Se muestra el valor de k4
      sprintf(a,"%*.*f",3,5,k4);
       TablaG->SetCellValue(i,5,a);
       //Se calcula el valor siguiente de y
      y[i+1]=y[i]+(h/6.0*(k1+2.0*k2+2.0*k3+k4));
      sprintf(a,"%*.*f",3,8,y[i+1]);
       TablaG->SetCellValue(i,6,a);

    }
    //Se escribe en la caja de texto el último valor de y calculado
    sprintf(a,"%*.*f",3,10,y[N]);
    ResultCT->WriteText(a);
    }//control de la ejecución
}


NUEVOS DATOS
*/
void Runge_KuttaDlg::NewDataBClick(wxCommandEvent& event)
{
       // insert your code here
       TablaG->ClearGrid();
       EcuacionCT->Clear();
       AvalorCT->Clear();
       BvalorCT->Clear();
       NvalorCT->Clear();
       IcsvalorCT->Clear();
       SintaxisCT->Clear();
       SintaxisCT->Enable(false);
       ResultCT->Clear();
       ResultCT->Enable(false);
       SintaxisST->Enable(false);
```

BIBLIOGRAFÍA

1. Chapra Steven C.,P. Canale Raymond. ***Métodos Numéricos para ingenieros***.Ed Mc Graw Hill 2007.
2. Ramos Carranza Rogelio, Aguilar Márquez Armando. ***Métodos Numéricos***. Edición 2009.
3. Herbert Schildt. ***C++ a begginer´s guide***. 2nd Edition. Ed. McGraw Hill. 2004.
4. Moreno Parra Rafael Alberto. ***Diseño de un evaluador de expresiones algebraicas***. Versión 2.0. 2013.
5. Pal Madhumangal. ***Numerical Analysis for scientists and engineers with programs in "C".*** Ed. Alpha Science. 2007.
6. Schoichiro Nakamura. ***Métodos numéricos Aplicados con Software.*** Ed. Prentice-Hall. 1997.
7. Burden L. Richard, Faires J. Douglas. ***Análisis Numérico .***7a Edición. Ed. Thomson-Learning. 2003.
8. Nieves Antonio, Dominguez Federico C. ***Métodos numéricos aplicados a la ingeniería.*** Ed. CECSA. 1997
9. Iriarte Rafael, Valderrama V. ***Métodos Numéricos.*** Ed. Trillas 1990.
10. D. Kincaid, W. Cheney. ***Análisis numérico, las matemáticas del cálculo científico.*** Ed. Addison-Wesley Iberoamericana 1994.