50521

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE ESTUDIOS SUPERIORES
ZARAGOZA

DESARROLLO DE PROGRAMAS DE APLICACION DE METODOS NUMERICOS EMPLEANDO VISUAL BASIC.

T E S I S

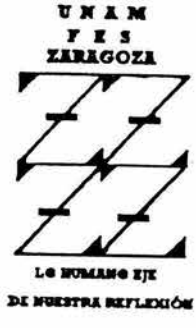
QUE PARA OBTENER EL TITULO DE:

INGENIERO QUÍMICO

P R E S E N T A :

RICARDO VAZQUEZ FRAGA

ASESOR: I.Q. JOSE ANTONIO ZAMORA PLATA



MEXICO, D. F.

MAYO 2004

FACULTAD DE ESTUDIOS
SUPERIORES ZARAGOZA

JEFATURA DE LA CARRERA
DE INGENIERIA QUIMICA

OFICIO: FESZ/JCIQ/083/03

ASUNTO: Asignación de Jurado

ALUMNO: VÁZQUEZ FRAGA RICARDO
P r e s e n t e.

En respuesta a su solicitud de asignación de jurado, la jefatura a mi cargo, ha propuesto a los siguientes sinodales:

Presidente:	I.Q. Eduardo Vázquez Zamora
Vocal:	I.Q. José Antonio Zamora Plata
Secretario:	Fis. Carlos Javier Martínez Gómez
Suplente:	M. en C. Genaro Altamirano García
Suplente:	I.I.Q. Alejandro Rubio Martínez

Sin más por el momento, reciba un cordial saludo.

A T E N T A M E N T E
"POR MI RAZA HABLARA EL ESPIRITU"
México, D. F., 3 de Octubre de 2003

EL JEFE DE LA CARRERA

FACULTAD DE ESTUDIOS SUPERIORES ZARAGOZA INGENIERIA QUIMICA SECRETARIA TECNICA

M. en C. ANDRÉS AQUINO CANCHOLA

A mi madre por el impulso y estimulo para
lograr lo que me proponga..

AGRADECIMIENTOS

Agradezco a **Dios** y a mi Madre por que gracias a ellos existo y soy gente de bien.

A los profesores que me transmitieron su conocimiento.

Gracias a los sinodales que dedicaron su tiempo a corregir mis frecuentes errores.

A Martín y Lucía por sus desveladas cuando comence a adentrarme en el mundo de las computadoras.

A todas aquellas personas que invirtieron su tiempo y esfuerzos en mi persona, no mencionando a ninguna pero teniendo la mayor gratitud a cada una.

A la familia que me ha soportado y que me estima.

Al apoyo especial de Angélica.

RESUMEN

El trabajo trata de llevar a cabo una relación entre los métodos numéricos, la programación y algunos ejemplos de manera que se integre el uso de la programación en Visual Basic, como una herramienta para facilitar y acelerar los cálculos que se llevan a cabo en las diversas actividades de los estudiantes y egresados de la carrera de ingeniería química.

Se seleccionaron algunos métodos numéricos con base en la cantidad de información disponible sobre ellos, de manera que se pueda centrar la atención sobre el proceso de programación, evitando que la falta de dominio sobre el algoritmo cause una incomprensión del proceso de desarrollo del código.

Los ejemplos tomados para resolverse mediante métodos numéricos, son representativos de los diferentes temas tratados por la ingeniería química, para dar un panorama general sin limitarse a un tema específico.

Se ha optado por analizar primero los diferentes lenguajes de programación, buscando que el lector conozca las características y ventajas que le proporcionará cada lenguaje, lo que le permitirá obtener el mayor provecho de esta habilidad.

Después se procede a detallar los métodos numéricos seleccionados y a describir su algoritmo, al tiempo que se codifican. Por último, se resuelven algunos ejemplos primero paso a paso y después con ayuda del programa desarrollado.

INDICE

Desarrollo de programas de aplicación de Métodos Numéricos empleando Visual Basic.

Introducción

Decir que la ingeniería química se basa en las matemáticas es tan claro como decir que la química es otro de sus soportes; los métodos numéricos son un componente que nos permite obtener una solución aproximada (en ocasiones exacta) a problemas en que su solución por métodos analíticos no es accesible, o con soluciones de tal forma, que no son convenientes para la interpretación numérica directa como en el caso de las funciones implícitas.

Por ejemplo, aunque se dispone de métodos rigurosos de cálculo para la resolución de problemas de separación de sistemas multicomponentes, los métodos aproximados siguen utilizándose en la práctica con fines tales como diseños preliminares, estudios paramétricos para establecer las condiciones óptimas de diseño, así como el estudio de síntesis del proceso para determinar las secuencias óptimas de separación.

Algunos métodos aproximados que se utilizan son:

- Fenske-Underwood-Gilliland.- Para la determinación del reflujo y las etapas necesarias en la destilación de sistemas multicomponentes.
- Método de Kremser.- Para separaciones en las que intervienen varias cascadas simples en contracorriente, tales como absorción, agotamiento y extracción líquido-líquido.
- Método de Edmister.- Para separaciones en las que intervienen cascadas en contracorriente con alimentaciones intermedias, tales como destilación.

Otro caso es que, para analizar el movimiento de los fluidos, se emplea una técnica que ha resultado exitosa denominada análisis diferencial. En la cuál se aplican las leyes básicas de conservación (masa, momentum y energía) a un volumen de control infinitesimal o, alternativamente, a un sistema fluido infinitesimal. La dificultad general

de las ecuaciones diferenciales esta llevando a la utilización de técnicas aproximadas, que forman parte y que han revitalizado al análisis numérico, por medio de las cuales las derivadas se sustituyen por relaciones algebraicas en un número finito de puntos del campo fluido y se podrían resolver empleando una computadora.

Así mismo los avances en materia de informática y cómputo han sido de una magnitud tal que han revolucionado una gran variedad de áreas. Los ingenieros han percibido desde un inicio el potencial que esta herramienta ha tenido y han hecho uso de ella. Sin embargo, con la evolución tan rápida en ocasiones han quedado algunas áreas un poco rezagadas.

Frecuentemente los ingenieros químicos por su proceder profesional necesitan resolver problemas que involucran a los métodos numéricos, para ello se recurre a emplear la programación como una herramienta de la que se dispone con el fin de simplificar los cálculos, pero se encuentran con la carencia de no saber como emplear las instrucciones de que se dispone; De ahí surge la idea de realizar un documento que si bien, no va a desarrollar completamente todos los métodos conocidos, ni empleará todos los lenguajes de programación, ni tampoco usará todas las instrucciones disponibles, si será una referencia para desarrollar la "intuición" que se requiere para plantear la secuencia de programación requerida. Se tratará de crear una forma de acercamiento al uso más cotidiano de los métodos numéricos, a la necesidad de la creación de programas que sean de uso sencillo en su uso, los cuales a su vez cuenten con una descripción de porque y para que se usaron las instrucciones o bloques de instrucciones y que sirvan de base para el desarrollo de nuevos y más poderosos programas.

El uso de computadoras y programas no es la panacea como se plantea en algunos círculos, los resultados a obtener dependen de cómo se implemente su uso: capacitación, software, soporte técnico, motivación y conocimiento del área o tema a manipular mediante medios electrónicos entre otros factores. Sin embargo, su utilización también ofrece muchas ventajas cuando su utilización se efectúa cuidando su correcta estructuración; rapidez, confiabilidad, precisión, repetitibilidad son algunas de ellas.

VisualBasic no fue el primer lenguaje de programación, con anterioridad se empleo el lenguaje de programación Fortran para DOS, el cuál permitía la obtención de resultados con una precisión aceptable y el cuál era "sencillo de usar", pero lamentablemente ha quedado en desventaja con los lenguajes visuales de programación. Estos últimos son cada vez más potentes y permiten una forma mas amigable de interactuar, tanto con el programador como con el usuario. VisualBasic mismo tiene sus orígenes en el lenguaje Basic que también ha quedado rezagado. En la sección de antecedentes se hará un breve resumen de algunos otros lenguajes de programación, los cuales pueden ser empleados (con ventajas o desventajas) para lograr los fines planteados.

Antecedentes

El desarrollo de un programa de cómputo de métodos numéricos tiene la finalidad de poner al alcance de el alumno de la carrera de Ingeniería Química un programa informático capaz de resolver problemas matemáticos de ingeniería química de solución compleja en los que la mejor vía es utilizar una aproximación haciendo uso de las métodos numéricos aprendidos en el cuarto semestre de la carrera. La meta de estos, es proporcionar técnicas convenientes para obtener información útil a partir de formulaciones matemáticas de problemas reales.

La solución de problemas de régimen de estado estacionario como son:

1) La distribución de temperatura en la conductividad térmica.
2) El equilibrio de las reacciones químicas
3) Los problemas de difusión constante.
4) Problemas de valores propios (eigenvalores).
5) problemas en los que se trata de hallar valores críticos de parámetros termodinámicos, además de
6) las configuraciones correspondientes de estado estacionario.

Y la solución de problemas de régimen de estado no estacionario como son:

1) Problemas de propagación
2) Problemas en estado transitorio o no estacionario en donde las variables son función del tiempo
3) Problemas que se presentan por lo regular en transmisión de calor en reacciones químicas y en la conducta transitoria de una columna de adsorción y de destilación.

Son algunos de los posibles campos de aplicación de lo que aquí se desarrollará.

En el prólogo de su libro, Ernest J. Henley y J. D. Seader señalan: "...Hace tan solo diez años, el diseño de fraccionadores, absorbedores, desorbedores y extractores se realizaba habitualmente por procedimientos aproximados de cálculo; por otra parte, los absorbedores con ebullición y las columnas de destilación extractiva eran con frecuencia estimadas a partir de la experiencia y de los datos obtenidos en planta

piloto. Actualmente, la utilización conjunta de desarrollos termodinámicos exactos y algoritmos de cálculo suficientemente rigurosos permiten al ingeniero resolver rápidamente a través de la terminal de un ordenador, sin abandonar su mesa de trabajo, problemas que antes eran considerados de resolución extremadamente complicada"[1]

Y añaden: "La disponibilidad de sistemas de cálculo comerciales para la simulación de procesos tales como CONCEPT, DESIGN/2000, FLOWTRAN, GPS-II y PROCESS ha reducido al ingeniero, en muchos casos, a jugar el papel de un soldado raso. Lo más frecuente es que su currículo no aborde los modernos algoritmos utilizados en estos sistemas, que el manual del usuario contenga solamente unas referencias vagas de las técnicas de cálculo utilizadas y que el manual de sistemas no este a su alcance de forma tal que el ejercicio de diseño degenere en la operación de una 'caja negra' quedando el usuario en la obscuridad"[2]

Estos párrafos engloban la idea principal que sirve de base para desarrollar el presente trabajo: Desarrollar una herramienta más, que los ingenieros químicos tenemos a nuestro alcance y que poco a poco irá ganando terreno en nuestras actividades hasta hacerse más necesaria... Tal vez indispensable.

Partiremos de la situación de analizar el lenguaje de programación y al sujeto de programación (métodos numéricos), como si fuesen dos áreas separadas con el fin de conocer un poco de su desarrollo, pero teniendo en cuenta que el fin último de este trabajo es el hecho de combinarlos y usarlos como elementos de nuestra formación.

Dependiendo del tipo de ordenador (computadora) hay una vasta variedad de sistemas operativos que permiten la utilización de estos equipos en su nivel más básico. Limitaremos el alcance de este trabajo al sistema operativo DOS –en sus diferentes versiones (MS-DOS, DR-DOS, OS/2)- y a los ambientes relacionados (Windows). Los cuales son los sistemas operativos más comunes a la gran mayoría de los usuarios de Computadoras Personales (PC´s) y compatibles.

---

[1] E. J. Henley; J. D. Seader. Operaciones de Separación por Etapas de Equilibrio en Ingeniería Química. Ediciones REPLA, México, 1990, p. VII.

[2] Idem.

Lenguajes de programación.

Hablar de programa es hablar de la necesidad de disponer de herramientas apropiadas para especificar los algoritmos o, lo que es lo mismo, para construir los programas informáticos que van a ser procesados en el ordenador.

El lenguaje de programación es un medio de comunicación entre el hombre y el ordenador. Para describir al ordenador un método para resolver un problema, es necesario formalizar el método adecuadamente para que lo pueda procesar. Esta noción de puente entre el hombre y el ordenador es la verdadera expresión de un lenguaje de programación.

Todo lenguaje esta definido por dos elementos fundamentales: un vocabulario y una gramática.

- El vocabulario se construye a partir de las palabras, según ciertas reglas morfológicas. Y las palabras se forman por medio de caracteres (símbolos formales que componen el alfabeto propio de un lenguaje). Se dice que un lenguaje no tiene formato si el número de caracteres que lo definen es variable. Este no es el caso de los lenguajes de programación.
- La gramática permite construir frases de acuerdo con determinadas reglas sintácticas. Y la formación de estas frases deben respetar ciertas reglas semánticas.

En todo lenguaje es necesario definir:

- Un alfabeto: Ej. (A,B,C,... Y,Z,0,1,... 9) alfabeto alfanumérico
- Una morfología: El estudio de las formas de las palabras; es decir, su estructura interna.
- Una sintaxis: Conjunto de reglas que gobiernan la formación de los programas.
- Una semántica: Definición de la función y del sentido asociado a cada instrucción del programa.
- Una potencia lexicográfica: El número de palabras distintas que se pueden formar a partir de un lenguaje. Ej. Si la base alfabética es el alfabeto decimal

(0,1,2,3,4,5,6,7,8,9) y la longitud de la palabra es 9, la potencia lexicográfica será: 10^9.

Lenguaje binario

Es el más simple de todos los lenguajes y se ajusta a las siguientes características:

- Alfabeto: Consta de una base alfabética de dos caracteres numéricos (0,1)
- Morfología: no se ajusta a reglas alfabéticas.
- Potencia de lenguaje: Para una longitud de palabra fija e igual a n será 2^n
- Así, para una palabra de longitud 4 existirán 2^4=16 combinaciones posibles: 0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1001, 1010, 1011, 1100, 1101, 1110, 1111

Lenguaje máquina

Dado que la circuitería del ordenador sólo acepta, e interpreta, dos estados físicos, el lenguaje binario es la base del "lenguaje máquina". Como la palabra lo dice, es particular de cada ordenador en concreto y aunque su base sea binaria existen entre unos y otros grandes diferencias.

Interesa precisar dos hechos:

- Un ordenador sólo entiende y acepta su propio lenguaje de máquina.
- La presencia de un lenguaje más evolucionado, y por lo tanto los programas escritos en ese lenguaje, exigirá la existencia de un "traductor" (escrito en el propio "lenguaje máquina") para que convierta estos programas al lenguaje máquina.

Este lenguaje es muy incómodo de utilizar. Por una parte, es necesario conocer los códigos de las operaciones y por otra, hay que disponer de las direcciones que los datos toman en la memoria central (lo que se hace más difícil a medida que el programa es más largo).

Todo ello obligo a que las casas constructoras de equipos informáticos se esforzarán desde el principio para permitir una programación más sencilla para sus ordenadores y plantearon diversas formas de automatizarla.

Un primer paso fue utilizar bases numéricas superiores a dos para la representación de las instrucciones, en la entrada y la salida del ordenador: octal, decimal, hexadecimal, etc.

Pero la verdadera evolución la marcan dos aspectos:

1) La simbolización
2) La utilización de lenguajes intermedios.

La simbolización

Un gran inconveniente en la programación, por el hecho de que el programador debe conocer en todo momento la ubicación de sus datos en memoria central.

Los lenguajes "simbólicos" obvian este inconveniente al permitir reemplazar los códigos de operación y las direcciones numéricas de memoria por símbolos que representan esos códigos y direcciones.

Los nombres simbólicos de las direcciones de memoria exigen la observancia de determinadas reglas de formación, por ejemplo: limitación del número de caracteres, obligación de que aparezca un carácter en particular, etc.

Las instrucciones escritas bajo forma simbólica deben ser transcritas en instrucciones en lenguaje máquina, ya que los símbolos no pueden ser interpretados directamente por el ordenador. Para ello, es necesario crear unas tablas de correspondencia a fin de traducir esos símbolos en códigos de operación y direcciones numéricas. Esta función la desarrolla un programa particular (que proporciona la casa constructora con el equipo) que se denomina TRADUCTOR.

Vemos que estos lenguajes simbólicos (denominados traductores intermedios) necesitan la presencia de un traductor, cuya función es la de traducir el programa fuente (programa origen) en el lenguaje máquina, dando como resultado un

programa objeto (programa resultante) que es directamente ejecutable por el ordenador.

Entre los lenguajes intermedios más importantes tenemos:

- Los "lenguajes de ensamble" (o denominados impropiamente ENSAMBLADORES)
- Los "lenguajes-autocode"

Lenguaje de ensamblaje

Es un lenguaje próximo a la máquina, en el que cada orden corresponde directamente a una instrucción en código máquina, o a la descripción de un dato elemental tal como se presenta en la memoria del ordenador: un octeto, una palabra, etc.

Este lenguaje posee, con respecto al lenguaje máquina, las siguientes ventajas:

- Direccionamiento simbólico.
- Pseudo-instrucciones: Instrucciones no traducibles a lenguaje máquina, pero aportando indicaciones útiles al traductor (ej. Implantación de variables).
- Reducción del tiempo de programación.
- Eliminación de errores de codificación.
- Puesta a punto del programa más rápida, ya que los traductores, además de efectuar la conversión, detectan los errores sintácticos del programa.

La acción del traductor (ensamblador) consiste en traducir todos los elementos simbólicos del programa fuente, tanto los códigos de operación como las direcciones de memoria. Con respecto a los códigos de operación, el ensamblador compara cada símbolo encontrado en el programa con los existentes en una tabla preestablecida, cargada en memoria en el momento del ensamblaje. Cuando se produzca la correspondencia, el código simbólico será substituido por su correspondiente numérico de la tabla. Para las direcciones, el proceso es algo diferente: en una primera pasada, el ensamblador crea en memoria una tabla, asignando una dirección real a cada dirección simbólica que encuentra. En la segunda pasada, el

ensamblador reexamina cada instrucción y reemplaza los símbolos por las direcciones correspondientes de la tabla. Vemos que es preciso que el ensamblador lea, por lo menos, dos veces el programa fuente para obtener el programa en lenguaje máquina (programa objeto).

La función del ensamblador no se reduce a estas operaciones que se acaban de indicar. También se encarga de reservar zonas de memoria, cargara algún dato inmediato en la memoria del ordenador, etc.

Lenguaje auto-code

También es un lenguaje próximo al "lenguaje máquina". Se diferencia del lenguaje de ensamblaje por la introducción de la MACRO-INSTRUCCIÓN. Se denomina así porque en el momento de la traducción genera varias instrucciones de código máquina (algunas macroinstrucciones pueden generar cientos de instrucciones).

La noción de macro-instrucción esta ligada al concepto de subprograma, procedimiento o rutina.

Existen dos estructuras diferentes en la incorporación de las macroinstrucciones al programa, que se describen a continuación:

1) Cada vez que se llama a la macro-instrucción, el TRADUCTOR genera las instrucciones que la definen a continuación del punto de llamada (código en línea).
2) Cuando se llama a la macro-instrucción, el traductor sustituye la macro por una referencia al punto de inicio de ese conjunto de instrucciones que constituye la macro-instrucción. Este es el concepto de subprograma.

La forma segunda representa, con respecto a la primera, la ventaja de una menor ocupación de memoria central, ya que sólo se genera el código máquina de esa macro una sola vez. Pero tiene el inconveniente de dar una ejecución más larga por el hecho de tener que saltar al subprograma y retornar a él.

En el "lenguaje auto-code", la técnica del ensamblaje es sensiblemente la misma que en el "lenguaje de ensamblaje". Al ensamblador del auto-code se le denomina

MACROENSAMBLADOR que tiene la ventaja de poder utilizar subprogramas ya existentes en el sistema, definidos por el constructor o por otro usuario del equipo.
Los "lenguajes intermedios" se suelen utilizar, bien porque las operaciones que se desean programar no son posibles con los lenguajes más evolucionados, o bien porque los programas a construir son de ejecución permanente y precisan una gran optimización tanto de memoria como de tiempo de ejecución. Este es el caso de los sistemas operativos y de los procesadores básicos: COBOL, FORTRAN, etc. No obstante, la tendencia actual es la de utilizar lenguajes de alto nivel, tipo de lenguaje C, en la programación de sistemas.

Los lenguajes evolucionados

La llegada de los lenguajes de alto nivel marcan una etapa importante en la evolución de los lenguajes intermedios. Mientras que los lenguajes descritos hasta ahora están "orientados a la máquina", los de alto nivel van "orientados al problema". Esto último explica que la estructura de los lenguajes evolucionados sea completamente diferente al de los lenguajes de ensamblaje, estando los primeros mucho más próximos al lenguaje natural. Los lenguajes evolucionados tienden a una universalidad de empleo, permitiendo ser procesados, casi, con independencia del tipo de ordenador.
Presentan, con respecto a los lenguajes intermedios, las siguientes ventajas:

- Un aprendizaje más rápido.
- Una disminución del tiempo de programación. La codificación es más simple en estos lenguajes.
- Una puesta a punto de los programas más rápida.

El programa de traducción se llama COMPILADOR, y la traducción es la COMPILACIÓN. Además, tienen entre ellos una estructura muy común:

- Un alfabeto: De tipo alfanumérico, con caracteres especiales.
- Una morfología: Sin restricciones alfabéticas en la formación de palabras.

- Una sintaxis: que permite la existencia de expresiones y frases para describir los datos, las sentencias*, los procedimientos, etc.
- Una semántica: Para dar un sentido concreto a las palabras empleadas.

Las palabras se clasifican en varias categorías:

- Identificadores: son nombres simbólicos que sirven para referirse a las variables, funciones y subrutinas del programa (son indicativos que se aplican a los datos y a los resultados).
- Etiquetas: Son indicativos que se refieren a una o varias instrucciones.
- Palabra clave: Son identificadores que utiliza el propio lenguaje para definir el tipo de los datos y las instrucciones. Tienen un significado concreto y el programador no las puede utilizar fuera de su contexto.
- Operadores: Son símbolos que permiten la ejecución de ciertas operaciones aritméticas y lógicas, tales como suma, resta, división entera, or, and, not, multiplicación, división real, exponenciación.

Tipos de lenguajes evolucionados: Se distinguen dos categorías tomando como criterio su campo de aplicación:

a) Los lenguajes de propósito general. Que permiten resolver, en principio cualquier tipo de problema. Se dividen en dos tipos, en función de la aplicación para la que han sido desarrollados.
   - Lenguajes definidos para aplicaciones técnico-científicas: El FORTRAN, ALGOL y BASIC que fueron los primeros en ser desarrollados, entre 1954 y 1960, y el PASCAL y el lenguaje C que se desarrollaron posteriormente, en 1970 y 1972 respectivamente, son los más significativos en esta categoría de lenguajes.
   - Lenguajes de gestión: el COBOL, RPG, etc. El COBOL ha sido básicamente estudiado y pensado para programar aplicaciones de gestión. Fue desarrollado, en 1959, por el Departamento de Defensa de los EE. UU.

* "Secuencia de expresiones que especifica una o varias operaciones". Diccionario Usual. Editorial Larousse. 7ª. Edición, México 1994,

b) Los lenguajes especializados: Que han sido diseñados para un tipo particular de aplicación (ej. Manejo de máquinas herramientas, enseñanza asistida por ordenador, cálculo de estructuras, etc.) Son a título de ejemplos el GPSS y SIMULA (para simulación de modelos), SNOBOL y LISP (para manipulación de cadenas de caracteres y programación funcional, en el caso de LISP), LP y FMPS (para programación lineal), etc. Incluso en la actualidad muchos programas de bases de datos incluyen sus propios lenguajes de programación con el fin de manipular los datos almacenados (DBASE, ACCESS, IMAGEPRO, etc.)

En la actualidad, casi todos los compiladores están concebidos para generar, a partir del programa fuente un módulo objeto que a su vez produce un programa objeto directamente ejecutable por el ordenador. Un compilador es un traductor de programas escrito en un lenguaje evolucionado. El compilador es un programa complejo (normalmente escrito en un lenguaje de ensamblaje) que tiene como misión traducir expresiones y sentencias, escritas en un lenguaje de alto nivel, en instrucciones que comprenda e interprete el ordenador.

Pascal

En particular, este lenguaje requiere que se declaren todos los identificadores antes de usarlos, ya que se requiere para el uso de subrutinas, las cuales demandan diseñar el programa antes de escribir una sola línea de código, marca indiscutible de que es un lenguaje estructurado. El diseño estructurado es de gran ayuda en la depuración de cualquier programa con una complejidad más que trivial y también es muy útil cuando se requiere modificar un programa. También se generaron versiones que apoyaban la programación orientada a objetos, la cual utiliza la idea de que las características internas de cada rutina deben estar ocultas de todas las demás partes. Difiere de la estructurada en la noción de que las partes del programa no son subrutinas, sino datos inteligentes que tienen sus propias subrutinas, que saben como hacer las cosas por ellos mismos.

Fortran

En 1954, un grupo en IBM propuso un sistema entre cuyas características estaban: niveles de asignación, do-loops (ciclos de control) y niveles de entrada/salida, aritmética de matrices, solución simultanea de sistemas de ecuaciones y resolución de ecuaciones diferenciales. El primer compilador (FORTRAN I) fue desarrollado con algoritmos para generar un código muy eficiente, es decir que no debía tener una degradación significativa en su desempeño comparado con el que se lograría con su equivalente en versiones en código máquina. La versión posterior (FORTRAN 66) presentó facilidades para el manejo de datos de caracteres, tipos de datos estructurados, construcción estructurada y recursión.

La siguiente versión, FORTRAN 77, añadió las variables de cadena, y la estructura IF; pero también dejo muchas secciones que no cubrían las necesidades de la comunidad científica por lo que casi inmediatamente surgió el siguiente estándar (FORTRAN 90). Las características más importantes de esta nueva versión es que era más fácil de escribir y de mantener, el uso de nuevas características en el manejo de arreglos, nuevos procedimientos intrínsecos e invocación elemental de los mismos lo que ayudo a que fueran más fáciles de leer los programas. Tipos derivados y apuntadores permitieron la creación de complejas estructuras de datos abstractos, tales como listas encadenadas y árboles de directorios, almacenamiento dinámico, el uso de módulos, procedimientos genéricos y tipos derivados. Todas estas características sirvieron para hacer de FORTRAN un excelente lenguaje para resolver los problemas encontrados en aplicaciones científicas, numéricas y de ingeniería.

La última versión de FORTRAN provee de velocidad y soporte en la escritura de programas o en la edición de aplicaciones existentes, mejoras en el procesamiento de condicionales lógicos, librerías en tiempo de ejecución (DLL´s), un paquete gráfico para diseño visual, compatibilidad con MATLAB, librerías de funciones y subrutinas científicas, una herramienta de prueba para mejorar el desempeño del programa y la posibilidad de integrar librerías DLL generadas en otros lenguajes.

C++

El C++ Está compuesto por declaraciones y enunciados en los que se especifican las instrucciones exactas que deben seguirse cuando se ejecute el programa.

El C++ fue creado por Bjarn Stroustrup en los laboratorios Bell. C++ supera al popular lenguaje C mediante el añadido de extensiones de lenguaje orientadas a objetos.

Un lenguaje orientado a objetos representa los atributos y operaciones de los objetos.

La programación en C++ requiere que usted tenga en mente las bibliotecas de soporte que ejecutan varias tareas, como la entrada, la salida, el manejo de texto, las operaciones matemáticas, la E/S (entrada/salida) de archivos, etc. En lenguajes como el BASIC, el soporte para este tipo de operaciones se manifiesta al trasluz de los programas debido a que se dispone de él automáticamente. En consecuencia, muchos programas se presentan como componentes individuales que son independientes de cualquier otro componente de programación. Por el contrario, la programación en C++ genera la conciencia de que se tiene más dependencia de diversas bibliotecas. La ventaja de esta característica del lenguaje es que se puede seleccionar entre bibliotecas similares. Por lo tanto, los programas de C++ son modulares. Los compiladores de C++, entre los que se puede contar el visual C++, emplean archivos de proyecto y archivos de programa. El banco de trabajo del visual C++ se vale de archivos de proyecto para administrar la creación y actualización de un programa.

Los archivos de proyecto especifican la biblioteca. Los archivos de programa crean una aplicación.

Visual Basic.

En 1960 John Kemeny y Thomas Kurtz desarrollaron el BASIC® (acrónimo de Beginner´s All-Purpouse Symbolic Instruction Code, Código de Instrucciones

Simbólicas de propósito General para Principiantes) como un lenguaje de aprendizaje. Sus características principales eran su naturaleza interactiva y su simplicidad. Estaba basado en el concepto de ser un interprete que permitía que se recibieran las respuestas inmediatamente, por lo que era posible hacer el seguimiento del programa conforme se iban escribiendo las líneas de código. Asimismo gozaba de la simplicidad de tener un vocabulario muy parecido al inglés, por lo que se podían escribir programas sin tener que interactuar con un vocabulario o una sintaxis extraña.

La aparición del entorno Windows® revolucionó de manera notable el desarrollo del software al facilitar el desarrollo de interfases gráficas sin la necesidad de procedimientos complejos y la utilización de lenguajes poco amigables.

Visual Basic® es una combinación de herramientas visuales del entorno Windows®, con códigos escritos en BASIC®. Una fusión que permite el desarrollo de ventanas de Windows® colocándoles botones de comandos, menús, cuadros de texto, etc. sin necesidad de escribir una sola línea de programación; Son una serie de herramientas que modifican ostensiblemente al BASIC® original al migrarlo a ser un lenguaje orientado a objetos, es decir, que a cada elemento gráfico se le asigna un código de programación que permitirá a ese elemento llevar una acción especifica en una situación dada.

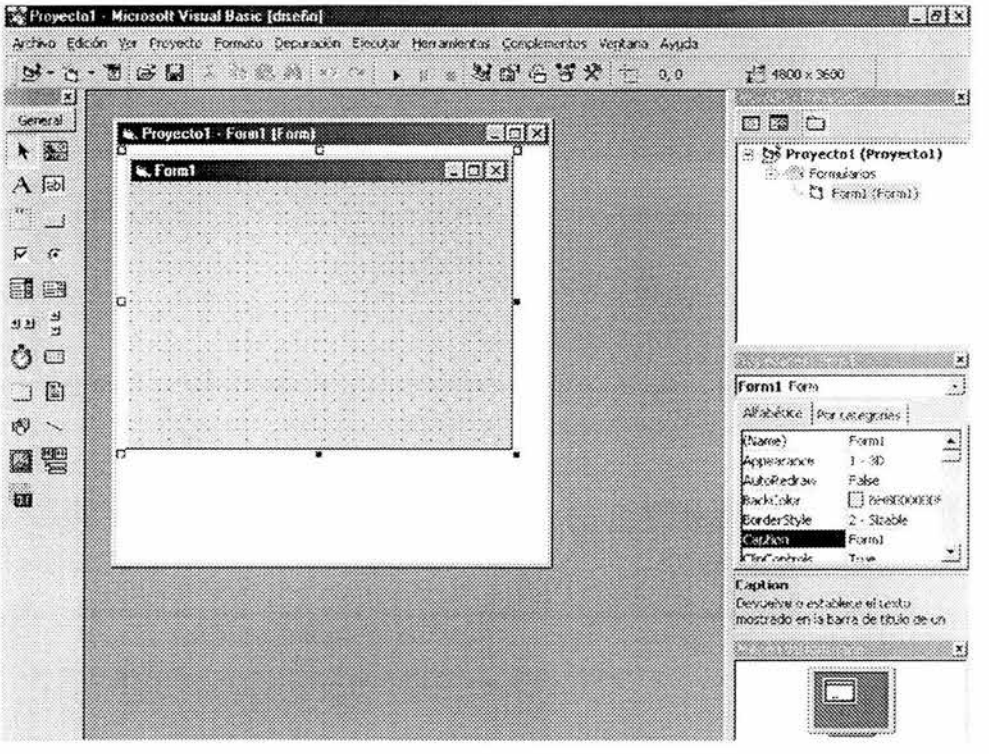
Se ha seleccionado este lenguaje de programación por ser uno de los más difundidos en la actualidad y a que el programa de aplicación Microsoft Office® (Word®, Excel®, Power Point®, Access®) lo integra como un elemento de substitución de las macros lo que aumenta las posibilidades y potencia para el usuario. Office® es hasta el momento empleado por casi un 90% de los usuarios de computadora personal.

Esta unificación es de gran ayuda también en la integración de los programas que desarrollemos con documentos generados a través de Office®.

Por otro lado, al ser un lenguaje de programación con bases en el idioma inglés, es relativamente sencillo trasladar el código a otro lenguaje de programación. Por lo que

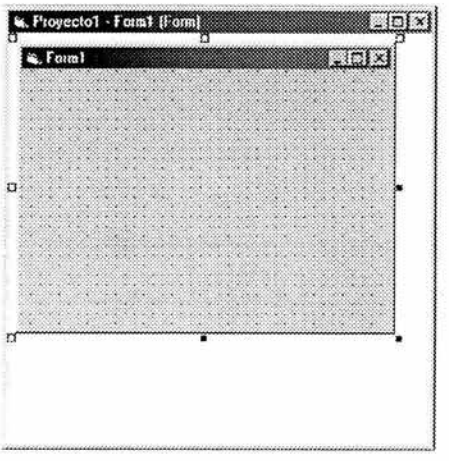
si tenemos alguna preferencia por un lenguaje diferente no tendremos que rediseñar el programa para hacerlo funcionar.

A continuación describiremos brevemente la interfase de programación (IP), con el propósito de que identifiquemos nuestro entorno de desarrollo, para posteriormente desarrollar programas que nos permitan la resolución de problemas en los que se empleen los métodos numéricos:



Normalmente esta debería ser una imagen típica de la IP al iniciar cualquier proyecto. Un proyecto puede constar de una o más formas; en ellas son colocados los elementos que servirán de base para la elaboración de la apariencia gráfica de nuestro programa. A dichos elementos bastará dar un doble clic para tener la posibilidad de adjuntarles el código de programación que debiera ejecutarse al ocurrir ciertos eventos. Por ejemplo, podemos colocar un botón de comando y agregarle

código, de manera que cuando al ejecutar nuestro programa; si ocurre el evento de darle clic, calcule una serie de formulas o agregar otro botón que mientras no se presione estará realizando una serie de iteraciones. El tipo de evento que esperaremos dependerá tan solo de nuestra creatividad o del tipo de problema a resolver.



Otra de las ventanas con que cuenta nuestra IP, es la ventana de propiedades. Esta ventana mostrará las propiedades del elemento seleccionado, es decir, que si damos click en la forma de programación, mostrará las propiedades de la forma y si seleccionamos un cuadro de texto, entonces se enlistarán las referidas a éste.

La barra de elementos permite seleccionar aquellos objetos que deseamos agregar a la forma; para ello es necesario dar clic sobre el elemento seleccionado y mover el

puntero hasta colocarlo sobre el formulario (o forma) y mientras se lleva a cabo la acción de arrastrar se dimensiona el elemento en tamaño y forma hasta que queda como se desea. Posteriormente en la ventana propiedades se le puede asignar un nombre, que es como se le identificará cuando se proceda a añadir código, así como detallar las características de color, etiqueta, estado inicial, etc.

La ventana del proyecto nos permitirán mostrar los elementos de que consiste el proyecto, por ejemplo si nuestro proyecto consta de dos formularios podemos hacer que con solo dar doble clic sobre el nombre de la forma, esta nos sea presentada.

La barra de herramientas consiste de elementos que nos permitirán controlar las acciones referidas al proyecto, como por ejemplo agregar formas, guardar el proyecto, mostrar algunas ventanas de la IP, hacer una depuración del código o llevar a cabo una ejecución de prueba antes de la compilación, etc.

Y por último, el elemento más importante, la barra de menús, que nos dará acceso a todas las acciones que son desarrolladas mediante las ventanas anteriormente descritas.

Software Comercial

Los pasos a seguir por un estudiante en su uso de computadoras durante su aprendizaje en la carrera de Ingeniería química son:

1) ubicar el modelo de ecuaciones para el problema en mano.
2) Encontrar el método numérico apropiado para resolver el modelo anterior.
3) Escribir y depurar programas para resolver el problema utilizando el algoritmo numérico seleccionado y
4) Analizar la validez y precisión de los resultados.

Sin embargo se ha identificado que estos dos últimos pasos son contribuciones menores en el aprendizaje del material objeto de estudio en el curso de ingeniería química, sin embargo son las que actualmente consumen la mayor cantidad de tiempo además de ser frustrantes para el estudiante.

La generación de programas permite la resolución de problemas reálistas, a pesar del tiempo gastado en detalles técnicos que son de relevancia menor para el tema.

Por ello hay una tendencia a proveer a los estudiantes con programas que pueden resolver un tipo particular de problemas. Listados o incluso discos conteniéndolos son incluidos en libros de texto o grandes programas de simulación están disponibles comercialmente. Esta vía tiene la desventaja que los programas son usados a

manera de "caja negra" en donde el estudiante solo provee los datos de entrada y observa los resultados. El muy importante paso de convertir un fenómeno físico a un modelo matemático se pierde, por tanto la conexión entre el modelo matemático y el problema se obscurece.

En los inicios de los 80´s paquetes de software numérico interactivo con capacidades gráficas comenzaron a emerger como herramientas computacionales mayores en la resolución de problemas de ingeniería (POLYMATH, MAPLE, EXCEL, MATHCAD, MATEMÁTICA, MATLAB). Sin embargo, con el uso de estos el estudiante se ve relevado de ejecutar los pasos técnicos de la solución.

Como sea la integración de software en los cursos iniciales de ingeniería es todavía baja: la resolución por aproximación numérica y métodos de cálculo son poco influenciados por la disponibilidad de computadoras. Parte de la respuesta a este bajo nivel de empleo de computadoras es que los educadores no están familiarizados lo suficiente con las herramientas computacionales; otro aspecto posible es que los paquetes no son amigables al usuario y no son tan fáciles de usar como ellos proclaman.

Los paquetes más integrados con los cursos de ingeniería deben tener capacidad suficiente para resolver los problemas encontrados durante el curso, pero a través de una interfase amigable, con una corta línea de aprendizaje y debe requerir la intervención mínima del usuario en los detalles técnicos del proceso de solución.

Los tipos de problemas más representativos en un curso típico de ingeniería química incluyen el uso de modelos matemáticos tales como:

1) Ecuaciones algebraicas no lineales sencillas y simultáneas. (Volumen molar y factores de compresibilidad usando la ecuación de Van Der Waals, equilibrio de reacciones para múltiples reacciones en fase gas, velocidad final en la caída de partículas, destilación batch binaria, etc.)
2) Ecuaciones lineales simultaneas (Balances de masa en estado estacionario en un tren de separación)
3) Ecuaciones diferenciales simultáneas con condiciones iniciales conocidas y con valores limite divididos (Intercambio de calor en estado no estacionario en una

serie de tanques agitados, difusión unidireccional con reacción química, destilación batch binaria, etc.)

4) Ajuste de curvas polinomiales (representación de datos de presión de vapor por polinomios y ecuaciones)
5) Regresión lineal, lineal múltiple y regresión no lineal. (representación de datos de presión de vapor por polinomios y ecuaciones)

Al hacer la comparación entre el uso de paquetes de software y el uso de un lenguaje de programación, se obtiene la ventaja de un ahorro de tiempo significativo pero solo después de invertir tiempo y esfuerzo en el aprendizaje del paquete de software. Sin embargo esto lleva a una "adicción" del paquete seleccionado y a pensar que es apropiado para toda clase de necesidad.

Polymath

Automáticamente reordena las ecuaciones, para permitir expresiones explícitas de las variables algebraicas, la pantalla de entrada también muestra variables no definidas durante la entrada del problema (muy útil en la depuración del modelo matemático); es manejado a través de menús y todas las opciones disponibles son presentadas en pantalla, ninguna información técnica del método de integración o criterio de error necesita ser proporcionada por el usuario. El software tiene una tolerancia de error relativo por default el cual detiene la integración con un mensaje de error si una solución no puede ser obtenida con la precisión deseada.

Los resultados pueden ser mostrados en forma gráfica con una calidad suficiente para incluirlos en reportes; La escala y etiquetas son incorporadas automáticamente pero pueden ser modificadas si así se desea. El modelo es documentado en forma clara y compacta. La documentación incluye todas las ecuaciones, valores iniciales de las variables dependientes e independientes y valores finales de las variables independientes.

Mathcad

La definición del modelo requiere cambios más extensivos a la ecuación original que los requeridos en POLYMATH, claras distinciones deben ser hechas entre las variables calculadas por ecuaciones algebraicas y diferenciales. Las variables diferenciales deben ser expresadas como vectores y las ecuaciones diferenciales como vectores de funciones.

El nombre de una variable algebraica debe incluir la lista de las variables diferenciales que utiliza. El programa esta basado en instrucciones y el usuario debe especificar el algoritmo de integración a ser usado y el número de puntos a ser guardados para su graficación. Los resultados son almacenados en una matriz y el usuario debe asignar nuevamente las columnas de la matriz a las variables para obtener una curva adecuada. La transformación de las variables y funciones hacia un vector y la transformación de los resultados nuevamente hacia las variables son detalles técnicos innecesarios que el usuario debe manejar.

Matlab

La solución requiere que el usuario escriba un programa donde MATHLAB provee los subprogramas para la integración numérica y la graficación. Al igual que MATHCAD, las variables diferenciales y funciones deben ser transferidas a vectores y los resultados asignados de vuelta a las variables.

El usuario debe seleccionar el método de integración a ser empleado. La creación de una gráfica requiere varias líneas de código. Para poder escribir un programa o entender la documentación del modelo, el usuario debe estar familiarizado con la sintaxis propia de MATLAB.

Análisis numérico

El análisis numérico trata de diseñar métodos para aproximar, de una manera eficiente, las soluciones de problemas expresados matemáticamente. La eficiencia del método depende tanto de la precisión que se requiera como de la facilidad con la que pueda implementarse.

Como la facilidad de implementación depende significativamente de los cambios tecnológicos en calculadoras y computadoras, los adelantos en estos dispositivos ha hecho que estos métodos sean más y más atractivos a pesar de la gran cantidad de operaciones matemáticas vinculadas con la resolución de problemas a través de métodos de aproximación.

Debido a que no es posible generar un texto completo que cubra todos y cada de los métodos existentes, el presente texto será una referencia para emplear otros métodos que pudiesen necesitarse más adelante para un problema dado.

Un método numérico que puede usarse para resolver un problema se llama algoritmo. El algoritmo consiste de una secuencia de operaciones algebraicas y lógicas en términos del programador que producen la aproximación al problema matemático y se espera que también al problema real, con una tolerancia o precisión predeterminada. La elección de un método depende de las opciones a la mano, sus características sobresalientes y su adecuación para un problema dado.

Un algoritmo no está bien definido a menos que no permita ambigüedad alguna acerca de cual es la operación que debe seguir a una operación cualquiera de éste. Causas obvias de interrupción de un algoritmo son las divisiones por cero, o (en el dominio real) las raíces cuadradas de números negativos. Más generalmente un algoritmo se interrumpirá siempre si una función ha de ser evaluada en un punto en donde no esté definida.

Los algoritmos diseñados para la solución de problemas de análisis numérico consisten de un número finito de operaciones que pueden efectuarse en cualquier operación práctica. Los métodos analíticos determinan que la precisión de la respuesta aumenta a medida que aumenta el número de operaciones implicadas,

esto es que la precisión puede hacerse tan sensible como se quiera con solo incrementar el número de pasos efectuados.

Muchos algoritmos sencillos pueden describirse perfectamente por medio de los símbolos convencionales del álgebra, suplementados, si es necesario por proposiciones en el lenguaje habitual.

Una vez que el algoritmo está propiamente formulado debemos conocer exactamente cuáles son las condiciones bajo las que el algoritmo produce la solución del problema considerado. Si el algoritmo da como resultado la construcción de una sucesión de números cada vez que se realiza el proceso, debemos conocer las condiciones para que esta sucesión sea convergente. Una vez resuelta la cuestión de la convergencia, se puede querer saber acerca de la velocidad de la misma o acerca del tamaño del error.

Para que sea confiable, un algoritmo debe permanecer inmune a la acumulación de errores de redondeo. A esta inmunidad se le llama *estabilidad numérica*. Esta estabilidad depende de la máquina calculadora en que se efectúa. Por lo tanto, es posible incrementar la estabilidad de ciertos algoritmos en el álgebra lineal numérica efectuando ciertas operaciones decisivas con precisión aumentada.

Capítulo 1.- Búsqueda y selección de métodos

La resolución de ecuaciones no lineales, es una de los problemas que aparecen más frecuentemente en ingeniería. Consiste en hallar los valores de x tales que f(x)=0, estos valores se denominan raíces ó ceros. Los métodos numéricos obtienen una sucesión de valores que se aproxima a la raíz partiendo de una ó más aproximaciones. Existen 3 técnicas para resolver este problema: métodos analíticos, métodos gráficos y métodos numéricos. Los últimos son generalmente la mejor opción.

La selección de los métodos se llevará a cabo en una forma arbitraria con base a que tan fácil es acceder a información sobre ellos en la literatura, es muy relativo y depende de la apreciación, pero muy útil para la situación de tener un conocimiento claro de su desarrollo con el fin de poder concentrarse en la programación de código para la computadora. Da la opción a que si no se entiende la forma de expresar el algoritmo se pueda recurrir a la literatura y aclarar la percepción. En el caso del método gráfico, su selección se basó en que permite tener una visión de las funciones para las que se trata de obtener una solución aproximada.

1.1. Ecuaciones no lineales

Los métodos que se presentan requieren que las funciones sean diferenciables, y por tanto continuas, en el intervalo donde se apliquen aquellos. También se puede intentar usarlos para funciones no diferenciables o discontinuas en algunos puntos, pero en este caso el llegar al resultado dependerá, aleatoriamente, de que durante la aplicación del método no se toquen esos puntos.

El problema de obtener las soluciones o raíces de una ecuación algebraica o trascendente de la forma F(x)=0 se presenta frecuentemente dentro del campo de la

ingeniería. Debido a ello, el desarrollo de métodos que permitan solucionarlo es amplio.

1.1.1. Método de Bisección

Este método también llamado de bipartición del intervalo[1], es el más simple, aunque también es el más seguro y sólido para encontrar una raíz en un intervalo dado donde se sabe que existe dicha raíz. Su única ventaja es que funciona aún para funciones no analíticas.

El método de bisección no puede encontrar una pareja de raíces dobles, sin embargo encontrará una de las raíces separadas en el intervalo dado.

Un defecto de éste método es que puede atrapar una singularidad como si fuera una raíz, debido a que dicho método no reconoce la diferencia entre una raíz y una singularidad. Un punto singular es aquel en el que el valor de la función tiende a infinito[2] o más precisamente como el punto de una curva $y=f(x)$ en el cual la derivada dy/dx toma la forma indeterminada $0/0$[3].

Converge muy lentamente, y una buena aproximación puede ser desechada sin que nos demos cuenta sin embargo, el hecho de que converge a una solución lo hace ideal para "poner en marcha" a los métodos más eficientes.

El método parte de una función $F(x)$ y un intervalo $[x_1,x_2]$ tal que $F(x_1)$ y $F(x_2)$ tienen signos contrarios. Si la función es continua en este intervalo, entonces existe una raíz de $F(x)$ entre x_1 y x_2.

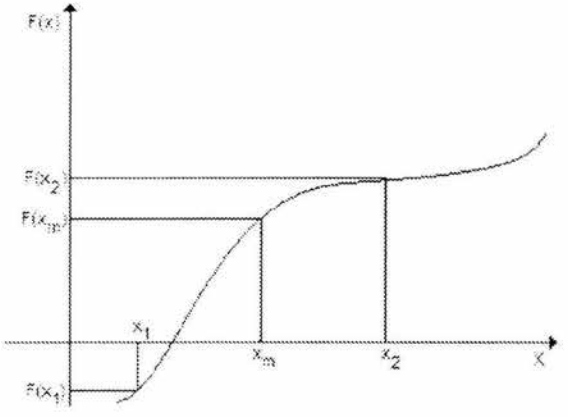
Una vez determinado el intervalo $[x_1,x_2]$ y asegurada la continuidad de la función en dicho intervalo, se valúa ésta en el punto medio x_m del intervalo. Si $F(x_m)$ y $F(x_1)$ tienen signos contrarios, se reducirá el intervalo de x_1 a x_m, ya que dentro de estos valores se encuentra la raíz buscada; en caso contrario, que $F(x_m)$ y $F(x_2)$ tengan signos contrarios, el intervalo se reduciría de x_m a x_2. Al repetir este proceso, hasta

---

[1] Métodos Numéricos Aplicados a la Ingeniería. Terrence J. Akai; LIMUSA-WILEY; México, 1ª ed., 1999, p. 141.

[2] Métodos Numéricos Aplicados con Software. Shoichiro Nakamura, Prentice-Hall Hispanoamericana S.A., México, 1992.

lograr que la diferencia entre estos dos valores de x_m sea menor que una tolerancia prefijada, el último valor de x_m será una buena aproximación de la raíz.



1.1.2. Método de Punto fijo

El método de punto fijo, de iteración funcional o de aproximaciones sucesivas es, junto con el de Bisección, uno de los primeros métodos que se utilizaron para resolver ecuaciones algebraicas y trascendentes. No obstante de existir otros métodos más eficientes, el de punto fijo se considera el más simple en sus principios y en él se pueden apreciar claramente todas las características de un método de aproximaciones sucesivas.

Sea F(x) = 0 una ecuación algebraica o trascendente cualquiera. Se suma una x en ambos miembros y se obtiene:

$$F(x) + x = x$$

Donde el miembro izquierdo es otra función de x que se define como

$$G(x) = F(x) + x$$

Se sustituye en la ecuación anterior:

$$X = G(x)$$

[3] Diccionario Especializado de Matemáticas, Ed. Norma, Bogota, Colombia, 2001

Obsérvese ahora que cualquier ecuación puede representarse en esta forma, siguiendo el procedimiento anterior. Si x = a es una raíz de la ecuación, entonces

$$F(a)= 0$$

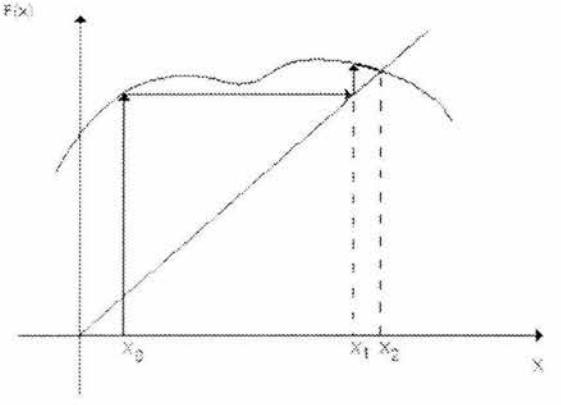
O bien, al sustituir en la ecuación anterior

$$A = G(a)$$

El método de aproximaciones sucesivas consiste en sustituir un valor inicial (x_0) apropiado (cercano a la raíz), si ocurre que $x_0 = G(x_0)$, entonces x_0 es la raíz.

Si $x_0 \neq G(x_0)$ entonces x1 = $G(x_0)$ hasta encontrar una solución aproximada a la precisión que se busca.

Puede afirmarse que si el método converge, la diferencia en el valor absoluto entre los valores proporcionados por dos iteraciones sucesivas será cada vez más pequeña a medida que el número de iteraciones aumente, y con esto se tendrá un criterio para saber cuando se termina la aplicación del método.



1.1.3. Método de Regula Falsi

Otro método que comúnmente se emplea es el de Regula Falsi. Este método también tiene otras denominaciones, como son: Regla falsa, Posición falsa o Interpolación Lineal. Su nombre original que esta en latín, denota su antigüedad. La idea del método es bastante similar al del método de Bisección. Requiere un intervalo que cumpla los mismos supuestos que el método de Bisección. En lugar de

obtener el punto medio en cada iteración, el método busca reemplazar la función original por otra a la cual sea más simple localizar su raíz. Dado que se comienza con solo un intervalo, es decir, sólo se tienen 2 puntos, se busca la curva más simple que pase por estos 2 puntos. Lógicamente se empleará una línea recta. Entonces en vez de obtener puntos medios, en este método se hallan las raíces de las rectas que pasen por los puntos que determinen los intervalos.

Se tiene que $y_0 = f(x_0)$ y $y_1 = f(x_1)$.

Primero se traza la recta que une los puntos (x_0, y_0), (x_1, y_1). El cero de esta recta esta dado por

$$x_2 = x_1 - y_1(x_1 - x_0) / (y_1 - y_0)$$

Si se evalúa la función en x_2 se obtiene $y_2 = f(x_2)$. Considerando los intervalos $[x_0, x_2]$ y $[x_2, x_1]$. Se detecta donde ocurre un cambio de signo, determinando que la raíz se encuentra en el intervalo $[x_0, x_2]$.

Dado que la raíz es única se puede descartar el intervalo $[x_2, x_1]$, dado que la raíz no esta ahí.

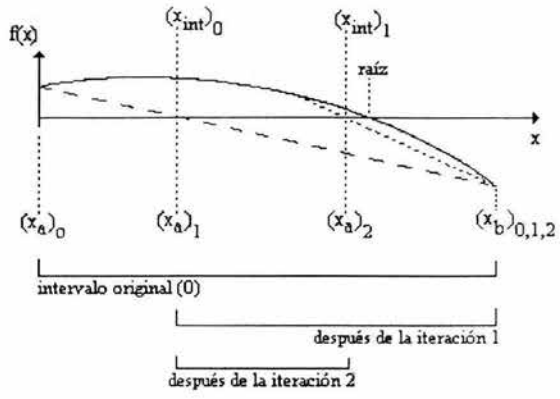
A continuación, se repite el procedimiento considerando el intervalo $[x_0, x_2]$. El cero de esta recta esta dado por

$$x_3 = x_2 - y_2(x_2-x_0)/(y_2-y_0)$$

Al evaluar en $f(x_3)$ se obtiene y_3

Se puede notar intuitivamente que se converge a la raíz. El procedimiento es esencialmente el mismo que para el método de Bisección. La diferencia estriba en que se obtiene el cero de la recta de que une los puntos del intervalo, en vez del punto medio del mismo.

El nombre del método de Posición o Regla Falsa se debe que se supone que el cero de la función coincide con el de una recta, lo cual obviamente no es cierto. El nombre Interpolación Lineal viene del hecho de que se esta obteniendo un valor de una función entre 2 puntos (polos).



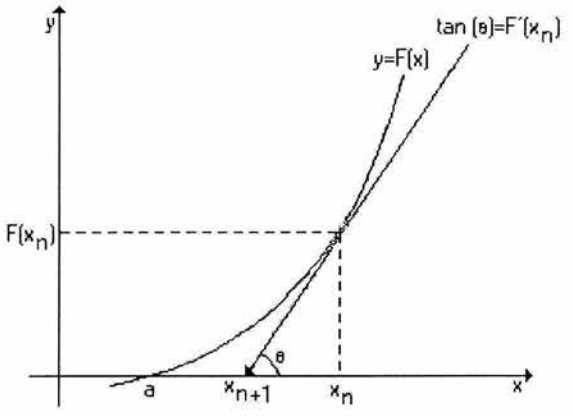
Para esté método la fórmula para cada iteración es

$$X_{n+1} = x_n - y_n(x_n - x_{n-1})/(y_n - y_{n-1})$$

El método es sencillo, pero un poco más complicado que el de bisección. El método es lento, pero más rápido que el de bisección. Puede demostrarse que si bien el método es lento, también es seguro, es decir, si se cumple todos los supuestos que el método exige, la convergencia está asegurada. Salvo raros casos, este método converge más rápido que el de bisección. Análogamente, al método de bisección, este método presenta convergencia lineal.

1.1.4. Método de Newton-Raphson

Este método como todos los de aproximaciones sucesivas, parte de una primera aproximación x_n y mediante la aplicación de una fórmula de recurrencia se acerca a la raíz buscada, de tal manera que la nueva aproximación x_{n+1} se localiza en la intersección de la tangente a la curva de la función en el punto y el eje de las abscisas, como se muestra en la siguiente figura.



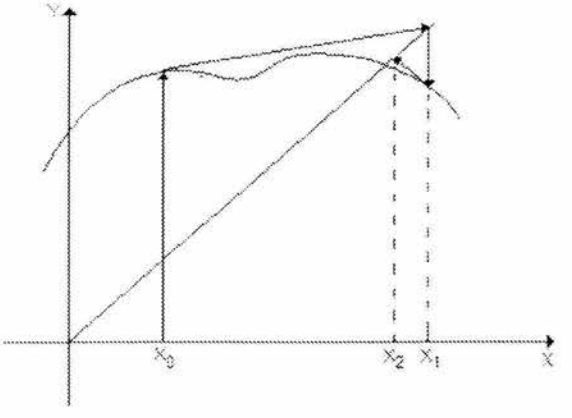
Además, por definición de tangente:

$$\text{Tan}(\theta) = F(x_n) / \{x_n - x_{n+1}\}$$

Por tanto:

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

Esta ecuación es la fórmula de recurrencia del método de Newton-Raphson, conocido también como el *método de las tangentes*.



El método de Newton es especialmente ventajoso por su rapidez de convergencia y en algunos casos es posible duplicar el número de cifras correctas en cada paso, sin embargo el método de la secante, él cual se verá más adelante, se usa frecuentemente para refinar las respuestas obtenidas por otras técnicas.

Para determinar los ceros de un polinomio, dispone de un algoritmo sencillo y eficiente, el cual se puede extender al cálculo del valor de su primera derivada. El método de Newton se puede aplicar al dominio complejo para hallar raíces complejas. También se puede extender a las ecuaciones no lineales simultaneas.
Las desventajas pueden resumirse en:

1) si $f'(x)$ se hace muy pequeño, entonces en estos puntos la tangente es paralela al eje X y las aproximaciones sucesivas pueden oscilar indefinidamente.
2) Si la curva $f(x)$ posee puntos de inflexión entre x_n y x^*, o entre x_n y x_{n+1}, entonces la iteración puede no converger a x^*, aunque puede converger a alguna otra raíz.
3) La aproximación inicial x_0 de una raíz de $f(x)=0$ tiene que estar suficientemente cerca de la raíz exacta (x^*) para garantizar la convergencia.
4) Al resolver problemas concretos, con frecuencia es difícil comprobar el cumplimiento de las limitaciones que se impone a la función $f(x)$, así como a la elección de la aproximación inicial x_0.
5) La necesidad de calcular $f'(x)$, cuando $f(x)$ no es un polinomio: en algunos casos $f'(x)$ no puede ser explícitamente utilizable, aún cuando puede ser evaluada, pues esto requeriría un trabajo computacional considerable.

Ante estas desventajas es necesario estudiar otros métodos con una eficiencia mayor o igual, pero con menos limitaciones.

1.1.5. Método de la secante.

Este método podría ser considerado una modificación del método de falsa posición, aunque en este caso $f(x_n)$ y $f(x_{n-1})$ no necesitan ser de signos opuestos. Es uno de los métodos más antiguos conocidos, aunque había sido poco apreciado hasta la actualidad, en que se han aprovechado sus ventajas, particularmente con el uso de computadoras; por esta razón y para ilustrar varios aspectos de los métodos iterativos se tomará como referencia.

Teniendo la curva de una función f(x) y tomando dos puntos P_n $(x_n, f(x_n))$ y P_{n-1} $(x_{n-1}, f(x_{n-1}))$ se traza una cuerda que una ambos puntos con lo que la distancia entre la intersección de la cuerda con el eje x está más cerca de $\dot{x}$.

Por tanto, si se considera a x_{n-1} y a x_n como las primeras aproximaciones de $\dot{x}$, la intersección de la cuerda con el eje x podrá ser considerada como la siguiente aproximación y se puede denotar como x_{n+1}.

Aproximando el valor de la derivada $f'(x_n)$, por el cociente de las diferencias:

$$\{f(x_n) - f(x_{n-1})\} / (x_n - X_{n-1})$$

y sustituyendo en la formula de iteración de Newton

$$x_{n+1} = x_n - [(x_n - x_{n-1})*f(x_n) / \{f(x_n) - f(x_{n-1})\}]$$

El cálculo mediante la sucesión definida por la fórmula anterior se denomina método de aproximación de la secante. Este método comienza a partir de dos estimaciones distintas x_n y x_{n-1} para estimar la raíz de la ecuación f(x)=0. Y para actualizar x_n y x_{n-1} y converger a una raíz se usa un procedimiento iterativo que se vale de la interpolación lineal.

El método de falsa posición tiene convergencia lineal para funciones convexas. Una situación similar se mantiene para funciones cóncavas. Debido a que la mayoría de las funciones son ya sea cóncavas o convexas en la vecindad de la raíz se puede lograr una convergencia lineal para casi todas las funciones. Entonces, la fuente de su pobre convergencia yace en que las iteraciones sucesivas se efectúan en el mismo lado de la raíz. La modificación adecuada del método puede esquivar esta deficiencia; La eficiencia del método de la secante se compara favorablemente con muchos métodos más sofisticados.

El hecho de que este método no requiera la evaluación de cualquier derivada puede ser una gran ventaja en ciertos problemas. El evaluar derivadas de *f* con respecto a *X* en algunos casos es frecuentemente impráctico.

Una desventaja del método es que se requiere una aritmética de doble precisión cuando la iteración se acerca a la convergencia debido a que involucra la diferencia de dos cantidades casi iguales.

Si se rescribe la ecuación principal se tiene:

$$X_{i+1} = X_i - ((x_i - x_{i-1})/(y_i - y_{i-1}))*y_i$$

En donde el segundo término puede ser considerado un término de corrección a x_i y como tal solo requiere de unos cuantos dígitos significativos cuando la convergencia es alcanzada. Como sea, la poca cantidad de dígitos significativos en el cociente determina que no haya la suficiente cantidad de cifras significativas en el cálculo de la raíz; ello repercute en la obtención de cantidades de simple precisión.

El método de la secante converge substancialmente más rápido al valor buscado que el método de falsa posición.

Mientras el método de falsa posición no depende del ordenamiento de los valores iniciales, el método de la secante y sus variantes (método Illinois y el método Pegaso) si lo hacen.

El caso problemático de la búsqueda de raíces lo constituyen las raíces repetidas. En este caso se emplean los métodos modificados los cuales solo se recomiendan cuando son realmente necesarios.

Los métodos modificados son más laboriosos que sus contrapartes, ya que requieren derivadas adicionales. El método de Newton Modificado requiere hasta la segunda derivada. El método de la secante modificado emplea la primera derivada.

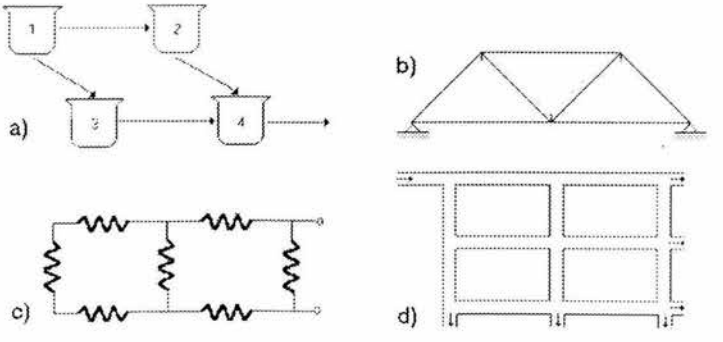
En una raíz simple es mejor emplear el método de Newton normal. El modificado requiere de más tiempo de maquina. En una raíz simple es mejor el método de la secante normal que el modificado, ya que ambos convergen igual de rápido, pero el primero no requiere la derivada. Además en este caso si se plantea calcular la primera derivada, es mejor emplear el método de Newton normal. Este converge más rápido que el de la secante. Por estas razones se recomienda usar estos métodos solo cuando sea requerido, es decir, en raíces dobles. También es recomendable realizar los cálculos en precisión doble

Resumen de los esquemas para encontrar raíces

Nombre	*Necesidad de especificar un intervalo que contenga a la raíz*	*Necesidad de la continuidad de f'*	*Tipos de ecuaciones*	*Otras características especiales*
Bisección	"Si"	No	cualquiera	Robusto, aplicable a funciones matemáticas
Falsa posición	"Si"	"Si"	cualquiera	Convergencia lenta en un intervalo grande
Falsa posición modificada	"Si"	"Si"	cualquiera	Más rápido que el método de falsa posición
Método de Newton	No	"Si"	cualquiera	Rápido; se necesita calcular f'; aplicable a raíces complejas
Método de la secante	No	"Si"	cualquiera	Rápido; no se requiere calcular f'
Sustitución sucesiva	No	"Si"	cualquiera	Puede no converger
Método de Bairstow	No	"Si"	polinomial	Factores cuadráticos

1.2. Sistemas de Ecuaciones Lineales

Las ecuaciones algebraicas lineales se presentan en el contexto de muchos problemas y en todos los campos de la ingeniería. En particular, se originan en los modelos matemáticos de grandes sistemas de elementos interconectados como reactores (a), estructuras (b), circuitos eléctricos (c) o redes de flujo (d).



En el tratamiento de sistemas de ecuaciones resulta generalmente conveniente expresar varias operaciones en función de matrices. Una matriz es una ordenación rectangular de números. Los números de la ordenación reciben el nombre de elementos de la matriz.

A continuación se definen diversos tipos de matrices:

Matriz cuadrada.- Es aquella que tiene el mismo número de filas y columnas.

Matriz diagonal.- Una matriz cuadrada que solamente tiene elementos a lo largo de su diagonal (mientras que todos los demás son ceros).

Matriz idéntica.- Una matriz cuadrada diagonal en que todos sus elementos a lo largo de la diagonal tienen un valor unitario (1).

Matriz transpuesta.- La matriz de tamaño n x m que se obtiene al intercambiar las filas y columnas de una matriz m x n que recibe el nombre de transpuesta de A y se representa por A^T.

Matriz simétrica.- Es una matriz cuadrada para la que $A^T = A$.

Matriz nula.- Todos los elementos de la matriz son iguales a cero.

Matriz de permutación.- Es una matriz que tiene precisamente un elemento cuyo valor es uno en cada columna y en cada renglón y todos sus demás elementos son cero.

Matriz triangular superior.- es una matriz n x n que tiene para cada j, los elementos $u_{ij}=0$ para cada $i=j+1, j+2,\ldots,n$

Matriz triangular inferior.- es aquella que tiene para cada j, los elementos Uij=0 para cada $i=1,2,\ldots,j-1$

Las matrices en forma triangular son llamadas también matrices reducidas.

Matriz no singular.- se dice que una matriz A de n x n es no singlar si existe una matriz A^{-1} de n x n tal que $AA^{-1} = A^{-1}A=1$.

Matriz singular.- Una matriz que no tiene inversa.

1.2.1. Método Gauss-Jordan

Muchos problemas relacionados no solo con la ingeniería se pueden expresar en términos de sistemas de ecuaciones algebraicas lineales, los cuales son un conjunto de ecuaciones de la forma:

$$a_{1,1}x_1 + a_{1,2}x_2 + a_{1,3}x_3 + \dots + a_{1,n}x_n = y_1$$
$$a_{2,1}x_1 + a_{2,2}x_2 + a_{2,3}x_3 + \dots + a_{2,n}x_n = y_2$$

$$a_{n,1}x_1 + a_{n,2}x_2 + a_{n,3}x_3 + \dots + a_{n,n}x_n = y_n$$

O bien de la forma matricial

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,n} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,n} \\ & & & & \\ a_{n,1} & a_{n,2} & a_{n,3} & \dots & a_{n,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \\ x_n \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ \\ b_n \end{bmatrix}$$

Donde los $a_{i,j}$ son coeficientes, los x_i son las incógnitas y los y_i son los términos independientes.

Un método útil y sencillo para encontrar la solución de un sistema de ecuaciones algebraicas lineales es el de eliminación completa; consiste en obtener sistemas equivalentes a partir del sistema original dado, utilizando las operaciones elementales sobre los renglones de la matriz que son:

- Intercambiar un renglón por otro; esto equivale a reordenar las ecuaciones del sistema.
- Multiplicar los elementos de un renglón por un escalar diferente de cero; operación que es equivalente a multiplicar una ecuación por una constante
- Sumar los elementos correspondientes de dos renglones, que es equivalente a sumar término a término las ecuaciones del sistema.

El método de Gauss-Jordan consiste en sistematizar la obtención de sistemas equivalentes hasta obtener uno en el cual la matriz del sistema se convierta a la matriz identidad [I]; a partir de este último se podrá observar su solución directamente.

$$\begin{bmatrix} 1 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ & & & & \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \\ x_n \end{bmatrix}$$

Los pasos a seguir para la obtención de sistemas equivalentes son:

- Seleccionar un elemento diferente de cero como elemento pivote, éste debe ser algún elemento de la matriz de coeficientes.
- Convertir en uno el elemento pivote mediante operaciones elementales.
- Convertir en ceros todos los elementos bajo de la columna donde se encuentra el elemento pivote.
- Seleccionar un nuevo elemento pivote, el cual no debe estar en la misma columna o renglón donde se encontraba(n) el (los) elemento(s) pivote anterior(es).
- Repetir los pasos anteriores hasta obtener una matriz de coeficientes formada solamente con ceros y unos; en caso contrario intercambiar renglones para obtener la matriz identidad. En el sistema de ecuaciones correspondiente a esta última matriz se observará la solución.

¿Porque usar un pivoteo?... La eliminación de Gauss se aplica a un problema ideal sencillo con coeficientes no nulos. Sin embargo, el método no funciona si el primer coeficiente del primer renglón es igual a cero o si un coeficiente de la diagonal se anula en el proceso de solución, ya que se usan como denominadores en la eliminación hacia adelante[4].

El pivoteo se usa para cambiar el orden secuencial de las ecuaciones con dos propósitos:

1) para evitar que los coeficientes de la diagonal se anulen
2) para hacer que cada coeficiente de la diagonal tenga magnitud mayor que cualquiera de los coeficientes por debajo de él.

---

[4] Los coeficientes de la diagonal –o pivotes- son los coeficientes de x_1 en al primera ecuación, el de x_2 en la segunda y el de x_3 en la tercera ecuación. En general, el coeficiente de x_n en la n-ésima ecuación es el coeficiente en la diagonal.

Las ecuaciones no se afectan matemáticamente por cambios en el orden secuencial, pero el cambio de orden hace posible el cálculo cuando el coeficiente de la diagonal se anula. Aún cuando los coeficientes de la diagonal fueran nulos, los cambios de orden aumentan la exactitud de los cálculos.

Ejemplo:

Para explicar el método de Gauss-Jordan con pivoteo, se considera el arreglo:

$$\begin{bmatrix} 0 & 10 & 1 \\ 1 & 3 & -1 \\ 2 & 4 & 1 \end{bmatrix} = \begin{bmatrix} 2 \\ 6 \\ 5 \end{bmatrix}$$

No se puede eliminar el primer número de los renglones segundo y tercero debido a que el primer número del primer renglón es igual a cero. En nuestro primer pivoteo, se intercambian el primer y último renglones, en base al coeficiente de mayor valor absoluto en la primera columna. El hecho de llevar el número más grande de la columna a la posición diagonal tiene la ventaja de reducir el error de redondeo

$$\begin{bmatrix} 2 & 4 & 1 \\ 1 & 3 & -1 \\ 0 & 10 & 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 6 \\ 2 \end{bmatrix}$$

$[R_2 - {}^1/_2 R_1]$

$$\begin{bmatrix} 2 & 4 & 1 \\ 0 & 1 & -3/2 \\ 0 & 10 & 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 7/2 \\ 2 \end{bmatrix}$$

Lo mismo se hace con el primer número del tercer renglón (aunque en este caso ya sea cero), restando a éste el primero multiplicado por 0/2:

$$\begin{bmatrix} 2 & 4 & 1 \\ 0 & 1 & -3/2 \\ 0 & 10 & 1 \end{bmatrix} = \begin{bmatrix} 5 \\ 7/2 \\ 2 \end{bmatrix}$$

En general, como ya se ha mencionado, no es recomendable eliminar un número mayor en magnitud que el término de la diagonal. Por lo tanto, se intercambia el orden de los renglones segundo y tercero:

$$\begin{bmatrix} 2 & 4 & 1 \\ 0 & 10 & 1 \\ 0 & 1 & -3/2 \end{bmatrix} = \begin{bmatrix} 5 \\ 2 \\ 7/2 \end{bmatrix}$$

Después se elimina el segundo número del tercer renglón, con lo que el arreglo se transforma en:

$$\begin{bmatrix} 2 & 4 & 1 \\ 0 & 1 & -3/2 \\ 0 & 0 & -8/5 \end{bmatrix} = \begin{bmatrix} 5 \\ 7/2 \\ 33/10 \end{bmatrix}$$

Los datos 2, 1 y –8/5 se llaman coeficientes de la diagonal o pivotes. Y mediante sustituciones hacia atrás o con eliminaciones se produce:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2.71875 \\ 0.40625 \\ -2.06255 \end{bmatrix}$$

Si a pesar del pivoteo, es inevitable un elemento nulo en la diagonal, esto indica que el problema es de los que no tienen solución única. Si así ocurre, se detiene el esfuerzo del cálculo.

Problemas sin solución única

No siempre es posible resolver un conjunto de ecuaciones lineales en forma numérica. Los siguientes conjuntos de ecuaciones lineales son tres ejemplos sencillos pero importantes:

a) $-x + y = 1$

$-2x + 2y = 2$

b) $-x + y = 1$

$-x + y = 0$

c) $-x + y = 1$

$x + 2y = -2$

$2x - y = 0$

En el conjunto a), la segunda ecuación es el doble de la primera ecuación, por lo que matemáticamente son idénticas. Cualquier punto (x,y) que satisfaga una de las ecuaciones, es también solución de la segunda. Por lo tato, el número de soluciones es infinito. En otras palabras, no existe una solución única.

Si una ecuación es múltiplo de otra, o se puede obtener sumando o restando otras ecuaciones, se dice que esa ecuación es *linealmente dependiente* de la(s) otra(s). Si ninguna de las ecuaciones es linealmente dependiente, se dice que todas las ecuaciones son *linealmente independientes*.

En el conjunto b), las dos ecuaciones son rectas paralelas que no se intersectan, por lo que no existe solución. Tal sistema recibe el nombre de sistema inconsistente. Un conjunto de ecuaciones es inconsistente si el lado izquierdo de al menos una de las ecuaciones se puede eliminar totalmente (sumando o restando otras ecuaciones), mientras que el lado derecho permanece distinto de cero.

En el tercer conjunto, existen tres ecuaciones independientes con dos incógnitas, las cuales no se pueden satisfacer simultáneamente.

Un caso como el de c) no puede ocurrir si el número de ecuaciones es igual al número de incógnitas. En este caso, sólo puede ocurrir la falta de independencia de independencia lineal –como en a)- o la inconsistencia, como en b).

Si el número de ecuaciones es mayor de dos, la carencia de independencia lineal o la inconsistencia son menos obvias. Sin embargo, un programa que ejecute la eliminación de Gauss y que intente llevarla a cabo en uno de tales conjuntos se detiene a mitad de los cálculos debido a un error aritmético, como un desbordamiento o una división entre cero.

De hecho si un conjunto de ecuaciones es inconsistente o linealmente dependiente, uno de los renglones de coeficientes en el arreglo (sin incluir el último número correspondiente al término del lado derecho) se anula durante la eliminación hacia delante. Es decir "*la matriz es singular*"

Comparación de tres métodos para la resolución de sistemas de ecuaciones lineales.

Método	*Ventajas*	*Desventajas*
Eliminación de Gauss	El algoritmo de solución más básico	Solución de un único conjunto de ecuaciones lineales a la vez.
Eliminación de Gauss-Jordan	La base para calcular la inversa; puede resolver conjuntos múltiples de ecuaciones.	Menos eficiente para un único conjunto de ecuaciones.
Descomposición LU	Eficaz si un conjunto de ecuaciones lineales se resuelve varias veces con distintos términos no homogéneos.	Menos eficiente y más complicado que la eliminación de Gauss si sólo se usa una sola vez.

1.3. Ecuaciones Diferenciales

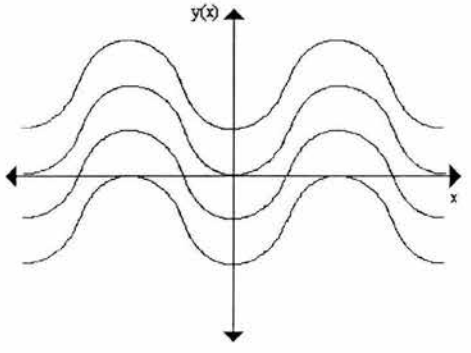
Se entiende por ecuación diferencial aquella que relaciona dos o más variables en términos de derivadas o diferenciales. Para su estudio conviene establecer las siguientes definiciones:

- Ecuación diferencial ordinaria. Es aquella en la que existe solamente una variable independiente; por lo tanto sus derivadas serán totales.
- Ecuación diferencial parcial. Es aquella en la que existen dos o más variables independientes, por lo que sus derivadas serán parciales.
- Orden de una ecuación diferencial. Es la derivada de mayor orden que aparece en la ecuación.
- Grado de una ecuación diferencial. Es el grado algebraico de la derivada de mayor orden que se encuentra en la ecuación.
- Ecuación diferencial lineal. Es una ecuación diferencial en la que no aparecen potencias de la variable dependiente por sus derivadas o productos entre derivadas.

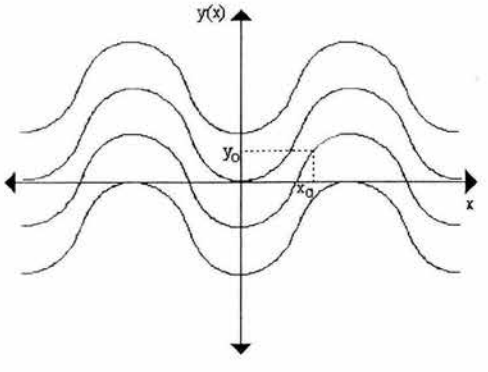
La solución de una ecuación diferencial, es la función que satisface tanto a la ecuación diferencial como a ciertas condiciones iniciales de la función. Al resolver analíticamente una función diferencial de ordinario se encuentra una solución general

que contiene constantes arbitrarias, las cuales se determinan de manera que la expresión concuerde con las condiciones iniciales. Para una ecuación de *n*-ésimo orden, se deben conocer *n* condiciones iniciales independientes.

La solución de una ecuación diferencial es una función que no contiene derivadas o integrales de funciones, y que verifica idénticamente la ecuación diferencial. Por ejemplo, la solución de la ecuación diferencial $y'(x) - \text{sen}(x) = 0$ es $y(x) = -\cos(x) + c$ ya que esta función verifica idénticamente la ecuación diferencial. Nótese que la solución resulta ser una familia de curvas.



Para determinar una solución particular, será necesaria una condición inicial del problema. Con lo cual será posible determinar el valor de c que corresponde a ese caso particular, y con ello seleccionar una sola curva que sea solución de la ecuación diferencial dada.

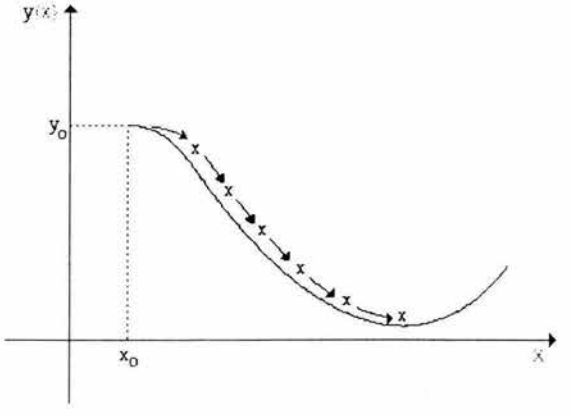


Cuando todas las condiciones están dadas en un valor de x y la solución procede a través de ese valor de x, se tiene un problema con **valor inicial**. Cuando las condiciones se proporcionan a diferentes valores de x, se tiene un problema con **valores a la frontera**.

En los métodos numéricos se obtiene la solución como una tabulación de valores de la función en los diversos valores de la variable independiente, y no como una relación funcional. De forma que se debe recalcular la tabla entera si se modifican las condiciones iniciales.

1.3.1. Método de Euler

Es uno de los métodos llamados de solución paso a paso[5], ya que a partir de uno o varios puntos conocidos se calcula el siguiente; una vez calculado, se apoyan en éste y en los anteriores para calcular uno más, y así sucesivamente.



Consiste en integrar la ecuación diferencial $y'(x) = f(x,y)$ entre un punto x_i y el siguiente; esto es

[5] Métodos Numéricos. Iriarte R. y Balderrama V., Ed. Trillas, México, 2ª reimpresión, 1999. p. 153.

$$\int_{x_i}^{x_{i+1}} y'(x)dx = \int_{x_i}^{x_{i+1}} f(x_i, y_i)dx \qquad (1)$$

Al integrar el miembro de la izquierda se obtiene:

$$y(x)\Big|_{x_i}^{x_{i+1}} = \int_{x_i}^{x_{i+1}} f(x_i, y_i)dx$$

$$y_{i+1} - y_i = \int_{x_i}^{x_{i+1}} f(x_i, y_i)dx$$

Sustituyendo el miembro de la derecha con una expresión de integración numérica generada a partir de la fórmula de interpolación de Newton[6]

$$\int_{x_i}^{x_{i+1}} f(x_i, y_i)dx = h\left[f(x_i, y_i) + \frac{1}{2}\left[f(x_{i+1}, y_{i+1}) - f(x_i, y_i)\right] + \ldots \right]$$

El método de Euler consiste en integrar el segundo miembro de la expresión (1), considerando únicamente el primer sumando de la expresión anterior con su correspondiente error:

$$\int_{x_i}^{x_{i+1}} f(x_i, y_i)dx = hf(x_i, y_i) + e_r$$

Si se desprecia el error y se sustituye esta última ecuación en la expresión (1), se obtiene:

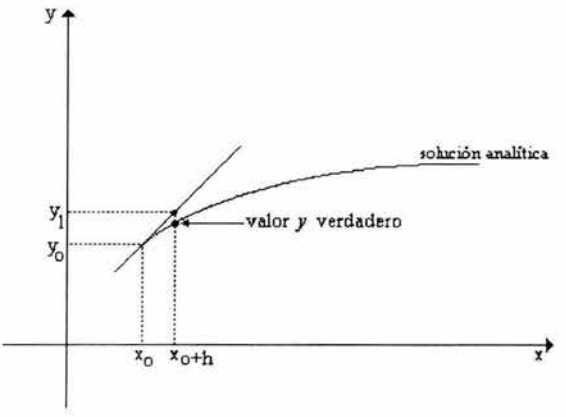
$$y_{i+1} - y_i = hf(x_i, y_i)$$

y al despejar y_{i+1}

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

con i = 0, 1, 2, 3...

El método usa la tangente al principio del intervalo y'_n para determinar el incremento de la función. Esto siempre es incorrecto, después de todo si la pendiente de la función fuera constante, la solución sería una relación lineal. Se necesita utilizar una pendiente promedio sobre el intervalo, si se espera estimar los cambios en *y* con precisión.



1.3.2. Método de Euler modificado.

El utilizar la media aritmética de las pendientes, al principio y al final del intervalo dará con seguridad una mejor estimación para *y* en x_{i+1}.

$$y_{i+1} = y_i + h\frac{y'_i + y'_{i+1}}{2}$$

Sin embargo, no es posible utilizar la ecuación anterior inmediatamente, debido a que la derivada es una función tanto de *x* como de *y*, y no se puede evaluar y'_{i+1} si se desconoce y_{i+1}. El método de Euler modificado supera la dificultad "prediciendo" un valor de y_{i+1} por la relación simple de Euler y utiliza este valor para calcular y'_{i+1} dando una mejor estimación (valor "corregido") para y_{i+1}.

Puesto que el valor de y'_{i+1} se calculó utilizando el valor pronosticado de menor precisión, uno está tentado a corregir el valor y_{i+1} tantas veces como para que la diferencia sea significativa. (Si se requieren más de dos o tres correcciones, resulta más eficiente reducir el tamaño de paso).

A este método también se le conoce como método pronosticador-corrector de Euler.

---

[6] El desarrollo completo se puede consultar en: Métodos Numéricos. Iriarte R. y Balderrama V., Ed. Trillas, México, 2ª reimpresión, 1999. p. 127

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

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

En donde el subíndice "p" indicará predicción y "c" corrección.

1.3.3. Método de Runge Kutta

Se llama así en honor de los dos matemáticos alemanes que lo desarrollaron hace aproximadamente 100 años. Este método hace un promedio ponderado de las pendientes dadas por la función f(x,y) que define la ecuación diferencial.

Las cuatro pendientes se determinan como sigue:

1.- La primera pendiente S se calcula igual que en el método de Euler; es decir $S = f(X_0, Y_0)$

2.- En el método de Euler mejorado usamos m_i para producir una segunda pendiente correspondiente a un punto con $t = t_i + 1 = t_i + h$. El método de Runge Kutta hace casi lo mismo excepto que va sólo a la mitad del camino a lo largo del eje X hasta $X_a = t_i + h/2$, esto significa que usamos S para producir un punto Xa, Ya, donde $Y_a = Y_i + S * h/2$.

Una vez que hemos determinado este punto, usamos la función f(x,y) para determinar la segunda pendiente N por medio de $N = f(X_a, Y_a)$

3.- Se calcula un punto en $X = X0 + h/2$ a lo largo de una línea con pendiente N, con lo que se obtiene $Y_b = Y_i + N * (h/2)$; se calcula una tercera pendiente $Q = f(X_a, Y_b)$.

4.- Finalmente obtenemos un cuarto punto sobre la linea X=Xi+1 usando Q.

$Y_c = Y_i + Q * h$

Después se calcula una cuarta pendiente en este punto

$P = f(X_i+1, Y_c)$

5.- Ahora que tenemos las cuatro pendientes, tomamos un promedio ponderado, en donde ponderamos al doble cada una de las pendientes que provienen de los puntos $X = X_a$, respecto a las otras dos pendientes.

$M = (S + 2*N + 2*Q + P) / 6$

y con éste calculamos el siguiente paso.

$Y_{i+1} = Y_i + M * h$

Capítulo 2.- Algoritmos de los métodos seleccionados

El que este trabajo se limite a mostrar unos cuantos algoritmos no quiere decir que el análisis numérico se limite a estos métodos. Existe una amplia bibliografía que nos permite tener acceso a los algoritmos requeridos (Burden, Carnahan, etc.)

Por ejemplo para resolver ecuaciones algebraicas no lineales se dispone de los métodos de iteración simples, de búsqueda incremental y de punto fijo, los métodos con acotamiento de bisección y de la falsa posición, así como los métodos de Newton-Raphson y de la secante basados en cálculo diferencial. La eliminación de Gauss, el método de Gauss-Jordan, el método de Jacobi, el método de Gauss-Seidel y la descomposición LU son métodos para resolver sistemas generales de ecuaciones algebraicas lineales.

Entre las herramientas para efectuar el análisis de datos se encuentran la estadística y métodos para ajustar curvas a los datos medidos (estadística elemental, método de mínimos cuadrados, interpolación polinómica, interpolación de Lagrange, algoritmo de Neville, etc.).

La diferenciación y la integración son dos procesos fundamentales del cálculo, por ello, aunque en principio, siempre es posible determinar una forma analítica de una derivada para una función dada, en algunos casos la forma analítica es demasiado complicada, y para el objetivo buscado puede ser suficiente una aproximación numérica, ello ha dado lugar al desarrollo de una gran cantidad de algoritmos, ya sea a través del ajuste de curvas o directamente en términos de los valores de la función (método de Euler, Runge-Kutta, regla del trapecio, regla de Simpson, integración de Romberg, cuadratura de Gauss, etc.).

Como se puede notar el arsenal es basto y tan solo es requisito delimitar el problema con el fin de buscar el método adecuado y analizar sus ventajas y desventajas antes de aplicar alguno de ellos.

Una sátira publicada en 1939 es muy buena para expresar el objeto de este trabajo, nos hace retroceder a la época en que se empezaba a formar a los jóvenes en el arte de capturar peces, cazar caballos lanudos a garrotazos y asustar con fuego a los tigres de dientes de sable. La cuestión era: ¿qué ocurriría

con estas venerables materias cuando alguien inventara la caña de pescar, los caballos lanudos se trasladaran a terrenos más altos y fueran reemplazados por antílopes, más veloces, y los tigres se murieran y ocuparan su lugar unos cuantos osos? ¿No se les debería jubilar o sustituir por estudios más pertinentes?

Los sabios ancianos dijeron: "No enseñamos a capturar peces con el fin de capturar peces; lo enseñamos para desarrollar una agilidad general que nunca se podrá obtener con una mera instrucción. No enseñamos a cazar caballos a garrotazos para cazar caballos; lo enseñamos para desarrollar una fuerza general en el aprendiz que nunca podrá obtener de una cosa tan prosaica y especializada como cazar antílopes con red. No enseñamos a asustar tigres con el fin de asustar tigres; lo enseñamos con el propósito de dar ese noble coraje que se aplica a todos los niveles de la vida y que nunca podría originarse en una actividad tan básica como matar osos"

Aún así hubo quien dijo: Pero con todo deberéis admitir que los tiempos han cambiado. ¿No podrías dignaros a probar estas otras actividades más modernas?

La respuesta fue: "La esencia de la verdadera educación es la intemporalidad. Es algo que permanece a través de las condiciones cambiantes como una roca firmemente plantada en medio de un tumultuoso torrente".

La idea es que no se trata de desarrollar el conocimiento de los métodos numéricos o de Visual Basic, sino de intentar crear el hábito de conocer la materia antes de querer programar, y el de ser una guía para desplegar la habilidad de mezclar las diferentes instrucciones, del lenguaje que se desee, y lograr que produzcan el resultado deseado.

2.1. Ecuaciones no lineales

2.1.1. Método de bisección.

Paso 1: Se eligen los valores limitantes X_a y X_b (con $X_b > X_a$).

Paso 2: Se calculan $f(X_a)$ y $f(X_b)$

Paso 3: Se calcula el punto medio del intervalo $x_m = (X_a \text{ y } X_b) / 2$ y $f(X_m)$

Paso 4: Se usa i o ii, dependiendo de:

i) Si $f(X_a) * f(X_m) > 0$, recolocar X_m en x_a

ii) $f(X_m) * f(X_b) > 0$, recolocar X_m en x_b

Paso 5: Si $|x_a - x_b| > \varepsilon$, se vuelve al paso 2; en caso contrario X_m es la raíz aproximada

2.1.2. Método de punto fijo

Paso 1: Se conjetura un valor inicial X_0, se elige un parámetro de convergencia ε y un número límite de iteraciones N.

Paso 2: Se calcula un valor mejorado a partir de $x_{mejorado} = f'(x_0)$, en donde $f'(x_0)$ es la primera derivada de $f(x_0)$

Paso 3: Si $|x_{mejorado} - x_0| > \varepsilon$, x_0 se iguala a $x_{mejorado}$ y se vuelve al paso 2; en caso contrario $x_{mejorado}$ es la raíz aproximada

2.1.3. Método Regula Falsi

Paso 1: Se eligen valores limitantes x_a y x_b (con $x_b > x_a$)

Paso 2: Se calcula $f_a=f(x_a)$ o $f_b=f(x_b)$, y un contador i se coloca en cero.

Paso 3: i se incrementa en 1 y se calcula el punto intermedio $x_{intermedio}$ a partir de una de las dos expresiones

$x_{intermedio}=x_a - (x_b - x_a)\, f(x_a) / [f(x_b) - f(x_a)]$ o bien,

$x_{intermedio}=x_b - (x_b - x_a)\, f(x_b) / [f(x_b) - f(x_a)]$

Paso 4: Se calcula $f_{intermedio} = f(x_{intermedio})$

Paso 5: Dependiendo de si f_a o f_b está disponible por el paso 2, se usa *i* o *ii*

i) Si $(f_a\, f_{intermedio}) > 0$, x_a se recoloca en $x_{intermedio}$; en caso contrario, x_b se recoloca en $x_{intermedio}$

ii) Si $(f_b\, f_{intermedio}) > 0$, x_b se recoloca en $x_{intermedio}$; en caso contrario, x_a se recoloca en $x_{intermedio}$.

Paso 6: Si $|f(x_{intermedio})|$ es suficientemente pequeño, es decir, menor o igual que alguna cantidad pequeña prescrita $\in$, o si *f* alcanza un límite de iteración *N*, $x_{intermedio}$ se considera como la raíz aproximada; en caso contrario, volver al paso 3.

2.1.4. Método de Newton-Raphson

Paso 1: Se elige un valor inicial X_0, se elige un parámetro de convergencia ε y un número límite de iteraciones N.

Paso 2: Se calculan $f(x_0)$ y $f'(x_0)$, en donde $f'(x_0)$ es la primera derivada de $f(x_0)$

Paso 3: Se calcula $x_{mejorado} = x_0 - f(x_0) / f'(x_0)$

Paso 3: Si $|x_{mejorado} - x_0| > \varepsilon$, x_0 se iguala a $x_{mejorado}$ y se vuelve al paso 2; en caso contrario $x_{mejorado}$ es la raíz aproximada

2.1.5. Método de la secante

Paso 1: Se eligen dos valores iniciales X_0 y X_1, se elige un parámetro de convergencia ε y un número límite de iteraciones N.

Paso 2: Se calculan $f(x_1)$ y $f(x_0)$.

Paso 3: Si $|f(x_1)| \leq \varepsilon$, entonces x_1 es la solución estimada, en caso contrario se procede al paso 4.

Paso 4: Se aplica la interpolación lineal para calcular $x_{intermedio} = x_1 - (x_1-x_0)*f(x_0) /[f(x_1)-f(x_0)]$

Paso 5: x_0 se iguala a x_1 y x_1 se iguala a $x_{intermedio}$ y se regresa al paso 2.

2.2 Sistemas de ecuaciones lineales

2.2.1 Método de Gauss-Jordan con pivoteo

Paso 1: Seleccionar un elemento diferente de cero como pivote; este debe ser algún elemento de la matriz de coeficientes.

Paso 2: Convertir en uno el elemento pivote mediante operaciones elementales.

Paso 3: Convertir en ceros todos los elementos de la columna donde se encuentra el elemento pivote.

Paso 4: Seleccionar un nuevo pivote, el cual no debe estar ni en el renglón ni en la columna donde se encontraba(n) el (los) pivote(s) anterior(es).

Paso 5: Repetir los pasos anteriores hasta obtener una matriz de coeficientes formada solamente por unos y ceros; en caso necesario intercambiar renglones para obtener la matriz identidad. En el sistema de ecuaciones correspondiente a esta última matriz se observara la solución.

2.3. Ecuaciones Diferenciales

2.3.1. Método de Euler

Dada la condición inicial $Y(X_0) = Y_0$ y el tamaño de paso h, calcule el punto (X_{i+1}, Y_{i+1}) a partir de (X_i, Y_i);

Paso 1: Use la ecuación diferencial para determinar la pendiente $m_i = f(X_i, Y_i)$.

Paso 2: Calcule el valor Y_{i+1} que resulta de la aplicación de $Y_{i+1} = Y_i + h * f(X_i, Y_i)$.

2.3.2. Método de Euler modificado

Dada la condición inicial $Y(X_0) = Y_0$ y el tamaño de paso h, calcule el punto (X_{i+1}, Y_{i+1}) a partir de (X_i, Y_i);

Paso 1: Use la ecuación diferencial para determinar la pendiente $m_i = f(X_i, Y_i)$.

Paso 2: Calcule el valor Y_{i+1} que resulta de la aplicación del método de Euler. Es decir, $Y_{i+1} = Y_i + h * f(X_i, Y_i)$.

Paso 3: Calcule $X_{i+1} = X_i + h$ y use la ecuación diferencial para calcular la pendiente $n_i = f(X_{i+1}, Y_{i+1})$ en el punto (X_{i+1}, Y_{i+1}).

Paso 4: Calcule Y_{i+1} por medio de $Y_{i+1} = Y_i + [(m_i + n_i) / 2] * h$.

2.3.3. Método de Runge Kutta

Paso 1: Use la ecuación diferencial para determinar la pendiente $S = f(X_i, Y_i)$.

Paso 2: Calcule el valor Y_a. Es decir, $Y_a = Y_i + (h/2) * S$.

Paso 3: $X_a = X_i + (h/2)$

Paso 4: $N = f(X_a, Y_a)$

Paso 5: $Y_b = Y_i + N * (h/2)$

Paso 6: $Q = f(X_a, Y_b)$

Paso 7: $Y_c = Y_i + Q * h$

Paso 8: $P = f(X_i + 1, Y_c)$

Paso 9: Calcule la pendiente ponderada $M = (S + 2*N + 2*Q + P)/6$.

Paso 10: Calcule Y_{i+1} por medio de $Y_{i+1} = Y_i + M * h$.

2.4. Codificación de los algoritmos

Para llevar a cabo la codificación en Basic o en cualquier otro lenguaje, hay que tener presente que antes de acometer la tarea, hay que tener una buena comprensión de el método que se desea programar, para que toda la concentración y las metas a lograr estén enfocadas a buscar la instrucción que nos facilite la tarea y evitar terminar perdidos en una serie de instrucciones que no se sabe ni para que se están poniendo.

Lo primero a realizar es el diseño de la interfase con la cual el usuario tendrá que interaccionar para trabajar con el programa, en este caso siendo Visual Basic un ambiente gráfico de desarrollo, está parte resulta muy sencilla.

2.4.1. Interfase gráfica

Partiendo del hecho que este sería nuestro primer programa, no se realizará un diseño muy complicado ni con mucho detalle; es decir que se elaborará un programa tan solo funcional que permitirá emplearlo para la solución, pero estando concientes de que se tendrá que modificarlo mediante programación, cuando sea necesario.

Al iniciar Visual Basic, el programa nos presenta una forma en blanco al iniciar un nuevo proyecto, en la cuál se irán añadiendo elementos (objetos) que servirán para lograr el objetivo. Normalmente al comenzar a utilizar Visual Basic, el programa nos presentará en pantalla las ventanas de menú, de herramientas, el explorador de proyecto, la ventana de propiedades y la de posición del formulario, en caso de que no fuese así, estas ventanas se pueden abrir en el menú VER.

De inicio se modificaran las propiedades de la form:

Nombre de la Propiedad	Que hace	Cambio que se hará
Caption	Muestra el dato en la barra de título	Solución de ecuaciones no lineales
Name	Es el nombre de la forma	ecnolineal
Icon	Se selecciona el icono que aparecerá en la parte izquierda de la barra de titulo.	En este caso se tendría que elaborar un icono o seleccionar uno disponible

Al modificar el tamaño de la forma, las propiedades width y Height se ajustarían; este ajuste se realiza tan solo arrastrando los lados o las esquinas de la forma. De esta manera es muy fácil el adecuar el tamaño de la forma para nuestros fines.

Una vez modificadas las propiedades de la forma llamada *ecnolineal*, es necesario el disponer de las ecuaciones no lineales sobre las que funcionarán los métodos. Como el programar para que se reconozcan caracteres y números (ya no hablar de funciones) es un procedimiento muy complicado y queda fuera de los límites de este trabajo (además de que se planteo que sería un programa de aprendizaje inicial); se recurrirá a fijar algunas ecuaciones no lineales que podrían ser modificadas por el programador.

Para ello, se tendrá que elegir la herramienta *frame* (marco)  y colocar uno de estos objetos ampliándolo lo suficiente para después colocar en su interior las funciones que interesan mediante la herramienta *option button* (botón de opción)  colocando unos cuantos en *ecnolineal* (10 en este caso).

Se selecciona uno de estos botones y se modifican sus propiedades (se sugiere comenzar con el primero)

Nombre de la Propiedad	Que hace	Cambio que se hace
Name	Asigna un nombre a el objeto	Option1
Caption	Muestra una cadena de caracteres a el lado derecho del botón de opción	La fórmula correspondiente a cada opción
Tabindex	Es el numero de índice en la tabulación, es decir que cuando se presione la tecla TABULADOR (dos flechas en sentidos opuestos), ira pasando de un elemento a otro en este orden	Colocar números del 0 en adelante según se crea conveniente

Al seleccionar otro botón y asignarle el mismo nombre, Visual Basic (VB), nos preguntará si se desea generar una matriz para estos elementos; a lo que se responderá afirmativamente, ya que de esta manera solo será posible seleccionar una y solo una ecuación (de manera que se limitará al usuario y se evitará generar código de programación para evitar fallas en la lógica del programa).

Aquí se puede también tomar la opción de no colocar la fórmula de la ecuación a resolver dejando la propiedad *caption* en blanco e insertando la fórmula en forma

de dibujo apoyándose para ello de algún programa de dibujo (Paint por ejemplo); en este programa se emplearán ambas.

Nótese que VB le asignara un número entre paréntesis a cada botón, con lo cuál se identifica que opción se seleccionó. Ejemplo: el botón de opción 1 se llamará option1(0), el segundo option1(1), etc.

Siendo en este caso, los métodos de Bisección, Punto Fijo, Newton-Raphson y de la Secante muy semejantes, y porque los tres se emplean para resolver ecuaciones no lineales, se agruparan en la misma sección, es decir, que existirá una sola pantalla para la captura de los datos de inicio. Asimismo esto será de utilidad para comparar los resultados que arroja cada método.

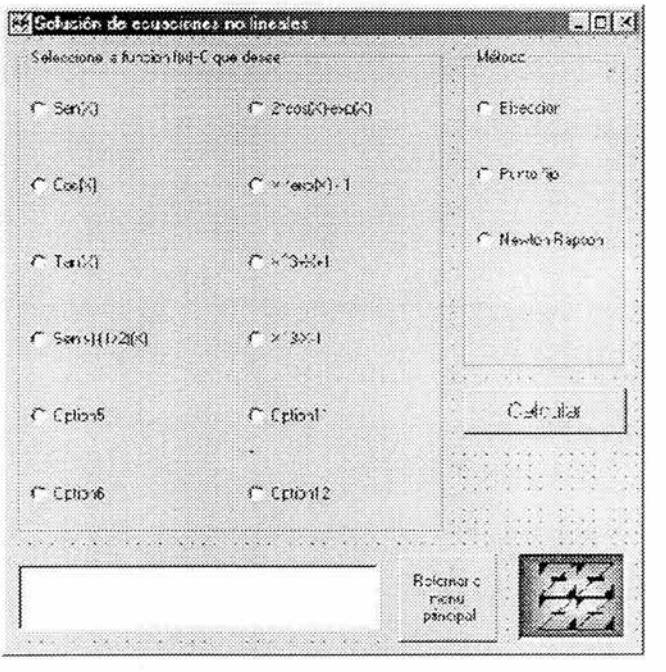
De la misma manera que con las ecuaciones, se procederá con los métodos. Primero se introduce un marco (frame), luego cuatro botones de opción los cuales tendrán por nombre option2 {en realidad option2(0), option2(1), option2(2), option2(3) y option2(4)}. Con lo que se limita que solo se seleccione un método a la vez.

Como se desea que el usuario sepa que es lo que esta seleccionando en cada caso, se necesita modificar la propiedad *caption* de los *option button*, para que muestre las fórmulas o los nombres de cada método según sea el caso, esto se logra dando clic en el botón deseado y modificar la propiedad *caption* del mismo. Cuando sea seleccionado por el usuario será referenciado por el número de botón, no por su contenido (el mismo que le fue asignado entre paréntesis cuando se genero la matriz de botones de opción).

Por último será necesario introducir un *text box* (cuadro de texto)  y un *command button* (botón de comando)  ; en el primero nos será mostrado el resultado o salida de datos y el segundo es con el fin de poner en ejecución el programa una vez que se seleccionó la ecuación y el método. Al *text box* le será modificada la propiedad *enabled* a false (de esta manera no se podrá introducir texto) y se eliminará el contenido de la propiedad *caption* para que no muestre nada. Al *command button* se le cambia la propiedad *caption* para que diga **calcular**.

Para mejorar la eficiencia en la selección de estos, modificaremos las propiedades de los marcos, cambiando la propiedad *caption* en el primero por "Seleccione la funcion f(x)=0 que desee" y por "Metodo" en el segundo. Téngase la precaución de no introducir acentos, porque debido a incompatibilidades en la configuración de idioma en las diferentes computadoras en que podría ser usado el programa, el uso de acentos podría provocar que no se mostrase bien el texto (aparecerían en ocasiones caracteres no deseados).

Para mejorar visualmente la forma, se le añade una imagen empleando la herramienta *image*  (en este caso el logotipo de FES Zaragoza). Asimismo, pensando en que está forma es parte de un sistema, se le agrega un botón de comando para regresar a la pantalla de inicio (o menú principal). Por el momento este botón no producirá ningún efecto al presionarlo.



En este punto nuestra forma *ecnolineal* debería parecerse a la siguiente.

2.4.2. Inserción del Código de programación

Hasta ahora solo se ha generado la interfaz gráfica con la que trabajará el usuario, la cual hasta el momento no tiene ninguna utilidad por si misma, es necesario

introducir el código de programación que le dirá a la computadora que es lo que debe hacer.

Para determinar que método fue seleccionado, se utiliza la función CASE de Visual Basic, la que permitirá asignar un valor a una variable dependiendo de la opción seleccionada, aquí hay que tener en cuenta que el primer botón de opción es el case 0, el siguiente es 1, etc.

Dando doble clic en alguno de los botones de opción (que corresponderían a los métodos) que están dentro de uno de los dos frame, se teclea el siguiente código, el cual asignara un valor a la variable "d"

```
Select Case Index
  Case 0
    d = 0
  Case 1
    d = 1
  Case 2
    d = 2
  Case 3
    d = 3
End Select
```

Después empleando la ventana de proyecto, se selecciona la forma ecnolineal, para repetir el proceso con uno de los botones que representarán las funciones (nótese que ahora se emplea una segunda variable "e", lo que permitirá que solo se pueda usar una función y un método a la vez)

```
Select Case Index
  Case 0
    e = 0
  Case 1
    e = 1
  Case 2
    e = 2
  Case 3
    e = 3
  Case 4
    e = 4
  Case 5
    e = 5
  Case 6
    e = 6
  Case 7
```

```
    e = 7
  Case 8
    e = 8
  Case 9
    e = 9
  End Select
```

Después, para el caso del método de bisección, se teclea el siguiente código, el cual será insertado dando doble clic en el botón de comando que se denomina calcular

```
'metodo de bisección
If d = 0 Then
  li# = InputBox("limite inferior del intervalo", "Datos iniciales", , 1, 1)
  ls# = InputBox("limite superior del intervalo", "Datos iniciales", , 1, 1)
  ni% = InputBox("numero maximo de iteraciones", "Datos iniciales", , 1, 1)
  td# = InputBox("tolerancia deseada", "Datos iniciales", , 1, 1)
  For z = 1 To ni
    X2# = (li + ls) / 2
      Case 0
        fli = Sin(li): fx2 = Sin(X2)
      Case 1
        fli = Sin(li) + 1 - (li) ^ 2: fx2 = Sin(X2) + 1 - (X2) ^ 2
      Case 2
        fli = Cos(li) - (li) ^ 2: fx2 = Cos(X2) - (X2) ^ 2
      Case 3
        fli = Sin(li) - (li / 2): fx2 = Sin(X2) - (X2 / 2)
      Case 4
        fli = ((li ^ 2) + 3) / ((li ^ 2) - 4): fx2 = (((X2) ^ 2) + 3) / (((X2) ^ 2) - 4)
      Case 5
        fli = 4 * li - Tan(li): fx2 = 4 * X2 - Tan(X2)
      Case 6
        fli = 2 * Cos(li) - Exp(li): fx2 = 2 * Cos(X2) - Exp(X2)
      Case 7
        fli = li * Exp(li) - 1: fx2 = X2 * Exp(X2) - 1
      Case 8
        fli = (((li) ^ 3) + (li) + 1): fx2 = (X2) ^ 3 + (X2) + 1
      Case 9
        fli = (((li) ^ 3) - (li) - 1): fx2 = (X2) ^ 3 - (X2) – 1
    End Select
    If (fli * fx2) > 0 Then
      li = X2
    Else
      ls = X2
    End If
    X1# = (li + ls) / 2
```

```
    er = Abs((X1 - X2) / X1)
    If er <= td Then
      r = "raiz= " & X1 & " encontrada en " & z & " iteraciones"
      GoTo respuesta:
    End If
    X1 = X2
  Next z
  r = "no converge en " & ni & " iteraciones"
respuesta:
Text1.Text = r
End If
```

Aquí parece que se estuviera dando un salto mortal, pero si se analiza el algoritmo del método de bisección, se determinará que es necesario que el usuario introduzca los valores limitantes del intervalo en los que se supone esta la raíz buscada, el número máximo de iteraciones y la precisión que se busca alcanzar. Esto se logra empleando las siguientes instrucciones:

```
li# = InputBox("limite inferior del intervalo", "Datos iniciales", , 1, 1)
ls# = InputBox("limite superior del intervalo", "Datos iniciales", , 1, 1)
ni% = InputBox("numero maximo de iteraciones", "Datos iniciales", , 1, 1)
td# = InputBox("tolerancia deseada", "Datos iniciales", , 1, 1)
```

En donde li#, ls#, td# son variables de doble precisión que almacenarán los datos que serán ingresados a través de cuadros de entrada y ni% es una variable entera (lo que evitará que se introduzcan números fraccionarios en la cantidad de iteraciones).

Después abrimos un ciclo que solo terminará cuando se encuentre la raíz con la precisión deseada o al alcanzarse el número máximo de iteraciones

```
For z = 1 To ni
```

Y como el método de bisección se basa en dividir el intervalo en dos

```
X2# = (li + ls) / 2
```

Después debemos utilizar la función f(x) que seleccionó el usuario, y como la computadora no es maga, pues simplemente introducimos la función en el mismo orden que en los botones de opción empleando nuevamente la instrucción

```
SELECT CASE
    Case 0
      fli = Sin(li): fx2 = Sin(X2)
    Case 1
      fli = Sin(li) + 1 - (li) ^ 2: fx2 = Sin(X2) + 1 - (X2) ^ 2
```

```
        Case 2
          fli = Cos(li) - (li) ^ 2: fx2 = Cos(X2) - (X2) ^ 2
        Case 3
          fli = Sin(li) - (li / 2): fx2 = Sin(X2) - (X2 / 2)
        Case 4
          fli = ((li ^ 2) + 3) / ((li ^ 2) - 4): fx2 = (((X2) ^ 2) + 3) / (((X2) ^ 2) - 4)
        Case 5
          fli = 4 * li - Tan(li): fx2 = 4 * X2 - Tan(X2)
        Case 6
          fli = 2 * Cos(li) - Exp(li): fx2 = 2 * Cos(X2) - Exp(X2)
        Case 7
          fli = li * Exp(li) - 1: fx2 = X2 * Exp(X2) - 1
        Case 8
          fli = (((li) ^ 3) + (li) + 1): fx2 = (X2) ^ 3 + (X2) + 1
        Case 9
          fli = (((li) ^ 3) - (li) - 1): fx2 = (X2) ^ 3 - (X2) – 1
    End Select
```

Como se puede notar repetimos la función, con diferente variable, ya que debe ser evaluada en el límite inferior y con el valor calculado, para determinar en que parte del intervalo se encuentra la raíz buscada, esta evaluación se lleva a cabo con las siguientes instrucciones:

```
If (fli * fx2) > 0 Then
  li = X2
Else
  ls = X2
End If
```

Es decir que si hay un cambio de signo en el intervalo (li, x2), entonces se debe sustituir el valor del limite superior y si es lo contrario, entonces se debe sustituir el valor del limite inferior con el nuevo valor calculado.

Luego se recalcula un nuevo valor usando el nuevo intervalo, se calcula la diferencia de los valores absolutos de las dos ultimas aproximaciones y se compara con la precisión deseada, si la precisión es la correcta, entonces se detiene el proceso de calculo (para ello se hace la variable r igual al resultado).

```
X1# = (li + ls) / 2
er = Abs((X1 - X2) / X1)
If er <= td Then
    r = "raiz= " & X1 & " encontrada en " & z & " iteraciones"
   GoTo respuesta:
End If
```

```
X1 = X2
```

De lo contrario se lleva a cabo una nueva iteración

```
Next z
```

Si el número de iteraciones se rebasa, entonces es necesario que se muestre un mensaje al usuario indicándole que posiblemente no fueron suficientes las iteraciones (se guarda un mensaje en r para después mostrarlo en el cuadro de texto).

```
r = "no converge en " & ni & " iteraciones"
```

Por ultimo, ya sea que fuese o no posible encontrar la raíz se muestra una respuesta al usuario en el cuadro de texto, igualando este a la variable r (la cual contiene el texto en cualquier caso

```
respuesta:
Text1.Text = r
End If
```

Nótese que se utilizó el uso de sangrías para poder tener una visión más clara de la relación entre instrucciones y el uso por pares de algunas otras (IF, THEN, ELSE, END IF... FOR, NEXT... SELECT CASE, END SELECT)

Con esto ya es posible probar el programa (para el caso del método de bisección) utilizando el control ▶ que se encuentra en la barra de herramientas de Visual Basic. Para detener la ejecución tan solo es necesario dar clic en el control ■

Para el caso del método de punto fijo, la codificación es semejante en cuanto su estructura de programación; Las variables iniciales se introducen mediante cuadros de entrada. Hay que notar que la primera línea revisa que método fue seleccionado (d=0 "método de bisección"; d=1 "método de punto fijo"; d=2 "método de Newton-Raphson" y d=4 "método de la secante").

```
'método de punto fijo
If d = 1 Then
  x0# = InputBox("aproximacion inicial", "Datos iniciales", , 1, 1)
  ni% = InputBox("numero maximo de iteraciones", "Datos iniciales", , 1, 1)
  td# = InputBox("tolerancia deseada", "Datos iniciales", , 1, 1)
  x00# = x0
  For z = 1 To ni
```

```
  Select Case e
    Case 0
      fx0 = Sin(X0): gx0 = Cos(X0)
    Case 1
      fx0 = Sin(X0) + 1 - (X0) ^ 2: gx0 = Cos(X0) - 2 * X0
    Case 2
      fx0 = Cos(X0) - (X0) ^ 2: gx0 = -Sin(X0) - 2 * X0
    Case 3
      fx0 = Sin(X0) - (X0 / 2): gx0 = Cos(X0) - 0.5
    Case 4
      fx0 = (((X0) ^ 2) + 3) / (((X0) ^ 2) - 4): gx0 = ((-14) * (X0)) / (((X0) ^ 2 - 4) ^ 2)
    Case 5
      fx0 = 4 * X0 - Tan(X0): gx0 = 4 - (1 / (Cos(X0) * Cos(X0)))
    Case 6
      fx0 = 2 * Cos(X0) - Exp(X0): gx0 = -2 * Sin(X0) - Exp(X0)
    Case 7
      fx0 = X0 * Exp(X0) - 1: gx0 = Exp(-X0)
    Case 8
      fx0 = (((X0) ^ 3) + (X0) + 1): gx0 = 3 * X0 ^ 2 + 1
    Case 9
      fx0 = (((X0) ^ 3) - (X0) - 1): gx0 = 3 * X0 ^ 2 - 1
  End Select
    X2 = gx0
    X1 = X2
    er = Abs((X1 - x0) / X1)
    If er <= td Then
      r = "raiz= " & X1 & " encontrada en " & z & " iteraciones"
      GoTo respuesta1:
    End If
    x0 = X1
    If x0 > (3 * x00) Then
      r = "el valor calculado esta divergiendo de el valor de la raiz"
      GoTo respuesta1:
    End If
  Next z
  r = "no converge en " & ni & " iteraciones"
respuesta1:
Text1.Text = r
End If
```

En este caso x0# y td# son las variables iniciales de doble precisión, ni% es el número de iteraciones y se añade una variable temporal x00# con el fin de conservar el valor original de la aproximación inicial.

```
    If d = 1 Then
      x0# = InputBox("aproximacion inicial", "Datos iniciales", , 1, 1)
```

```
ni% = InputBox("numero maximo de iteraciones", "Datos iniciales", , 1, 1)
td# = InputBox("tolerancia deseada", "Datos iniciales", , 1, 1)
x00# = x0
```

Luego se emplean nuevamente las instrucciones SELECT CASE para determinar cual fue la función seleccionada y hacer la evaluación; en este caso debido a que el método hace uso de la derivada de la función, es necesario introducir la función (fx) y a continuación su derivada (gx) con el fin de disponer de las operaciones de cálculo.

```
Select Case e
  Case 0
    fx0 = Sin(X0): gx0 = Cos(X0)
  Case 1
    fx0 = Sin(X0) + 1 - (X0) ^ 2: gx0 = Cos(X0) - 2 * X0
  Case 2
    fx0 = Cos(X0) - (X0) ^ 2: gx0 = -Sin(X0) - 2 * X0
  Case 3
    fx0 = Sin(X0) - (X0 / 2): gx0 = Cos(X0) - 0.5
  Case 4
    fx0 = (((X0) ^ 2) + 3) / (((X0) ^ 2) - 4): gx0 = ((-14) * (X0)) / (((X0) ^ 2 - 4) ^ 2)
  Case 5
    fx0 = 4 * X0 - Tan(X0): gx0 = 4 - (1 / (Cos(X0) * Cos(X0)))
  Case 6
    fx0 = 2 * Cos(X0) - Exp(X0): gx0 = -2 * Sin(X0) - Exp(X0)
  Case 7
    fx0 = X0 * Exp(X0) - 1: gx0 = Exp(-X0)
  Case 8
    fx0 = (((X0) ^ 3) + (X0) + 1): gx0 = 3 * X0 ^ 2 + 1
  Case 9
    fx0 = (((X0) ^ 3) - (X0) - 1): gx0 = 3 * X0 ^ 2 - 1
End Select
```

Después se compara si ya se llego a la precisión deseada en la aproximación y si esta convergiendo o no.

```
er = Abs((X1 - x0) / X1)
If er <= td Then
  r = "raiz= " & X1 & " encontrada en " & z & " iteraciones"
  GoTo respuesta1:
End If
x0 = X1
If x0 > (3 * x00) Then
  r = "el valor calculado esta divergiendo de el valor de la raíz"
```

```
    GoTo respuesta1:
End If
```

Y dependiendo del caso se guarda la posible respuesta para mostrarse en el cuadro de texto a través de la variable r.

Por último se muestra en el cuadro de texto el resultado.

```
respuesta1:
Text1.Text = r
End If
```

Se hace uso de la etiqueta "respuesta1:" para indicar a que sección debe saltar el control del programa en el momento de haber obtenido una respuesta ya sea positiva o no, en la sección correspondiente al método de bisección se empleo una etiqueta "respuesta:" con el fin de que no salte el control del programa a un sitio distinto del deseado.

Como se pudo notar hay espacios en que no se introdujeron las ecuaciones, esto se debe a que no necesariamente debo tener completas las ecuaciones para que el programa funcione, pero su ausencia provocara un error en caso de que se seleccione esa opción. En caso de introducir la fórmula en forma de imagen en la *form*, es obligatorio introducir la ecuación en la sección *case* correspondiente de cada uno de los métodos (si es que se desea evaluar esa función en cada método). Recordemos que el hecho de colocar algo en la *form* es solo con el fin de hacer una presentación visual al usuario, pero la programación es lo que realmente le dará las instrucciones a la computadora de lo que debe hacer.

Cuando se selecciona el caso de que d=2 (es decir, se selecciono el método de Newton Raphsón)

Añadimos algunas sentencias REM (') con el fin de documentar el programa para futuras modificaciones.

```
'método de newton-raphson
If d = 2 Then
```

Aquí estamos verificando el método seleccionado.

```
  X0# = InputBox("aproximacion inicial", "Datos iniciales", , 1, 1)
```

```
ni% = InputBox("numero maximo de iteraciones", "Datos iniciales", , 1, 1)
td# = InputBox("tolerancia deseada", "Datos iniciales", , 1, 1)
```

Se solicita que introduzca el usuario los valores iniciales e inmediatamente se inicia la primera iteración.

```
  For z = 1 To ni
Select Case e
    Case 0
      fx0 = Sin(X0): gx0 = Cos(X0)
    Case 1
      fx0 = Sin(X0) + 1 - (X0) ^ 2: gx0 = Cos(X0) - 2 * X0
    Case 2
      fx0 = Cos(X0) - (X0) ^ 2: gx0 = -Sin(X0) - 2 * X0
    Case 3
      fx0 = Sin(X0) - (X0 / 2): gx0 = Cos(X0) - 0.5
    Case 4
      fx0 = ((X0) ^ 2 + 3) / ((X0) ^ 2 - 4): gx0 = ((-14) * (X0)) / ((((X0) ^ 2) - 4) ^ 2)
    Case 5
      fx0 = 4 * X0 - Tan(X0): gx0 = 4 - (1 / (Cos(X0) * Cos(X0)))
    Case 6
      fx0 = 2 * Cos(X0) - Exp(X0): gx0 = -2 * Sin(X0) - Exp(X0)
    Case 7
      fx0 = X0 * Exp(X0) - 1: gx0 = Exp(-X0)
    Case 8
      fx0 = (((X0) ^ 3) + (X0) + 1): gx0 = 3 * X0 ^ 2 + 1
    Case 9
      fx0 = (((X0) ^ 3) - (X0) - 1): gx0 = 3 * X0 ^ 2 - 1
End Select
    X2 = X0 - fx0 / gx0
    X0 = X1: X1 = X2
    er = Abs((X1 - X0) / X1)
    If er <= td Then
      r = "raiz= " & X1 & " encontrada en " & z & " iteraciones"
      GoTo respuesta2:
    End If
  Next z
```

Al igual que en los métodos anteriores se agregan las diferentes ecuaciones y su derivada apoyándonos en las instrucciones *select case* para procesar una solo función a la vez.

```
  r = "no converge en " & ni & " iteraciones"
respuesta2:
```

```
Text1.Text = r
End If
```

Al final se muestra el resultado ya sea que se encontró respuesta o que divergió el resultado.

Como podemos observar hasta el momento la forma de programar los tres métodos es muy semejante, es por ello que para evitar el tener que hacer el diseño de tres *forms*, simplemente se procedió a reutilizar la misma interfase.

Añadiremos un método más, el método de la secante.

```
'método de la secante
If d = 3 Then
  X0# = InputBox("aproximacion inicial 1", "Datos iniciales", , 1, 1)
  X1# = InputBox("aproximacion inicial 2", "Datos iniciales", , 1, 1)
  ni% = InputBox("numero maximo de iteraciones", "Datos iniciales", , 1, 1)
  td# = InputBox("tolerancia deseada", "Datos iniciales", , 1, 1)
  For z = 1 To ni
  Select Case e
      Case 0
        fx0 = Sin(X0): gx0 = Cos(X0)
      Case 1
        fx0 = Sin(X0) + 1 - (X0) ^ 2: gx0 = Cos(X0) - 2 * X0
      Case 2
        fx0 = Cos(X0) - (X0) ^ 2: gx0 = -Sin(X0) - 2 * X0
      Case 3
        fx0 = Sin(X0) - (X0 / 2): gx0 = Cos(X0) - 0.5
      Case 4
        fx0 = ((X0) ^ 2 + 3) / ((X0) ^ 2 - 4): gx0 = ((-14) * (X0)) / ((((X0) ^ 2) - 4) ^ 2)
      Case 5
        fx0 = 4 * X0 - Tan(X0): gx0 = 4 - (1 / (Cos(X0) * Cos(X0)))
      Case 6
        fx0 = 2 * Cos(X0) - Exp(X0): gx0 = -2 * Sin(X0) - Exp(X0)
      Case 7
        fx0 = X0 * Exp(X0) - 1: gx0 = Exp(-X0)
      Case 8
        fx0 = (((X0) ^ 3) + (X0) + 1): gx0 = 3 * X0 ^ 2 + 1
      Case 9
        fx0 = (((X0) ^ 3) - (X0) - 1): gx0 = 3 * X0 ^ 2 - 1
    End Select
    If fx1 <= td Then
      r = "raiz= " & X1 & " encontrada en " & z & " iteraciones"
      GoTo respuesta3:
    End If
    X2 = X1 - (X1 - X0) * fx1 / (fx1 - fx0)
    X0 = X1: X1 = X2
```

```
  Next z
  r = "no converge en " & ni & " iteraciones"
respuesta3:
Text1.Text = r
End If
```

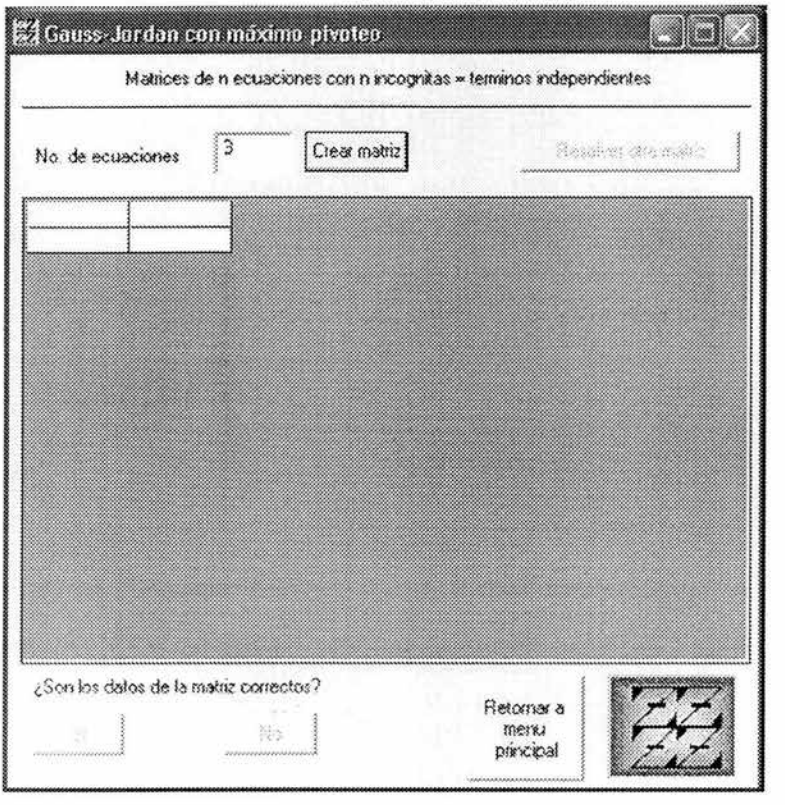
Comparando la programación ¿Ya estarías en posición de analizar el porqué de las instrucciones?

Nos hemos dedicado tan solo a analizar la combinación de las instrucciones con el fin de relegar la tarea de explicar la sintaxis de cada instrucción al manual de programación del lenguaje. Ya que la parte que suponemos ha sido descuidada, es en donde hacemos la estructuración de las diferentes instrucciones para codificar los algoritmos.

Para el siguiente programa se necesitará una *form* llamada gaussj, la cual será almacenada con el nombre gauss_jordan.frm, en esta forma serán colocados cinco botones de comando con las siguientes propiedades:

Name	**Caption**	**Enabled**	**Default**
cmdcrear	Crear matriz	False	True
Cmdsi	Si	False	False
Cmdno	No	False	False
cmdmenu	Retornar a menu principal	True	False
cmdotra	Resolver otra matriz	False	False

Y se agregará un cuadro de texto con la propiedad name *txtecuaciones*. Tambien un objeto llamado MSFlexGrid y se añadiran algunas etiquetas, para dar la siguiente apariencia:



Una vez generada la forma, se continuará con la generación del código que lleve a cabo el algoritmo. Se cuidará en este ejemplo no solo obtener el resultado, sino cuidar los detalles en la elaboración de software para terceros, es decir, programas que usan otras personas.

Con las siguientes instrucciones se oculta la pantalla de menú principal (que se elaborará posteriormente) y se define que habrá una matriz flexible. El siguiente código debe introducirse dando doble clic en el botón de comando "*cmdcrear*"

```
cmdmenu.Enabled = False
MSFlexGrid1.Row = 0
MSFlexGrid1.Col = 0
MSFlexGrid1.Text = "Ec.\Col."
```

La primer línea deshabilita el botón de regreso al menú, para evitar que se deje incompleta la captura de los datos de la matriz. Las siguientes dimensionan la matriz a cero líneas y cero columnas, es decir una posición inicial antes de comenzar el cálculo.

Se abrirá un ciclo do-while con el fin de verificar que se introduzca un valor adecuado, para definir el número de ecuaciones que manejará el método. En este programa se manejará un nivel mayor de conocimiento, cuidando que se ingresen valores adecuados, es decir haciendo la validación de los datos.

```
Do ' validacion del numero de ecuaciones
If IsNumeric(txtecuaciones.Text) Then
  n = txtecuaciones.Text
  flag = 1
  nn$ = Int(n)
  If nn$ <> n Then
    txtecuaciones.Text = InputBox("necesita introducir solo números enteros", "Aviso")
    n = txtecuaciones.Text
    txtecuaciones.SetFocus
    Cmdcrear.Enabled = False
    flag = 0
  Else
    flag = 1
  End If
Else
  MsgBox "necesita introducir solo números", 48, "Aviso"
  txtecuaciones.SetFocus
  Cmdcrear.Enabled = False
End If
```

Interesa primero que se introduzca un valor numérico y eliminar aquellas entradas que contengan caracteres, asimismo que no se introduzca un número fraccionario (no se debe generar una fracción de línea o columna); para ello, se incluirá un condicional *IF* y la opción *IsNumeric()*, colocando la etiqueta del contenido del cuadro de texto (txtecuaciones.text);

```
If IsNumeric(txtecuaciones.Text) Then
```

Por lo que, si la entrada no es de tipo numérica, se volverá a solicitar el ingreso de un valor adecuado.

```
Else
  MsgBox "necesita introducir solo números", 48, "Aviso"
  txtecuaciones.SetFocus
  Cmdcrear.Enabled = False
```

Una vez que se determina que la información de entrada si es numérica, se procede a guardar el valor de entrada en la variable de tipo variant *n*, después se guarda el entero de esta variable en una variable de tipo cadena *nn$*; Si al comparar las variables son iguales, el número ingresado es entero y no debe haber ningún problema para proseguir; en caso contrario se mostrará un mensaje avisando del error y se solicitará se vuelva a ingresar un valor adecuado.

```
If nn$ <> n Then
  txtecuaciones.Text = InputBox("necesita introducir solo números enteros", "Aviso")
  n = txtecuaciones.Text
  txtecuaciones.SetFocus
  Cmdcrear.Enabled = False
  flag = 0
```

Se agregó una variable (flag) para indicar si la validación fue positiva y puede continuar (flag=1) o que se debe ingresar un valor correcto (flag=0).

Otra validación que hay que llevar a cabo, es que no puede haber un número negativo de ecuaciones, por ello se debe validar que no se introduzcan números de este tipo.

```
If n <= 0 Then
  txtecuaciones.Text = InputBox("No se aceptan numeros negativos", "Aviso")
  flag = 0
End If
```

Hay que tener en cuenta la cantidad de memoria disponible en la computadora y de acuerdo con las propiedades de una matriz flexible se debe evitar que se soliciten mas de 500 ecuaciones para una PC con 128 Mb en RAM[1]; porque se busca que el programa ejecute en la mayoría de las computadoras personales sin tener que modificar el programa fuente).

```
If n >= 500 Then
  txtecuaciones.Text = InputBox("No es posible generar una matriz de ese tamaño", "Aviso")
  flag = 0
End If
```

La variable dentro del código anterior (*flag*) ha servido como llave para comprobar que se ha verificado la información de entrada, a esta variable se le asigna el valor de 1 cuando se ha cumplido con los requisitos y el de 0 cuando no, de esa manera

[1] Se toma esta referencia, porque en la actualidad una PC promedio cuenta como mínimo con esta cantidad de memoria.

el ciclo do-while solo se abandona hasta haber comprobado que se introdujo un valor entero positivo.

```
Loop While flag <> 1
```

Una vez conocido el número de ecuaciones con las que se trabajará, hay que redimensionar la matriz flexible al tamaño deseado y dimensionar los arreglos matriciales para almacenar los valores ingresados y los que se calcularán.

```
'redimensionamiento de las matrices
ReDim a(n, n + 1), b(n, n + 1), c(n), X(n)
MSFlexGrid1.Rows = n + 1
MSFlexGrid1.Cols = n + 2
z% = 0
For y% = 1 To n
    MSFlexGrid1.Row = z%
    MSFlexGrid1.Col = y%
    MSFlexGrid1.Text = "X " & y%
Next y%
MSFlexGrid1.Row = 0
MSFlexGrid1.Col = n + 1
MSFlexGrid1.Text = "Constante"
```

Después, hay que proporcionar el mecanismo para la introducción de las constantes de los términos de cada ecuación y validar que se introduzcan en forma numérica.

```
For z% = 1 To n   'Introduccion y validacion de datos
  Title$ = "ecuacion " & z%
  For y% = 1 To n   'elementos
    title2$ = "elemento " & z% & "," & y%
    Do
      temp = InputBox(title2$, Title$, , 1, 1)
      If IsNumeric(temp) Then
        flag = 1
      Else
        If temp = "" Then   'cancelacion del proceso para introducir valores
          flagcancel = 1
          GoTo cancelado:
        End If
        MsgBox "Introduzca solo números", 48, "Aviso"
        flag = 0
      End If
    Loop While flag <> 1
    b(z%, y%) = temp: flag = 0
    MSFlexGrid1.Row = z%
    MSFlexGrid1.Col = y%
    MSFlexGrid1.Text = b(z%, y%)
  Next y%
  title2$ = "constante " & z%   'constante
    Do
```

```
      temp = InputBox(title2$, Title$, , 1, 1)
      If IsNumeric(temp) Then
        flag = 1
      Else
        MsgBox "Introduzca solo números", 48, "Aviso"
        flag = 0
      End If
    Loop While flag <> 1
  b(z%, n + 1) = temp: flag = 0
  MSFlexGrid1.Row = z%
  MSFlexGrid1.Col = y%
  MSFlexGrid1.Text = b(z%, y%)
Next z%
cancelado:
If flagcancel = 1 Then
  Cmdsi.Enabled = False
  Cmdno.Enabled = False
  cmdmenu.Enabled = True
Else
  txtecuaciones.Enabled = False
  Cmdcrear.Enabled = False
  cmdmenu.Enabled = True
  Cmdsi.Enabled = True
  Cmdno.Enabled = True
End If
End Sub
```

El surgimiento de algún error en la captura de los términos que impidiese seguir con la introducción de valores, o simplemente el deseo del usuario de finalizar el proceso en cualquier momento, requiere que se habilite una opción de cancelación; por ello se tiene que incorporar una variable (*flagcancel*) con el fin de determinar si se ha hecho una llamada de cancelación.

```
If temp = "" Then   'cancelacion del proceso para introducir valores
```

Los ciclos For-Next incluidos son con el fin de hacer un barrido por los términos de las ecuaciones y por las ecuaciones mismas. Al mismo tiempo que se introduce cada término, se valida que cada uno sea del tipo numérico (entero o fraccionario, positivo o negativo).

```
For z% = 1 To n   'Introduccion y validacion de datos
  Title$ = "ecuacion " & z%
  For y% = 1 To n   'elementos
    title2$ = "elemento " & z% & "," & y%
    Do
      temp = InputBox(title2$, Title$, , 1, 1)
      If IsNumeric(temp) Then
        flag = 1
      Else
        If temp = "" Then   'cancelacion del proceso para introducir valores
          flagcancel = 1
```

```
          GoTo cancelado:
        End If
        MsgBox "Introduzca solo números", 48, "Aviso"
        flag = 0
      End If
    Loop While flag <> 1
    b(z%, y%) = temp: flag = 0
    MSFlexGrid1.Row = z%
    MSFlexGrid1.Col = y%
    MSFlexGrid1.Text = b(z%, y%)
  Next y%
  title2$ = "constante " & z%   'constante
    Do
      temp = InputBox(title2$, Title$, , 1, 1)
      If IsNumeric(temp) Then
        flag = 1
      Else
        MsgBox "Introduzca solo números", 48, "Aviso"
        flag = 0
      End If
    Loop While flag <> 1
  b(z%, n + 1) = temp: flag = 0
  MSFlexGrid1.Row = z%
  MSFlexGrid1.Col = y%
  MSFlexGrid1.Text = b(z%, y%)
Next z%
```

Las entradas se guardan en el arreglo matricial b(n,n+1) {hay que recordar que este método es para resolver n ecuaciones con n incógnitas, el espacio adicional es para el termino independiente}.

Si se abandona esta sección del programa y se regresa al menú principal, entonces hay que tratar de retornar a las condiciones iniciales para evitar que se produzca un error en caso de regresar. Desgraciadamente ya no se puede restaurar totalmente al mismo estado, lo que se puede hacer entonces es minimizar tanto la matriz flexible como el dimencionamiento de los arreglos matriciales de las variables. Para ello se introduce el siguiente código dando doble click en el botón de comando *cmdmenu*.

```
n = txtecuaciones.Text
If txtecuaciones.Text = "" Then
  n = 2
  ReDim a(n, 2 * n + 1), b(n, n + 1), c(n), X(n)
  MSFlexGrid1.Rows = n + 1
  MSFlexGrid1.Cols = n + 2
End If
For z% = 1 To n
  For y% = 1 To n + 1
```

```
    a(z%, y%) = 0: b(z%, y%) = 0
  Next y%
Next z%
For z% = 1 To n
  For y% = 1 To n + 1
    MSFlexGrid1.Row = z%
    MSFlexGrid1.Col = y%
    MSFlexGrid1.Text = " "
  Next y%
Next z%
Cmdcrear.Enabled = True
txtecuaciones.Text = ""
txtecuaciones.Enabled = True
txtecuaciones.SetFocus
Cmdotra.Enabled = False
gaussj.Hide
Unload gaussj
menu.Show
```

Otra posibilidad es que se halla cometido un error al introducir un valor, por ello es que se creará la opción de modificar aquel valor erróneo, para no tener que ingresar la matriz completa nuevamente. Esto se logra solicitando el número de ecuación y el término a modificar, se pide el nuevo valor, el cual se almacena en la misma posición. El código se añade en el botón de comando *cmdno* dando doble clic; aquí se debe validar que los valores de columna y renglón sean numéricos enteros, para que pueda ser localizado el término a corregir.

```
' correcion de datos en la matriz
    Do
      temp1 = InputBox("¿ecuacion a modificar?", "Corrección", , 1, 1)
      If IsNumeric(temp1) Then
        flag = 1
        If temp1 > txtecuaciones.Text Or temp1 < 1 Then
          aviso$ = "Ecuaciones disponibles: " & txtecuaciones.Text
          MsgBox aviso$, 48, "Aviso"
          flag = 0
        Else
          flag = 1
        End If
      Else
        MsgBox "Introduzca solo números enteros", 48, "aviso"
        flag = 0
      End If
    Loop While flag <> 1
    flag = 0
    Do
      temp2 = InputBox("¿columna a modificar?", "Corrección", , 1, 1)
      If IsNumeric(temp2) Then
        flag = 1
        If temp2 > txtecuaciones.Text + 1 Or temp2 < 1 Then
```

```
        aviso$ = "Columnas disponibles: " & txtecuaciones.Text + 1
        MsgBox aviso$, 48, "Aviso"
        flag = 0
      Else
        flag = 1
      End If
    Else
      MsgBox "Introduzca solo números enteros", 48, "aviso"
      flag = 0
    End If
  Loop While flag <> 1
  title2$ = "elemento " & temp1 & "," & temp2: flag = 0
  Do
   temp = InputBox(title2$, Title$, , 1, 1)
   If IsNumeric(temp) Then
     flag = 1
   Else
     MsgBox "Introduzca solo números", 48, "Aviso"
     flag = 0
   End If
  Loop While flag <> 1
b(temp1, temp2) = temp: flag = 0
MSFlexGrid1.Row = temp1
MSFlexGrid1.Col = temp2
MSFlexGrid1.Text = b(temp1, temp2)
```

Como se puede observar, para mantener la seguridad de que se ha introducido un valor adecuado, también hay que controlar en este punto que sea un valor sin caracteres (aquí el valor puede ser fraccionario y/o negativo); para ello se ha agregado el código necesario para garantizarlo, al tiempo que se evita dejar el espacio vacío.

En caso de querer calcular un segundo sistema de ecuaciones, entonces es necesario permitir al usuario introducir un nuevo dimensionamiento del sistema de ecuaciones y validarlo (numero de ecuaciones). En el botón de comando cmdotra se añade el siguiente código:

```
n = txtecuaciones.Text
For z% = 1 To n
  For y% = 1 To n + 1
    a(z%, y%) = 0: b(z%, y%) = 0
  Next y%
Next z%
For z% = 1 To n
  For y% = 1 To n + 1
    MSFlexGrid1.Row = z%
    MSFlexGrid1.Col = y%
    MSFlexGrid1.Text = " "
  Next y%
```

```
Next z%
Cmdcrear.Enabled = True
txtecuaciones.Text = ""
txtecuaciones.Enabled = True
Cmdotra.Enabled = False
```

Por cuestiones de seguridad de los datos, mientras se calcula, se genera una copia de la matriz B en la A y se trabajará con esta última.

```
Cmdno.Enabled = False
Cmdsi.Enabled = False
'Se genera la matriz A como copia de B
n = Val(txtecuaciones.Text)
For z% = 1 To n
  For y% = 1 To n + 1
    a(z%, y%) = b(z%, y%)
  Next y%
Next z%
```

Y finalmente si ya se genero la matriz en forma adecuada, lo que procede es el cálculo propiamente dicho a través del método de Gauss-Jordan con pivoteo. El código siguiente debe ejecutarse al dar clic en el botón "si" (*cmdsi*).

A partir de la copia de la matriz, se trabajará de acuerdo con el algoritmo, para ello se introducirán algunas sentencias REM como ayuda al programador para indicar que es lo que hace cada sección del código y para poder modificar el programa en caso de alguna falla o modificación.

```
'generacion de coeficientes de la diagonal o pivotes
maxpivot# = 0.00000000000001: Row = 1: Col = 1
For j = 1 To n
  maxpivot# = 0.00000000000001
' deteccion del maximo pivote
  For z = n To j Step -1
    If maxpivot# < Abs(a(z, j)) Then
      maxpivot# = Abs(a(z, j)): Row = z
    End If
  Next z
' intercambio de renglones
  For z = 1 To (n + 1)
    a(0, z) = a(j, z)
    a(j, z) = a(Row, z)
    a(Row, z) = a(0, z)
  Next z
' eliminacion hacia adelante
  For zz = 1 To (n - j)
  If j < n Then
    a(0, 0) = a(j + zz, j) / a(j, j)
    For z = 1 To (n + zz)
      a(j + zz, z) = a(j + zz, z) - a(j, z) * a(0, 0)
```

```
    Next z
  Else
    a(0, 0) = a(n, n)
    For z = 1 To (n + 1)
      a(j, z) = a(j, z) / a(0, 0)
    Next z
  End If
  Next zz
' eliminacion de elementos no cero (errores de redondeo)
  For zz = j - 1 To 0 Step -1
    If a(j, zz) < 0.1 Then
      a(j, zz) = 0
    End If
  Next zz
  maxpivot# = 0.000000000000001
Next j
'sustitucion hacia atras
For j = n To 1 Step -1
  If j = n Then
    a(0, 0) = 0
  Else
    For z = j + 1 To n
      a(0, 0) = a(0, 0) - a(j, z) * a(z, z)
    Next z
  End If
  a(j, j) = (a(j, n + 1) + a(0, 0)) / a(j, j)
  a(0, 0) = 0
Next j
For j = n To 1 Step -1
  a(j, n + 1) = a(j, j)
Next j
' igualar a cero los elementos posteriores a la diagonal
For j = 1 To n
  For zz = j To n
    If j = zz Then
      a(j, zz) = 1
    Else
      a(j, zz) = 0
    End If
  Next zz
Next j
'salida de resultados
For z% = 1 To n
  For y% = 1 To n + 1
    MSFlexGrid1.Row = z%
    MSFlexGrid1.Col = y%
    MSFlexGrid1.Text = a(z%, y%)
  Next y%
Next z%
Cmdotra.Enabled = True
```

Para finalizar, siempre es bueno recordarle al usuario algunas limitantes y el alcance de la precisión. Este código se introduce en la sección que ejecuta al cargar la forma

```
Private Sub Form_Load()
aviso1$ = "El máximo número de ecuaciones que es posible manejar es de 499 (con una memoria
RAM de 128 Mb)"
aviso2$ = " y la precisión máxima es de 0.00000001"
aviso$ = aviso1$ & aviso2$
MsgBox aviso$, 48, "Aviso importante"
```

La utilización de un lenguaje visual dirigido a objetos implica que cada sección se ejecuta dependiendo del elemento empleado, lo que obliga a introducir código que habilite o deshabilite elementos, con el fin de evitar que el usuario lleve a cabo acciones fuera de secuencia que podrían provocar algún error en la ejecución del programa.

Por ejemplo, en un cuadro de texto se añadió el siguiente código:

```
Cmdcrear.Enabled = Trae
```

con el fin de que al introducir el tamaño de la matriz se habilite el botón para crear la matriz, que a su vez validará el valor tecleado por el usuario; si esté es validado, se habilitarán los botones de si está o no correcta la matriz.

Siempre hay que tener en cuenta que el usuario no siempre responderá en la forma en que debe, o tratará de averiguar "que pasa si..."

Por ejemplo, hasta el momento se ha supuesto que el usuario empleará el mouse como medio para aceptar o introducir el dato del número de ecuaciones, pero, ¿que pasaría si emplea el teclado y presiona *Enter*?

Ello se tendría que prever, seleccionando el evento *keydown* que verificará la presión de cada tecla y si alguna de ellas es la tecla Enter, entonces llevar a cabo lo mismo que haría si se hubiese dado click con el mouse, introduciendo el siguiente código en el cuadro de texto donde se introducirá el número de ecuaciones.

```
If KeyCode = vbEnter Then
  Cmdcrear.Enabled = True
  Cmdcrear.SetFocus
End If
```

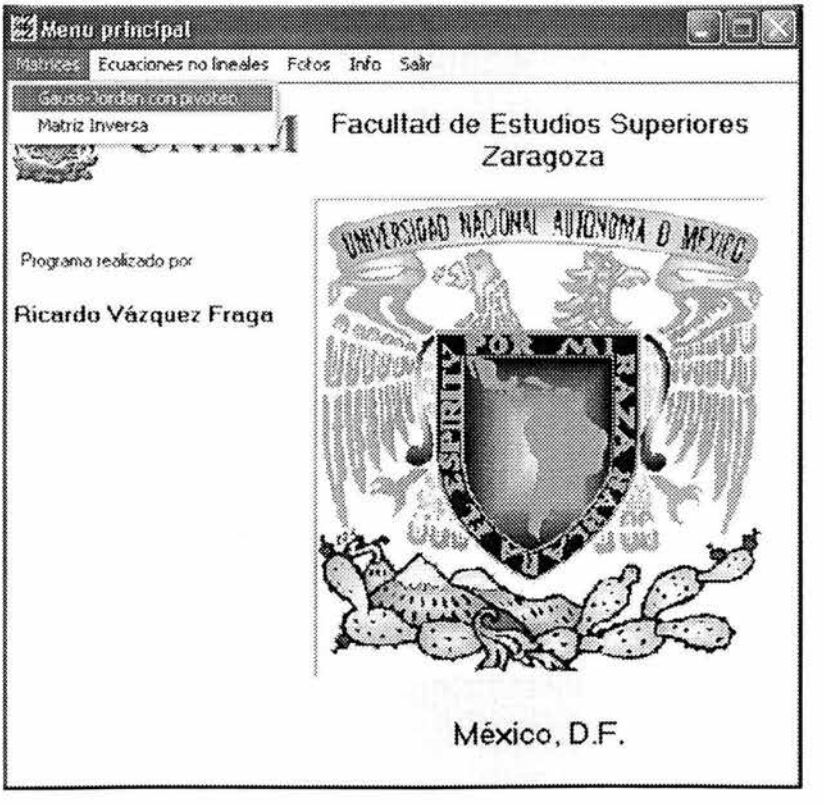
Con el fin de elaborar un menú en donde se pueda integrar este programa con el ya elaborado y algún otro, procederemos a elaborar una *form* a la que llamaremos menú y que guardaremos con el nombre de menu.frm

Este menú servirá a la vez de portada; se introducirán algunas imágenes y etiquetas que serán la presentación. En el menú de herramientas esta el editor de menús, al activarlo se abrirá un cuadro con el que se puede insertar el texto que llevará cada menú (en la sección de *caption*); en la sección *name* se coloca el nombre de una variable que identificará a ese menú y el orden en que aparecerán se regula mediante unas flechitas que aparecen del lado izquierdo de dicho cuadro.

De manera que por el momento se elaborarán tres menús

1) Matrices
2) Ecuaciones no lineales
3) Salir

Una vez elaborado este menú, se añadirá una opción dentro del menú de matrices "Gauss-Jordan con pivoteo", logrando un resultado como el de la siguiente figura.



¿para qué tomarse tantos problemas para programar algo tan complicado, si el o los método son sencillos?

La respuesta es:

- Se busca tener un ejemplo claro de cómo programar y de los cuidados que hay que tener en la elaboración de software cuando este podría ser o será empleado por otros.
- La realización de un programa se justifica, cuando la cantidad de trabajo a realizarse con él es mayor que el esfuerzo requerido para programar; también cuando el programa podría estar a disposición de otros usuarios.
- La práctica obtenida con estos ejercicios nos da la pericia necesaria para afrontar proyectos mayores como en el caso de tratar de generar programas que cubran la necesidad de un software comercial caro o que no logra cubrir las necesidades particulares.

Capítulo 3.- Programas y ejemplos de ejecución

Por el diseño que se hizo del programa para resolver ecuaciones no lineales, en el cual los cuatro métodos seleccionados se trabajan en la misma pantalla y a que se pueden resolver las ecuaciones por los cuatro métodos, Mostraremos únicamente como se emplea el programa con uno de los métodos, el proceso de introducir los datos es semejante en cada uno de ellos y solo se tendrán diferencias en cuanto a la precisión.

3.1. Ejemplos de resolución de ecuaciones no lineales.

3.1.1. Método de bisección

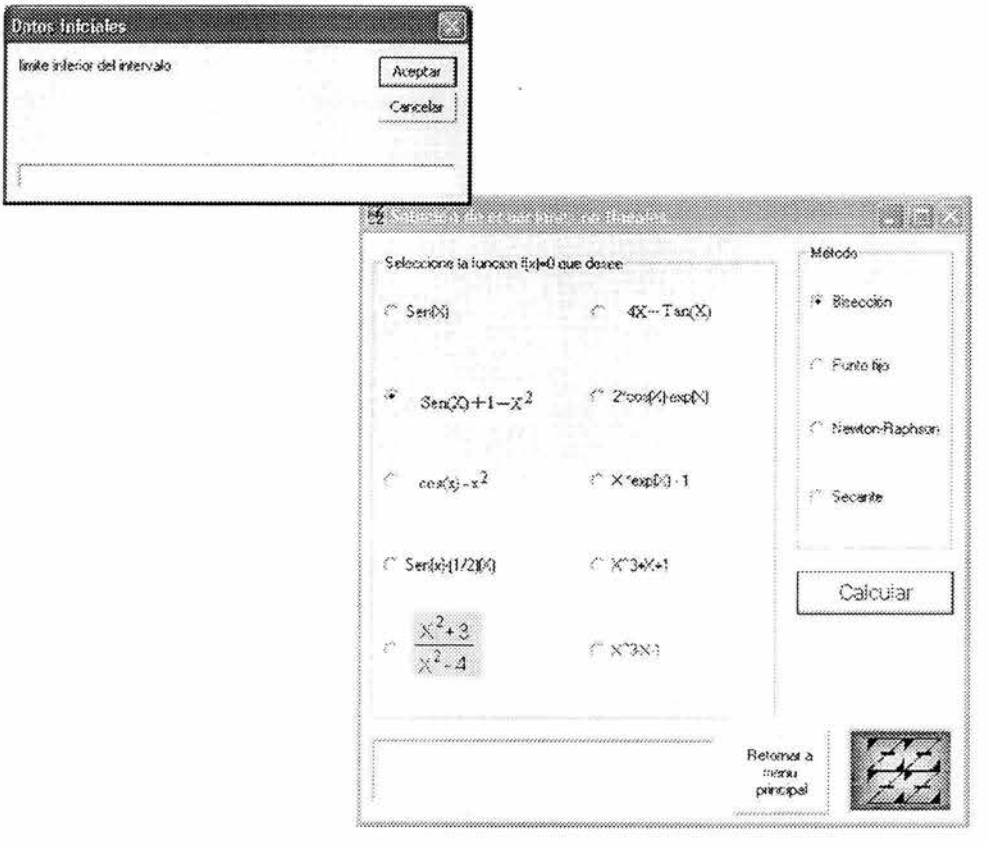
El primer paso es seleccionar la función con que se desea llevar a cabo el cálculo, en caso de que la función que se desea no se encuentre en las opciones, se deberá introducir la función en el código fuente. Ello se debe a que Visual Basic no cuenta con alguna instrucción para la definición de funciones y a que el proceso de "desmenuzar" e interpretar el texto introducido por el usuario es sumamente laborioso y complicado.

Dicho proceso implica generar un programa que interprete letra por letra y/o número la cadena de caracteres con que el usuario describe la función de interés para posteriormente reconstruir la función internamente para llevar a cabo el proceso de cálculo y dar un resultado.

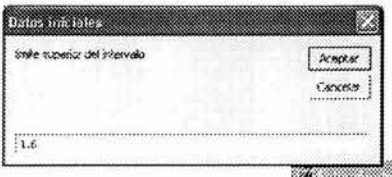
Otra posible opción es generar una librería en algún lenguaje de programación que si cuente con el medio de introducir funciones y darle la orden a Visual Basic para que la utilice.

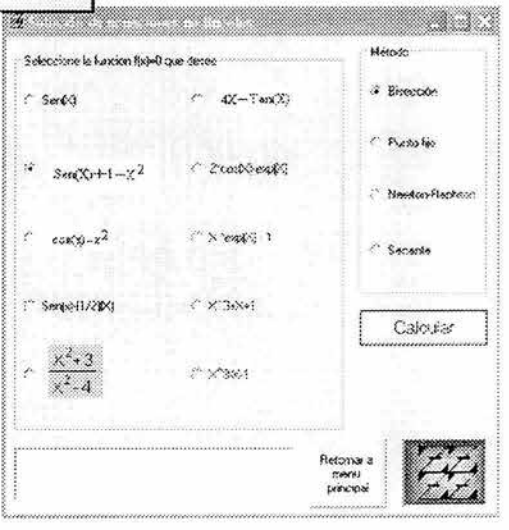
Una vez seleccionada la función, hay que marcar el método con el que se desea llevar a cabo el proceso de cálculo y dar click en el botón "Calcular".

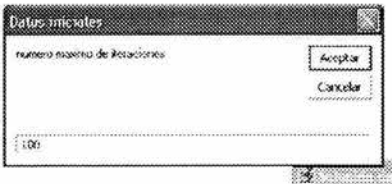
Se nos pedirá que se introduzca el límite inferior del intervalo en donde se piensa que existe una raíz

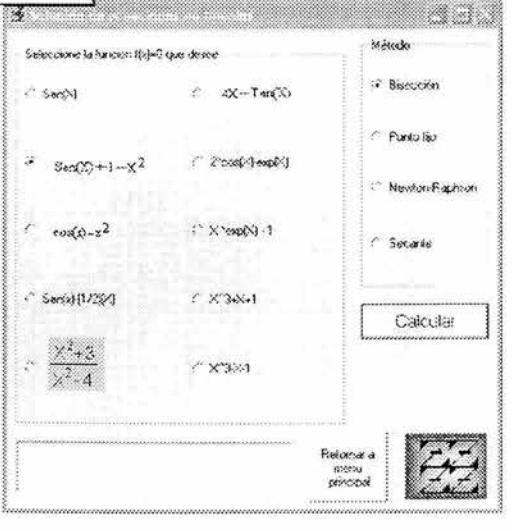


Después de introducir el valor del límite inferior, se solicitará el límite superior y el número máximo de iteraciones, esto se podría haber evitado incluyendo dentro del código de programación una cantidad fija y algunos criterios de paro, pero se prefirió dejar al usuario la libertad de determinarlo.

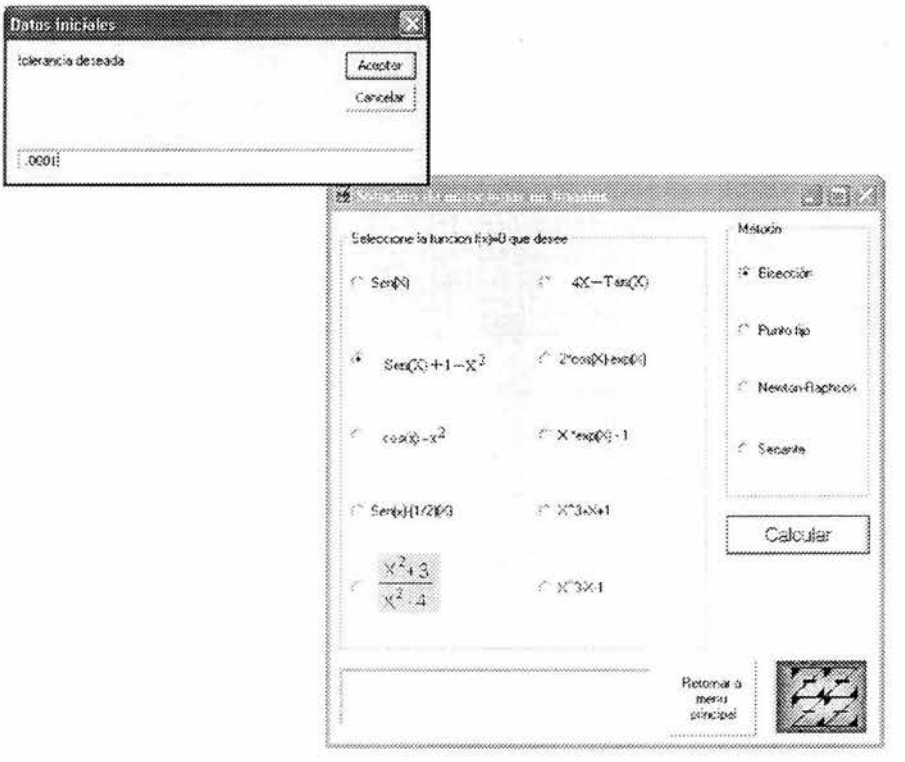

Datos iniciales
límite superior del intervalo
Aceptar
Cancelar
1.6


Seleccione la función f(x)=0 que desee
Método
Bisección
Punto fijo
Newton-Raphson
Secante
Calcular
Retornar a menú principal

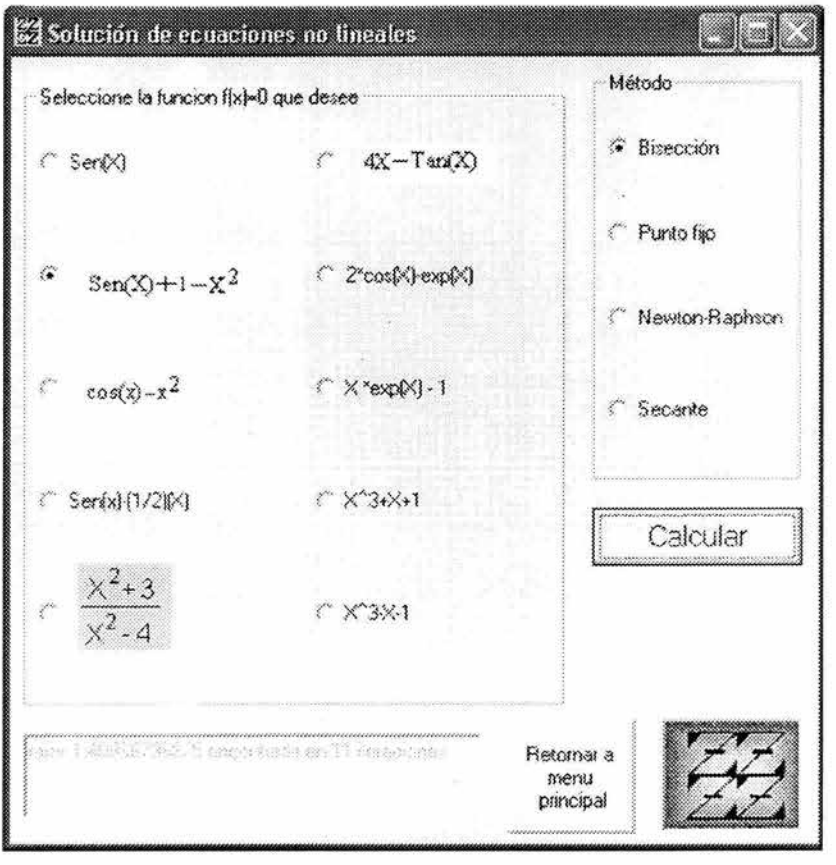

Datos iniciales
número máximo de iteraciones
Aceptar
Cancelar
100


Seleccione la función f(x)=0 que desee
Método
Bisección
Punto fijo
Newton-Raphson
Secante
Calcular
Retornar a menú principal

También se pedirá se introduzca el nivel de precisión deseado en los cálculos, es decir la tolerancia.



En el recuadro a la izquierda del botón de retorno a Menú Principal se mostrará el resultado (sea que se encontrará la raíz o no) y en caso de una solución positiva, también se mostrará el número de iteraciones en que fue posible detectarla.
Se optó por deshabilitar el cuadro de texto para evitar posible confusión al momento de introducir datos.



3.2. Ejemplos de resolución de matrices.

Ejemplo 1.

$3X_1$	$-6X_2$	$7X_3$	4
$8X_1$	0	$-5X_3$	19
X_1	$-2X_2$	$6X_3$	5

X_1	X_2	X_3	
3	-6	7	4
8	0	-5	19
1	-2	6	5

Se selecciona el elemento (2,1) como elemento pivote y se dividen todos los elementos entre 8

X_1	X_2	X_3	
3	-6	7	4
1	0	-0.625	2.375
1	-2	6	5

Ahora debemos convertir en ceros los elementos (1,1) y (3,1) que son los elementos de la misma columna de nuestro elemento pivote. Para ello multiplicamos por -3 los elementos del renglón 2 (de nuestro elemento pivote) y se suman con sus respectivos elementos del renglón 1.

X_1	X_2	X_3	
0	-6	8.875	-3.125
1	0	-0.625	2.375
1	-2	6	5

(-3)(2º. R)+(1er. R)

Para el 3er. Renglón multiplicamos por -1 el 2º. Renglón y se suma al 3ero.

X_1	X_2	X_3	
0	-6	8.875	-3.125
1	0	-0.625	2.375
0	-2	6.625	2.625

(-1)(2º. R)+(3er. R)

Después debemos seleccionar un nuevo elemento pivote que no debe estar en la 1er. Columna o el 2º. Renglón, ya que de ahí extrajimos el primer elemento pivote, por lo que nuestro siguiente elemento será el (1,3). Dividiendo los elementos del primer renglón entre 8.875 obtenemos:

X_1	X_2	X_3	
0	-0.676	1	-0.352
1	0	-0.625	2.375
0	-2	6.625	2.625

(1er. R)(1/8.875)

Después requerimos convertir los elementos (2,3) y (3,3) en ceros para ello multiplicamos el 1er. Renglón por 0.625 y se lo sumamos al 2º. Renglón. Después multiplicamos el 1er. Renglón por -6.625 y se lo sumamos al 3er. Renglón.

X_1	X_2	X_3	
0	-0.676	1	-0.352
1	-0.423	0	2.155
0	2.479	0	4.957

(1er. R)(0.625)+(2º. R)
(1er. R)(-6.625)+(3er. R)

Por ultimo, seleccionamos el elemento pivote que no se encuentre ni en el 1ero o 2º. Renglón ni en la 1ra. o 3era. columna, es decir el elemento (3,2). Procediendo en forma similar se multiplica el 3er. Renglón por (1/2.479).

X_1	X_2	X_3	
0	-0.676	1	-0.352
1	-0.423	0	2.155
0	1	0	2

(3er. R)(1/2.479)

Se multiplica el 3er. Renglón por 0.676 y se suma al 1er. Renglón. Se multiplica el 3er. Renglón por 0.423 y se suma al 2º. Renglón.

X_1	X_2	X_3	
0	0	1	1
1	0	0	3
0	1	0	2

(3er. R)(0.676)+(1er. R)
(3er. R)(0.423)+(2º. R)

Se reacomodan los renglones

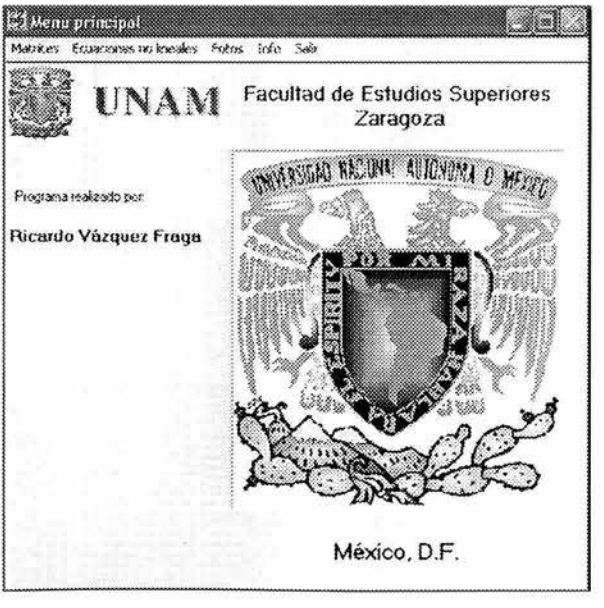
X_1	X_2	X_3	
1	0	0	3
0	1	0	2
0	0	1	1

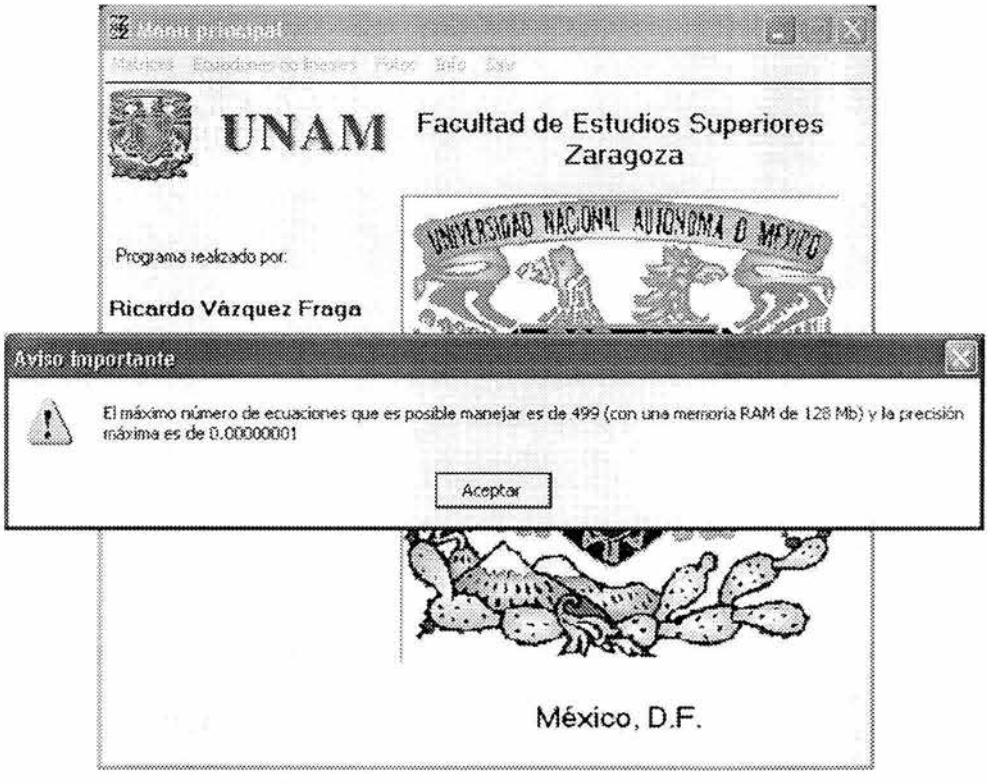
Por lo que la solución es:

$X_1 = 3$, $X_2 = 2$ y $X_3 = 1$

Empleando el programa:

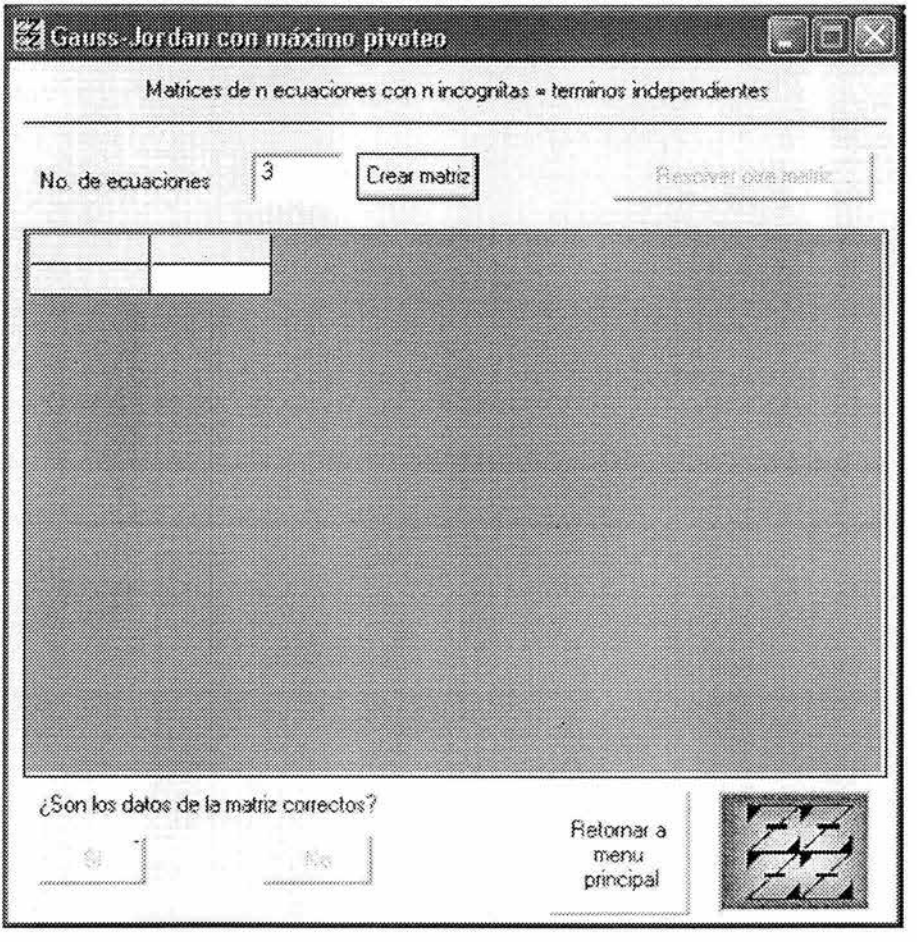
Se selecciona la opción "matrices" y luego en el submenu "Gauss-Jordan con pivoteo" se da clic, ello nos mostrará un mensaje sobre las limitantes del programa. Posteriormente debemos introducir le número de renglones de que constará la matriz e iremos insertando los valores según se nos requiera, por último después de corregir los posibles errores, se aceptará la matriz y se nos mostrará el resultado. Quedando en posibilidad de llevar a cabo otro cálculo.





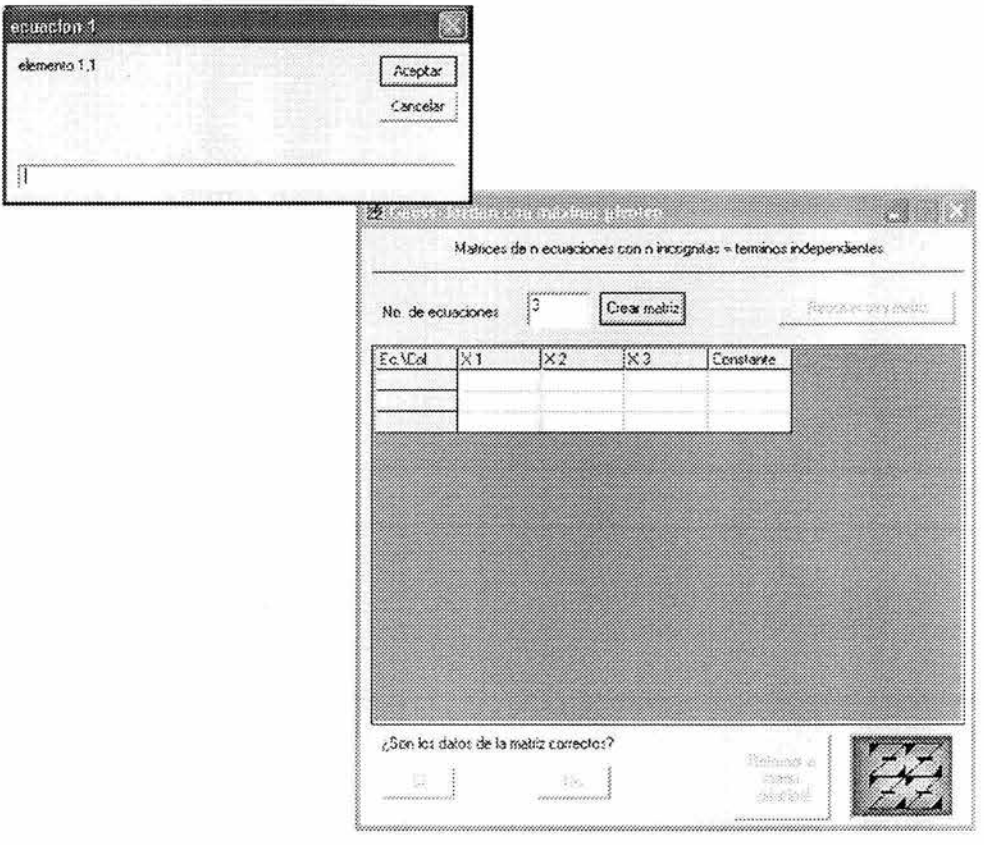
La limitación con respecto al máximo número de ecuaciones que se pueden resolver se relaciona directamente con la cantidad de memoria RAM del equipo de cómputo con el que se este trabajando, por ello se hace la observación que de acuerdo con pruebas realizadas se han podido resolver hasta un máximo de 499 ecuaciones con una memoria de 128 Mb, así que si se dispusiera de una cantidad mayor de RAM, sería posible resolver una mayor cantidad.

La relación entre memoria RAM y el número de ecuaciones no es lineal, así que el doble de memoria RAM no implica resolver el doble de ecuaciones; el número de cálculos se incrementa con el número de ecuaciones requeridas.



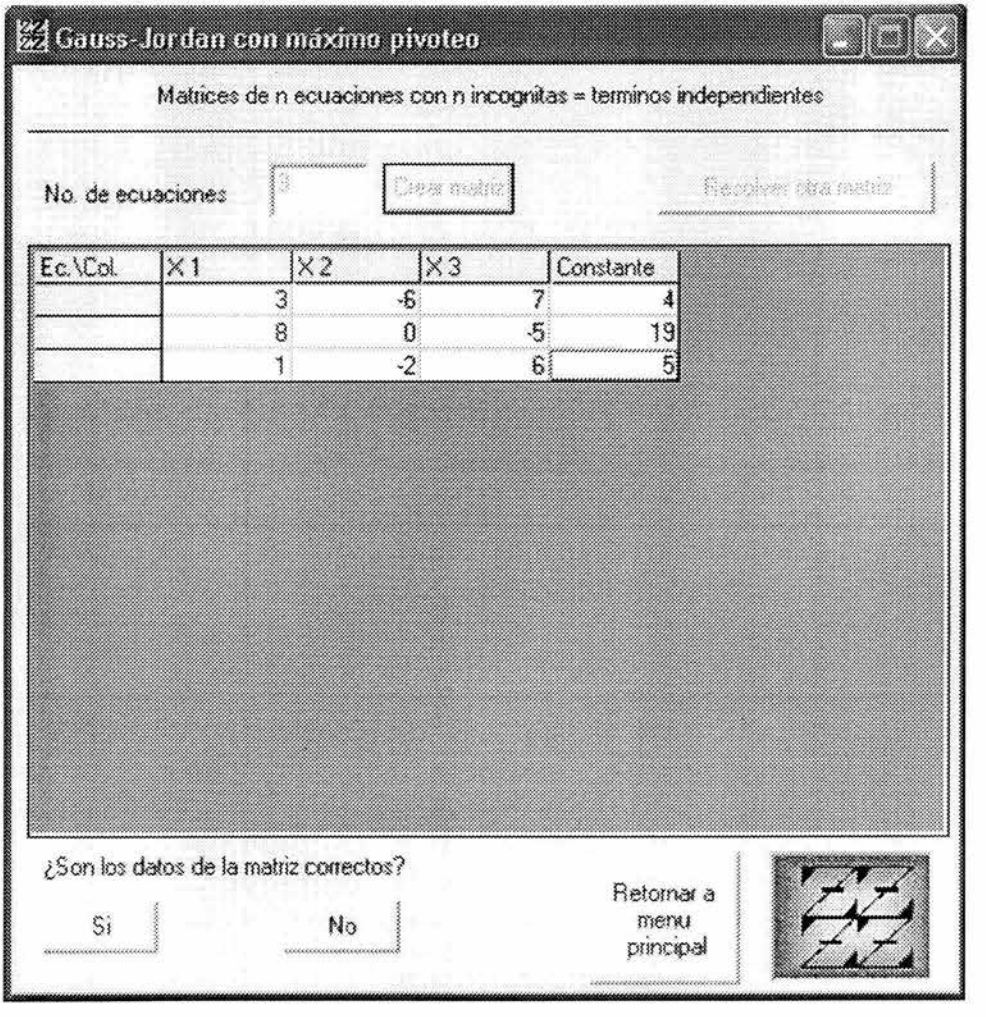
Después de introducir el número de ecuaciones, se dimensionara la matriz para permitir el ingreso de los elementos de la matriz, los cuales serán validados para impedir el ingreso de letras que podrían provocar errores en la lógica de la computadora.

Para evitar confusión, el ingreso de cada elemento se hará en un recuadro independiente en donde se hace notar de qué elemento se trata, p. e. el elemento (2,4) se refiere al elemento del renglón 2 y la columna 4

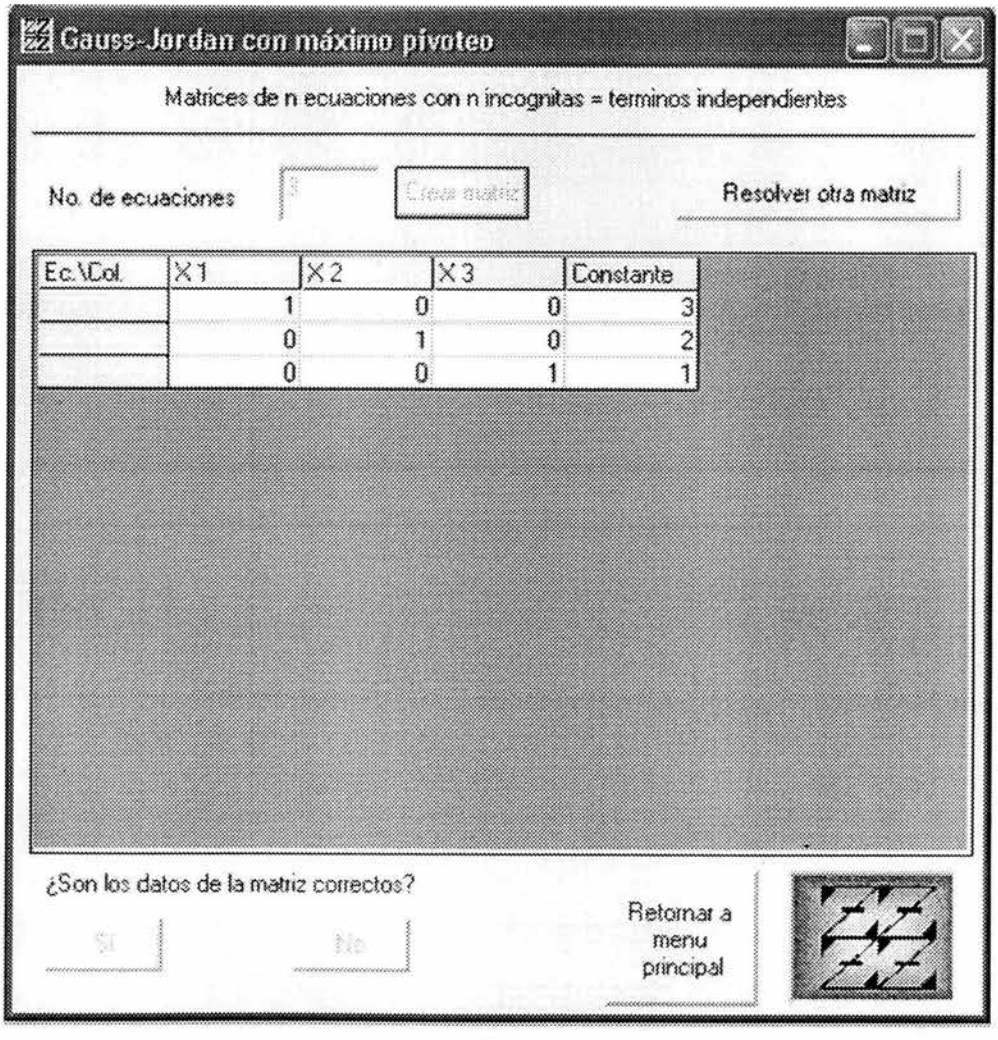


Una vez que se introducen los elementos, se pregunta si los datos son correctos, debido a que por error se pudo haber introducido un elemento erróneo y no por ello se tiene que capturar nuevamente la matriz entera.

Para hacer la corrección se pedirá ingresar la ubicación (coordenada) del elemento o elementos a corregir (en caso de ser más de uno, la modificación se hará uno por uno).



Una vez que se ingresó la matriz correctamente se procede a realizar el cálculo y los resultados nos son mostrados dentro de la misma matriz como si está fuese una matriz idéntica en donde el resultado está en la columna "constante".



Ejemplos de resolución de matrices.

Ejemplo 2.

$2X_1$	$6X_2$	$-X_3$	-12
$5X_1$	$-X_2$	$2X_3$	29
$-3X_1$	$-4X_2$	X_3	5

X_1	X_2	X_3	
2	6	-1	-12
5	-1	2	29
-3	-4	1	5

Se selecciona el elemento (1,2) como elemento pivote y se dividen todos los elementos entre 6

X_1	X_2	X_3	
0.33333333	1	-0.16666667	-2
5	-1	2	29
-3	-4	1	5

Ahora debemos convertir en ceros los elementos (2,2) y (3,2) que son los elementos de la misma columna de nuestro elemento pivote. Para ello sumamos los elementos del renglón 1 (de nuestro elemento pivote) con sus respectivos elementos del renglón 2.

X_1	X_2	X_3	
0.33333333	1	-0.16666667	-2
5.33333333	0	1.83333333	27
-3	-4	1	5

(1er. R) + (2°. R)

Para el 3er. Renglón multiplicamos por 4 el 1er. Renglón y se suma al 3ero.

X_1	X_2	X_3	
0.33333333	1	-0.16666667	-2
5.33333333	0	1.83333333	27
-1.66666667	0	0.33333332	-3

(4)(1er. R)+(3er. R)

Después debemos seleccionar un nuevo elemento pivote que no debe estar en la 2a. Columna o el 1er. Renglón, ya que de ahí extrajimos el primer elemento pivote, por lo

que nuestro siguiente elemento será el (2,1). Dividiendo los elementos del segundo renglón entre 5.33333333 obtenemos:

X_1	X_2	X_3	
0.33333333	1	-0.16666667	-2
1	0	0.34375	5.0625
-1.66666667	0	0.33333332	-3

(2°. R)(1/5.33333333)

Después requerimos convertir los elementos (1,1) y (3,1) en ceros para ello multiplicamos el 2o. Renglón por -0.33333333 y se lo sumamos al 1er. Renglón. Después multiplicamos el 2o. Renglón por 1.66666667 y se lo sumamos al 3er. Renglón.

X_1	X_2	X_3	
0	1	-0.28125735	-3.6875
1	0	0.34375	5.0625
0	0	0.90625	5.4375

(2o. R)(-0.33333333)+(1er. R)

(2o. R)(1.66666667)+(3er. R)

Por ultimo seleccionamos el elemento pivote que no se encuentre ni en el 1ero o 2°. Renglón ni en la 1ra. o 2a columna, es decir el elemento (3,3). Procediendo en forma similar se multiplica el 3er. Renglón por (1/0.90625).

X_1	X_2	X_3	
0	1	-0.28125735	-3.6875
1	0	0.34375	5.0625
0	0	1	6

(3er. R)(1/0.90625)

Se multiplica el 3er. Renglón por 0.28125735 y se suma al 1er. Renglón. Se multiplica el 3er. Renglón por -0.34375 y se suma al 2°. Renglón.

X_1	X_2	X_3	
0	1	0	-2
1	0	0	3
0	0	1	6

(3er. R)(0.28125735)+(1er. R)

(3er. R)(-0.34375)+(2°. R)

Se reacomodan los renglones

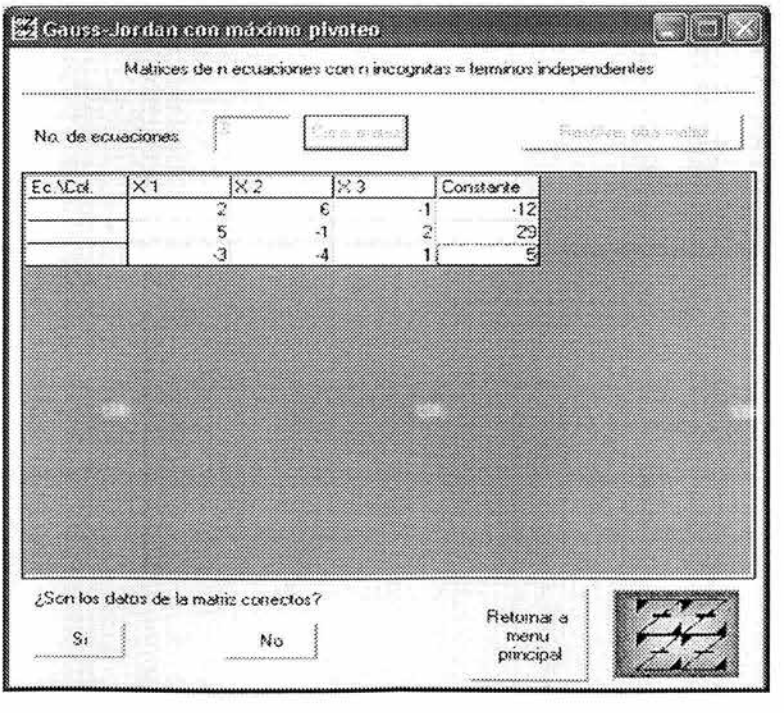
X_1	X_2	X_3	
1	0	0	3
0	1	0	-2
0	0	1	6

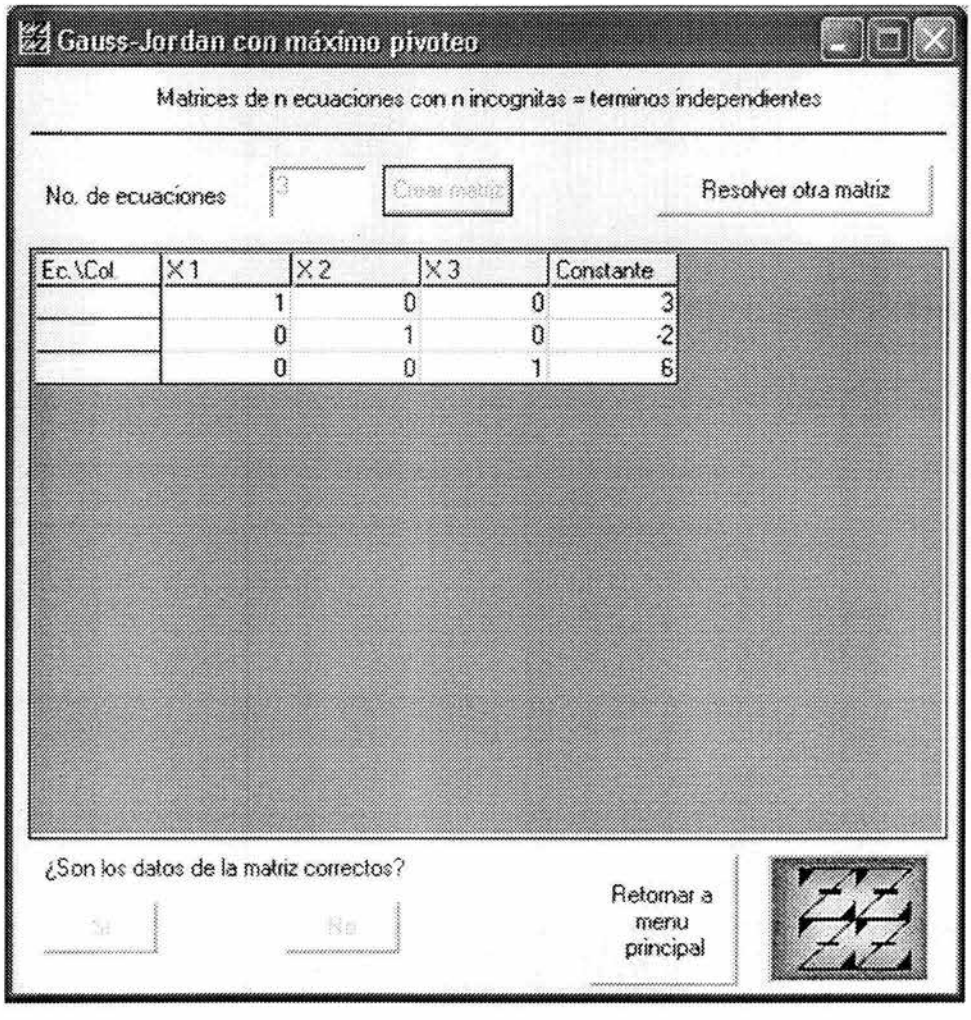
Por lo que la solución es:

$X_1 = 3$, $X_2 = -2$ y $X_3 = 6$

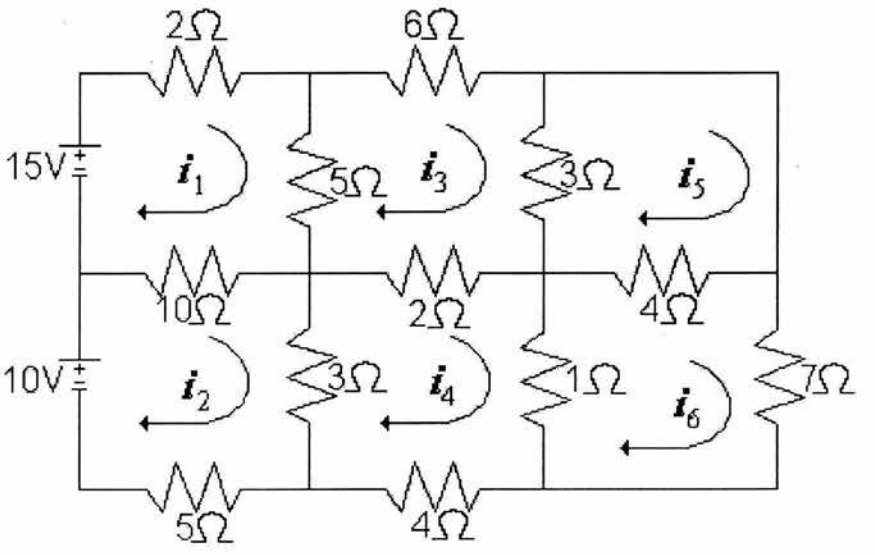
Como se pudo observar, fue necesario el emplear una mayor precisión en las cifras, de no haberlo hecho así, no se hubiese llegado al resultado. El emplear cifras más largas nos puede llevar a errores, es aquí en donde comienzan a mostrarse las ventajas del uso de la computadora.

Empleando el programa:




Gauss-Jordan con máximo pivoteo
Matrices de n ecuaciones con n incognitas = terminos independientes
No. de ecuaciones
Crear matriz
Resolver otra matriz
Ec.\Col
X 1
X 2
X 3
Constante
1 0 0 3
0 1 0 -2
0 0 1 6
¿Son los datos de la matriz correctos?
Si
No
Retornar a menu principal

Consideremos el siguiente problema de Ingeniería Eléctrica.

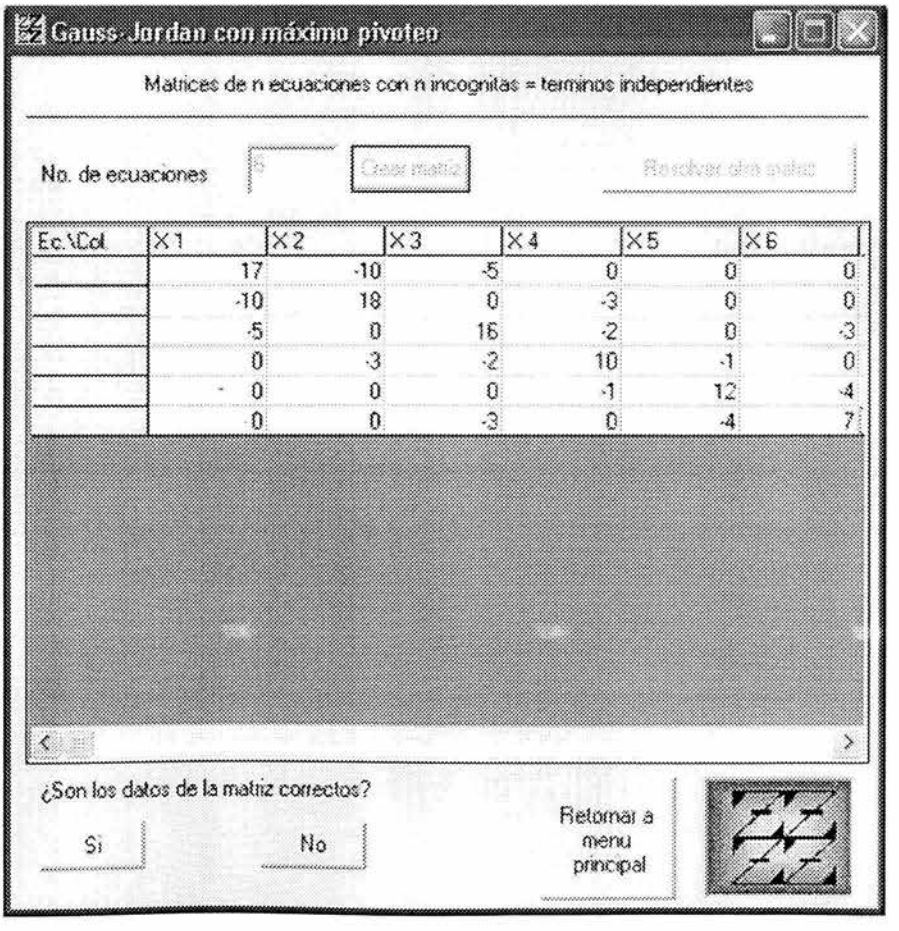


Se desea conocer la corriente que circula por cada rama de este circuito, para ello con ayuda de las Leyes de Kirchhoff estas corrientes se pueden conocer resolviendo el siguiente sistema de ecuaciones algebraicas lineales:

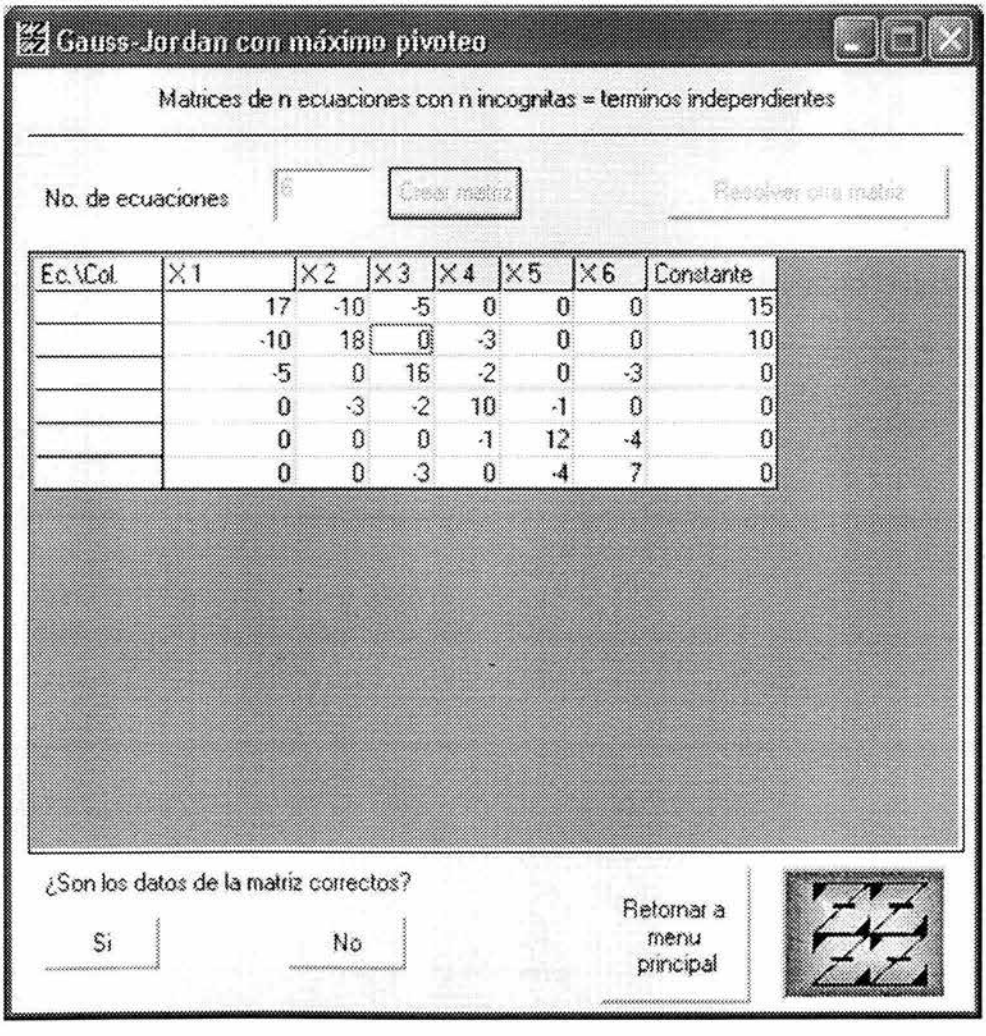
$17i_1$	$-10i_2$	$-5i_3$				=	15
$-10i_1$	$+18i_2$		$-3i_4$			=	10
$-5i_1$		$+16i_3$	$-2i_4$		$-3i_6$	=	0
	$-3i_2$	$-2i_3$	$+10i_4$	$-i_5$		=	0
			$-i_4$	$+12i_5$	$-4i_6$	=	0
		$-3i_3$		$-4i_5$	$+7i_6$	=	0

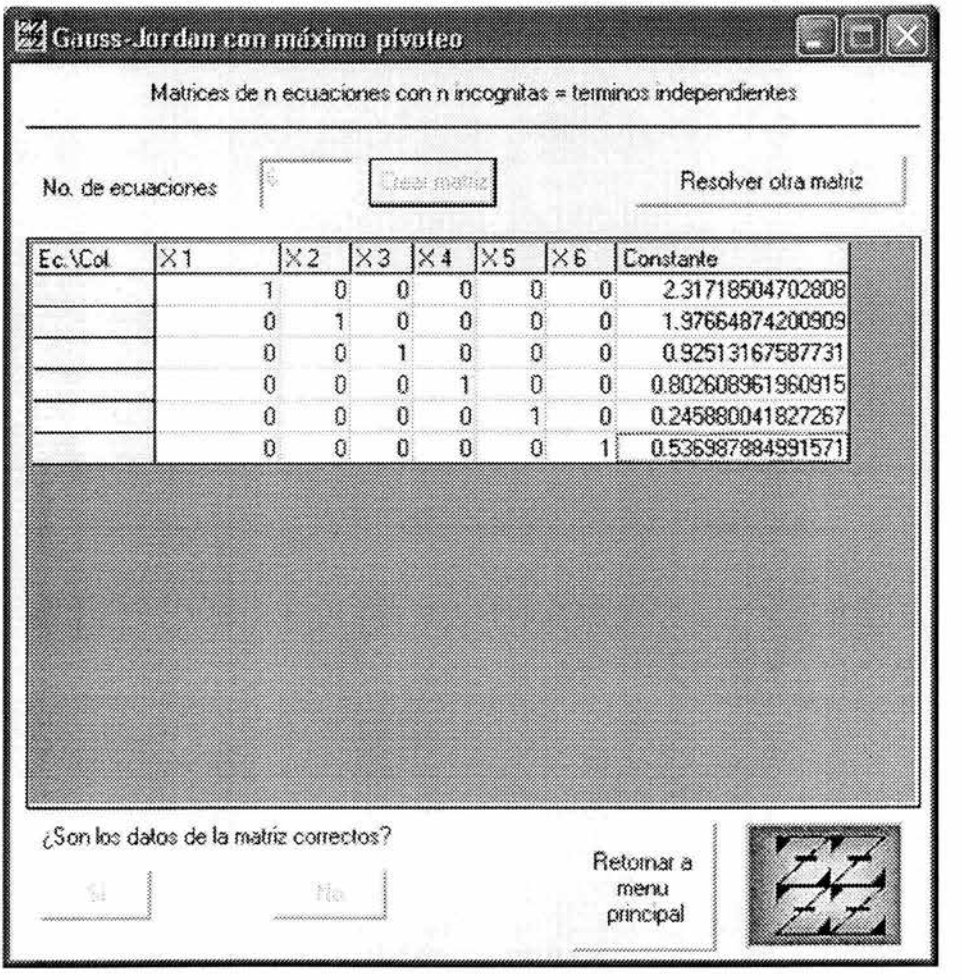
i_1	i_2	i_3	i_4	i_5	i_6		
17	-10	-5				=	15
-10	+18		-3			=	10
-5		+16	-2		-3	=	0
	-3	-2	+10	-1		=	0
			-1	+12	-4	=	0
		-3		-4	+7	=	0

Empleando el programa para resolver el sistema:



Como podemos notar, debido a la cantidad de columnas es necesario que si queremos ver todas las columnas desplacemos la barra de desplazamiento mostrada o que ajustemos el ancho de las columnas.





Como podemos ver el programa es de uso sencillo y nos guía paso a paso, al haber previsto que botones se pueden o no usar dependiendo de la situación.

3.3. Ejemplo de inversión de matriz.

Obtener la matriz inversa de:

3	4	-1
1	1	3
4	2	2

Se obtiene la matriz ampliada

3	4	-1	1	0	0
1	1	3	0	1	0
4	2	2	0	0	1

Se toma el elemento (1,2) como pivote, después de realizar las operaciones elementales indicadas se obtiene:

0.75	1	-0.25	0.25	0	0	(1er. R)(1/4)
0.25	0	3.25	-0.25	1	0	(1er. R)(-1)+(2º. R)
2.5	0	2.5	-0.5	0	1	(1er. R)(-2)+(3er. R)

El siguiente pivote debe ser seleccionado fuera del 1er. Renglón o la 2ª. Columna, para ello seleccionamos el elemento (2,3). Se realizan nuevamente las operaciones indicadas y se llega a:

0.77	1	0	0.23	0.08	0	(2o. R)(0.25)+(1er. R)
0.08	0	1	-0.08	0.31	0	(2o. R)(1/3.25)
2.3	0	0	-0.3	-0.78	1	(2o. R)(-2.5)+(3er. R)

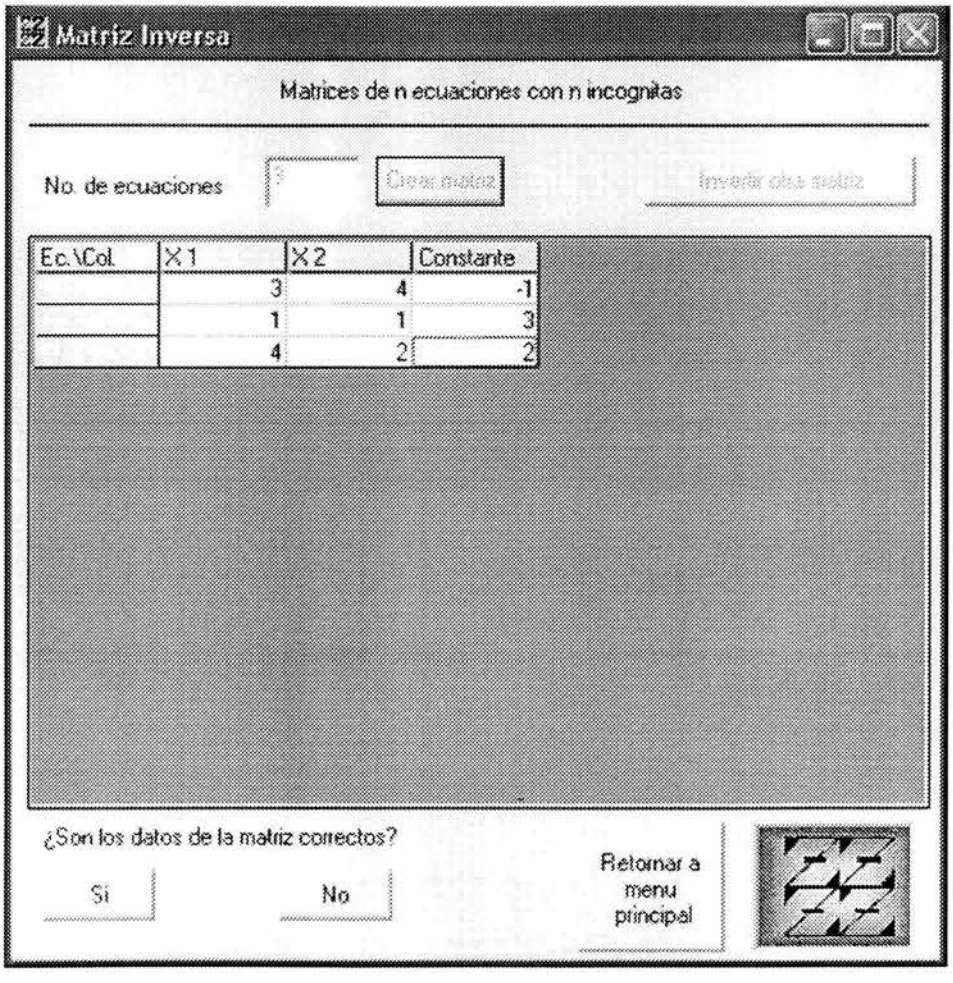
Finalmente tomamos el elemento pivote (3,1)

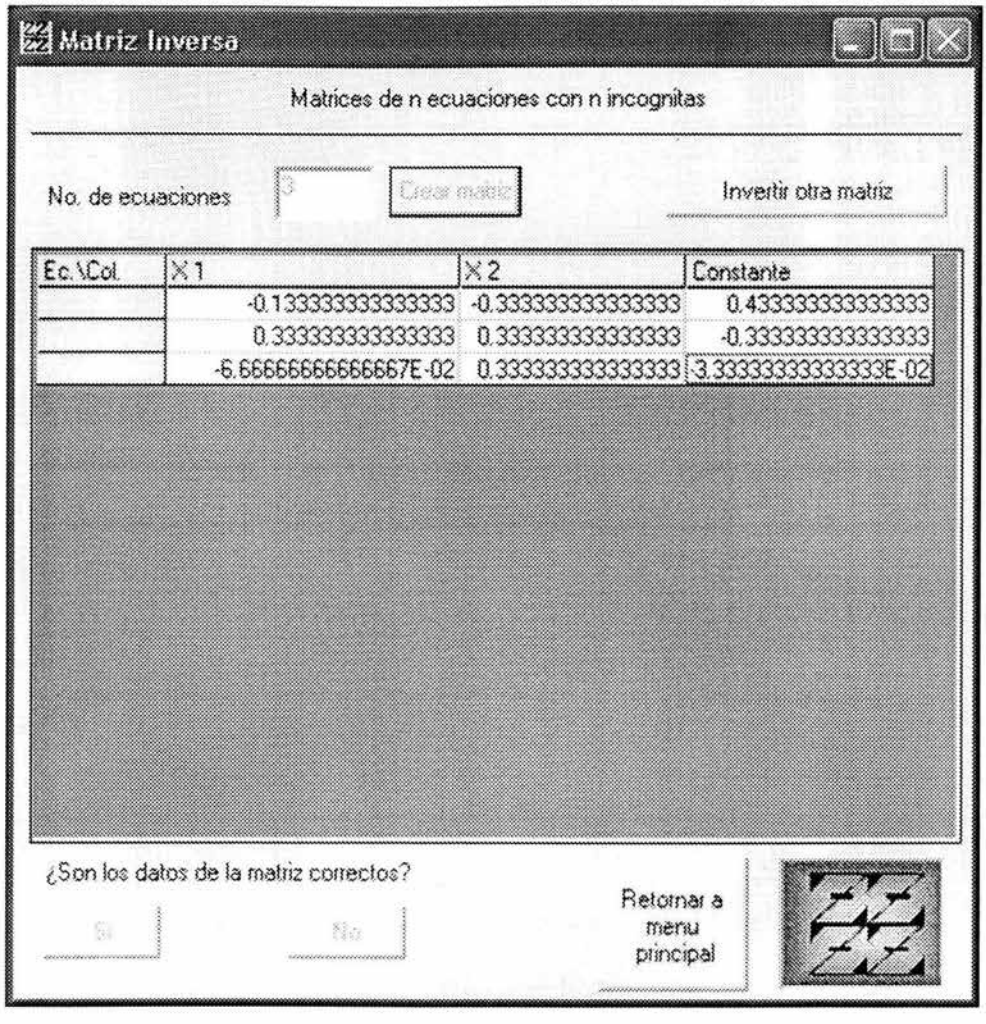
0	1	0	0.33	0.34	-0.33	(3er. R)(-0.77)+(1er. R)
0	0	1	-0.77	0.34	-0.03	(3er. R)(-0.08)+(2º. R)
1	0	0	-0.13	-0.34	0.43	(3er. R)(1/2.3)

Se reacomodan los renglones y se elimina la matriz identidad, obteniéndose la matriz inversa:

-0.13	-0.34	0.43
0.33	0.34	-0.33
-0.77	0.34	-0.03

Empleando el programa:





3.4. Ejemplos de resolución de ecuaciones diferenciales

a) Resolver la ecuación diferencial $y' - 2xy = x$ con condición inicial $y(0.5) = 1$, utilizando el método de Euler, en el intervalo $0.5 \leq x \leq 1.0$ con $h = 0.1$

partiendo de la ecuación diferencial de primer orden

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

se tiene que

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

empleando las condiciones iniciales (i = 0)

$y_0 = 1 \quad y \quad h = 0.1$

$f(x_0,y_0) = x_0 + 2x_0y_0$

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

$f(x_0,y_0) = 1.5$

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

$y_1 = 1 + 0.1(1.5)$

$y_1 = 1.15$

Se repite el proceso para (i = 1)

$y_2 = y_1 + h\, f(x_1,y_1)$

donde

$y_1 = 1.15 \quad y \quad h = 0.1$

$f(x_1,y_1) = x_1 + 2x_1y_1$

$f(x_1,y_1) = 0.6 + 2(0.6)\,(1.15)$

$f(x_1,y_1) = 1.98$

$y_2 = y_1 + h\, f(x_1,y_1)$

$y_2 = 1.15 + 0.1(1.98)$

$y_2 = 1.348$

En forma similar

i = 2 $y_3 = y_2 + h\, f(x_2,y_2)$

$y_3 = 1.348 + 0.1[0.7 + 2(0.7)(1.348)]$

$y_3 = 1.60672$

i = 3 $y_4 = y_3 + h\, f(x_3,y_3)$

$y_4 = 1.60672 + 0.1[0.8 + 2(0.8)(1.60672)]$

$y_4 = 1.94380$

i = 4 $y_5 = y_4 + h\, f(x_4,y_4)$

$y_5 = 1.94380 + 0.1[0.9 + 2(0.9)(1.94380)]$

$y_5 = 2.38368$

Por lo que la solución de la ecuación diferencial en el intervalo $0.5 \leq x \leq 1.0$ es:

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

b) Se tiene un círculo definido por la ecuación $x_2 + y_2 = 4$, tomando x=1 se despeja el valor de $y = \sqrt{3}$. Con lo cual se tiene la condición inicial (1, $\sqrt{3}$)

Mediante el método de Euler, ¿cual sería el valor de *y* en x = 0.5 ?

$x_0 = 1$, $y_0 = \sqrt{3}$, h = -0.1

$f(x_0, y_0)$ = y´(x) = $-\frac{y}{x}$

$f(x_0, y_0)$ = -($\sqrt{3}$ /1)

$y = y_0 + hf(x_0, y_0)$

y = $\sqrt{3}$ +(-0.1)(- $\sqrt{3}$)

y = $\sqrt{3}$ (1+0.1) = $\sqrt{3}$ (1.1)

y = 1.9053

$x^2 + y^2 = 4$ con x = 0.5

$(0.5)^2 + y^2 = 4$

y2 = 4 – 0.25 = $\sqrt{3.75}$

y = 1.9365

c) Resolver la ecuación diferencial y´- 2xy = x con condición inicial y(0.5) = 1, utilizando el método de Euler modificado, en el intervalo $0.5 \leq x \leq 1.0$ con h = 0.1

La ecuación diferencial es:

y´(x) = x + 2xy con condición inicial y(0.5) = 1

para i = 0

$y_0 = 1$ y $h = 0.1$

$f(x_0,y_0) = x_0 + 2x_0y_0$

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

$f(x_0,y_0) = 1.5$

$y_{1p} = y_0 + h\, f(x_0,y_0)$

$y_{1p} = 1 + 0.1(1.5)$

$y_{1p} = 1.15$

y la corrección de este valor es:

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

donde $f(x_0,y_0) = 1.5$

$f(x_1,y_{1p}) = x_1 + 2x_1y_{1p}$

$f(x_1,y_{1p}) = 0.6 + 2(0.6)(1.15)$

$f(x_1,y_{1p}) = 1.98$

Se sustituye:

$$y_{1_c} = 1 + \frac{0.1}{2}\left[1.5 + 1.98\right]$$

$y_{1c} = 1.174$

para i = 1

$y_1 = 1.174$ y $h = 0.1$

$f(x_1,y_1) = x_1 + 2x_1y_1$

$f(x_1,y_1) = 0.6 + 2(0.6)(1.174)$

$f(x_1,y_1) = 2.0088$

$y_{2p} = y_1 + h\, f(x_1,y_1)$

$y_{2p} = 1.174 + 0.1(2.0088)$

$y_{2p} = 1.37488$

y la corrección de este valor es:

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

donde $f(x_1,y_1) = 2.0088$

$f(x_2,y_{2p}) = x_2 + 2x_2y_{2p}$

$f(x_2,y_{2p}) = 0.7 + 2(0.7)(1.37488)$

$f(x_2,y_{2p}) = 2.624832$

Se sustituye:

$$y_{2_c} = 1.174 + \frac{0.1}{2}\left[2.0088 + 2.624832\right]$$

$y_{2c} = 1.40568$

para i = 2

$y_2 = 1.40568$ y $h = 0.1$

$f(x_2,y_2) = x_2 + 2x_2y_2$

$f(x_2,y_2) = 0.7 + 2(0.7)(1.40568)$

$f(x_2,y_2) = 2.667952$

$y_{3p} = y_2 + h\, f(x_2,y_2)$

$y_{3p} = 1.40568 + 0.1(2.667952)$

$y_{3p} = 1.6724752$

y la corrección de este valor es:

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

donde $f(x_2,y_2) = 2.667952$

$f(x_3,y_{3p}) = x_3 + 2x_3y_{3p}$

$f(x_3,y_{3p}) = 0.8 + 2(0.8)(1.6724752)$

$f(x_3,y_{3p}) = 3.47596032$

Se sustituye:

$$y_{3_c} = 1.40568 + \frac{0.1}{2}\left[2.667952 + 3.47596032\right]$$

$y_{3c} = 1.712875616$

para i = 3

$y_3 = 1.71288$ y $h = 0.1$

$f(x_3,y_3) = x_3 + 2x_3y_3$

$f(x_3,y_3) = 0.8 + 2(0.8)(1.71288)$

$f(x_3,y_3) = 3.540608$

$y_{4p} = y_3 + h\, f(x_3,y_3)$

$y_{4p} = 1.71288 + 0.1(3.540608)$

$y_{4p} = 2.0669408$

y la corrección de este valor es:

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

donde $f(x_3,y_3) = 3.540608$

$f(x_4,y_{4p}) = x_4 + 2x_4y_{4p}$

$f(x_4,y_{4p}) = 0.9 + 2(0.9)(2.0669408)$

$f(x_4,y_{4p}) = 4.62049344$

Se sustituye:

$$y_{4_c} = 1.71288 + \frac{0.1}{2}\left[3.540608 + 4.62049344\right]$$

$y_{4c} = 2.120935072$

para i = 4

$y_4 = 2.12094$ y $h = 0.1$

$f(x_4,y_4) = x_4 + 2x_4y_4$

$f(x_4,y_4) = 0.9 + 2(0.9)(2.12094)$

$f(x_4,y_4) = 4.717692$

$y_{5p} = y_4 + h\, f(x_4,y_4)$

$y_{5p} = 2.12094 + 0.1(4.717692)$

$y_{5p} = 2.5927092$

y la corrección de este valor es:

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

donde $f(x_4,y_4) = 4.717692$

$f(x_5,y_{5p}) = x_5 + 2x_5y_{5p}$

$f(x_5,y_{5p}) = 1 + 2(1)(2.5927092)$

$f(x_5,y_{5p}) = 6.1854184$

Se sustituye:

$$y_{5_c} = 2.12094 + \frac{0.1}{2}\left[4.717692 + 6.1854184\right]$$

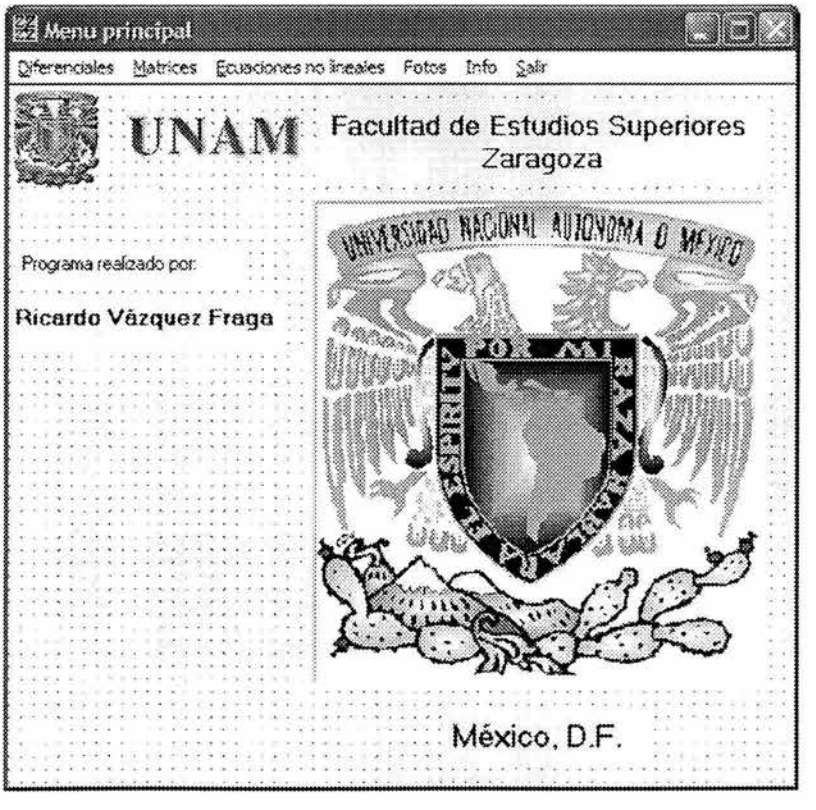
$y_{5c} = 2.66609552$

Concentrando los resultados en la siguiente tabla:

i	x	y_{ip}	y_{ic}
0	0.5	1.00000	1.00000
1	0.6	1.15000	1.17400
2	0.7	1.37488	1.40568
3	0.8	1.67248	1.71288
4	0.9	2.06694	2.12094
	1.0	2.59271	2.66610

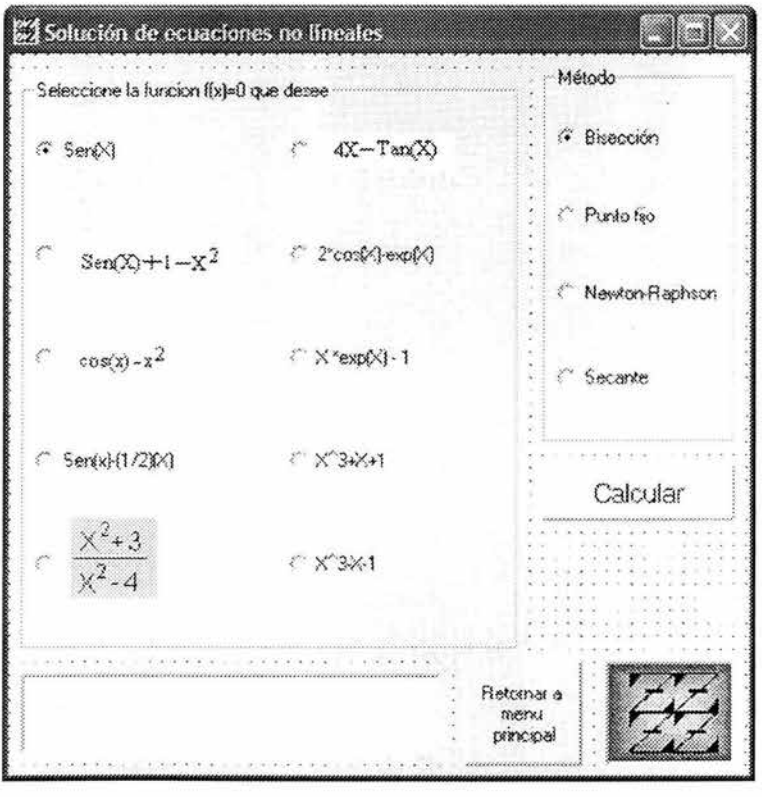
3.5. Listado del programa.

A continuación se presenta la impresión del código de programación junto con las interfases gráficas de usuario a las que pertenecen.


Menu principal
Diferenciales Matrices Ecuaciones no lineales Fotos Info Salir
UNAM
Facultad de Estudios Superiores
Zaragoza
Programa realizado por:
Ricardo Vázquez Fraga
México, D.F.

menu - 1

```
Private Sub about_Click()
MsgBox "Programa de métodos numéricos elaborado por Ricardo Vázquez Fraga con Asesoria del Ing.
Quim. José Antonio Zamora Plata como parte del proyecto de Tesis; México D.F. -2003-", 64, "Info
rmación"
End Sub

Private Sub biseccion_Click()
ecnolineal.Show
End Sub

Private Sub EM_Click()
EulerModi.Show
End Sub

Private Sub eu_Click()
Euler.Show
End Sub

Private Sub inv_Click()
inversa.Show
End Sub

Private Sub MGJ_Click()
gaussj.Show
End Sub

Private Sub newton_Click()
ecnolineal.Show
End Sub

Private Sub pics_Click()
Fotos.Show
End Sub

Private Sub puntofijo_Click()
ecnolineal.Show
End Sub

Private Sub rk_Click()
Runge.Show
End Sub

Private Sub Salir_Click()
End
End Sub

Private Sub secante_Click()
ecnolineal.Show
End Sub
```


Solución de ecuaciones no lineales
Seleccione la funcion f(x)=0 que desee
Sen(X)
4X−Tan(X)
Sen(X)+1−X^2
2*cos(X)-exp(X)
cos(x)-x^2
X*exp(X)-1
Sen(x)-(1/2)X
X^3+X+1
(X^2+3)/(X^2-4)
X^3-X-1
Método
Bisección
Punto fijo
Newton-Raphson
Secante
Calcular
Retornar a menu principal

```
ecnolineal - 1

Private Sub cmdmenu_Click()
ecnolineal.Hide
Unload ecnolineal
menu.Show
End Sub

Private Sub Command1_Click()
'metodo de bisección
If d = 0 Then
   li# = InputBox("limite inferior del intervalo", "Datos iniciales", , 1, 1)
   ls# = InputBox("limite superior del intervalo", "Datos iniciales", , 1, 1)
   ni% = InputBox("numero maximo de iteraciones", "Datos iniciales", , 1, 1)
   td# = InputBox("tolerancia deseada", "Datos iniciales", , 1, 1)
   For z = 1 To ni
      X2# = (li + ls) / 2
      Select Case e
         Case 0
            fli = Sin(li): fx2 = Sin(X2)
         Case 1
            fli = Sin(li) + 1 - (li) ^ 2: fx2 = Sin(X2) + 1 - (X2) ^ 2
         Case 2
            fli = Cos(li) - (li) ^ 2: fx2 = Cos(X2) - (X2) ^ 2
         Case 3
            fli = Sin(li) - (li / 2): fx2 = Sin(X2) - (X2 / 2)
         Case 4
            fli = ((li ^ 2) + 3) / ((li ^ 2) - 4): fx2 = (((X2) ^ 2) + 3) / (((X2) ^ 2) - 4)
         Case 5
            fli = 4 * li - Tan(li): fx2 = 4 * X2 - Tan(X2)
         Case 6
            fli = 2 * Cos(li) - Exp(li): fx2 = 2 * Cos(X2) - Exp(X2)
         Case 7
            fli = li * Exp(li) - 1: fx2 = X2 * Exp(X2) - 1
         Case 8
            fli = (((li) ^ 3) + (li) + 1): fx2 = (X2) ^ 3 + (X2) + 1
         Case 9
            fli = (((li) ^ 3) - (li) - 1): fx2 = (X2) ^ 3 - (X2) - 1
      End Select
      If (fli * fx2) > 0 Then
         li = X2
      Else
         ls = X2
      End If
      X1# = (li + ls) / 2
      er = Abs((X1 - X2) / X1)
      If er <= td Then
         r = "raiz= " & X1 & " encontrada en " & z & " iteraciones"
         GoTo respuesta:
      End If
      X1 = X2
    Next z
    r = "no converge en " & ni & " iteraciones"
respuesta:
Text1.Text = r
End If
'método de punto fijo
If d = 1 Then
   x0# = InputBox("aproximacion inicial", "Datos iniciales", , 1, 1)
   ni% = InputBox("numero maximo de iteraciones", "Datos iniciales", , 1, 1)
   td# = InputBox("tolerancia deseada", "Datos iniciales", , 1, 1)
   x00# = x0
   For z = 1 To ni
         Select Case e
         Case 0
            fx0 = Sin(x0): gx0 = Cos(x0)
         Case 1
            fx0 = Sin(x0) + 1 - (x0) ^ 2: gx0 = Cos(x0) - 2 * x0
         Case 2
            fx0 = Cos(x0) - (x0) ^ 2: gx0 = -Sin(x0) - 2 * x0
         Case 3
```

```
ecnolineal - 2

            fx0 = Sin(x0) - (x0 / 2): gx0 = Cos(x0) - 0.5
         Case 4
            fx0 = (((x0) ^ 2) + 3) / (((x0) ^ 2) - 4): gx0 = ((-14) * (x0)) / (((x0) ^ 2 - 4) ^
2)
         Case 5
            fx0 = 4 * x0 - Tan(x0): gx0 = 4 - (1 / (Cos(x0) * Cos(x0)))
         Case 6
            fx0 = 2 * Cos(x0) - Exp(x0): gx0 = -2 * Sin(x0) - Exp(x0)
         Case 7
            fx0 = x0 * Exp(x0) - 1: gx0 = Exp(-x0)
         Case 8
            fx0 = (((x0) ^ 3) + (x0) + 1): gx0 = 3 * x0 ^ 2 + 1
         Case 9
            fx0 = (((x0) ^ 3) - (x0) - 1): gx0 = 3 * x0 ^ 2 - 1
      End Select
      X2 = gx0
      X1 = X2
      er = Abs((X1 - x0) / X1)
      If er <= td Then
         r = "raiz= " & X1 & " encontrada en " & z & " iteraciones"
         GoTo respuestal:
      End If
      x0 = X1
      If x0 > (3 * x00) Then
         r = "el valor calculado esta divergiendo de el valor de la raiz"
         GoTo respuestal:
      End If
   Next z
   r = "no converge en " & ni & " iteraciones"
respuestal:
Text1.Text = r
End If
'método de newton-raphson
If d = 2 Then
   x0# = InputBox("aproximacion inicial", "Datos iniciales", , 1, 1)
   ni% = InputBox("numero maximo de iteraciones", "Datos iniciales", , 1, 1)
   td# = InputBox("tolerancia deseada", "Datos iniciales", , 1, 1)
   For z = 1 To ni
   Select Case e
         Case 0
            fx0 = Sin(x0): gx0 = Cos(x0)
         Case 1
            fx0 = Sin(x0) + 1 - (x0) ^ 2: gx0 = Cos(x0) - 2 * x0
         Case 2
            fx0 = Cos(x0) - (x0) ^ 2: gx0 = -Sin(x0) - 2 * x0
         Case 3
            fx0 = Sin(x0) - (x0 / 2): gx0 = Cos(x0) - 0.5
         Case 4
            fx0 = ((x0) ^ 2 + 3) / ((x0) ^ 2 - 4): gx0 = ((-14) * (x0)) / ((((x0) ^ 2) - 4) ^ 2
         Case 5
            fx0 = 4 * x0 - Tan(x0): gx0 = 4 - (1 / (Cos(x0) * Cos(x0)))
         Case 6
            fx0 = 2 * Cos(x0) - Exp(x0): gx0 = -2 * Sin(x0) - Exp(x0)
         Case 7
            fx0 = x0 * Exp(x0) - 1: gx0 = Exp(-x0)
         Case 8
            fx0 = (((x0) ^ 3) + (x0) + 1): gx0 = 3 * x0 ^ 2 + 1
         Case 9
            fx0 = (((x0) ^ 3) - (x0) - 1): gx0 = 3 * x0 ^ 2 - 1
      End Select
      X2 = x0 - fx0 / gx0
      x0 = X1: X1 = X2
      er = Abs((X1 - x0) / X1)
      If er <= td Then
         r = "raiz= " & X1 & " encontrada en " & z & " iteraciones"
         GoTo respuesta2:
      End If
   Next z
   r = "no converge en " & ni & " iteraciones"
```

```
ecnolineal - 3

respuesta2:
Text1.Text = r
End If
'método de la secante
If d = 3 Then
   x0# = InputBox("aproximacion inicial 1", "Datos iniciales", , 1, 1)
   X1# = InputBox("aproximacion inicial 2", "Datos iniciales", , 1, 1)
   ni# = InputBox("numero maximo de iteraciones", "Datos iniciales", , 1, 1)
   td# = InputBox("tolerancia deseada", "Datos iniciales", , 1, 1)
   For z = 1 To ni
   Select Case e
         Case 0
            fx1 = Sin(X1): fx1 = Sin(x0)
         Case 1
            fx1 = Sin(X1) + 1 - (X1) ^ 2: fx0 = Sin(x0) + 1 - (x0) ^ 2
         Case 2
            fx1 = Cos(X1) - (X1) ^ 2: fx0 = Cos(x0) - (x0) ^ 2
         Case 3
            fx1 = Sin(X1) - (X1 / 2): fx0 = Sin(x0) - (x0 / 2)
         Case 4
            fx1 = (((X1) ^ 2) + 3) / (((X1) ^ 2) - 4): fx0 = (((x0) ^ 2) + 3) / (((x0
         Case 5
            fx1 = 4 * X1 - Tan(X1): fx0 = 4 * x0 - Tan(x0)
         Case 6
            fx1 = 2 * Cos(X1) - Exp(X1): fx0 = 2 * Cos(x0) - Exp(x0)
         Case 7
            fx1 = X1 * Exp(X1) - 1: fx0 = x0 * Exp(x0) - 1
         Case 8
            fx1 = (((X1) ^ 3) + (X1) + 1): fx0 = (((x0) ^ 3) + (x0) + 1)
         Case 9
            fx1 = (((X1) ^ 3) - (X1) - 1): fx0 = (((x0) ^ 3) - (x0) - 1)
      End Select
      If fx1 <= td Then
         r = "raiz= " & X1 & " encontrada en " & z & " iteraciones"
         GoTo respuesta3:
      End If
      X2 = X1 - (X1 - x0) * fx1 / (fx1 - fx0)
      x0 = X1: X1 = X2
   Next z
   r = "no converge en " & ni & " iteraciones"
respuesta3:
Text1.Text = r
End If
End Sub

Private Sub Form_Load()
Dim e, d As Integer
End Sub

Private Sub Option1_Click(Index As Integer)
Select Case Index
    Case 0
       e = 0
    Case 1
       e = 1
    Case 2
       e = 2
    Case 3
       e = 3
    Case 4
       e = 4
    Case 5
       e = 5
    Case 6
       e = 6
    Case 7
       e = 7
    Case 8
       e = 8
```

```
ecnolineal - 4

    Case 9
       e = 9
    End Select
End Sub

Private Sub Option2_Click(Index As Integer)
Select Case Index
   Case 0
      d = 0
   Case 1
      d = 1
   Case 2
      d = 2
   Case 3
      d = 3
End Select
End Sub
```

Module1 - 1

```
'dimensionamiento de las variables para el trabajo con matrices
Public a() As Double, b() As Double, c() As Double, x() As Double, e, d As Integer
```


Gauss-Jordan con máximo pivoteo
Matrices de n ecuaciones con n incognitas = terminos independientes
No. de ecuaciones
Crear matriz
Resolver otra matriz
¿Son los datos de la matriz correctos?
Si
No
Retornar a menu principal

gaussj - 1

```
Private Sub Cmdcrear_Click()
'Generacion de la matriz e introduccion de datos
cmdmenu.Enabled = False
MSFlexGrid1.Row = 0
MSFlexGrid1.Col = 0
MSFlexGrid1.Text = "Ec.\Col."
Do ' validacion del numero de ecuaciones
If IsNumeric(txtecuaciones.Text) Then
   n = txtecuaciones.Text
   flag = 1
   nn$ = Int(n)
   If nn$ <> n Then
      txtecuaciones.Text = InputBox("necesita introducir solo números enteros", "Aviso")
      n = txtecuaciones.Text
      txtecuaciones.SetFocus
      Cmdcrear.Enabled = False
      flag = 0
   Else
      flag = 1
   End If
Else
   MsgBox "necesita introducir solo números", 48, "Aviso"
   txtecuaciones.SetFocus
   Cmdcrear.Enabled = False
End If
If n <= 0 Then
   txtecuaciones.Text = InputBox("No se aceptan numeros negativos", "Aviso")
   flag = 0
End If
If n >= 500 Then
   txtecuaciones.Text = InputBox("No es posible generar una matriz de ese tamaño", "Aviso"
   flag = 0
End If

Loop While flag <> 1
'redimensionamiento de las matrices
ReDim a(n, 2 * n + 1), b(n, n + 1), c(n), x(n)
MSFlexGrid1.Rows = n + 1
MSFlexGrid1.Cols = n + 2
z% = 0
For y% = 1 To n
      MSFlexGrid1.Row = z%
      MSFlexGrid1.Col = y%
      MSFlexGrid1.Text = "X " & y%
Next y%
MSFlexGrid1.Row = 0
MSFlexGrid1.Col = n + 1
MSFlexGrid1.Text = "Constante"
For z% = 1 To n    'Introduccion y validacion de datos
   Title$ = "ecuacion " & z%
   For y% = 1 To n    'elementos
      title2$ = "elemento " & z% & "," & y%
      Do
        temp = InputBox(title2$, Title$, , 1, 1)
        If IsNumeric(temp) Then
           flag = 1
        Else
           If temp = "" Then    'cancelacion del proceso para introducir valores
              flagcancel = 1
              GoTo cancelado:
           End If
           MsgBox "Introduzca solo números", 48, "Aviso"
           flag = 0
        End If
      Loop While flag <> 1
      b(z%, y%) = temp: flag = 0
      MSFlexGrid1.Row = z%
      MSFlexGrid1.Col = y%
      MSFlexGrid1.Text = b(z%, y%)
```

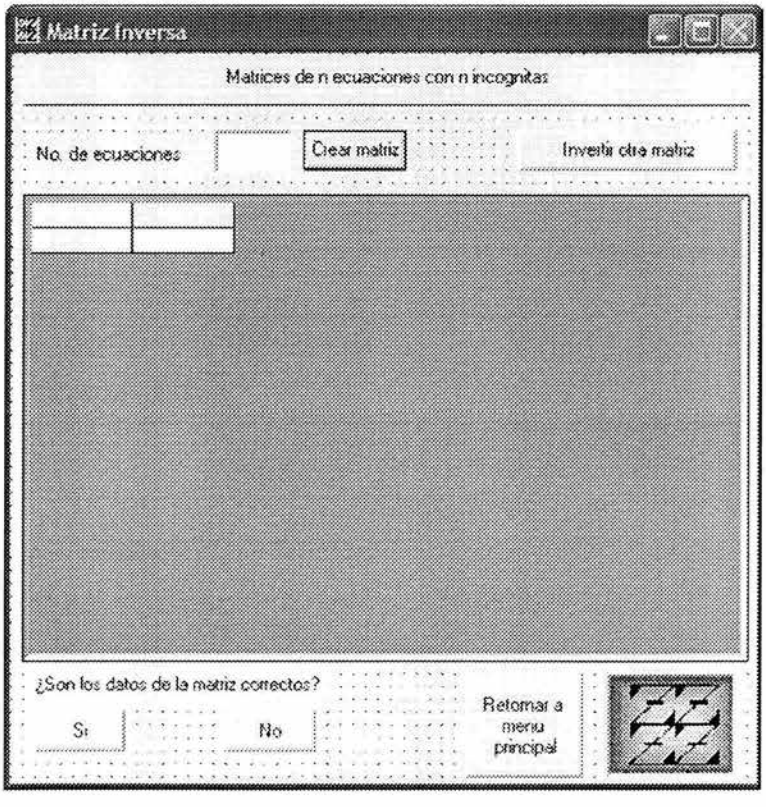
```
    Next y%
    title2$ = "constante " & z%    'constante
       Do
         temp = InputBox(title2$, Title$, , 1, 1)
         If IsNumeric(temp) Then
            flag = 1
         Else
            MsgBox "Introduzca solo números", 48, "Aviso"
            flag = 0
         End If
       Loop While flag <> 1
    b(z%, n + 1) = temp: flag = 0
    MSFlexGrid1.Row = z%
    MSFlexGrid1.Col = y%
    MSFlexGrid1.Text = b(z%, y%)
Next z%
cancelado:
If flagcancel = 1 Then
    Cmdsi.Enabled = False
    Cmdno.Enabled = False
    cmdmenu.Enabled = True
Else
    txtecuaciones.Enabled = False
    Cmdcrear.Enabled = False
    cmdmenu.Enabled = True
    Cmdsi.Enabled = True
    Cmdno.Enabled = True
End If
End Sub

Private Sub cmdmenu_Click()
n = txtecuaciones.Text
If txtecuaciones.Text = "" Then
    n = 2
    ReDim a(n, 2 * n + 1), b(n, n + 1), c(n), x(n)
    MSFlexGrid1.Rows = n + 1
    MSFlexGrid1.Cols = n + 2
End If
For z% = 1 To n
    For y% = 1 To n + 1
       a(z%, y%) = 0: b(z%, y%) = 0
    Next y%
Next z%
For z% = 1 To n
    For y% = 1 To n + 1
       MSFlexGrid1.Row = z%
       MSFlexGrid1.Col = y%
       MSFlexGrid1.Text = " "
    Next y%
Next z%
Cmdcrear.Enabled = True
txtecuaciones.Text = ""
txtecuaciones.Enabled = True
txtecuaciones.SetFocus
Cmdotra.Enabled = False
gaussj.Hide
Unload gaussj
menu.Show
End Sub

Private Sub Cmdno_Click()
' correcion de datos en la matriz
      Do
         templ = InputBox("¿ecuacion a modificar?", "Corrección", , 1, 1)
         If IsNumeric(templ) Then
            flag = 1
            If templ > txtecuaciones.Text Or templ < 1 Then
               aviso$ = "Ecuaciones disponibles: " & txtecuaciones.Text
               MsgBox aviso$, 48, "Aviso"
```

```
            flag = 0
         Else
            flag = 1
         End If
      Else
         MsgBox "Introduzca solo números enteros", 48, "aviso"
         flag = 0
      End If
   Loop While flag <> 1
   flag = 0
   Do
      temp2 = InputBox("¿columna a modificar?", "Corrección", , 1, 1)
      If IsNumeric(temp2) Then
         flag = 1
         If temp2 > txtecuaciones.Text + 1 Or temp2 < 1 Then
            aviso$ = "Columnas disponibles: " & txtecuaciones.Text + 1
            MsgBox aviso$, 48, "Aviso"
            flag = 0
         Else
            flag = 1
         End If
      Else
         MsgBox "Introduzca solo números enteros", 48, "aviso"
         flag = 0
      End If
   Loop While flag <> 1
   title2$ = "elemento " & temp1 & "," & temp2: flag = 0
   Do
     temp = InputBox(title2$, Title$, , 1, 1)
     If IsNumeric(temp) Then
        flag = 1
     Else
        MsgBox "Introduzca solo números", 48, "Aviso"
        flag = 0
     End If
   Loop While flag <> 1
b(temp1, temp2) = temp: flag = 0
MSFlexGrid1.Row = temp1
MSFlexGrid1.Col = temp2
MSFlexGrid1.Text = b(temp1, temp2)
End Sub

Private Sub Cmdotra_Click()
n = txtecuaciones.Text
For z% = 1 To n
   For y% = 1 To n + 1
      a(z%, y%) = 0: b(z%, y%) = 0
   Next y%
Next z%
For z% = 1 To n
   For y% = 1 To n + 1
      MSFlexGrid1.Row = z%
      MSFlexGrid1.Col = y%
      MSFlexGrid1.Text = " "
   Next y%
Next z%
Cmdcrear.Enabled = True
txtecuaciones.Text = ""
txtecuaciones.Enabled = True
Cmdotra.Enabled = False
End Sub

Private Sub Cmdsi_Click()
Cmdno.Enabled = False
Cmdsi.Enabled = False
'Se genera la matriz A como copia de B
n = Val(txtecuaciones.Text)
For z% = 1 To n
   For y% = 1 To n + 1
```

gaussj - 4

```
      a(z%, y%) = b(z%, y%)
   Next y%
Next z%
'generacion de coeficientes de la diagonal o pivotes
maxpivot# = 0.00000000000001: Row = 1: Col = 1
For j = 1 To n
   maxpivot# = 0.00000000000001
'  deteccion del maximo pivote
   For z = n To j Step -1
      If maxpivot# < Abs(a(z, j)) Then
         maxpivot# = Abs(a(z, j)): Row = z
      End If
   Next z
   If Abs(maxpivot) < 0.00000001 Then
      MsgBox "Elemento pivote menor a 0.00000001", 48, "Aviso"
      MsgBox "la Matriz podria ser singular. De rango " & z% - 1, 48, "aviso"
      MsgBox "Fin del programa", 48, "Aviso"
      End
   End If
'  intercambio de renglones
   For z = 1 To (n + 1)
      a(0, z) = a(j, z)
      a(j, z) = a(Row, z)
      a(Row, z) = a(0, z)
   Next z
'  eliminacion hacia adelante
   For zz = 1 To (n - j)
   If j < n Then
      a(0, 0) = a(j + zz, j) / a(j, j)
      For z = 1 To (n + zz)
         a(j + zz, z) = a(j + zz, z) - a(j, z) * a(0, 0)
      Next z
   Else
      a(0, 0) = a(n, n)
      For z = 1 To (n + 1)
         a(j, z) = a(j, z) / a(0, 0)
      Next z
   End If
   Next zz
'  eliminacion de elementos no cero (errores de redondeo)
   For zz = j - 1 To 0 Step -1
      If a(j, zz) < 0.1 Then
         a(j, zz) = 0
      End If
   Next zz
   maxpivot# = 0.000000000000001
Next j
'sustitucion hacia atras
For j = n To 1 Step -1
   If j = n Then
      a(0, 0) = 0
   Else
      For z = j + 1 To n
         a(0, 0) = a(0, 0) - a(j, z) * a(z, z)
      Next z
   End If
   a(j, j) = (a(j, n + 1) + a(0, 0)) / a(j, j)
   a(0, 0) = 0
Next j
For j = n To 1 Step -1
   a(j, n + 1) = a(j, j)
Next j
'  igualar a cero los elementos posteriores a la diagonal
For j = 1 To n
   For zz = j To n
      If j = zz Then
         a(j, zz) = 1
      Else
         a(j, zz) = 0
```

```
      End If
   Next zz
Next j
'salida de resultados
For z% = 1 To n
   For y% = 1 To n + 1
      MSFlexGrid1.Row = z%
      MSFlexGrid1.Col = y%
      MSFlexGrid1.Text = a(z%, y%)
   Next y%
Next z%
Cmdotra.Enabled = True
End Sub
Private Sub Form_Load()
aviso1$ = "El máximo número de ecuaciones que es posible manejar es de 499 (con una memoria RAM
de 128 Mb)"
aviso2$ = " y la precisión máxima es de 0.00000001"
aviso$ = aviso1$ & aviso2$
MsgBox aviso$, 48, "Aviso importante"
End Sub


Private Sub txtecuaciones_Change()
Cmdcrear.Enabled = True
End Sub

Private Sub txtecuaciones_KeyDown(KeyCode As Integer, Shift As Integer)
If KeyCode = vbEnter Then
   Cmdcrear.Enabled = True
   Cmdcrear.SetFocus
End If
End Sub
```


Matriz Inversa
Matrices de n ecuaciones con n incognitas
No. de ecuaciones
Crear matriz
Invertir otra matriz
¿Son los datos de la matriz correctos?
Si
No
Retomar a menu principal

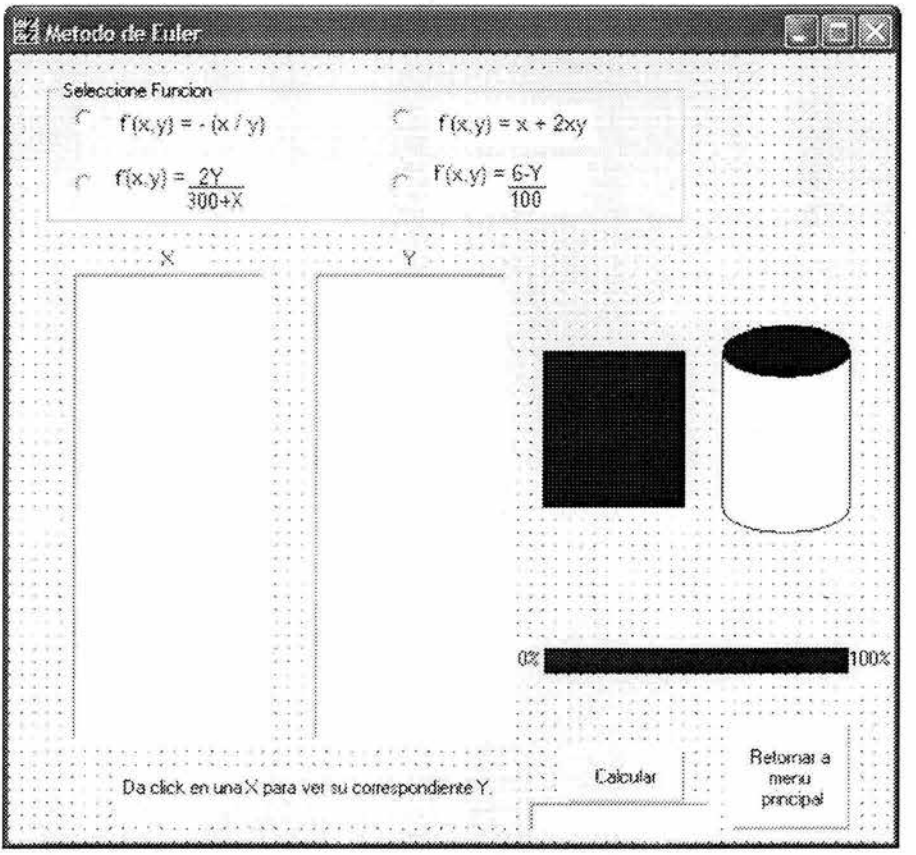
```
inversa - 1

Private Sub Cmdcrear_Click()
'Generacion de la matriz e introduccion de datos
cmdmenu.Enabled = False
MSFlexGrid1.Row = 0
MSFlexGrid1.Col = 0
MSFlexGrid1.Text = "Ec.\Col."
Do ' validacion del numero de ecuaciones
If IsNumeric(txtecuaciones.Text) Then
   n = txtecuaciones.Text
   flag = 1
   nn$ = Int(n)
   If nn$ <> n Then
      txtecuaciones.Text = InputBox("necesita introducir solo números enteros", "Aviso")
      n = txtecuaciones.Text
      txtecuaciones.SetFocus
      Cmdcrear.Enabled = False
      flag = 0
   Else
      flag = 1
   End If
Else
   MsgBox "necesita introducir solo números", 48, "Aviso"
   txtecuaciones.SetFocus
   Cmdcrear.Enabled = False
End If
If n <= 0 Then
   txtecuaciones.Text = InputBox("No se aceptan numeros negativos", "Aviso")
   flag = 0
End If
If n >= 500 Then
   txtecuaciones.Text = InputBox("No es posible generar una matriz de ese tamaño", "Aviso")
   flag = 0
End If

Loop While flag <> 1
'redimensionamiento de las matrices
ReDim a(n, 2 * n), b(n, n), c(n), x(n)
MSFlexGrid1.Rows = n + 1
MSFlexGrid1.Cols = n + 1
z% = 0
For y% = 1 To n
      MSFlexGrid1.Row = z%
      MSFlexGrid1.Col = y%
      MSFlexGrid1.Text = "X " & y%
Next y%
MSFlexGrid1.Row = 0
MSFlexGrid1.Col = n
MSFlexGrid1.Text = "Constante"
For z% = 1 To n    'Introduccion y validacion de datos
   Title$ = "ecuacion " & z%
   For y% = 1 To n    'elementos
      title2$ = "elemento " & z% & "," & y%
      Do
        temp = InputBox(title2$, Title$, , 1, 1)
        If IsNumeric(temp) Then
           flag = 1
        Else
           If temp = "" Then    'cancelacion del proceso para introducir valores
              flagcancel = 1
              GoTo cancelado:
           End If
           MsgBox "Introduzca solo números", 48, "Aviso"
           flag = 0
        End If
      Loop While flag <> 1
      b(z%, y%) = temp: flag = 0
      MSFlexGrid1.Row = z%
      MSFlexGrid1.Col = y%
      MSFlexGrid1.Text = b(z%, y%)
```

```
inversa - 2

   Next y%
Next z%
cancelado:
If flagcancel = 1 Then
   Cmdsi.Enabled = False
   Cmdno.Enabled = False
   cmdmenu.Enabled = True
Else
   txtecuaciones.Enabled = False
   Cmdcrear.Enabled = False
   cmdmenu.Enabled = True
   Cmdsi.Enabled = True
   Cmdno.Enabled = True
End If
End Sub

Private Sub cmdmenu_Click()
n = txtecuaciones.Text
If txtecuaciones.Text = "" Then
   n = 2
   ReDim a(n, 2 * n + 1), b(n, n + 1), c(n), x(n)
   MSFlexGrid1.Rows = n + 1
   MSFlexGrid1.Cols = n + 2
End If
For z% = 1 To n
   For y% = 1 To n
      a(z%, y%) = 0: b(z%, y%) = 0
   Next y%
Next z%
For z% = 1 To n
   For y% = 1 To n
      MSFlexGrid1.Row = z%
      MSFlexGrid1.Col = y%
      MSFlexGrid1.Text = " "
   Next y%
Next z%
Cmdcrear.Enabled = True
txtecuaciones.Text = ""
txtecuaciones.Enabled = True
txtecuaciones.SetFocus
Cmdotra.Enabled = False
inversa.Hide
Unload inversa
menu.Show
End Sub

Private Sub Cmdno_Click()
' correcion de datos en la matriz
      Do
         templ = InputBox("¿ecuacion a modificar?", "Corrección", , 1, 1)
         If IsNumeric(templ) Then
            flag = 1
            If templ > txtecuaciones.Text Or templ < 1 Then
               aviso$ = "Ecuaciones disponibles: " & txtecuaciones.Text
               MsgBox aviso$, 48, "Aviso"
               flag = 0
            Else
               flag = 1
            End If
         Else
            MsgBox "Introduzca solo números enteros", 48, "aviso"
            flag = 0
         End If
      Loop While flag <> 1
      flag = 0
      Do
         temp2 = InputBox("¿columna a modificar?", "Corrección", , 1, 1)
         If IsNumeric(temp2) Then
            flag = 1
```

```
inversa = 3

            If temp2 > txtecuaciones.Text + 1 Or temp2 < 1 Then
               aviso$ = "Columnas disponibles: " & txtecuaciones.Text + 1
               MsgBox aviso$, 48, "Aviso"
               flag = 0
            Else
               flag = 1
            End If
         Else
            MsgBox "Introduzca solo números enteros", 48, "aviso"
            flag = 0
         End If
      Loop While flag <> 1
      title2$ = "elemento " & templ & "," & temp2: flag = 0
      Do
        temp = InputBox(title2$, Title$, , 1, 1)
        If IsNumeric(temp) Then
           flag = 1
        Else
           MsgBox "Introduzca solo números", 48, "Aviso"
           flag = 0
        End If
      Loop While flag <> 1
   b(templ, temp2) = temp: flag = 0
   MSFlexGrid1.Row = templ
   MSFlexGrid1.Col = temp2
   MSFlexGrid1.Text = b(templ, temp2)
End Sub

Private Sub Cmdotra_Click()
n = txtecuaciones.Text
For z% = 1 To n
   For y% = 1 To n
      a(z%, y%) = 0: b(z%, y%) = 0
   Next y%
Next z%
For z% = 1 To n
   For y% = 1 To n
      MSFlexGrid1.Row = z%
      MSFlexGrid1.Col = y%
      MSFlexGrid1.Text = " "
   Next y%
Next z%
Cmdcrear.Enabled = True
txtecuaciones.Text = ""
txtecuaciones.Enabled = True
Cmdotra.Enabled = False
End Sub

Private Sub Cmdsi_Click()
Cmdno.Enabled = False
Cmdsi.Enabled = False
'Se genera la matriz A como copia de B
n = Val(txtecuaciones.Text)
For z% = 1 To n
   For y% = 1 To n
      a(z%, y%) = b(z%, y%)
   Next y%
'Next z%
'ampliando la matriz
For z% = 1 To n
   For y% = (n + 1) To (2 * n)
      If (n + z) = y Then
         a(z%, y%) = 1
      Else
         a(z%, y%) = 0
      End If
   Next y%
Next z%
'generacion de coeficientes de la diagonal o pivotes
```

```
inversa - 4

maxpivot# = 0.00000000000001: Row = 1: Col = 1
For j = 1 To n
   maxpivot# = 0.00000000000001
For z = 1 To n
   For zz = 1 To n
      b(z, zz) = a(z, zz)
   Next zz
Next z
'  deteccion del maximo pivote
   For z = 1 To n
      If maxpivot# < Abs(a(j, z)) Then
         maxpivot# = Abs(a(j, z)): Col = z
      End If
   Next z
   inv = 1 / a(j, Col)
   For z = 1 To (2 * n)
      a(j, z) = inv * a(j, z)
   Next z
   For z = 1 To n
      If j = z Then

      Else
         For zz = 1 To (2 * n)
            a(z, zz) = a(j, zz) * (-1) * b(z, Col) + a(z, zz)
         Next zz
      End If
   Next z
Next j
'reacomodo de renglones para obtener matriz identidad
maxpivot# = 0.00000000000001: Row = 1: Col = 1
For j = 1 To n
   maxpivot# = 0.00000000000001
'  deteccion del maximo pivote
   For z = n To j Step -1
      If maxpivot# < Abs(a(z, j)) Then
         maxpivot# = Abs(a(z, j)): Row = z
      End If
   Next z
'  intercambio de renglones
   For z = 1 To (2 * n)
      a(0, z) = a(j, z)
      a(j, z) = a(Row, z)
      a(Row, z) = a(0, z)
   Next z
Next j
'recuperacion de la matriz inversa
For z = 1 To n
   For zz = 1 To n
      b(z, zz) = a(z, zz + n)
   Next zz
Next z
'salida de resultados
For z% = 1 To n
   For y% = 1 To n
      MSFlexGrid1.Row = z%
      MSFlexGrid1.Col = y%
      MSFlexGrid1.Text = b(z%, y%)
   Next y%
Next z%
Cmdotra.Enabled = True
End Sub

Private Sub Form_Load()
aviso1$ = "El máximo número de ecuaciones que es posible manejar es de 499 (con una memoria RAM
de 128 Mb)"
aviso2$ = " y la precisión máxima es de 0.00000001"
aviso$ = aviso1$ & aviso2$
MsgBox aviso$, 48, "Aviso importante"
End Sub
```

```
inversa - 5


Private Sub txtecuaciones_Change()
Cmdcrear.Enabled = True
End Sub

Private Sub txtecuaciones_KeyDown(KeyCode As Integer, Shift As Integer)
If KeyCode = vbEnter Then
   Cmdcrear.Enabled = True
   Cmdcrear.SetFocus
End If
End Sub
```

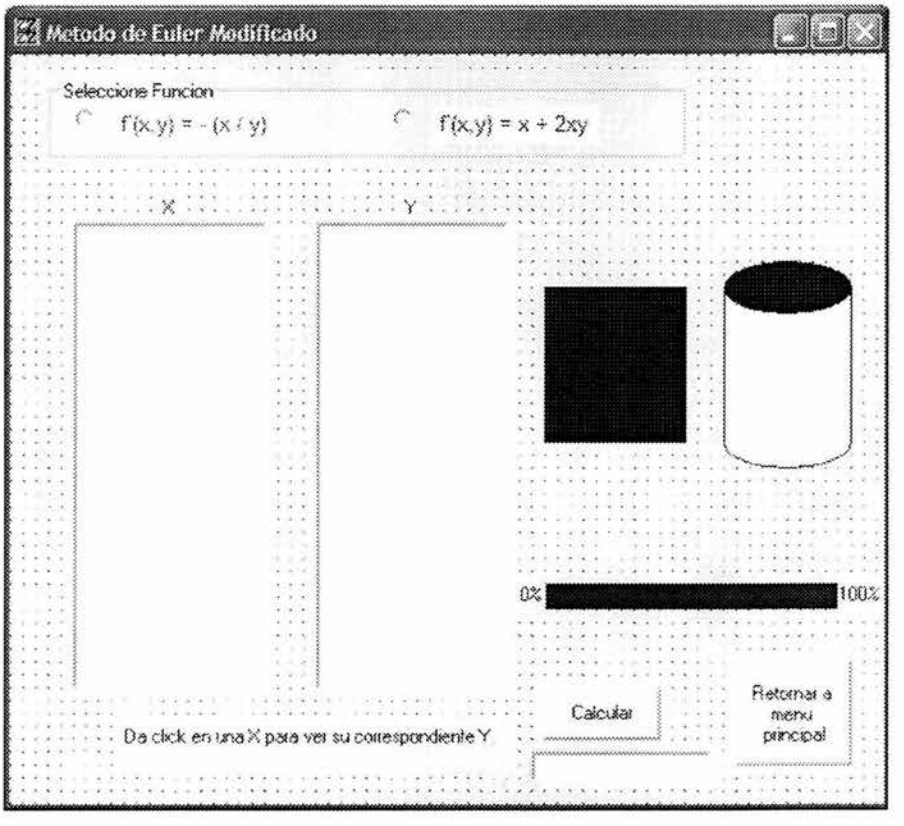

Metodo de Euler
Seleccione Funcion
f(x,y) = - (x / y)
f(x,y) = x + 2xy
f(x,y) = 2Y / 300+X
f(x,y) = 6-Y / 100
X
Y
0%
100%
Da click en una X para ver su correspondiente Y.
Calcular
Retornar a menu principal

```
Euler - 1

Private Sub cmdmenu_Click()
Euler.Hide
Unload Euler
menu.Show
End Sub

Private Sub Command1_Click()
List1.Clear
List2.Clear
i = 0
Do
   temp = InputBox("Xo", "Datos iniciales", , 1, 1)
   If IsNumeric(temp) Then
      flag = 1
   Else
      If temp = "" Then   'cancelacion del proceso para introducir valores
         GoTo Flag3:
      End If
      MsgBox "Introduzca solo números", 48, "Aviso"
      flag = 0
   End If
Loop While flag <> 1
x0# = temp: flag = 0
Do
   temp = InputBox("Yo", "Datos iniciales", , 1, 1)
   If IsNumeric(temp) Then
      flag = 1
   Else
      If temp = "" Then   'cancelacion del proceso para introducir valores
         GoTo Flag3:
      End If
      MsgBox "Introduzca solo números", 48, "Aviso"
      flag = 0
   End If
Loop While flag <> 1
Y0# = temp: flag = 0
Do
   temp = InputBox("h", "Datos iniciales", , 1, 1)
   If IsNumeric(temp) Then
      flag = 1
   Else
      If temp = "" Then   'cancelacion del proceso para introducir valores
         GoTo Flag3:
      End If
      MsgBox "Introduzca solo números", 48, "Aviso"
      flag = 0
   End If
Loop While flag <> 1
h! = temp: flag = 0
Do
   temp = InputBox("X final", "Datos iniciales", , 1, 1)
   If IsNumeric(temp) Then
      flag = 1
   Else
      If temp = "" Then   'cancelacion del proceso para introducir valores
         GoTo Flag3:
      End If
      MsgBox "Introduzca solo números", 48, "Aviso"
      flag = 0
   End If
Loop While flag <> 1
limite! = temp: flag = 0
List1.AddItem x0, i: List2.AddItem Y0, i
'las siguientes 2 lineas son para la animacion
inter = (limite - x0) / h
Text1.Text = inter
If x0 > limite Then flag2 = 1
If x0 < limite Then flag2 = 0
flag1:
```

Euler - 2

```
'proceso de calculo con la funcion seleccionada
If d = 0 Then
   X1 = x0 + h: i = i + 1
   'si cambia la funcion hay que cambiar la siguiente linea
   Y1 = Y0 - h * (x0 / Y0)
   List1.AddItem X1, i
   List2.AddItem Y1, i
   If Abs(X1 - limite) > Abs(x0 - limite) Then GoTo Flag3
   x0 = X1: Y0 = Y1
End If
If d = 1 Then
   X1 = x0 + h: i = i + 1
   'si cambia la funcion hay que cambiar la siguiente linea
   Y1 = Y0 + h * x0 * (2 * Y0 + 1)
   List1.AddItem X1, i
   List2.AddItem Y1, i
   If Abs(X1 - limite) > Abs(x0 - limite) Then GoTo Flag3
   x0 = X1: Y0 = Y1
End If
If d = 2 Then
   X1 = x0 + h: i = i + 1
   'si cambia la funcion hay que cambiar la siguiente linea
   Y1 = Y0 + h * (2 * Y0) / (300 + x0)
   List1.AddItem X1, i
   List2.AddItem Y1, i
   If Abs(X1 - limite) > Abs(x0 - limite) Then GoTo Flag3
   x0 = X1: Y0 = Y1
End If
If d = 3 Then
   X1 = x0 + h: i = i + 1
   'si cambia la funcion hay que cambiar la siguiente linea
   Y1 = Y0 + h * (6 - (Y0 / 100))
   List1.AddItem X1, i
   List2.AddItem Y1, i
   If Abs(X1 - limite) > Abs(x0 - limite) Then GoTo Flag3
   x0 = X1: Y0 = Y1
End If
If flag2 = 1 And X1 > limite Then GoTo flag1
If flag2 = 0 And X1 < limite Then GoTo flag1
Flag3:
End Sub

Private Sub Form_Load()
Dim x As Integer
End Sub

Private Sub List1_Click()
'con la siguiente linea se sincronizan los cuadros de lista
List2.ListIndex = List1.ListIndex
'las siguientes 4 lineas determinan los intervalos para la animacion grafica y el color
inter = Text1.Text
tono = 255 / inter
m = List1.ListIndex
cajacolor = 255 - (m * tono)
'se limpia la imagen del cuadro
For j = Shape5.Top To Shape5.Top + Shape5.Height Step 3
   For i = Shape5.Left To Shape5.Left + Shape5.Width Step 3
      PSet (i, j), &HC00000
   Next i
Next j
'calculo de los limites de las figuras
aa = Shape6.Top + Shape6.Height - 1
aaa = Shape6.Left + (m * (Shape6.Width / inter))
bb = m * ((Shape5.Width / 2) / inter)
bbb = (Shape5.Width / 2)
cc = m * ((Shape5.Height / 2) / inter)
ccc = Shape5.Top + Shape5.Height / 2
'se limpia la imagen del rectangulo
For i = Shape6.Left To Shape6.Left + Shape6.Width Step 3
```

```
Euler - 3

   For j = Shape6.Top To aa Step 3
      PSet (i, j), &HC00000
   Next j
Next i
'se colorea el cilindro
Shape2.FillColor = RGB(cajacolor, cajacolor, cajacolor)
Shape3.FillColor = RGB(cajacolor, cajacolor, cajacolor)
'se dibuja gradualmente el cuadrado
For j = (ccc) - Abs(cc) To ccc + Abs(cc) Step 3
   For i = Shape5.Left + bbb - bb To Shape5.Left + bbb + bb Step 3
      PSet (i, j), RGB(0, 0, 100)
   Next i
Next j
'Se marca el avance del calculo en el rectangulo
For i = Shape6.Left To aaa Step 3
   For j = Shape6.Top To aa Step 3
      PSet (i, j), RGB(0, 0, 100)
   Next j
Next i
End Sub

Private Sub Option1_Click(Index As Integer)
'reconocimiento de la funcion seleccionada
Select Case Index
   Case 0
      d = 0
   Case 1
      d = 1
   Case 2
      d = 2
   Case 3
      d = 3
End Select
End Sub
```

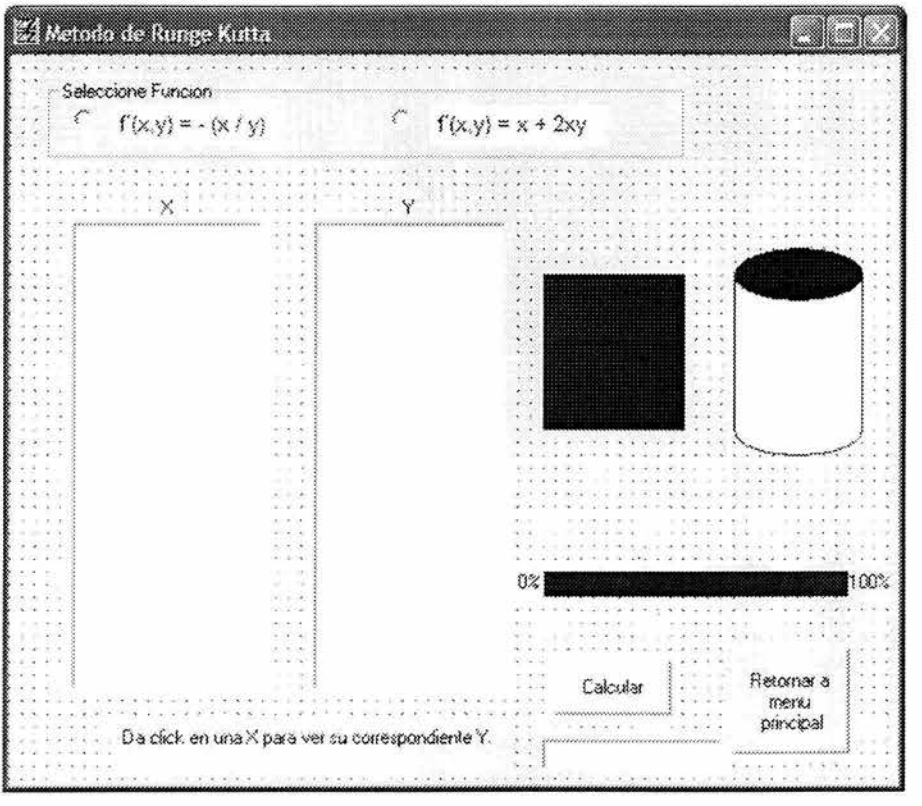

Metodo de Euler Modificado
Seleccione Funcion
f'(x,y) = - (x / y)
f'(x,y) = x + 2xy
X
Y
0%
100%
Da click en una X para ver su correspondiente Y
Calcular
Retornar a menu principal

```
EulerModi - 1

Private Sub cmdmenu_Click()
Euler.Hide
Unload Euler
menu.Show
End Sub

Private Sub Command1_Click()
List1.Clear
List2.Clear
i = 0
Do
   temp = InputBox("Xo", "Datos iniciales", , 1, 1)
   If IsNumeric(temp) Then
      flag = 1
   Else
      If temp = "" Then   'cancelacion del proceso para introducir valores
         GoTo Flag3:
      End If
      MsgBox "Introduzca solo números", 48, "Aviso"
      flag = 0
   End If
Loop While flag <> 1
x0# = temp: flag = 0
Do
   temp = InputBox("Yo", "Datos iniciales", , 1, 1)
   If IsNumeric(temp) Then
      flag = 1
   Else
      If temp = "" Then    'cancelacion del proceso para introducir valores
         GoTo Flag3:
      End If
      MsgBox "Introduzca solo números", 48, "Aviso"
      flag = 0
   End If
Loop While flag <> 1
Y0# = temp: flag = 0
Do
   temp = InputBox("h", "Datos iniciales", , 1, 1)
   If IsNumeric(temp) Then
      flag = 1
   Else
      If temp = "" Then    'cancelacion del proceso para introducir valores
         GoTo Flag3:
      End If
      MsgBox "Introduzca solo números", 48, "Aviso"
      flag = 0
   End If
Loop While flag <> 1
h! = temp: flag = 0
Do
   temp = InputBox("X final", "Datos iniciales", , 1, 1)
   If IsNumeric(temp) Then
      flag = 1
   Else
      If temp = "" Then    'cancelacion del proceso para introducir valores
         GoTo Flag3:
      End If
      MsgBox "Introduzca solo números", 48, "Aviso"
      flag = 0
   End If
Loop While flag <> 1
limite! = temp: flag = 0
List1.AddItem x0, i: List2.AddItem Y0, i
'las siguientes lineas son para la animacion
inter = (limite - x0) / h
Text1.Text = inter

If x0 > limite Then flag2 = 1
If x0 < limite Then flag2 = 0
```

```
EulerModi - 2

flag1:
If d = 0 Then
   X1 = x0 + h: i = i + 1
   'si cambia la funcion hay que cambiar la siguiente linea
   fx0 = -1 * (x0 / Y0)
   Y1 = Y0 + h * fx0
   fx1 = -1 * (X1 / Y1)
   Y1 = Y0 + h * ((fx0 + fx1) / 2)
   List1.AddItem X1, i
   List2.AddItem Y1, i
   If Abs(X1 - limite) > Abs(x0 - limite) Then GoTo Flag3
   x0 = X1: Y0 = Y1
End If
If d = 1 Then
   X1 = x0 + h: i = i + 1
   'si cambia la funcion hay que cambiar la siguiente linea
   fx0 = x0 + 2 * x0 * Y0
   Y1 = Y0 + h * x0 * (2 * Y0 + 1)
   fx1 = X1 + 2 * X1 * Y1
   Y1 = Y0 + h * ((fx0 + fx1) / 2)
   List1.AddItem X1, i
   List2.AddItem Y1, i
   If Abs(X1 - limite) > Abs(x0 - limite) Then GoTo Flag3
   x0 = X1: Y0 = Y1
End If
If flag2 = 1 And X1 > limite Then GoTo flag1
If flag2 = 0 And X1 < limite Then GoTo flag1
Flag3:
End Sub

Private Sub Form_Load()
Dim x As Integer
End Sub

Private Sub List1_Click()
List2.ListIndex = List1.ListIndex
inter = Text1.Text
tono = 255 / inter
m = List1.ListIndex
cajacolor = 255 - (m * tono)
For j = Shape5.Top To Shape5.Top + Shape5.Height Step 3
   For i = Shape5.Left To Shape5.Left + Shape5.Width Step 3
      PSet (i, j), &HC00000
   Next i
Next j
aa = Shape6.Top + Shape6.Height - 1
aaa = Shape6.Left + (m * (Shape6.Width / inter))
bb = m * ((Shape5.Width / 2) / inter)
bbb = (Shape5.Width / 2)
cc = m * ((Shape5.Height / 2) / inter)
ccc = Shape5.Top + Shape5.Height / 2
For i = Shape6.Left To Shape6.Left + Shape6.Width Step 3
   For j = Shape6.Top To aa Step 3
      PSet (i, j), &HC00000
   Next j
Next i
Shape2.FillColor = RGB(cajacolor, cajacolor, cajacolor)
Shape3.FillColor = RGB(cajacolor, cajacolor, cajacolor)
For j = (ccc) - Abs(cc) To ccc + Abs(cc) Step 3
   For i = Shape5.Left + bbb - bb To Shape5.Left + bbb + bb Step 3
      PSet (i, j), RGB(0, 0, 100)
   Next i
Next j
For i = Shape6.Left To aaa Step 3
   For j = Shape6.Top To aa Step 3
      PSet (i, j), RGB(0, 0, 100)
   Next j
Next i
End Sub
```

EulerModi - 3

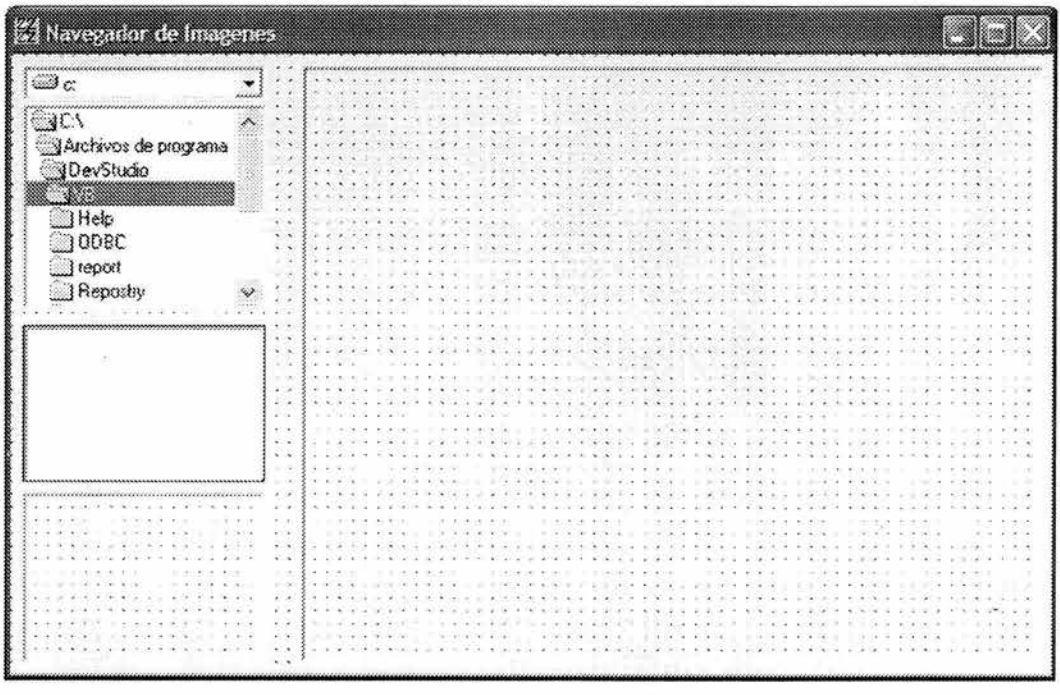
```
Private Sub Option1_Click(Index As Integer)
Select Case Index
   Case 0
      d = 0
   Case 1
      d = 1
End Select
End Sub
```


Metodo de Runge Kutta
Seleccione Funcion
f'(x,y) = - (x / y)
f'(x,y) = x + 2xy
X
Y
0%
100%
Calcular
Retornar a menu principal
Da click en una X para ver su correspondiente Y.

```
Runge - 1

Private Sub cmdmenu_Click()
Euler.Hide
Unload Euler
menu.Show
End Sub

Private Sub Command1_Click()
List1.Clear
List2.Clear
i = 0
Do
   temp = InputBox("Xo", "Datos iniciales", , 1, 1)
   If IsNumeric(temp) Then
      flag = 1
   Else
      If temp = "" Then   'cancelacion del proceso para introducir valores
         GoTo Flag3:
      End If
      MsgBox "Introduzca solo números", 48, "Aviso"
      flag = 0
   End If
Loop While flag <> 1
x0# = temp: flag = 0
Do
   temp = InputBox("Yo", "Datos iniciales", , 1, 1)
   If IsNumeric(temp) Then
      flag = 1
   Else
      If temp = "" Then   'cancelacion del proceso para introducir valores
         GoTo Flag3:
      End If
      MsgBox "Introduzca solo números", 48, "Aviso"
      flag = 0
   End If
Loop While flag <> 1
Y0# = temp: flag = 0
Do
   temp = InputBox("h", "Datos iniciales", , 1, 1)
   If IsNumeric(temp) Then
      flag = 1
   Else
      If temp = "" Then   'cancelacion del proceso para introducir valores
         GoTo Flag3:
      End If
      MsgBox "Introduzca solo números", 48, "Aviso"
      flag = 0
   End If
Loop While flag <> 1
h! = temp: flag = 0
Do
   temp = InputBox("X final", "Datos iniciales", , 1, 1)
   If IsNumeric(temp) Then
      flag = 1
   Else
      If temp = "" Then   'cancelacion del proceso para introducir valores
         GoTo Flag3:
      End If
      MsgBox "Introduzca solo números", 48, "Aviso"
      flag = 0
   End If
Loop While flag <> 1
limite! = temp: flag = 0
List1.AddItem x0, i: List2.AddItem Y0, i
'las siguientes lineas son para la animacion
inter = (limite - x0) / h
Text1.Text = inter
If x0 > limite Then flag2 = 1
If x0 < limite Then flag2 = 0
flag1:
```

```
Runge - 2

If d = 0 Then
   X1 = x0 + h: i = i + 1
   'si cambia la funcion hay que cambiar la siguiente linea
   fx0 = -1 * (x0 / Y0)
   m0 = fx0: xx = x0 + h / 2: yy = Y0 + m0 * h / 2
   m1 = -1 * (xx / yy): yyy = Y0 + m1 * h / 2
   m2 = -1 * (xx / yyy): yyyy = Y0 + m2 * h
   m3 = -1 * (X1 / yyyy)
   m = (m0 + 2 * m1 + 2 * m2 + m3) / 6
   Y1 = Y0 + h * m
   List1.AddItem X1, i
   List2.AddItem Y1, i
   If Abs(X1 - limite) > Abs(x0 - limite) Then GoTo Flag3
   x0 = X1: Y0 = Y1
End If
If d = 1 Then
   X1 = x0 + h: i = i + 1
   'si cambia la funcion hay que cambiar la siguiente linea
   fx0 = x0 + 2 * x0 * Y0
   m0 = fx0: xx = x0 + h / 2: yy = Y0 + m0 * h / 2
   m1 = xx + 2 * xx * yy: yyy = Y0 + m1 * h / 2
   m2 = xx + 2 * xx * yyy: yyyy = Y0 + m2 * h
   m3 = X1 + 2 * X1 * yyyy
   m = (m0 + 2 * m1 + 2 * m2 + m3) / 6
   Y1 = Y0 + h * m
   List1.AddItem X1, i
   List2.AddItem Y1, i
   If Abs(X1 - limite) > Abs(x0 - limite) Then GoTo Flag3
   x0 = X1: Y0 = Y1
End If
If flag2 = 1 And X1 > limite Then GoTo flag1
If flag2 = 0 And X1 < limite Then GoTo flag1
Flag3:
End Sub

Private Sub Form_Load()
Dim x As Integer
End Sub

Private Sub List1_Click()
List2.ListIndex = List1.ListIndex
inter = Text1.Text
tono = 255 / inter
m = List1.ListIndex
cajacolor = 255 - (m * tono)
For j = Shape5.Top To Shape5.Top + Shape5.Height Step 3
   For i = Shape5.Left To Shape5.Left + Shape5.Width Step 3
      PSet (i, j), &HC00000
   Next i
Next j
aa = Shape6.Top + Shape6.Height - 1
aaa = Shape6.Left + (m * (Shape6.Width / inter))
bb = m * ((Shape5.Width / 2) / inter)
bbb = (Shape5.Width / 2)
cc = m * ((Shape5.Height / 2) / inter)
ccc = Shape5.Top + Shape5.Height / 2
For i = Shape6.Left To Shape6.Left + Shape6.Width Step 3
   For j = Shape6.Top To aa Step 3
      PSet (i, j), &HC00000
   Next j
Next i
Shape2.FillColor = RGB(cajacolor, cajacolor, cajacolor)
Shape3.FillColor = RGB(cajacolor, cajacolor, cajacolor)
For j = (ccc) - Abs(cc) To ccc + Abs(cc) Step 3
   For i = Shape5.Left + bbb - bb To Shape5.Left + bbb + bb Step 3
      PSet (i, j), RGB(0, 0, 100)
   Next i
Next j
For i = Shape6.Left To aaa Step 3
```

```
Runge - 3

   For j = Shape6.Top To aa Step 3
      PSet (i, j), RGB(0, 0, 100)
   Next j
Next i
End Sub

Private Sub Option1_Click(Index As Integer)
Select Case Index
   Case 0
      d = 0
   Case 1
      d = 1
End Select
End Sub
```


Navegador de Imagenes
c:
c:\
Archivos de programa
DevStudio
VB
Help
ODBC
report
Repostry

```
Private Sub Drive1_Change()
Dir1.Path = Drive1.Drive
End Sub

Private Sub File1_Click()
selectedfile = File1.Path & "\" & File1.filename
Image1.Picture = LoadPicture(selectedfile)
Image2.Picture = LoadPicture(selectedfile)
End Sub
```

CONCLUSIONES

Antes de acometer la tarea de realizar un programa, hay que entender como funciona el algoritmo, de manera que será mucho más fácil poder comprender la estructuración que se hace y se tendrán recursos en los cuales basarse para predecir el comportamiento que debe tener el programa en cada situación.

El seguimiento a lápiz es algo muy recomendable para comprobar el funcionamiento de un programa, dicho seguimiento consiste en realizar por uno mismo, con ayuda de papel y lápiz, las instrucciones que la computadora interpretará.

Visual Basic es el lenguaje de programación ideal para comenzar a programar, ya que permite con su simplicidad y potencia, enfrentar desde pequeños a grandes proyectos; al tiempo que en cuanto se domine otro lenguaje, ya sea para aprovechar alguna nueva cualidad, por gusto o para aumentar su potencia de cálculo, se pueda hacer la "traducción" de programas ya realizados al nuevo lenguaje en una forma sencilla o simplemente adicionarlos como librerías o subrutinas.

Los métodos numéricos son una potente herramienta en conjunción con equipos de cálculo electrónicos para alcanzar soluciones razonables o al menos lo suficientemente satisfactorias, a la vez que permiten evadir el enfrentar procesos más complicados para obtener resultados semejantes, en tiempos menores.

Una vez que se ha comprendido la técnica de programación, se puede intentar llevar a cabo proyectos cada vez más ambiciosos como el diseño de equipos, el análisis de procesos, el control de los proyectos, etc.

El uso cada vez más frecuente de equipos electrónicos, no solamente en la industria sino también en la vida diaria, nos obliga como ingenieros, a conocer su empleo y hacer el uso más provechoso de estos recursos.

El ahorro de tiempo obtenido con el desarrollo de programas de cómputo para tareas repetitivas o con gran cantidad de cálculos, es el principal motor para que los Ingenieros se deban apropiar de la habilidad para programar y generar, con su conocimiento del área, los programas más adecuados para llevar a cabo sus labores.

Existen una gran cantidad de programas disponibles en libros de texto, cdroms e internet que no se emplean por que no llevan a cabo totalmente las tareas que se necesitan, pero que gracias a la habilidad de programar es posible modificarlos y añadirles las secciones necesarias. Así no se tendrá que partir todo el tiempo de cero; esto es muy útil sobretodo en programas de empleo breve o en grandes proyectos que acortan el tiempo de programación.

La programación de aplicaciones no es la única vía para la solución de problemas simples y complejos, también tenemos disponible el uso de hojas de cálculo. Sin embargo, después de adquirir la capacidad de hacerlo, se obtiene una habilidad mayor en el tratamiento y análisis de temas abstractos y de modelado.

Gran parte del código de un programa no tiene que ver con el algoritmo del método, sino con la captura, presentación y validación de datos. Pero, el llevar a cabo un buen diseño de la interfase de entrada de datos, ayuda mucho en la eliminación de errores evitando la introducción de valores inadecuados; y el tener una interfase de salida apropiada, evita problemas de no entendimiento o mala interpretación de los resultados.

BIBLIOGRAFÍA

- Applied Numerical Methods. Brice Carnahan, H. A. Luther, James O. Wilkes; Ed. John Wiley & Sons, USA

- Introducción a la Computación para Ingenieros. C. Chapra Steven, P. Canale Raymond; Ed. McGraw-Hill, México, 1989

- Métodos Numéricos. Rodolfo Luthe, Antonio Olivera, Fernando Schutz; Ed. Limusa, 7ª reimpresión, México, 1988.

- Numerical Methods for Engineers and Computer Scientists. Paul F. Hultquist; The Benjamin/Cummings Publishing Company Inc., USA, 1988

- Análisis Numérico Elemental, un Enfoque Algorítmico. S. D. Conte, Carl de Boor; Ed. McGraw-Hill, 2ª edición, México, 1977

- Numerical Methods, a Software Approach. R. L. Johnston; Ed. John Wiley and sons, USA, 1982

- Métodos Numéricos Aplicados con Software. Shoichiro Nakamura, Prentice-Hall Hispanoamericana S.A., México, 1992

- Métodos Numéricos Aplicados a la Ingeniería química. Nieves, Domínguez

- Métodos Numéricos. Iriarte V. Balderrama, Rafael; Ed. Trillas-UNAM, México, 1ª ed., 1990 (reimp. 1999)

- Métodos Numéricos Aplicados a la Ingeniería. Terrence J. Akai; LIMUSA-WILEY; México, 1ª ed., 1999

- Calculus, serie educativa (cd-rom), CEDISAC Sistemas Automatizados, Cuba, 1999

- Métodos Numéricos, Manual de teoría y Problemas. Galicia Tapia, Jorge; UNAM, México, 1988

- Visual Basic 3.0. Achával, Manuel; Ventura Ediciones, México, 1994

- Visual Basic 4.0, Guía de Autoenseñanza. Osborne McGraw-Hill; España, 1996

- Técnicas de Programación. F. Alonso Amo, A. Morales Lozano; Ed. Paraninfo, Madrid, 1988

- Fortran 90 A Reference Guide. Luc Chamberland; Prentice may, USA, 1995

- Domine Turbo Pascal 6. Scott D. Palmer; (Ed. Sybex, USA, 1991) Ventura Ediciones, México, 1992

- C Programming. Augie Hansen; Ed. Addison-Wesley, USA, 1989

- Programación Metódica. José Luis Balcázar; Mc-Graw Hill, Madrid, 1993

- Aprendiendo Visual C++ 2. Namir Clement Shammas; Sams Publishing, México, 1996

- Diccionario Especializado de Matemáticas, Ed. Norma, Bogota, Colombia, 2001

Tesis (Universidad Nacional Autónoma de México.)

- Vázquez Zamora, Eduardo. Microcomputadoras y métodos numéricos; 1984.

- Mendoza Romero, David Ismael. Métodos Numéricos Con Pascal; 1985
- Flores Villalobos, Martha. Solución de ecuaciones, diferenciales parciales por métodos numéricos; 1986.
- Cernuda Benavente, Rocío. Apuntes de métodos numéricos; 1987.
- Hernández Loyola, Armando. Comparación de técnicas de resolución de ecuaciones diferenciales ordinarias por métodos numéricos y su aplicación a reactores; 1988.
- Hernández Martínez. Métodos numéricos para la resolución de mínimos cuadrados por el método de Gram-Schmitd; 1990.

Tesis (Instituto Politécnico Nacional)

- Sánchez López, José. Programación de métodos numéricos y codificación en Pascal; 1993.
- Perea Guzmán, Bernardino. Tutorial de métodos numéricos en Delphi; 2000.
- Caballero Jasso, Eloy, Jaén Sañudo, Minerva de los Remedios. "sistema de métodos numéricos" / Eloy Caballero Jasso, Minerva de los Remedios Jaén Sañudo México; UPIICSA, 1994