



Universidad Nacional Autónoma de México

FACULTAD DE CIENCIAS

MÉTODOS NUMÉRICOS CON PASCAL

T E S I S

QUE PARA OBTENER EL TÍTULO DE

M A T E M Á T I C O

P R E S E N T A

DAVID ISMAEL MENDOZA ROMERO

MEXICO, D. F.

1985



Universidad Nacional
Autónoma de México



UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

DERECHOS RESERVADOS © PROHIBIDA SU REPRODUCCIÓN TOTAL O PARCIAL

Todo el material contenido en esta tesis está protegido por la Ley Federal del Derecho de Autor (LFDA) de los Estados Unidos Mexicanos (México).

El uso de imágenes, fragmentos de videos, y demás material que sea objeto de protección de los derechos de autor, será exclusivamente para fines educativos e informativos y deberá citar la fuente donde la obtuvo mencionando el autor o autores. Cualquier uso distinto como el lucro, reproducción, edición o modificación, será perseguido y sancionado por el respectivo titular de los Derechos de Autor.

PROLOGO

La solución de una inmensa cantidad de problemas a los cuales nos enfrentamos en la vida cotidiana encuentra respuesta en la aplicación de los métodos numéricos. Bajo este panorama se hace imprescindible hacer llegar a los estudiantes los conocimientos sobre esta disciplina. El material aquí presentado busca constituir un apoyo de consulta para estudiantes de licenciatura (Matemáticas, Ingeniería, etc), que por primera vez se enfrentan al estudio de los métodos numéricos.

Se eligió Pascal como lenguaje en los programas que se presentan debido a las ventajas del lenguaje y a la rápida difusión que ha experimentado en los últimos años, tanto en los Estados Unidos como en Europa, predominantemente en el área educativa. Tal es el caso que en la Facultad de Ciencias se da este lenguaje en cursos como Computación I y se prefiere en Estructura de datos.

Un sentir muy personal es que se debe dar un panorama general de los métodos numéricos, expuestos de una manera sencilla y clara, para que esto motive a un estudio y análisis más profundo de los mismos. Es por ello que este material puede ser una introducción práctica y a la vez didáctica a un curso de análisis numérico.

La estructura que se le ha dado a este trabajo, pretende seguir una meta fundamental: Que el lector aprenda y se interese, mediante la lectura y práctica del material incluido en este trabajo, en los métodos numéricos y que esto le permita resolver problemas en los que se requiera la aplicación de este material o bien, que lo motive a un estudio más profundo de los métodos nu-

mericos y en especial al area de las matematicas denominada Analisis Numerico.

Este trabajo es apto para cualquier estudiante de licenciatura con conocimientos de Matematicas, Calculo y Algebra Superior, asi como con conocimiento del lenguaje Pascal.

Se ha tratado de seguir, en lo posible, los lineamientos de la Programacion Estructurada, la cual constituye una de las tecnicas mas modernas para programar y que conduce a la realizacion de programas altamente modulares, sencillos de depurar y actualizar, asi como de facil comprension para personas no involucradas en su realizacion. De esta manera, cada tema es tratado como un modulo independiente, pero que puede ser usado como parte de otro que lo requiera.

Quiero dar una disculpa al lector, ya que como habra observado este documento adolece de acentos y de la letra "eñe", esto se debe a que se edito en el editor de textos TIA (tipografo automatizado) en la Burroughs B7800 y las impresoras no tienen la facilidad del retorno de carro.

Cabe mencionar que la mayoría de los programas que se incluyen estan contenidos en un manual editado en el Programa Universitario de Computo, cuyo nombre es "manual de metodos numericos con Pascal", realizado por el autor en colaboracion de otras dos personas.

Quiero agradecer al Ins. Jorge Gil Mendieta, director general del Programa Universitario de Computo, por las facilidades prestadas para la realizacion de este trabajo, al Dr. Manuel

Portilla Robertson la orientacion, colaboracion y entusiasmo
brindados. Finalmente debo mencionar a dos companeros becarios de
Programa Universitario de Computo que colaboraron conmigo en la
mayor parte de este trabajo; ellos son Jaime Besprosbany y Alber-
to Villarreal; asi como tambien agradezco el apoyo de la srta.
Maria Guadalupe Castellanos.

David I. Mendoza Romero

Mexico, D.F. ----- 1983.

C O N T E N I D O

INTRODUCCION	1
- PROGRAMACION ESTRUCTURADA	3
- ESTRUCTURA GENERAL	5
I) GRAFICAS	
1.1) GRAFICA DE FUNCIONES	11
1.2) HISTOGRAMAS	21
II) ALGEBRA MATRICIAL	
2.1) INTRODUCCION	35
2.2) SUMA DE MATRICES	39
2.3) MULTIPLICACION DE MATRICES	45
2.4) INVERSION MATRICIAL	50
III) SOLUCION DE SISTEMAS DE ECUACIONES LINEALES	
3.1) INTRODUCCION	59
3.2) METODO DE GAUSS-JORDAN	61
3.3) METODO DE GAUSS-SEIDEL	71
3.4) METODO DE LA DESCOMPOSICION TRIANGULAR	81
3.5) COMPARACION DE METODOS	93
IV) RAICES DE ECUACIONES	

4.1)	INTRODUCCION	96
4.2)	METODO DE BISECCION Y METODO DE LA SECANTE	97
4.3)	METODO DE NEWTON-RAPHSON	100
4.4)	METODO DE LIN-BAIRSTOW PARA POLINOMIOS	108
4.5)	COMENTARIOS DE LOS METODOS	122
V) VECTORES Y VALORES CARACTERISTICOS		
5.1)	INTRODUCCION	124
5.2)	METODO DE JACOBI	126
5.3)	METODO DE KRYLOV	133
5.4)	COMENTARIOS DE LOS METODOS	140
VI) INTERPOLACION		
6.1)	INTRODUCCION	142
6.2)	METODO DE LAGRANGE	144
6.3)	COMENTARIOS	150
VII) METODO DE LOS MINIMOS CUADRADOS		
7.1)	INTRODUCCION	153
7.2)	APROXIMACION POLINOMIAL	154
7.3)	COMENTARIOS	166
VIII) INTEGRACION NUMERICA		

8.1) INTRODUCCION	169
8.2) METODO DEL TRAPECIO, METODOS DE SIMPSON	169
8.3) COMENTARIOS	183
IX) SOLUCION DE ECUACIONES DIFERENCIALES DE PRIMER ORDEN	
9.1) INTRODUCCION	186
9.2) METODO DE EULER Y METODO DE EULER MODIFICADO	189
9.3) METODO DE RUNGE-KUTTA	200
9.4) METODO DE MILNE	
9.5) COMPARACION DE METODOS	220
X) BIBLIOGRAFIA	222

INTRODUCCION

Las computadoras electronicas de alta velocidad han traído un cambio revolucionario en la solución de problemas científicos y de tipo ingenieril. Este cambio libero al hombre de muchas tareas de calculos tediosos, y also muy importante, hicieron posible la solución de problemas científicos e ingenieriles de gran complejidad.

La mayoría de problemas de Ingenieria, Fisica, Quimica, etc., requieren de calculos numericos para su solución, como en la sustitucion de parametros en el diseño de una ecuación, la estimación de el costo de cambio en una planta de operación o el analisis estadístico de datos de control de calidad.

Es importante hacer notar que la computadora es unicamente una herramienta para el profesionista; esta no lo puede reemplazar. Es el quien debe formular la solución correcta a su problema, reunir todos los datos que necesite, ver los resultados y analizar su valor. En suma, necesita saber como usar la computadora, el instrumento que realizara los calculos aritmeticos.

Los pasos en la solución de un problema por la computadora son:

- 1) Formulación del modelo matemático, es decir, convertir el sistema físico a un modelo matemático idealizado y, de acuerdo con este modelo, formular las ecuaciones matemáticas.

- 2) Seleccionar un procedimiento numerico adecuado para una computadora digital.

3) Dibujar con detalle un diagrama de flujo (una representación gráfica de la secuencia de operaciones, tales como lecturas, cálculos, comparaciones, etc.) o bien un pseudocódigo, que también nos da la secuencia que puede tener un algoritmo, pero no es gráfico y usa las técnicas de la programación estructurada.

4) De acuerdo al diagrama de flujo (o pseudocódigo) escribir un programa en un lenguaje propio (en nuestro caso en lenguaje de alto nivel Pascal).

5) Ejecutar el programa, es decir, introducirlo a la máquina y correrlo.

6) Analizar resultados.

INTRODUCCION

Las computadoras electronicas de alta velocidad han traído un cambio revolucionario en la solución de problemas científicos y de tipo insenieril. Este cambio libero al hombre de muchas tareas de calculos tediosos, y also muy importante, hicieron posible la solución de problemas científicos e insenieriles de gran complejidad.

La mayoría de problemas de Ingeniería, Física, Química, etc., requieren de calculos numericos para su solución, como en la sustitución de parametros en el diseño de una ecuación, la estimación de el costo de cambio en una planta de operación o el análisis estadístico de datos de control de calidad.

Es importante hacer notar que la computadora es unicamente una herramienta para el profesionista; esta no lo puede reemplazar. Es el quien debe formular la solución correcta a su problema, reunir todos los datos que necesite, ver los resultados y analizar su valor. En suma, necesita saber como usar la computadora, el instrumento que realizara los calculos aritmeticos.

Los pasos en la solución de un problema por la computadora son:

- 1) Formulacion del modelo matemático, es decir, convertir el sistema físico a un modelo matemático idealizado y, de acuerdo con este modelo, formular las ecuaciones matemáticas.

- 2) Seleccionar un procedimiento numerico adecuado para una computadora digital.

3) Dibujar con detalle un diagrama de flujo (una representación gráfica de la secuencia de operaciones, tales como lecturas, cálculos, comparaciones, etc.) o bien un pseudocódigo, que también nos da la secuencia que puede tener un algoritmo, pero no es gráfico y usa las técnicas de la programación estructurada.

4) De acuerdo al diagrama de flujo (o pseudocódigo) escribir un programa en un lenguaje propio (en nuestro caso en lenguaje de alto nivel Pascal).

5) Ejecutar el programa, es decir, introducirlo a la máquina y correrlo.

6) Analizar resultados.

PROGRAMACION ESTRUCTURADA

La programación estructurada es una rama de la computación que consta de un conjunto de técnicas para desarrollar programas que tengan las características siguientes:

- Confiables.
- Fáciles de probar y corregir.
- Fáciles de comprender.
- Susceptibles de ser modificados y actualizados.
- Eficientes en cuanto a memoria, tiempo y uso.

Tomando en cuenta los puntos anteriores, para que un programa sea claro, entendible, corto, y para que pueda ser comprendido mejorado o modificado, debe mantener las siguientes normas específicas:

1) Secuencia. Al leer ordenadamente un programa, nuestra atención no se dispersa al intentar atrapar la lógica del mismo (tratar evitar la sentencia GOTO).

2) Estructurado. El programa se escribe en módulos empleando solo las unidades lógicas: IF-THEN-ELSE, WHILE (en Pascal podemos agregar FOR y REPEAT).

3) Corto. Los módulos no deben sobrepasar cierto número de registros.

4) Fragmentado. Debido a que en la realidad los programas resultan ser muy grandes, es importante poder dividirlos en blo-

ques (en Pascal funciones y subrutinas).

5) Estilo. Entre mas formal se haga el programa, mas facil sera de entender.

6) Claridad.

7) Senstrado. Mediante este se podran marcar las estructuras.

8) Documentacion. Mediante referencias adicionales al programa este se lograra comprender mejor.

9) Universal. El programa debere poder ser entendido por cualquiera.

VENTAJAS DEL PASCAL

Todas las características mencionadas anteriormente pueden ser llevadas a cabo satisfactoriamente por un programa en Pascal, sin embargo tiene algunas desventajas, sobre todo en la carencia de doble precision que puede ser requerida para cierto tipo de problemas numericos; por lo demas consideramos que las ventajas que tiene sobre otros lenguajes como el Fortran es sobresaliente.

ESTRUCTURA GENERAL

En este trabajo no se hace un analisis extenso de cada metodo sino basicamente contiene una introduccion didactica a cada metodo; un Programa implementado del mismo con un representacion formal (pseudocodiso); un listado del Programa y algun ejemplo ilustrativo de su aplicacion con la corrida del Programa.

Este trabajo pretende introducir a cualquier persona interesada en los metodos numericos al aprendizaje y a la practica directa de los mismos. Se escogieron una serie de problemas y metodos que consideramos de mayor importancia y que resuelven una gran variedad de problemas; es decir, tienen cierta flexibilidad dependiendo del tipo de problema a tratar; algun lector interesado en el tema podria ahondar sobre cada metodo; mejorarlo o implementar cualquier otro mas eficiente tomando en cuenta caracteristicas como:

- a) Menor tiempo de CPU (minimizando numero de operaciones).
- b) Mejor aprovechamiento de la memoria.
- c) Obtencion de los resultados con una mayor exactitud (mejoramiento del error de truncamiento y redondeo).

Para este fin existe en la literatura una bibliografia muy amplia sobre estos temas que son tratados en la rama de las matematicas denominada "Analisis Numerico".

Todos los temas estan desarrollados en modulos; tratando de ser independientes entre si; sin embargo esto no quita la posibilidad de usar determinado bloque o subrutina de modulos anterior-

res al considerado.

Cada capítulo consta de uno a tres métodos del tema tratado y se ilustran con un ejemplo que da una idea del tipo de problemas que pueden ser resueltos con el método en cuestión. Los programas se han implementado en la computadora B7800 y están diseñados de tal manera que, al ejecutarlos o correrlos, se sepa el tipo de información (datos) que requieren el programa en cuestión despliegue un letrero para pedir el tipo de dato requerido. Además la información se puede realimentar; es decir, se pueden resolver varios problemas del mismo tipo. Inmediatamente después de ejecutarse un problema se piden datos para otro.

Los programas se diseñaron para trabajarse en terminal, sin embargo el lector puede modificarlos a su criterio, en el caso que quiera una salida distinta.

Cada tema se desarrolle con un formato estándar con el fin de darle claridad y fácil entendimiento al lector. El formato general es:

1) METODO TRATADO <Nombre del metodo>.

1.1) INTRODUCCION

Se da una descripción matemática general del método empleado. Las demostraciones se excluyen por estar fuera de los propósitos de esta tesis (si se desea un estudio más detallado del tema consultar la bibliografía dada).

1.2) PROGRAMA <Nombre>

1.2.1) OBJETIVO:

Se establece el tipo de problema que resuelve el Programa.

1.2.2) DESCRIPCION:

Se da un bosquejo general de las características del Programa, tales como entrada, salida, variables que pueden ampliar la dimensión del programa, etc.

1.2.3) ALGORITMO EN PSEUDOCODIGO:

Esta parte contiene una descripción de todas las subrutinas del Programa (incluyendo el programa principal). Las subrutinas están numeradas según el orden en que se les llama y la profundidad en que se encuentran. Cada descripción de subrutina incluye:

1.2.3.1) Objetivo. Se describe en términos generales lo que hace la subrutina.

1.2.3.2) VARIABLES. Se da una tabla con todas las variables usadas en la subrutina. Para cada variable se describe:

1.2.3.2.1) Uso. En el programa principal, Las iniciales siguientes denotan: E = Variable de entrada, S = Variable de Salida, '-' = Constante o Variable global. En las subrutinas las iniciales siguientes denotan: E = Variable-Parametro de entrada, S = Variable-Parametro de salida, '-' = Constante, L = Variable local.

1.2.3.2.2) Tipo. Se refiere a si una variable es de tipo real, entero, de caracteres, arreglo o constante.

1.2.3.2.3) Funcion. Enuncia el papel que desempeña la variable en la subrutina.

1.2.3.3) Pseudocodiso:

Tomando en cuenta las técnicas y características de la programación estructurada es un algoritmo que sigue paso a paso la secuencia del programa pero con la diferencia de que está escrito en un lenguaje más informal. Es por tanto una buena guía para entender el programa. Contiene las siguientes características:

1) El programa principal y las subrutinas empiezan con BEGIN y terminan con END.

2) El pseudocodiso es secuencial; en algunos casos se emplea la misma palabra reservada de PASCAL para hacer indicaciones como: IF - THEN - ELSE, WHILE, FOR, REPEAT, etc.

3) En instrucciones como las mencionadas anteriormente se pondrá 3 espacios a la derecha todo el bloque que afecten; y las siguientes instrucciones van con el margen de la instrucción anterior, por ejemplo:

```

WHILE EPSILON > 0.06 DO
  Escribe EPSILON.
  IF RENGLON > COLUMNA THEN
    Intercambia el renglon.
    EPSILON <--- EPSILON - 0.1.
    A <--- A + B.
  B <--- A / C.
FOR I = 1 TO 100 DO
  A <--- A - B.
  CONT <--- CONT + 1.

```

4) Se usa el símbolo ' <--- ' para hacer asignaciones.

5) Todas las instrucciones terminan con un punto (".") y empiezan con una letra mayuscula. En caso de que en una linea del pseudocodigo se empiece con una letra minuscula, entonces dicha linea es continuacion de la instruccion anterior.

6) La llamada a las subrutinas se realiza mediante la palabra "CALL".

1.3) LISTADO DEL PROGRAMA

Impresion del programa en Pascal; se excluyen subrutinas empleadas en capitulos anteriores.

1.4) EJEMPLO

Se presenta un problema practico. El programa al ser ejecutado despliega el tipo de informacion requerida; es importante mencionar que el programa, al ser ejecutado, seguira pidiendo datos hasta que no se le de fin de archivo (en la B7800 el fin de archivo es "TEND").

1.5) COMPARACION DE METODOS

En esta seccion, mas que una comparacion lo que se da son algunos comentarios generales de los metodos incluidos y de algunos que aparecen en la Literatura mas interesantes y eficientes.

El numero que se le da a las partes que constituyen el capitulo no aparece necesariamente en ellos, (aqui se dan como suia).

I) GRAFICAS.

1.1) GRAFICAS.

1.1.1) INTRODUCCION

Casi en todas las ramas de la ciencia y la ingeniería son muy frecuentes los casos en los que es necesario darse una idea del comportamiento de una función, y para esto, nada mejor que visualizarla gráficamente. En esta sección se describirá un programa que generará una o más gráficas a partir de un conjunto de puntos muestrales.

1.1.2) PROGRAMA GRAFICAS

OBJETIVO:

Crear gráficas a partir de conjuntos de puntos.

DESCRIPCION:

Este programa recibirá como datos de entrada el número de puntos a graficar, el número de gráficas deseado y los puntos muestrales a graficar. Como salida, producirá una gráfica en la cual el eje de las ordenadas para todas las funciones a graficar es transversal a la hoja y el eje de las abscisas es una columna hacia abajo de la hoja, dependiendo la longitud de este eje de la cantidad de puntos existente.

ALGORITMO EN PSEUDOCODIGO:

PROGRAMA PRINCIPAL

VARIABLES	USO	TIPO	FUNCION
CAMPO	-	Const.	Numero maximo de puntos a graficar.
MAXORD	-	Const.	Numero maximo de funciones a graficar.
NUMABS	E	Entera	Numero de puntos a graficar por la subrutina GRAFICA.
NUMFUNC	E	Entera	Numero de funciones a graficar por la subrutina GRAFICA.
ARREGLO	E	Arreglo de reales	Matriz que contiene las abscisas y valores de las funciones a graficar.
I, J	-	Enteros	Contadores.

Pseudocodiso:

```

BEGIN
Lee NUMABS y NUMFUNC.
Lee ARREGLO[I,J].
CALL GRAFICA (ARREGLO, NUMABS, NUMFUNC).
END.

```

SUBROUTINAS:

1) GRAFICA (ARREGLO, NUMABS, NUMFUNC).

OBJETIVO: Graficar los puntos contenidos en ARREGLO.

VARIABLES	USO	TIPO	FUNCION
NUMABS	E	Entera	Numero de puntos a graficar por la subrutina GRAFICA.
NUMFUNC	E	Entera	Numero de funciones a graficar por la subrutina GRAFICA.
ARREGLO	E	Arreglo de reales	Matriz que contiene las abscisas y valores de las funciones a graficar.
BLANCO	-	Const.	Representa un ' '.
GUION	-	Const.	Representa un '-'.
PUNTO	-	Const.	Representa un '.'.
ASTERISCO	-	Const.	Representa un '*'.
CARACTER	L	Arreglo de caracteres	Numeros empleados para las funciones (del 1 al 6).
RENGLON	L	Arreglo de Caracteres	Vector que contiene las 101 posiciones de cada renglon.
ORDMAX	L	Real	Guarda la ordenada de mayor valor.
ORDMIN	L	Real	Guarda la ordenada de menor valor.
ORDENADA	L	Arreglo de Reales	Cotas de las ordenadas.
IAUX	L	Arreglo de Caracteres	Indica superposicion de funciones.
LT, LK	L	Entero	Contadores.
L	L	Arreglo de enteros	Indica la posicion de una de las ordenadas dentro de la grafica.
PASO	L	Real	Indica la separacion entre las abscisas.
LTEM	L	Arreglo de enteros	Identifica los caracteres de las ordenadas superpuestas.
POSORD	L	Arreglo de enteros	Posicion asociada al valor de una ordenada.
AUXILIARI	L	Entero	Indica cuantas funciones se superponen.
ESFACIAMIENTO	L	Entero	Guarda el espaciamiento entre las abscisas.
H	L	Entero	Cantidad de columnas de la matriz ARREGLO.
I, J, K	L	Enteros	Contadores.

Pseudocodiso:

BEGIN

Busca el valor maximo y minimo de las ordenadas.

Selecciona espaciamiento entre las abscisas.

Obtiene e imprime los valores de las ordenadas de referencia.

WHILE existen puntos para graficar DO

 Inicializa el vector RENGLON.

 Normaliza el valor de las ordenadas.

 Asigna un numero (del 1 al 101) a la posicion de la ordenada.

 Busca ordenadas superpuestas.

 Indica en la variable IAUX las ordenadas superpuestas existentes.

 Imprime RENGLON.

END.

1.1.3) LISTADO DEL PROGRAMA#

```

PROGRAM GRAFICACION(INPUT,OUTPUT);
CONST
  CAMPO = 101;
  MAXORD= 6;
TYPE
  MATRIZ=ARRAY[1..CAMPO,1..MAXORD] OF REAL;
VAR
  ARREGLO      : MATRIZ;
  NUMABS,
  NUMFUNC,
  I,
  J              : INTEGER;
PROCEDURE GRAFICA(ARREGLO :MATRIZ; NUMABS,NUMFUNC:INTEGER);
CONST
  BLANCO = ' ';
  GUION = '-';
  PUNTO = '.';
  ASTERISCO = '*';
VAR
  CARACTER      : PACKED ARRAY[1..MAXORD] OF CHAR;
  RENGLON       : PACKED ARRAY[1..CAMPO] OF CHAR;
  ORDMAX,
  ORDMIN        : REAL;
  ORDENADA      : ARRAY[1..11] OF REAL;
  IAUX          : ARRAY[1..7] OF CHAR;
  LT,
  LK            : INTEGER;
  L             : ARRAY[1..7] OF INTEGER;
  PASO          : REAL;
  LTEM,
  POSORD        : ARRAY[1..7] OF INTEGER;
  AUXILIAR1,
  ESPACIAMIENTO,
  H,
  I,
  J,
  K              : INTEGER;
BEGIN (DEL PROCEDURE GRAFICA);
CARACTER[1]:= '1';
CARACTER[2]:= '2';
CARACTER[3]:= '3';
CARACTER[4]:= '4';
CARACTER[5]:= '5';
CARACTER[6]:= '6';
NUMFUNC := NUMFUNC + 1;
IF NUMABS>20 THEN
  IF NUMABS>40 THEN
    ESPACIAMIENTO:=0
  ELSE
    ESPACIAMIENTO:=1
ELSE
  ESPACIAMIENTO:=2;
ORDMAX:=ARREGLO[1,2];

```

```

ORDMIN:=ARREGLOC[1,2];
FOR I:=1 TO NUMABS DO
  FOR J:=2 TO NUMFUNC DO
    BEGIN
      IF (ORDMAX<ARREGLOC[I,J]) THEN
        ORDMAX:=ARREGLOC[I,J];
      IF (ORDMIN>ARREGLOC[I,J]) THEN
        ORDMIN:=ARREGLOC[I,J];
      END(FOR);
    (END FOR);
  IF (ORDMAX=ORDMIN) THEN
    ORDMAX:=ORDMAX + 10;
  PASO:=((ORDMAX-ORDMIN)/10);
  ORDENADAC[1]:=ORDMIN;
  ORDENADAC[11]:=ORDMAX;
  FOR I:=2 TO 10 DO
    ORDENADAC[I]:=ORDENADAC[I-1]+PASO;
  FOR I:=1 TO CAMPO DO
    RENGLONC[I]:=GUION;
  FOR I:=0 TO 10 DO
    BEGIN
      AUXILIAR1:=(10*I)+1;
      RENGLONCAUXILIAR1:=ASTERISCO;
    END(FOR);
  WRITELN;
  WRITELN;
  WRITELN;
  WRITE(' '4);
  FOR I:=1 TO 11 DO
    WRITE(' ',ORDENADAC[I]:9:3);
  WRITELN;
  WRITE(' '11);
  FOR I:=1 TO CAMPO DO
    WRITE(RENGLONC[I]);
  WRITELN;
  FOR I:=1 TO NUMABS DO
    BEGIN
      FOR J:=1 TO 7 DO
        IAUXC[J]:=BLANCO;
      FOR J:=2 TO 100 DO
        RENGLONC[J]:=BLANCO;
      FOR J:=1 TO 10 DO
        BEGIN
          AUXILIAR1:=(10*J)+1;
          RENGLONCAUXILIAR1:=PUNTO;
        END(FOR);
      RENGLONC[1]:=ASTERISCO;
      RENGLONC[CAMPO]:=ASTERISCO;
      IF (ESPACIAMIENTO<>0) THEN
        FOR K:=1 TO ESPACIAMIENTO DO
          BEGIN
            WRITE(' '11);
            FOR H:=1 TO CAMPO DO
              WRITE(RENGLONCH);
            WRITELN;
          END(FOR);
        END(FOR);
      END(FOR);
    END(FOR);
  END(FOR);

```

```

(ENDE IF)
FOR J:=2 TO NUMFUNC DO
  BEGIN
    LCJJ:=TRUNC((ARREGLOCJ, JJ-ORDMIN)*100/(ORDMAX-ORDMIN)+1.5)
    POSORDEJJ:=0
    ENDE(FOR)
  LK:=0
  LT:=1
  FOR J:=2 TO NUMFUNC DO
    IF (POSORDEJJ<>J) THEN
      BEGIN
        RENGLONCLJJ:=CARACTERCJ-1
        POSORDEJJ:=J
        FOR K:=2 TO NUMFUNC DO
          BEGIN
            IF (POSORDCK<>K) THEN
              IF (LCJJ=LCK) THEN
                BEGIN
                  LK:=LK+1
                  LTEMCK:=K
                  POSORDCK:=K
                  ENDE(IF)
                (ENDE IF)
              ENDE(FOR)
            IF (LK<>0) THEN
              BEGIN
                IAUXCLT+1:=CARACTERCJ-1
                FOR K:=1 TO LK DO
                  IAUXCLT+K+1:=CARACTERCLTEMCK-1
                LT:=LT+LK+2
                IAUXCLT:=QUION
                LK:=0
                ENDE(IF)
              ENDE(IF)
            (ENDE FOR)
            WRITE(' ', ARREGLOCJ, 11:9:3, ' ')
            FOR J:=1 TO CAMPO DO
              WRITE(RENGLONCJJ)
            FOR J:=1 TO 7 DO
              WRITE(IAUXCJJ)
            WRITELN
            ENDE(FOR)
            FOR I:=2 TO 100 DO
              RENGLONCIJ:=QUION
            FOR I:=1 TO 10 DO
              BEGIN
                AUXILIARI:=(10*I)+1
                RENGLONCAUXILIARI:=ASTERISCO
              ENDE(FOR)
            WRITE(' '11)
            FOR I:=1 TO CAMPO DO
              WRITE(RENGLONCIJ)
            WRITELN
            ENDE(GRAFICA)
          (PROGRAMA PRINCIPAL)
        BEGIN (PROGRAMA PRINCIPAL)

```

```

WRITELN('NUM. DE PUNTOS MUESTRALES ?, NUM DE GRAFICAS ?')#
READLN#READ(NUMABS,NUMFUNC)#
WRITELN('PUNTOS MUESTRALES ? (UNO POR RENGLON)')#
FOR I:=1 TO NUMABS DO
  BEGIN
  READLN#
  FOR J:=1 TO NUMFUNC + 1 DO
    READ(ARREGLOI,JJ)#
  END(FOR)#
GRAFICA(ARREGLO,NUMABS,NUMFUNC)#
END. (PROGRAMA PRINCIPAL)

```

1.1.4) EJEMPLO

Obtener la grafica para las funciones $\text{Sen}(x)$ y $(1/x)\text{Sen}(x)$ en el intervalo $(0,10)$ y tomando incrementos de 0.2. La ejecucion del programa GRAFICAS para los puntos generados por dichas funciones en el intervalo dado quedaria como se muestra en las paginas siguientes.

NUM. DE PUNTOS MUESTRALES ? NUM DE GRAFICAS ?

50 2
PUNTOS MUESTRALES ? (UND POARENGLON)

0.200	0.199	0.993
0.400	0.389	0.974
0.600	0.565	0.941
0.800	0.717	0.897
1.000	0.841	0.841
1.200	0.932	0.777
1.400	0.985	0.704
1.600	1.000	0.625
1.800	0.974	0.541
2.000	0.909	0.455
2.200	0.808	0.367
2.400	0.675	0.281
2.600	0.516	0.198
2.800	0.335	0.120
3.000	0.141	0.047
3.200	-0.058	-0.018
3.400	-0.256	-0.075
3.600	-0.443	-0.123
3.800	-0.612	-0.161
4.000	-0.757	-0.189
4.200	-0.872	-0.208
4.400	-0.952	-0.216
4.600	-0.994	-0.216
4.800	-0.996	-0.208
5.000	-0.959	-0.192
5.200	-0.883	-0.170
5.400	-0.773	-0.143
5.600	-0.631	-0.113
5.800	-0.465	-0.080
6.000	-0.279	-0.047
6.200	-0.083	-0.013
6.400	0.117	0.018
6.600	0.312	0.047
6.800	0.494	0.073
7.000	0.657	0.094
7.200	0.794	0.110
7.400	0.899	0.121
7.600	0.968	0.127
7.800	0.999	0.128
8.000	0.989	0.124
8.200	0.941	0.115
8.400	0.855	0.102
8.600	0.734	0.085
8.800	0.585	0.066
9.000	0.412	0.046
9.200	0.223	0.024
9.400	0.025	0.003
9.600	-0.174	-0.018

1.2) HISTOGRAMAS

1.2.1) INTRODUCCION

En estadística, al querer analizarse un conjunto de datos, es útil distribuirlos en grupos ordenados llamados clases o categorías y determinar el número de ocurrencias de los datos en cada clase, siendo esto la frecuencia de clase.

En esta forma, los datos quedan agrupados en intervalos limitados por dos valores, cuyo valor intermedio puede ser usado para representar a todo el intervalo. A este valor se le conoce como punto medio de intervalo de clase. Una ordenación de datos de esta manera es una distribución de frecuencia.

Un ejemplo de esto sería el tener el número de alturas medidas en alguna población. Se podrían ordenar estas alturas en clases de por ejemplo 1.50-1.60 m., 1.60-1.70 m., 1.80-1.90 m., etc. y medir cuántas medidas quedan comprendidas en cada intervalo, es decir, su frecuencia. De esta forma, se obtendrían rangos de altura representados por un punto medio de intervalo de clase (en este caso serían 1.55, 1.65, 1.75, etc.) con una correspondiente frecuencia, o sea, el número de individuos cuya altura estuvo en ese intervalo.

La distribución de frecuencia se aplica en muchos otros casos tales como el coeficiente intelectual en una población, la calidad de un producto en una línea de producción, etc.

Un histograma es una representación gráfica de la distribución de frecuencia en un conjunto de datos. El objeto de un his-

tosrama es facilitar la visualización de la distribución de frecuencia.

Por convención, en el eje X se representan los intervalos de clase y sobre el eje Y se representan las frecuencias de clase. Es factible dividir las frecuencias también en intervalos.

Sobre cada intervalo de clase se construye un rectángulo cuya base representa el tamaño del intervalo y cuya altura representa la frecuencia.

1.2.2) PROGRAMA HISTOGRAMA

OBJETIVO:

Crear histogramas a partir de pares numéricos clase/frecuencia.

DESCRIPCION:

El programa que presentamos a continuación tiene como entrada el número de pares a introducirse y posteriormente los pares clase/frecuencia en sí. El programa supone que los valores del punto medio de intervalo de clase se dan en orden y que además, representan intervalos de clase usuales.

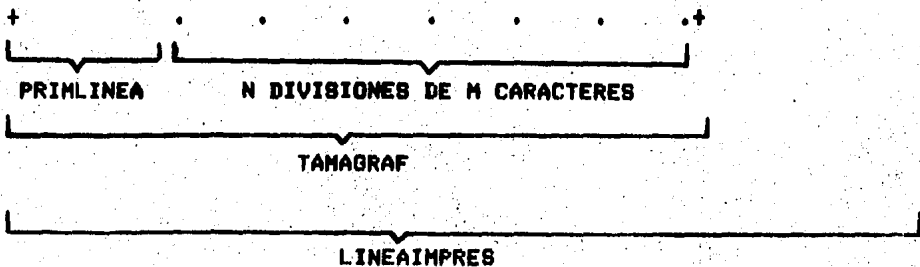
El programa permite la creación de tantos histogramas como se quiera.

El formato de la gráfica es el siguiente: Cada histograma es enmarcado con una línea de signos '+' y es separado de los otros por un espacio. Considerando la línea de impresión, horizontalmente se representan las frecuencias (eje Y) y verticalmente las

clases (eje X). Verticalmente y del lado izquierdo, se colocan numeros que representan cada punto medio de intervalo de clase y horizontalmente y arriba se escriben los valores correspondientes a cada division (o intervalo) de las frecuencias. Las frecuencias y por tanto, sus etiquetas correspondientes se grafican en relacion al mayor valor de estas de tal forma que al menos una frecuencia ocupa toda la grafica. Cada frecuencia es representada por tres lineas de asteriscos de un tamaño que la representa y todas estan separadas por una linea punteada que marca ademas, los intervalos de frecuencia.

El tamaño de la grafica, el numero de divisiones de intervalos de frecuencia y el espacio para escribir el valor del punto medio de intervalo de clase pueden ser determinados a gusto del usuario mediante el manejo de las constantes del programa TAMAGRAF, PRIMLINEA y DIVISIONES, respectivamente.

La eleccion de estos valores esta limitada por el hecho de que el tamaño de la grafica no debe ser menor que el de la linea de impresion (LINEAIMPRES), por que las divisiones de frecuencia correspondan a un valor entero de caracteres de impresion y por que las etiquetas no sobrepasen el espacio asignado. El siguiente diagrama representa el sentido de cada constante en el programa:



Las mismas condiciones expresadas en forma matemática son:

$$1) \text{TAMAGRAF} \leq \text{LINEAIMPRES}$$

2) $2 + \text{PRIMLINEA} + (\text{INTERVALO} * \text{DIVISIONES}) = \text{TAMAGRAF}$, donde INTERVALO es un número entero.

3) Valor Máximo de Clase $< 10 * \text{Tamaño Permitido}$ y Valor Máximo de Frecuencia $< 10 * \text{Tamaño Permitido}$.

Por ejemplo, con una línea de impresión (LINEAIMPRES) de 80, sería correcto hacer un histograma de 68 caracteres (TAMAGRAF) de 6 divisiones (DIVISIONES), con un tamaño de intervalo (INTERVALO) de 10 y con la primera línea (PRIMLINEA) de 6.

Alguna falta del cumplimiento de estas condiciones será avisada por el programa.

ALGORITMO EN PSEUDOCODIGO:

PROGRAMA HISTOGRAMA

VARIABLES	USO	TIPO	FUNCION
LIMITE	-	Constante	Tamaño del arreslo
I	-	Entero	Contador
N	E	Entero	Numero de pares de datos
CLASE	E/S	Arreslo	Valores de punto medio de
		de Reales	intervalo de clase
FRECUENCIA	E/S	Arreslo	Valores de frecuencia
		de Reales	

Pseudocodigo:

```

BEGIN
Pide datos
Lee N
WHILE not eof DO
  FOR I := 1 TO N
    Lee CLASE[I] y FRECUENCIA[I]
    CALL HISTOGRAMA (N, CLASE, FRECUENCIA)
  Lee N
END
  
```

SUBRUTINAS

1) HISTOGRAMA (N, X, NX)

Objetivo: Crear la grafica de un histograma a partir de los valores que le son dados.

VARIABLES	USO	TIPO	FUNCION
N	E	Entero	Numero de pares de valores
X	E	Arreglo de Reales	Valores de punto medio de intervalo de clase
NX	E	Arreglo de Reales	Valores de frecuencia
DIVISIONES	L	Constante	Numero de intervalos de frecuencia
PRIMLINEA	L	Constante	Tamano de primera parte de la linea
TAMAGRAF	L	Constante	Tamano de la grafica
LINEAIMPRES	L	Constante	Tamano linea de impresion
INTERVALO	E/S	Entero	Tamano de intervalos de frecuencia
NXMAX	E/S	Real	Maximo valor de frecuencias

Pseudocodiso:

```

BEGIN
CALL MAXIMO (NX, NXMAX, N)
CALL ETIQUETAS (NX, NXMAX, INTERVALO)
CALL CREAFIGRIFICA (X, NX, N, NXMAX)
Saltar un renglon
asigna a NXMAX.
END

```

1.2) ETIQUETAS (NXMAX, INTERVALO)

Objetivo: Colocar los valores correspondientes a los intervalos de frecuencia en la parte superior de la grafica segun las divisiones marcadas.

VARIABLES	USO	TIPO	FUNCION
ARRE	E	Arreglo Caracteres	Guarda caracteres de linea de salida
NMAX	E	Real	Maximo valor de frecuencia
I			
INTERVA	E	Entero	Guarda valor del intervalo (en caracteres)
ESC	L	Real	Tamano del intervalo de frecuencia (en numero)
PASO	L	Real	Guarda valores sucesivos de las etiquetas
CORRECTOR	L	Booleano	Indicador de correccion de formato

Pseudocódigo:

```

BEGIN
Inicializa variables
Calcula PASO
Calcula INTERVA
IF division incorrecta THEN
  Mandar mensaje
IF etiquetas no caben en el espacio asignado THEN
  Mandar mensaje
  CORRECTOR <--- false
Dibuje marco superior
Escribe titulos
Escribe etiqueta 0
FOR J := 1 TO DIVISIONES DO
  Escribe etiqueta de intervalo de frecuencia w un punto divisor

  Saltar un renglon
END

```

1.3) CREAGRAFICA (X, NX, N, NMAX)

Objetivo: Delinear la grafica del histograma, traduciendo numericos de los datos a sus equivalentes graficos.

VARIABLES	USO	TIPO	FUNCION
X	E	Arreglo de Reales	Valores de punto medio de intervalo de clase
NX	E	Arreglo	Valores de frecuencia
N	E	Entero	Numero de pares de valores
NMAX	E	Real	Maximo valor de frecuencia
I, J, K	L	Enteros	a representar una frecuencia
LSAL	L	Arreglo de Caracteres	Contadores
	L	Arreglo de Caracteres	Lineas de salida que guarda un numero de asteriscos correspondiente a la frecuencia
CORRECTOR	L	Booleano	Indicador de correccion de formato

Pseudocodiso:

```

BEGIN
Inicializa CORRECTOR
Coloca marco derecho e izquierdo en LSAL
FOR I := 1 TO N DO
  Calcula NXSALI
  CALL LINEAPUNTO (LSAL)
  CALL LINEABLANCO (LSAL)
  Escribe puntos separadores
  Coloca NXSALI asteriscos en LSAL
  Escribe LSAL
  Escribe valor correspondiente a punto
  medio de intervalo de clase (XCII) sobre PRIMLINEA
  IF XCII no cabe en PRIMLINEA THEN
    CORRECTOR <--- false
  Escribe LSAL y una vez mas
  CALL LINEABLANCO (LSAL)
  CALL LINEAPUNTO (LSAL)
  IF CORRECTOR = false THEN
    Mandar mensaje
END

```

1.3.1) LINEABLANCO (LSALII)

Objetivo: Coloca blancos en LSALI

1.3.2) LINEAPUNTO (LSALII)

Objetivo: Coloca puntos en LSALI segun las divisiones y el intervalo dado.

1.2.3) LISTADO DEL PROGRAMA

```
PROGRAM HISTOGRAMA (INPUT, OUTPUT);
```

```
CONST
```

```
  LIMITE = 20;
```

```
TYPE
```

```
  ARREG = ARRAY [1..LIMITE] OF REAL;
```

```
VAR
```

```
  I, N          : INTEGER;
```

```
  CLASE, FRECUENCIA : ARREG;
```

```
PROCEDURE HISTOGRAMA (N : INTEGER; X : ARREG; NX : ARREG);
```

```
CONST
```

```
  DIVISIONES = 5;
```

```
  TAMAGRAF   = 62;
```

```
  LINEAIMPRES = 132;
```

```
VAR
```

```
  INTERVALO : INTEGER;
```

```
  NXMAX      : REAL;
```

```
(BUSCA EL VALOR MAXIMO DEL ARREGLO)
```

```
PROCEDURE MAXIMO (ARRE : ARREG; VAR MAX : REAL; N : INTEGER);
```

```
VAR
```

```
  J : INTEGER;
```

```
BEGIN
```

```
  J := 1;
```

```
  MAX := ARRE[J];
```

```
  FOR J := 1 TO N DO
```

```
    IF ARRE[J] > MAX THEN
```

```
      MAX := ARRE[J];
```

```
    END (FOR);
```

```
  END (PROCEDURE);
```

```
(ESCRIBE ETIQUETAS EN LA LINEA DE SALIDA)
```

```
PROCEDURE ETIQUETAS (NXMAX : REAL; VAR INTERVA : INTEGER);
```

```
VAR
```

```
  J          : INTEGER;
```

```
  ESC, PASO : REAL;
```

```
  CORRECTOR : BOOLEAN;
```

```
BEGIN
```

```
  ESC := 0;
```

```
  CORRECTOR := TRUE;
```

```
  PASO := NXMAX/DIVISIONES;
```

```
  INTERVA := ROUND ((TAMAGRAF - PRIMLINEA - 2) / DIVISIONES);
```

```
  IF 2 + PRIMLINEA + DIVISIONES * INTERVA <> TAMAGRAF THEN
```

```
    WRITELN ('DIVISION INCORRECTA DE LINEA DE SALIDA');
```

```
  IF (ROUND (100 * NXMAX)) / 100 >=
```

```
    ROUND (EXP ((INTERVA - 5) * LN (10))) THEN
```

```
    BEGIN
```

```
      WRITE ('TAMANO DE INTERVALOS NO ES SUFICIENTE ');
```

```

WRITELN ('PARA ESCRIBIRSE EN EL ESPACIO DE ETIQUETAS');
CORRECTOR := FALSE;
END (IF);
FOR J := 1 TO TAMAGRAF DO
  WRITE ('+');
  WRITE (' ' ; LINEAIMPRES - TAMAGRAF);
  WRITELN ('PTO. MEDIO DE INT. DE CLASE/FRECUENCIA -->');
  WRITE ('+');
  WRITE (ESC ; PRIMLINEA - 1 ; 2);
  WRITE ('.')
  FOR J := 1 TO DIVISIONES DO
    BEGIN
      ESC := ESC + PASO;
      WRITE (ESC ; INTERVA - 1 ; 2);
      WRITE ('.')
    END (FOR);
  WRITE ('+');
  WRITE (' ' ; LINEAIMPRES - TAMAGRAF);
  IF CORRECTOR = FALSE THEN
    WRITELN;
  END (PROCEDURE);

```

<GRAFICA LOS DATOS>

```

PROCEDURE CREAGRAFICA (X : ARREG; NX : ARREG; N : INTEGER;
  NMAX : REAL; INTERVA : INTEGER);

```

TYPE

```

  SALI = ARRAY [1..TAMAGRAF] OF CHAR;

```

VAR

```

  I, J, K, NXVALIDA : INTEGER;

```

```

  LSAL : SALI;

```

```

  CORRECTOR : BOOLEAN;

```

<LLENA LA LINEA DE SALIDA CON BLANCOS>

```

PROCEDURE LINEABLANCO (VAR LSALII : SALI);

```

BEGIN

```

  FOR J := 2 TO TAMAGRAF - 1 DO

```

```

    LSALII[J] := ' ';

```

```

  END (PROCEDURE);

```

<COLOCA PUNTOS PARA DIVIDIR INTERVALOS>

```

PROCEDURE LINEAPUNTO (VAR LSALII : SALI);

```

BEGIN

```

  FOR J := PRIMLINEA + 1 TO TAMAGRAF - 1 DO

```

BEGIN

```

    LSALII[J] := '.';

```

```

    J := J + INTERVA - 1;

```

```

  END (FOR);

```

```

  END (PROCEDURE);

```

BEGIN

```

  CORRECTOR := TRUE;

```

```

  LSAL[1] := '+';

```

```

  LSAL[TAMAGRAF] := '+';

```

```

  FOR I := 1 TO N DO

```

BEGIN

```

    NXVALIDA := ROUND ((NX[I] / NMAX) * (TAMAGRAF -

```



```

        PRIMLINEA - 2)) + PRIMLINEA + 1;
LINEABLANCO (LSAL);
LINEAPUNTO (LSAL);
FOR K := 1 TO TAMAGRAF DO
    WRITE (LSAL[K]);
WRITE ('' : LINEAIMPRES - TAMAGRAF);
FOR J := PRIMLINEA + 1 TO NXVALIDA DO
    LSAL[J] := '*';
FOR K := 1 TO TAMAGRAF DO
    WRITE (LSAL[K]);
WRITE ('' : LINEAIMPRES - TAMAGRAF);
WRITE ('+');
WRITE (XC[I] : PRIMLINEA - 2 : 1);
WRITE(' ');
IF (ROUND (10 * XC[I]) / 10 >=
    ROUND (EXP ((PRIMLINEA - 5) * LN (10))) THEN
    CORRECTOR := FALSE;
FOR K := PRIMLINEA + 1 TO TAMAGRAF DO
    WRITE (LSAL[K]);
WRITE ('' : LINEAIMPRES - TAMAGRAF);
FOR K := 1 TO TAMAGRAF DO
    WRITE (LSAL[K]);
WRITE ('' : LINEAIMPRES - TAMAGRAF);
END (FOR);
LINEABLANCO (LSAL);
LINEAPUNTO (LSAL);
FOR K := 1 TO TAMAGRAF DO
    WRITE (LSAL[K]);
WRITE ('' : LINEAIMPRES - TAMAGRAF);
FOR J := 1 TO TAMAGRAF DO
    WRITE ('+');
WRITE ('' : LINEAIMPRES - TAMAGRAF);
IF CORRECTOR = FALSE THEN
    BEGIN
        WRITE ('FRECUENCIAS DEMASIADO GRANDES PARA ');
        WRITELN ('ESCRIBIRSE. CORREGIR PRIMLINEA');
    END (IF);
END (PROCEDURE);

```

```

BEGIN
MAXIMO (NX, NXMAX, N);
ETIQUETAS (NXMAX, INTERVALO);
CREAGRAFICA (X, NX, N, NXMAX, INTERVALO);
WRITELN;
END (PROCEDURE PRINCIPAL);

```

(PROGRAMA PRINCIPAL)

```

BEGIN
WRITELN ('NUMERO DE DATOS?');
READ (N);
WHILE NOT EOF DO
    BEGIN
        WRITE ('PARES? DE VALORES (CADA PUNTO MEDIO ');
        WRITE ('DE INTERVALO DE CLASE CON SU CORRESPONDIENTE ');
        WRITELN ('CANTIDAD DE SUCEOS');
        FOR I := 1 TO N DO

```

```

    READLN (CLASE[I], FRECUENCIA[I]);
    HISTOGRAMA (N, CLASE, FRECUENCIA);
    WRITELN ('NUMERO DE DATOS?');
    READ (N);
    END (WHILE);
END.

```

1.2.4) EJEMPLO

Un biologo desea obtener una imagen de la abundancia de ciertos arboles en relacion a las altitudes a las que se encuentran en una determinada area. Si cuenta con la siguiente tabla de distribucion de frecuencias, obtener la grafica de un histograma a partir de los datos.

Rangos de Altura	No de arboles (en cierta area)
2000-2100	6
2100-2200	3
2200-2300	21
2300-2400	57
2400-2500	101
2500-2600	166
2600-2700	179
2700-2800	139
2800-2900	122
2900-3000	71
3000-3100	29
3100-3200	2

CORRIDA

NUMERO DE DATOS?

11

PARES DE VALORES? (CADA PUNTO MEDIO DE INTERVALO DE CLASE CON
CORRESPONDIENTE CANTIDAD DE SUCEOS)

2050	6
2150	3
2250	21
2350	57
2450	101
2550	166
2650	179
2750	139
2850	122
2950	71
3050	29
3150	2

+++++
PTO. MEDIO DE INT. DE CLASE/FRECUENCIA -->

	0.00.	35.80.	71.60.	107.40.	143.20.	179.00.	+
+	+
+	***	+
+	2050.0	***	+
+	.	***	+
+	+
+	***	+
+	2150.0	**	+
+	.	**	+
+	+
+	*****	+
+	2250.0	*****	+
+	.	*****	+
+	+
+	*****	+
+	2350.0	*****	+
+	.	*****	+
+	+
+	*****	+
+	2450.0	*****	+
+	.	*****	+
+	+
+	*****	+
+	2550.0	*****	+
+	.	*****	+
+	+
+	*****	+
+	2650.0	*****	+
+	.	*****	+
+	+
+	*****	+
+	2650.0	*****	+
+	.	*****	+
+	+
+	*****	+
+	2850.0	*****	+
+	.	*****	+
+	+
+	*****	+
+	2950.0	*****	+
+	.	*****	+
+	+
+	*****	+
+	3050.0	*****	+
+	.	*****	+
+	+
+	**	+
+	3150.0	**	+
+	.	**	+
+	+
+	*****	+

II) ALGEBRA MATRICIAL

2.1) INTRODUCCION

La teoria de matrices tiene su origen en los trabajos de tres matematicos ingleses: Hamilton, Silverster y Cayley, publicados alrededor de 1850. Desde entonces ha sido una de las ramas de la matematica de mas fecunda aplicacion en los mas diversos campos de la ciencia, tan indispensable para quienes trabajan en disciplinas tales como la fisica, la ingenieria, la estadistica, la economia, la administracion, la psicologia, la sociologia, etc.

A pesar de todo, aun hoy existe un grupo numeroso de profesionales no familiarizado con esta materia. A continuacion presentamos los conceptos basicos, antes de su manejo en la computadora:

Una matriz es un arreglo de elementos pertenecientes a un campo K , distribuido en m renglones y n columnas; si denotamos a una matriz con la letra A , entonces la matriz A tiene la forma:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

donde a_{ij} es el elemento del i -esimo renglon y de la j -esima columna.

Se dice que esta es una matriz de m por n o bien, de $m \times n$. Los elementos de la matriz, como se menciono pertenecen a un campo K (reales, complejos, funciones en el espacio del tiempo,

etc.).

Al ser una matriz un arreglo ordenado de elementos, esto permite que al aplicar cierta metodología a dicho arreglo se obtengan una serie de resultados que respondan a los interrogantes por los que se origino el arreglo; entre algunos de los procesos en los que se utilizan arreglos matriciales se tiene: Jerarquizacion de actividades, almacenamiento de datos, inventarios, representacion de sistemas dinamicos, sistemas de ecuaciones, etc.

Debido a la distribucion particular que presentan los elementos de una matriz, estas se clasifican en distintos grupos entre los que se tienen:

a) Matriz Cuadrada. Es aquella matriz en la cual el numero de renglones es igual al numero de columnas. Ejemplo:

$$A = \begin{bmatrix} 1 & 2 & -4 \\ 8 & 0 & 12 \\ 3 & 1 & -7 \end{bmatrix}$$

b) Matriz Nula. Es aquella matriz de cualquier orden en la que $a_{ij} = 0$ para todo i y $j = 1 \dots N$, o sea, en la que todos sus elementos son nulos. Se representa con la letra 0. Ejemplo:

$$0 = B = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

c) Matriz Identidad. Es una matriz cuadrada en la cual los elementos de la diagonal principal (aquella que va del extremo superior izquierdo al extremo inferior derecho) son unitarios y el resto nulos, es decir, $c_{ij} = 0, i < j$; $c_{ij} = 1, i = j$. Esta matriz se denota con I_n donde 'n' es el orden de la matriz. Ejem-

Ejemplo:

$$I_4 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

d) Matriz Diagonal. Es una matriz cuadrada cuyos elementos son cero, con excepción de los que forman la diagonal principal, es decir, $d_{ij} = 0$ si $i \neq j$. Ejemplo:

$$D = \begin{bmatrix} 5 & 0 & 0 \\ 0 & -4 & 0 \\ 0 & 0 & 6 \end{bmatrix}$$

Un tipo especial de matriz diagonal es aquella en la que todos los elementos D_{ij} de la diagonal principal son iguales. Este tipo de matriz se denomina matriz escalar. Ejemplo:

$$E = \begin{bmatrix} 8 & 0 & 0 \\ 0 & 8 & 0 \\ 0 & 0 & 8 \end{bmatrix}$$

e) Matriz Transpuesta. Es una matriz de $n \times n$ que se obtiene a partir de una matriz A intercambiando renglones por columnas, es decir $a_{ij} = a_{ji}^t$. Se denota por A^t . Ejemplo:

$$A = \begin{bmatrix} a & a & a \\ 11 & 12 & 13 \\ a & a & a \\ 21 & 22 & 23 \end{bmatrix} \quad A^t = \begin{bmatrix} a & a & a \\ 11 & 12 & 13 \\ a & a & a \\ 21 & 22 & 23 \end{bmatrix}$$

f) Matriz Simétrica. Es aquella matriz cuadrada que es igual a su transpuesta o $A^t = A$, es decir, $a_{ij} = a_{ji}$ para todo $i, j = 1, \dots, N$. Ejemplo:

$$A = \begin{bmatrix} 2 & -3 & 7 \\ -3 & 4 & 8 \\ 7 & 8 & 6 \end{bmatrix} \quad A^t = \begin{bmatrix} 2 & -3 & 7 \\ -3 & 4 & 8 \\ 7 & 8 & 6 \end{bmatrix}$$

Con el conjunto de matrices se define un algebra con las operaciones siguientes: Suma, multiplicacion por un escalar elemento de K y la multiplicacion matricial.

El producto de un escalar por una matriz se define asi: Sea x en K y A una matriz cualquier entonces $xA = (x a_{ij}) = c_{ij}$, para todo i y j . Ejemplo:

$$x = 3 \quad A = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

$$xA = \begin{bmatrix} 3*2 & 3*4 \\ 3*6 & 3*8 \end{bmatrix} = \begin{bmatrix} 6 & 12 \\ 18 & 24 \end{bmatrix}$$

Para usos practicos no es necesario implementar una subrutina para este fin. Observacion: Las matrices $n \times n$ con componentes en un campo K , forman un espacio vectorial sobre K .

2.2) SUMA DE MATRICES

2.2.1) INTRODUCCION

Se define la adición de matrices solo cuando tienen el mismo orden. Se define la matriz $C = A + B$ como aquella matriz cuya componente en el renglón "i" y la columna "j" es:

$$c_{ij} = a_{ij} + b_{ij}$$

La resta de matrices es equivalente a multiplicar escalarmente una matriz por -1 y sumarla a otra, es decir:

$$A - B = A + (-B)$$

2.2.2) PROGRAMA SUMAT

OBJETIVO:

Obtener la suma matricial de uno o varios pares de matrices de la misma dimensión.

DESCRIPCION

El programa efectúa la suma de dos matrices A y B de orden MxN los datos que se le daran son los valores de M y N, además los coeficientes de las matrices a sumar. Se recomienda dar los componentes de las matrices renglón por renglón para evitar equivocaciones.

ALGORITMO EN PSEUDOCODIGO

PROGRAMA PRINCIPAL

VARIABLE	USO	TIPO	FUNCION
REN	-	Constante	Numero maximo de renglones para la matriz usada en el Programa (es modificable).
COL	-	Constante	Numero maximo de columnas para la matriz usada en el Programa (es modificable).
A, B	E	Arreglo de reales	Matrices usadas para hacer la operacion matricial.
C	E	Arreglo de reales	Matriz que contiene la suma de las matrices A y B.
N, M	E	Entero	Son las dimensiones de las matrices a sumar.

Pseudocodiso:

BEGIN

 Escribir letrero de entrada

 leer M y N

 WHILE no sea fin de archivo DO

 CALL LEEMAT (A,M,N)

 CALL LEEMAT (B,M,N)

 CALL SUMAT (A,B,C,M,N)

 Escribir letrero de salida

 Escribe letrero para pedir mas datos

END.

SUBROUTINAS

1) LEEMAT (D,M,N)

Objetivo: Esta subrutina lee una matriz de orden $M \times N$, D es el nombre de la matriz.

2) ESCMAT (D,M,N)

Objetivo: Esta subrutina escribe una matriz de orden $M \times N$, estos valores le entran como parametros asi como la matriz a escribir.

3) SUMAT (A,B,C,M,N)

Objetivo: Efectuar la suma matricial de dos matrices del mismo

orden, las matrices sumando son A y B, la matriz C es la matriz suma.

VARIABLE	USO	TIPO	FUNCION
A	E	Arreglo de reales	Matriz sumando de orden MxN.
B	E	Arreglo de reales	Matriz sumando de orden MxN.
C	S	Arreglo de reales	Matriz que contiene la suma de las matrices A y B.
M, N	E	Entero	Parametros que indican las dimensiones de las matrices a sumar.

Pseudocodiso:

```

BEGIN
  FOR I := 1 TO M DO
    FOR J := 1 TO N DO
      C[I,J] := A[I,J] + B[I,J]
    END
  END
END.
```

2.2.3) LISTADO DEL PROGRAMA

```

PROGRAM SUMAT (INPUT,OUTPUT);
CONST
  REN = 12;
  CDL = 10;
TYPE
  MATRIZ = ARRAY[1..REN,1..COL] OF REAL;
VAR
  A, B, C : MATRIZ;
  N, M : 1..REN;

  PROCEDURE LEEMAT (VAR D : MATRIZ; M, N : INTEGER);
  VAR
    I, J : INTEGER;
  BEGIN
    FOR I:= 1 TO M DO
      FOR J:= 1 TO N DO
        READ(DCI,JJ);
      WRITELN
    END(*LEEMAT*);

  PROCEDURE ESCMAT (D : MATRIZ; M, N : INTEGER);
  VAR
    I, J : INTEGER;
  BEGIN
    WRITELN;
    FOR I := 1 TO M DO
      BEGIN
        WRITE ('C');
        FOR J := 1 TO N DO
          WRITE (DCI, JJ : 12 : 4);
        WRITELN ('J');
      END
    END(* ESCMAT *);

  PROCEDURE SUMAT (A, B : MATRIZ; VAR C : MATRIZ;
    M, N : INTEGER);
  VAR
    I, J : INTEGER;
  BEGIN
    FOR I := 1 TO M DO
      FOR J := 1 TO N DO
        CCI, JJ := ACI, JJ + BCI, JJ
      END(* SUMAT *)
    END

      ( PROGRAMA PRINCIPAL )

  BEGIN
    WRITELN ('DAR LAS DIMENSIONES DE LAS MATRICES ');
    READ (M,N);
    WHILE NOT EOF DO
      BEGIN
        WRITELN ('DAR COMPONENTES DE LA MATRIZ A');
        LEEMAT (A,M,N);
        WRITELN ('DAR COMPONENTES DE LA MATRIZ B');
      END
    END
  END

```

```

LEEMAT (B,M,N);
SUMAT (A,B,C,M,N);
WRITELN (' LA MATRIZ SUMA ES:');
ESCHAT (C,M,N);
WRITELN;
WRITELN (' DAR LAS DIMENSIONES DE LAS MATRICES ');
READ (M,N);

```

END

END.

2.2.4) EJEMPLO

Una empresa fabrica los productos A, B, C, que se procesan en los talleres T1, T2, T3, y T4. Si las existencias al principio de semana son:

	A	B	C
T1	200	380	275
T2	500	250	215
T3	600	225	150
T4	660	380	220

y durante la semana hay la siguiente producción:

	A	B	C
T1	60	70	57
T2	95	65	60
T3	90	100	110
T4	75	80	55

Determine la producción total que tendrá el inventario semanal si se sabe que la empresa no hizo repartos en la semana.

La solución a este problema, introduciendo directamente los datos a la computadora es:

DAR DIMENSIONES DE LAS MATRICES

4 3

DAR COMPONENTES DE LA MATRIZ A

200	380	275
500	250	215
600	225	150
660	380	220

DAR COMPONENTES DE LA MATRIZ B

60	70	57
95	65	60

90	100	110
75	80	55

LA MATRIZ SUMA ES:

260.0000	450.0000	332.0000
595.0000	315.0000	275.0000
690.0000	325.0000	260.0000
735.0000	460.0000	275.0000

2.3) MULTIPLICACION DE MATRICES

2.3.1) INTRODUCCION

Para efectuar el producto matricial entre dos matrices dadas A y B , se debe cumplir que el numero de renglones de A sea igual al numero de columnas de B . Esta condicion se conoce como conformidad para multiplicacion de matrices, es decir: si A es de orden $M \times N$ y B es de orden $N \times S$, se define el producto AB como la matriz C de orden $M \times S$ cuya entrada (i, j) es igual a:

$$c_{ij} = \sum_{k=1}^N a_{ik} b_{kj}$$

$i=1, \dots, M \quad j=1, \dots, S$

Cabe mencionar que el producto matricial es distributivo respecto a la suma, es asociativo y no es conmutativo, es decir cumple con las propiedades: dadas las matrices A, B, C conformables.

- a) $A(B + C) = AB + AC$
- b) SI x pertenece a K entonces $A(xB) = x(AB)$
- c) $A(BC) = (AB)C$
- d) AB diferente de BA en general.

2.3.2) PROGRAMA MULTMAT

OBJETIVO:

Implementar el producto matricial de matrices conformables en la computadora, para varias parejas de matrices conformables.

DESCRIPCION

El programa multiplica cualquier par de matrices conformables hasta de orden 12×10 , en caso que se quieran multiplicar matrices de mayor orden, modificar las constantes del programa REN y COL segun se requiera. Si se quiere multiplicar dos matrices de orden $L \times M$ y $M \times N$, el programa pedira dimensiones de las matrices a multiplicar; se deben dar tres valores: L, M y N en este orden. Se recomienda dar los componentes de las matrices por renglones, para mejor claridad.

ALGORITMO EN PSEUDOCODIGO:

PROGRAMA PRINCIPAL

VARIABLE	USD	TIPO	FUNCION
REN	-	Constante	Numero maximo de renglones para la matriz usada en el programa (es modificable).
COL	-	Constante	Numero maximo de columnas para la matriz usada en el programa (es modificable).
A	E	Arreglo	Matriz premultiplicadora de orden $L \times M$.
B	E	Arreglo	Matriz postmultiplicadora de orden $M \times N$.
C	S	Arreglo	Matriz producto de orden $L \times N$.
L, M, N	E	Entero	Son las dimensiones de las matrices a multiplicar.

Pseudocodiso:

BEGIN

```

Leer dimensiones de las matrices, L, M y N.
WHILE no sea fin de archivo DO
  CALL LEEHAT (A,L,M)
  CALL LEEHAT (B,M,N)
  CALL MULTMAT (A,B,C,L,M,N)
  Escribe letrero de salida
  CALL ESCHAT (C,L,N)
  Leer dimensiones para otras matrices.

```

END.

SUBROUTINAS

1) LEEHAT (D,M,N)

(Lee una matriz de orden $M \times N$, ver sec. 2.2.2).

2) ESCHAT (D1,D2,P,L,M,N)

(Escribe una matriz de orden $M \times N$, ver sec. 2.2.2).

3) MULTMAT(D1,D2,P,L,M,N)

Objetivo: Multiplicar dos matrices conformables.

VARIABLE	USO	TIPO	FUNCION
D1	E	Arreglo de reales	Matriz premultiplicadora de orden $L \times M$.
D2	E	Arreglo de reales	Matriz postmultiplicadora de orden $M \times N$.
P	S	Arreglo de reales	Matriz producto de orden $L \times N$.
AUX	L	Real	Variable auxiliar para efectuar el producto matricial.

Pseudocodiso:

BEGIN

```

FOR I := 1 TO L DO
  FOR J := 1 TO N DO
    AUX <--- 0
    FOR K := 1 TO M DO
      AUX <--- AUX + D1(I,K) * D2(K,J)
    PCI,JJ <--- AUX

```

END.

2.3.3) LISTADO DEL PROGRAMA

```

PROGRAM MULTMAT (INPUT,OUTPUT);
CONST
  REN = 12;
  COL = 10;
TYPE
  MATRIZ = ARRAY[1..REN,1..COL] OF REAL;
VAR
  A, B, C : MATRIZ;
  N, M ,L : 1..REN;

PROCEDURE MULTMAT (D1, D2 : MATRIZ) VAR P : MATRIZ;
  L, M, N : INTEGER);
VAR
  I, J,K : INTEGER;
  AUX : REAL;
BEGIN
  FOR I := 1 TO L DO
    FOR J := 1 TO N DO
      BEGIN
        AUX := 0.0;
        FOR K := 1 TO M DO
          AUX := AUX + D1[I,K] * D2[K,J];
        P[I,J] := AUX;
      END
    END
  END(* MULTMAT *);

  ( PROGRAMA PRINCIPAL )

BEGIN
  WRITELN ('DAR DIMENSIONES DE LAS MATRICES');
  READ (L,M,N);
  WHILE NOT EOF DO
    BEGIN
      WRITELN ('DAR COMPONENTES DE LA MATRIZ A');
      LEEMAT (A,L,M);
      WRITELN ('DAR COMPONENTES DE LA MATRIZ B');
      LEEMAT (B,M,N);
      MULTMAT (A,B,C,L,M,N);
      WRITELN (' LA MATRIZ PRODUCTO ES:');
      ESCMAT (C,L,N);
      WRITELN (' DAME LAS DIMENSIONES DE LAS MATRICES ');
      READ (L,M,N);
    END
  END.

```

2.3.4) EJEMPLO

Un agricultor produce 3 productos, maiz, trigo y cebada, con tres factores de producción: tierra, mano de obra y yuntas de bueves. Las unidades (por semana) que requiere de cada factor de producción para tener una buena cosecha son:

	Tierra	mano de obra	yuntas
maiz	20	20	18
trigo	10	12	13
cebada	5	8	9

Si los costos unitarios de cada factor de producción son:

tierra	300
mano de obra	3 500
yuntas	1 000

Determinar el costo total de cada producto debido al factor de producción que requiere.

La solución a este problema, introduciendo los datos directamente a la computadora es:

DAR LAS DIMENSIONES DE LA MATRICES

3 3 1

DAR COMPONENTES DE LA MATRIZ A

20 20 18

10 12 13

5 8 9

DAR COMPONENTES DE LA MATRIZ B

300

3500

1000

LA MATRIZ PRODUCTO ES

94000.0000

58000.0000

38500.0000

2.4) INVERSION DE MATRICES

2.4.1) INTRODUCCION

Sea A una matriz cuadrada de orden N . Se dice que A es invertible o no singular si existe una matriz B cuadrada de orden N tal que:

$$AB = BA = I_n$$

Tal matriz B esta determinada de forma unica y se conoce como la inversa de A , y se denota:

$$A^{-1}$$

Una definicion formal de la inversa es:

$$A^{-1} = \frac{A^*}{|A|}$$

Donde A^* "estrella", es la matriz adjunta de A . $|A|$ es el determinante de la matriz A . De la definicion dada anteriormente se infiere que para que exista la inversa se requiere que $|A|$ no sea cero, es decir, una matriz A se llama no singular si $|A|$ no es cero.

Sin embargo para la obtencion numerica de la matriz inversa, no es conveniente implementar un algoritmo para una computadora digital aplicando directamente la definicion dada, ya que se requiere una gran cantidad de operaciones y consecuentemente tiempo. Para obtener la inversa de una matriz de 10×10 se requieren mas de 340 millones de operaciones con el metodo directo.

Para resolver el problema existen metodos completamente sistematicos y relativamente eficientes como el metodo de GAUSS-JORDAN modificado el cual consiste en lo siguiente: partiendo del arreglo

$$(A_n, I_n)$$

donde A_n es la matriz a la que se busca la inversa, I_n es la matriz identidad (tanto A_n como I_n son de orden n), y aplicando alguna de las siguientes transformaciones a dicho arreglo

- a) intercambio del renglon R_i por el R_j
- b) multiplicacion de un renglon R_i por un escalar c diferente de cero, es decir, reemplazando R_i por cR_i
- c) suma de equimultiplos de un renglon a otro, es decir, reemplazando el renglon R_i por $R_i + cR_j$ ($j \neq i$).

Se llega al arreglo:

$$\begin{matrix} -1 \\ (I_n, A_n) \end{matrix}$$

Se transforma la matriz original en una matriz identidad I_n y a su vez esta ultima se transforma en la matriz inversa.

El metodo parte de la suposicion de que A es una matriz no singular, en caso de no serlo el metodo lo puede detectar, dicha situacion se presenta cuando todos los elementos de un renglon de la matriz A o de sus matrices transformadas, son nulos.

Con el fin de minimizar los errores de redondeo, la eliminacion de los elementos se efectua pivoteando sobre los menores elementos que quedan en la matriz A o en sus matrices obtenidas a

partir de esta última por transformación; se tiene cuidado de no emplear como pivotes elementos de renglones que ya han sido utilizados como pivotes.

2.4.2) PROGRAMA INVMAT

OBJETIVO:

Implementar la inversión de matrices en la computadora por el método de GAUSS-JORDAN; para una o varias matrices no singulares.

DESCRIPCION

La entrada de este programa es la dimensión de la matriz a invertir así como los coeficientes de la misma. Las constantes definidas pueden ser modificadas según se requiera. Se recomienda dar los componentes de la matriz por renglones.

ALGORITMO EN PSEUDOCODIGO:

PROGRAMA PRINCIPAL

VARIABLE	USO	TIPO	FUNCION
EPS	-	Constante	Criterio para determinar si el determinante de la matriz considerada es nulo.
N1	-	Constante	Orden máximo posible para las matrices usadas en el programa (es modificable).
A	e	Arreglo de reales	Matriz a la que se le busca la inversa.
NC	e	Entero	Orden de la matriz A.
DET	-	Real	Parámetro que indica si el determinante de la matriz es nulo.

Pseudocodiso:

BEGIN

```

Leer el orden de la matriz.
WHILE no sea fin de archivo DO
  CALL LEEMAT (A,MC,MC)
  INVMAT (A,MC,EPS,DET)
  IF determinante de la matriz es
  mayor que EPS THEN
    Escribe la matriz inversa
  ELSE
    Escribe letrero
  Leer orden para otra Matriz.

```

END.

SUBRUTINAS

1) LEEMAT(D,M,N)

(Lee una matriz de orden MxN; ver sec. 2.2.2)

2) ESCMAT (D,M,N)

(Escribe una matriz de orden MxN; ver sec. 2.2.2)

3) INVMAT (A,N,EPS,DET)

Objetivo: Obtener la inversa de una matriz no singular.

VARIABLE	USO	TIPO	FUNCION
A	E/S	Arreglo de reales	Matriz a la que se buscara la inversa.
N	E	Entero	Orden de la matriz A.
EPS	E	Real	Criterio para determinar si el determinante de A es nulo.
DET	S	Real	Parametro que indica si el determinante es cero.
C	L	Arreglo de reales	Matriz identidad empleada para obtener la matriz inversa.
LC,LR	L	Vector entero	indicadores de renglon y columna que se utilizan.
MVR,MV C	L	Vector	contadores que indican cuales renglones y cuales columnas fueron empleados como pivotes.
RA MAX	L		Menor elemento de la matriz A o de sus transformaciones empleado como elemento pivote.
TEMP	L	Real	Variable de localizacion temporal.

Pseudocódigo:

BEGIN

Inicializa en cero los contadores MVCCIJ
y MVR[II] para toda I.

Se obtiene la matriz identidad en C

K <--- 1

DET <--- 1

WHILE (K <= N) AND (DET > EPS) DO

RAMAX <--- 0.0

LC <---0

LR <---0

Encuentra el mayor pivote A[I,J] De los
no considerados y lo deposita en RAMAX, deja
posición en LR y LC para I y J respectivamente.

DET <--- valor absoluto de RAMAX

IF DET mayor EPS THEN

Hace permutaciones de renglones para que el pivote
quede en la diagonal.

Normaliza el pivote (divide entre RAMAX todo
el renglón donde está el pivote, en A y C).

Hace ceros arriba y abajo del pivote.

Asigna LC a MVR[LC] y MVCCI[LC] para
no usar ese pivote posteriormente.

Incrementa K en uno.

Deposita los valores de C en A.

END.

2.4.3) LISTADO DEL PROGRAMA

```

PROGRAM INVMAT (INPUT,OUTPUT)
CONST
  EPS = 0.0000001
  N1 = 15
TYPE
  MATRIZ = ARRAY[1..N1,1..N1] OF REAL
VAR
  MC : 1..N1
  DET : REAL
  A : MATRIZ

PROCEDURE INVMAT (VAR A : MATRIZ; N : INTEGER; EPS : REAL;
                  VAR DET : REAL)
TYPE
  VECTOR = ARRAY[1..N1] OF INTEGER
  MAT = ARRAY[1..N1,1..N1] OF REAL
VAR
  C : MAT
  I, J, K : INTEGER
  LC, LR : INTEGER
  MVR, MVC : VECTOR
  RAMAX, TEMP : REAL
BEGIN
  < ACTUALIZACION DE VALORES PARA INICIAR PROCESO >
  FOR I := 1 TO N DO
    BEGIN
      MVR[I] := 0
      MVC[I] := 0
    END
  < OBTENCION DE LA MATRIZ IDENTIDAD >
  FOR I := 1 TO N DO
    FOR J := 1 TO N DO
      IF I = J THEN C[I,J] := 1.0
      ELSE
        C[I,J] := 0.0
      < OBTENCION DE LA MATRIZ INVERSA >
      K := 1
      DET := 1
      WHILE ((K <= N) AND (DET > EPS)) DO
        BEGIN
          RAMAX := 0
          LC := 0
          LR := 0
          FOR I := 1 TO N DO
            IF MVR[I] <> I THEN
              BEGIN
                FOR J := 1 TO N DO
                  IF MVC[J] <> J THEN
                    IF ABS(RAMAX) < ABS(A[I,J]) THEN
                      BEGIN
                        RAMAX := A[I,J]
                        LR := I
                        LC := J

```

```

        END;
    DET := ABS(RAMAX);
    IF DET > EPS THEN
        BEGIN
            IF LR <> LC THEN
                FOR I := 1 TO N DO
                    BEGIN
                        TEMP := ACLR,I;
                        ACLR,I := ACLC,I;
                        ACLC,I := TEMP;
                        TEMP := CCLR,I;
                        CCLR,I := CCLC,I;
                        CCLC,I := TEMP;
                    END;
                FOR I := 1 TO N DO
                    BEGIN
                        ACLC,I := ACLC,I / RAMAX;
                        CCLC,I := CCLC,I / RAMAX;
                    END;
                FOR I := 1 TO N DO
                    IF I <> LC THEN
                        BEGIN
                            TEMP := ACI,LC;
                            FOR J := 1 TO N DO
                                BEGIN
                                    ACI,JJ := ACI,JJ - TEMP * ACLC,JJ;
                                    CCI,JJ := CCI,JJ - TEMP * CCLC,JJ;
                                END
                            END;
                            MVRCLC := LC;
                            MVCCLC := LC;
                        END;
                    K := K + 1;
                END(* DEL WHILE *);
                FOR I := 1 TO N DO
                    FOR J := 1 TO N DO
                        ACI,JJ := CCI,JJ;
                    END(** DE INVMAT **);
                END(** DE INVMAT **);

```

(PROGRAMA PRINCIPAL)

```

BEGIN
    WRITELN ('DAR EL ORDEN DE LA MATRIZ');
    READ (MC);
    WHILE NOT EOF DO
        BEGIN
            WRITELN ('DAR COMPONENTES DE LA MATRIZ A');
            LEEMAT (A,MC,MC);
            INVMAT (A,MC,EPS,DET);
            IF DET > EPS THEN
                BEGIN
                    WRITELN ('LA MATRIZ INVERSA ES');
                    ESCMAT (A,MC,MC);
                END
            ELSE

```

```

WRITELN ('NO EXISTE LA MATRIZ INVERSA');
WRITELN ('DAR EL ORDEN DE LA MATRIZ');
READ (MC);

```

```
END
```

```
END.
```

2.4.4) EJEMPLO

Obtener la inversa de la matriz:

1	1	2
1	2	3
3	2	1

Introduciendo directamente los datos a la computadora:

DAR EL ORDEN DE LA MATRIZ

3

DAR COMPONENTES DE LA MATRIZ A

1	1	2
1	2	3
3	2	1

LA MATRIZ INVERSA ES:

1.0000	-0.7500	0.2500
-2.0000	1.2500	0.2500
1.0000	-0.2500	-0.2500

III) SOLUCION DE SISTEMAS DE ECUACIONES LINEALES

3.1) INTRODUCCION

Un sistema de ecuaciones lineales de M ecuaciones con N incógnitas, tiene la siguiente representación:

$$\begin{array}{cccccc}
 a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n & = & b_1 \\
 a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n & = & b_2 \\
 \dots & & \dots \\
 a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n & = & b_m
 \end{array} \quad (1)$$

Donde a_{ij} y b_j ($i=1,2,\dots,m; j=1,2,\dots,n$) son constantes y

Las x_i son las incógnitas del sistema.

Matricialmente este sistema se representa en la forma:

$$Ax = b \quad (2)$$

A se conoce como la matriz de coeficientes del sistema

b Vector de términos independientes.

x Vector de incógnitas.

Ahora veremos condiciones bajo las cuales (1) tiene solución.

Hay que aclarar que estos resultados no son constructivos; no nos dicen cómo obtener la solución, x de (2); esto lo veremos en la siguiente sección.

Hay alguna terminología asociada con el sistema (1) y la solución matricial (2). Sea A matriz de orden $M \times N$; si $M > N$ el sistema se le llama sobredeterminado (hay más ecuaciones que incógnitas) si $M < N$ el sistema se llama indeterminado (hay más incógnitas).

tas que ecuaciones); si $b = 0$, el sistema se llama homogéneo.

Un criterio clásico para determinar una solución de (2) está contenido en el teorema:

TEOREMA.— La ecuación $Ax = b$ tiene una solución si y solo si

$$\text{Rango } (A,b) = \text{Rango } (A)$$

Donde:

(A,b) Se le conoce como matriz aumentada del sistema
 $\text{Rango } (A)$ es la cantidad de vectores linealmente
 independientes del conjunto de vectores
 columna (enslon) que forman la matriz A .

Sin embargo un sistema de ecuaciones puede tener una única solución o una infinidad de soluciones, en el primer caso se llama sistema compatible determinado, y en el segundo sistema compatible indeterminado. En el caso que un sistema no tenga solución se le llama sistema incompatible.

Un sistema compatible determinado se caracteriza por:

$$\text{Rango } (A) = n \quad (\text{numero de incógnitas})$$

Un sistema compatible indeterminado se caracteriza por:

$$\text{Rango } (A) < n$$

Un sistema incompatible se caracteriza por:

$$\text{Rango } (A) < \text{Rango } (A,b)$$

3.2) METODO DE GAUSS-JORDAN

3.2.1) INTRODUCCION

Dado el sistema de ecuaciones $Ax = b$ el metodo consiste en trabajar con la matriz de coeficientes A el vector de terminos independientes, es decir, con la matriz ampliada del sistema:

$$(A, b) \quad (2)$$

A dicha matriz se le aplican una serie de transformaciones que conducen a obtener otra matriz ampliada equivalente:

$$(I_n, C) \quad (3)$$

Donde C representa la solucion de cada una de las incognitas del sistema. El proceso equivale a premultiplicar la ecuacion (2) por la inversa de A , es decir, el metodo de la matriz inversa, solo que este metodo consiste en una eliminacion sistemática de valores. La transformacion de la matriz (2) en la matriz (3) se efectua basandose en tres operaciones que no alteran el sistema de ecuaciones, sino que proporcionan sistemas de ecuaciones equivalentes, ellas son:

- a) Intercambio del renglon R_i por el R_j .
- b) Multiplicacion de un renglon R_i por un escalar, c , diferente de cero, es decir reemplazando R_i por cR_i .
- c) Suma de equimultiplos de un renglon a otro renglon, es decir, reemplazando el renglon R_i por $R_i + cR_j$ ($i \neq j$).

Para aplicar las operaciones anteriores se procede en la si-

siguiente forma:

(1) Seleccionar un renglon pivote y un elemento pivote dentro de dicho renglon.

(2) Normalizar el elemento pivote, es decir, hacerlo unitario.

(3) Cancelar elementos que se encuentren en la columna arriba y/o abajo del elemento pivote, mediante la transformacion c).

(4) Resresar al paso (1) y asi sucesivamente, hasta que la matriz de coeficientes original, A, queda transformada en una matriz identidad In.

Debido a que durante el proceso se presentan errores por redondeo, la forma optima de escoger los elementos pivotes⁺ es seleccionar el mayor elemento que quede en la matriz A o en sus transformaciones, en la k-esima iteracion. Hay que tener presente que los elementos de un renglon que ya fue seleccionado como linea pivote no se pueden usar como pivotes, aun cuando el mayor elemento quede colocado en dicho renglon.

Al seleccionar los pivotes en la forma antes mencionada, el error se reduce al minimo, y debido a que puede quedar una matriz no identidad al termino de las iteraciones, es necesario efectuar un intercambio de lineas hasta obtener In. Cabe mencionar que el presente metodo es un metodo directo de solucion que no requiere que se determine con anterioridad si el sistema es compatible y determinado, el metodo durante el proceso proporciona dicha informacion.

+ El concepto de pivots se entendera mas claramente en la siguiente seccion.

Si el sistema es compatible y determinado, el procedimiento descrito se puede llevar a cabo sin contratiempos hasta llegar a:

$$(I_n, C)$$

Si el sistema es compatible pero indeterminado, la matriz ampliada adquirirá una configuración como la siguiente:

$$\left[\begin{array}{ccc|c} 1 & 0 & 2 & 1 \\ 0 & 1 & 2 & 2 \\ 0 & 0 & 0 & 0 \end{array} \right]$$

Es decir, un renglón será nulo; en esta situación se obtienen las ecuaciones independientes que restan en el sistema y se aplica la metodología correspondiente a sistemas indeterminados.

Si el sistema es incompatible, se presenta also como:

$$\left[\begin{array}{ccc|c} 1 & 1 & 2 & 1 \\ 0 & 2 & 3 & 2 \\ 0 & 0 & 0 & K \end{array} \right]$$

Donde, si $K \neq 0$ esto daría una contradicción.

3.2.2) PROGRAMA GAUSSJORDAN

OBJETIVO:

Obtener la solución del sistema de ecuaciones lineales $Ax = b$ para varios paquetes de datos, es decir, para varias matrices A y varios vectores de términos independientes b (usando el método antes descrito).

DESCRIPCION

Este programa pide como entrada la dimension del sistema a resolver, la matriz de coeficientes del sistema y el vector de terminos independientes. Las constantes definidas en el programa pueden ser modificadas si se requieren resolver sistemas de orden mayor que 15 y con otro grado de precision.

ALGORITMO EN PSEUDOCODIGO:

PROGRAMA PRINCIPAL

VARIABLE	USO	TIPO	FUNCION
EPS	-	Constante	Criterio para determinar si el determinante de la matriz considerada es nulo.
N1	-	Constante	Orden maximo posible para las matrices usadas en el programa (es modificable).
A	E	Arreglo de reales	Matriz de coeficientes del sistema de ecuaciones.
B	E/S	Arreglo de reales	Vector de terminos independientes del sistema de ecuaciones; durante el proceso se transforma en la solucion.
MC	E	Entero	Orden de la matriz.
DET	-	Real	Parametro que indica si el determinante de la matriz es nulo.

Pseudocodiso:

BEGIN

 Dar orden de la matriz del sistema, MC.

REPEAT

 Leer la matriz A.

 Leer el vector de terminos independ., B.

 CALL GAUSJOR (A,B,MC,EPS,DET)

 IF DET > EPS THEN

 Escribe el vector solucion.

 ELSE

 Escribe letrero de indicacion.

 Dar orden de la matriz para otro sistema.

 UNTIL fin de archivo.

END.

SUBROUTINAS

1) LEEMAT (D,M,N)

(Lee una matriz de orden $M \times N$, ver seccion 2.2.2).

2) ESCMAT (D,M,N)

(Escribe una matriz de orden $M \times N$, ver seccion 2.2.2).

3) LEEVECT(D,M)

(Lee un vector de orden M , entran como parametros la matriz a leer y el orden).

3.1) ESCVECT (D,M)

(Escribe un vector de orden M).

4) GAUSJOR (A,B,N,EPS,DET)

OBJETIVO: Obtener la solucion del sistema de ecuaciones lineales $Ax = B$, en B se obtiene la solucion.

VARIABLE	USO	TIPO	FUNCION
A	E	Arreglo de Reales	Matriz de coeficientes del sistema de ecuaciones.
B	E/S	Arreglo de Reales	Vector de terminos independientes del sistema de ecuaciones, durante el proceso se transforman en la solucion.
N	E	Entero	Orden del sistema de ecuaciones.
EPS	E	Real	Criterio para determinar si el determinante de la matriz A es nulo.
DET	E	Real	Parametro que indica si el determinante de la matriz es cero.
LR,LC	L	Entero	Indicadores del renglon y columna que se utilizan.
MVR,MVC	L	Arreglo	Contadores que indican que renglon y columna ya fueron usados.
RAMAX	L	Real	Mayor elemento de la matriz A que se emplea como pivote.
TEMP	L	Real	Variable de localizacion temporal

Pseudocodiso:

BEGIN

Inicializa en cero los contadores MURCII,

MUCIJJ para toda 'I'.

K <--- 1.

DET <--- 1.

WHILE (K <= N) y (DET > EPS) DO

RAMAX <---0.

LC <---0.

LR <---0.

Encuentra el mayor pivote de los no considerados

la posición (I,J) las deja en LR y LC respect.

DET <--- valor absoluto de RAMAX.

IF DET > EPS THEN

Hacer permutaciones de renglones para que

el pivote quede en la diagonal.

Normaliza el pivote (divide entre RAMAX todo

el renglon donde esta el pivote, en A y B).

Hacer ceros arriba y abajo del pivote.

Asigna LC a MURCLCJ y MUCCLCJ para ya no

usar este pivote.

Incrementa K en uno.

END.

3.2.3) LISTADO DEL PROGRAMA

```

PROGRAM GAUSSJORDAN (INPUT,OUTPUT);
CONST
  EPS = 0.0000001;
  N1 = 15;
TYPE
  MATRIZ = ARRAY[1..N1,1..N1] OF REAL;
  VECTOR = ARRAY[1..N1] OF REAL;
VAR
  MC : 1..N1;
  DET : REAL;
  A : MATRIZ;
  B : VECTOR;

PROCEDURE LEEVECT (VAR D : VECTOR; M : INTEGER);
VAR
  I : INTEGER;
BEGIN
  FOR I := 1 TO M DO
    READ(D[I]);
  Writeln
END;

PROCEDURE ESCVECT (D : VECTOR; M : INTEGER);
VAR
  I : INTEGER;
BEGIN
  FOR I:= 1 TO M DO
    WRITE(D[I] : 12 : 4);
  Writeln
END;

PROCEDURE GAUSJOR (VAR A : MATRIZ; VAR B : VECTOR; N : INTEGER;
  EPS : REAL; VAR DET : REAL);
TYPE
  VECTOR = ARRAY[1..N1] OF INTEGER;
VAR
  I, J, K : INTEGER;
  LC, LR : INTEGER;
  MVR, MVC : VECTOR;
  RAMAX, TEMP : REAL;
BEGIN
  (* ACTUALIZACION DE VALORES PARA INICIAR PROCESO *)
  FOR I := 1 TO N DO
    BEGIN
      MVR[I] := 0;
      MVCC[I] := 0;
    END;

  (* SOLUCION DEL SISTEMA DE ECUACIONES *)
  K:=1;
  DET:=1;
  WHILE ((K <= N) AND (DET > EPS)) DO
    BEGIN

```

```

RAMAX := 0.0;
LC := 0;
LR := 0;

FOR I:= 1 TO N DO
  BEGIN
    IF MVR(I) <> I THEN
      FOR J := 1 TO N DO
        IF MVCE(J) <> J THEN
          IF ABS(RAMAX) < ABS(ACI,J) THEN
            BEGIN
              RAMAX := ACI,J;
              LR := I;
              LC := J;
            END;
          END;
        END;
      END;
    END;

  DET := ABS(RAMAX);
  IF DET > EPS THEN
    BEGIN
      IF LR <> LC THEN
        BEGIN
          FOR I := 1 TO N DO
            BEGIN
              TEMP := ACLR,I;
              ACLR,I := AELC,I;
              AELC,I := TEMP;
            END;
            TEMP := BELR,J;
            BELR,J := BELC,J;
            BELC,J := TEMP;
          END;
          FOR I := 1 TO N DO
            AELC,I := AELC,I / RAMAX;
            BELC,J := BELC,J / RAMAX;
          END;
          FOR I := 1 TO N DO
            IF I <> LC THEN
              BEGIN
                TEMP := ACI,LC;
                BC(I) := BC(I) - TEMP * BELC,J;
                FOR J := 1 TO N DO
                  ACI,J := ACI,J - TEMP * AELC,J;
                END;
              END;
            END;
          MVR(LC) := LC;
          MVCE(LC) := LC;
        END;
      K := K + 1;
    END(* DEL WHILE *);
  END(* DE GAUSJOR *);

```

(PROGRAMA PRINCIPAL)

```

BEGIN
  WRITELN (' DAR EL ORDEN DE LA MATRIZ ');

```

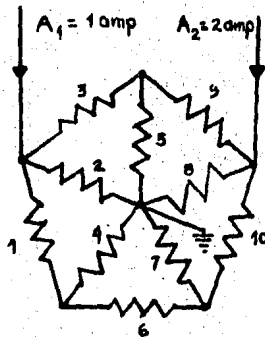
```

READ (MC);
REPEAT
  WRITELN ('DAR COMPONENTES DE LA MATRIZ A');
  LEEMAT (A,MC,MC);
  WRITELN ('DAR COMPONENTES DEL VECTOR B');
  LEEVECT (B,MC);
  GAUSJOR (A,B,MC,EPS,DET);
  IF DET > EPS THEN
    BEGIN
      WRITELN ('LA SOLUCION ES:');
      ESCVECT (B,MC)
    END
  ELSE
    WRITELN ('NO EXISTE LA SOLUCION');
  WRITELN;
  WRITELN ('DAR EL ORDEN DE LA MATRIZ');
  READ (MC);
UNTIL EOF;
END.

```

3.2.4) EJEMPLO

Empleando las leyes de Kirchhoff en el circuito mostrado en la figura:



FIGURA

Se establece el siguiente conjunto de ecuaciones:

$$\begin{array}{rcl}
 I_1 + I_2 & & I_3 = A_1 \\
 +I_8 + I_9 & & +I_{10} = A_2 \\
 -I_1 + I_4 & & -I_6 = 0
 \end{array}$$

$$\begin{array}{r}
 -I_3 + I_5 - I_9 = 0 \\
 I_6 + I_7 - I_{10} = 0 \\
 -R_7 I_7 + R_8 I_8 - R_{10} I_{10} = 0 \\
 -R_5 I_5 + R_8 I_8 - R_9 I_9 = 0 \\
 R_2 I_2 - R_3 I_3 - R_5 I_5 = 0 \\
 -R_1 I_1 + R_2 I_2 - R_4 I_4 = 0 \\
 -R_1 I_1 - R_4 I_4 + R_7 I_7 = 0
 \end{array}$$

Donde $R_1=1$, $R_2=10$, $R_3=35$, $R_4=23$, $R_5=100$, $R_6=25$,

$R_7=50$, $R_8=75$, $R_9=5$, $R_{10}=50$ Ohms.

Resolver este sistema para las corrientes de I_1 a I_{10} .

Al correr el programa, pide datos que se introducen directamente:

DAR EL ORDEN DE LA MATRIZ
10

DAR COMPONENTES DE LA MATRIZ A

1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	1	1	1
-1	0	0	1	0	-1	0	0	0	0
0	0	-1	0	1	0	0	0	-1	0
0	0	0	0	0	1	1	0	0	-1
0	0	0	0	0	0	-50	75	0	-50
0	0	0	0	-100	0	0	75	-5	0
0	10	-35	0	-100	0	0	0	0	0
-1	10	0	-23	0	0	0	0	0	0
0	0	0	-23	0	-25	50	0	0	0

DAR VECTOR DE TERMINOS INDEPENDIENTES

.1 2 0 0 0 0 0 0 0 0

LA SOLUCION DEL SISTEMA ES:

0.3776	1.2622	-0.6399	0.5324	0.3502
0.1548	0.3223	0.5329	0.9900	0.4771

3.3) METODO DE GAUSS-SEIDEL

3.3.1) INTRODUCCION

El metodo de Gauss-Seidel es un metodo de tipo iterativo que sirve para la solucion de sistemas del tipo:

$$Ax = b \quad (1)$$

cuando los valores numericos de los elementos de la diazonal principal son mayores que los demas de su correspondiente rension. Para asegurar la convergencia se requiere que:

a) Los elementos no nulos de la matriz de coeficientes (A) se acumulen en la diazonal principal.

b)

$$|a_{ii}| > \sum_{j=1, j \neq i}^n |a_{ij}| \quad i=1,2,\dots,n$$

Cabe mencionar que el criterio de convergencia depende del tipo de metrica que se este usando (en este caso estamos usando la distancia euclidiana).

Para aplicar el metodo se procede a despejar una incognita del arreglo:

$$\begin{array}{r} a_{11} x_1 + a_{12} x_2 + \dots + a_{1n} x_n = b_1 \\ a_{21} x_1 + a_{22} x_2 + \dots + a_{2n} x_n = b_2 \\ \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \\ \cdot \quad \cdot \quad \cdot \quad \cdot \quad \cdot \end{array} \quad (2)$$

$$a_{n1} x_1 + a_{n2} x_2 + \dots + a_{nn} x_n = b_n$$

De cada ecuación, sin perder generalidad podemos despejar la incógnita X_i de la 'i-esima' ecuación, o sea:

$$\begin{aligned} x_1 &= \frac{1}{a_{11}} (b_1 - a_{12} x_2 - a_{13} x_3 - \dots - a_{1n} x_n) \\ x_2 &= \frac{1}{a_{22}} (b_2 - a_{21} x_1 - a_{23} x_3 - \dots - a_{2n} x_n) \\ &\dots \\ x_n &= \frac{1}{a_{nn}} (b_n - a_{n1} x_1 - a_{n2} x_2 - \dots - a_{n,n-1} x_{n-1}) \end{aligned} \quad (3)$$

y se establecen las siguientes ecuaciones iterativas:

$$\begin{aligned} x_1^{(k+1)} &= \frac{1}{a_{11}} (b_1 - a_{12} x_2^{(k)} - a_{13} x_3^{(k)} - \dots - a_{1n} x_n^{(k)}) \\ x_2^{(k+1)} &= \frac{1}{a_{22}} (b_2 - a_{21} x_1^{(k+1)} - a_{23} x_3^{(k)} - \dots - a_{2n} x_n^{(k)}) \\ &\dots \\ x_n^{(k+1)} &= \frac{1}{a_{nn}} (b_n - a_{n1} x_1^{(k+1)} - a_{n2} x_2^{(k+1)} - \dots - a_{n,n-1} x_{n-1}^{(k+1)}) \end{aligned} \quad (4)$$

Donde:

$x_i^{(k+1)}$ Indica el valor de la 'i-esima' incógnita en la iteración 'k+1'.

Para arrancar el método se establece una solución inicial:

$$X = \begin{pmatrix} 0 & 0 & \dots & 0 \\ x_1 & x_2 & \dots & x_n \\ 0 & 1 & 2 & \dots & n \end{pmatrix}$$

Dichos valores se sustituyen en el lado derecho de la ecuacion (4) para obtener la siguiente solucion aproximada:

$$X_1 = (x_1^1, x_2^1, \dots, x_n^1) \quad \text{sea} \quad \text{RESID} = \max |x_i^1 - x_i^0|$$

$$\text{luego} \quad X_2 = (x_1^2, x_2^2, \dots, x_n^2), \quad \text{RESID} = \max |x_i^2 - x_i^1|$$

$$\text{y asi hasta:} \quad \text{RESID} = \max |x_i^{k+1} - x_i^k| < \text{EPS}$$

Para poder emplear este metodo es necesario verificar con anterioridad que el sistema sea compatible y determinado; ademas que cumpla con las condiciones de convergencia.

Algunos sistemas a primera vista no cumplen los requisistos del metodo, y sin embargo pueden cumplirlos mediante un simple intercambio en la posicion de las ecuaciones, un apropiado cambio de variable o definiendo otra distancia.

3.3.2) PROGRAMA GAUSSEIDEL

OBJETIVO:

Resolver el sistema $Ax = b$, via aproximaciones sucesivas (usando el metodo antes descrito), para varios paquetes de datos.

DESCRIPCION

La entrada a este programa es el orden de la matriz del sistema, maximo numero de iteraciones, coeficientes de la matriz,

vector de terminos independientes, vector inicial para arrancar al metodo. Las constantes definidas en el programa pueden modificarse en caso de querer resolver sistemas de orden mayor que 15 y otro grado de precision.

ALGORITMO EN PSEUDOCODIGO:

PROGRAMA PRINCIPAL

VARIABLE	USO	TIPO	FUNCION
EPS	-	Constante	Criterio para determinar si se convergio el metodo.
NI	-	Constante	Orden maximo posible para las matrices usadas en el programa (es modificable).
MC	E	Entero	Orden de la matriz del sistema.
A	E	Arreglo de Reales	Matriz de coeficientes del sistema.
B	E	Arreglo de Reales	Vector de terminos independientes.
XANT	E/S	Arreglo de Reales	Vector con el valor inicial de las incognitas del sistema.
XPOST	-	Arreglo de Reales	Vector con el valor de las incognitas en nueva iteracion.
M	E	Entero	Maximo numero de iteraciones a efectuar.
RESID	-	Real	Variable que contiene el mayor elemento de las diferencias: $ x_i^{k+1} - x_i^k $ para toda "i" en la ultima iteracion, $k+1$.

Pseudocodiso:

BEGIN

Leer orden de la matriz y maximo numero de iteraciones, MC y M respect.

REPEAT

Leer la matriz A

Leer el vector de terminos independientes.

Leer vector inicial (solucion inicial).

CALL GAUSSEIDEL (A,B,XANT,M,MC,EPS,RESID).

IF RESID < EPS THEN

Escribe vector solucion del sistema.

Leer orden y maximo numero de iteraciones para otra matriz, si se quiere.

UNTIL fin de archivo.

END.

SUBROUTINAS

1) LEEMAT (D,M,N)

(Lee una matriz de orden $M \times N$, ver sec. 2.2.2).

2) ESCMAT (D,M,N)

(Escribe una matriz de orden $M \times N$, ver sec 2.2.2).

3) LEEVECT (D,M)

(Lee un vector de orden M , ver sec. 3.2.2).

4) GAUSSEIDEL (A,B,XANT,M,MC,EPS,RESID)

Objetivo: Obtener la solución del sistema $Ax = b$, la solución se da en el vector XANT.

VARIABLE	USO	TIPO	FUNCION
A	E	Arreglo de Reales	Matriz de coeficientes del sistema.
B	E	Arreglo	Vector de terminos independientes.
XANT	E/S	Arreglo	Valor inicial de las incognitas sistema; finalmente este vector contiene la solución del sistema.
XPOST	L	Arreglo de Reales	Vector con el valor de las incognitas en la siguiente iteración.
M	E	Entero	Maximo numero de iteraciones a efectuar.
MC	E	Entero	Orden de la matriz considerada.
EPS	E	Real	Criterio de convergencia.
RESID	E/S	Real	Variable que contiene la diferencia menor en valor absoluto de cada una incognitas, en la iteración "i-esima" y la siguiente.
SUMA	L	Real	Variable usada como sumador.
OK	L	Booleana	Variable que indica si se cumplen las condiciones de convergencia.

Pseudocódigo:

BEGIN

Checar condición de convergencia colocando en la variable OK verdadero o falso según el caso.

IF OK = FALSE THEN

 Escribe mensaje

ELSE

 Asigna los valores del vector XANT al vector XPOST.

 RESID <--- 1.

 K <--- 1.

 WHILE K <= M y RESID >= EPS DO

 RESID <--- 0.0.

 FOR I := 1 TO N DO

 Obtiene la aproximación siguiente del elemento "I" del vector solución y lo deposita en XPOST[I].

 IF valor absoluto de (XPOST[I] - XANT[I]) es mayor que RESID THEN

 RESID <--- valor absoluto de XPOST[I]

 XANT[I] <--- XPOST[I].

 Incrementa K en uno.

 IF RESID >= EPS THEN

 Escribe mensaje de no convergencia.

END.

3.3.3) LISTADO DEL PROGRAMA

```

PROGRAM GAUSSEIDEL (INPUT,OUTPUT);
CONST
  EPS = 0.0000001;
  N1 = 15;
TYPE
  MATRIZ = ARRAY[1..N1,1..N1] OF REAL;
  MATRIZ1 = ARRAY[1..N1] OF REAL;
VAR
  MC, M : 1..N1;
  RESID : REAL;
  A : MATRIZ;
  B,XANT : MATRIZ1;

PROCEDURE GAUSEIDEL (VAR A : MATRIZ; VAR B, XANT : MATRIZ1;
  M, N : INTEGER) EPS : REAL;VAR RESID : REAL);
TYPE
  VECTOR = ARRAY[1..N1] OF REAL;
VAR
  I, J, K : INTEGER;
  SUMA : REAL;
  XPOST : VECTOR;
  OK : BOOLEAN;
BEGIN
  (* CHECAR CONDICION DE CONVERGENCIA *)
  I := 1;
  OK := TRUE;
  WHILE ((I <= N) AND OK) DO
    BEGIN
      SUMA := 0.0;
      FOR J := 1 TO N DO
        IF I <> J THEN
          SUMA := SUMA + ABS(A[I,J]);
        IF (ABS(A[I,I]) - SUMA) < 0 THEN
          OK := FALSE;
        I := I + 1;
      END(*DEL WHILE *);
    IF OK = FALSE THEN
      BEGIN
        WRITELN (' EL SISTEMA NO CUMPLE CONDICION DE CONVERGENCIA');
        RESID := 1;
      END
    ELSE
      BEGIN
        FOR I := 1 TO N DO
          XPOST[I] := XANT[I];
          RESID := 1;
          K := 1;
          WHILE ((K <= M) AND (RESID >= EPS)) DO
            BEGIN
              RESID := 0.0;
              FOR I := 1 TO N DO
                BEGIN
                  SUMA := 0.0;

```



```

FOR J := 1 TO N DO
  IF J <> I THEN
    SUMA := SUMA + A(I,J) * XPOST(I,J)
  XPOST(I,J) := (B(I) - SUMA)/A(I,I)
  IF ABS(XPOST(I,J) - XANT(I,J)) > RESID THEN
    RESID := ABS(XPOST(I,J))
    XANT(I,J) := XPOST(I,J)
  END;
  (* SE COMPLETA UNA PASADA SE PUEDEN IMPRIMIR DATOS*)
  K := K + 1;
END(* DEL WHILE *);
IF RESID >= EPS THEN
  WRITELN ('EL PROCESO NO CONVERGIO EN',M,' ITERACIONES')
END;
WRITELN
END(* DE GAUSSEIDEL *);

(***** PROGRAMA PRINCIPAL *****)

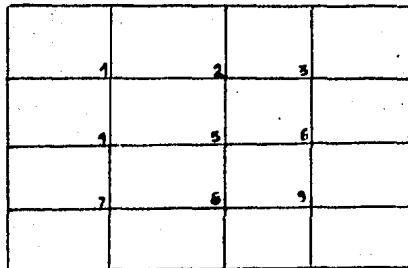
BEGIN
  WRITELN ('DAR ORDEN DE LA MATRIZ Y MAXIMO NO. DE ITERACIONES');
  READ (M,M);
  REPEAT
    WRITELN('DAR COMPONENTES DE LA MATRIZ A');
    LEEMAT (A,M,M);
    WRITELN('DAR EL VECTOR DE TERMINOS INDEPENDIENTES');
    LEEVECT (B,M);
    WRITELN('DAR SOLUCION INICIAL');
    LEEVECT (XANT,M);
    GAUSSEIDEL (A,B,XANT,M,M,EPS,RESID);
    IF RESID < EPS THEN
      BEGIN
        WRITELN ('LA SOLUCION DEL SISTEMA ES: ');
        ESCVECT (XANT,M);
      END;
    WRITELN;
    WRITELN ('DAR ORDEN DE LA MATRIZ Y MAXIMO NO. DE ITERACIONES');
    READ (M,M);
  UNTIL EOF;
END.

```

3.3.4) EJEMPLO

Una persona se encuentra perdida en un laberinto cuadrado de corredores (ver figura). En cada interseccion escoge una direccion al azar y sigue hasta la siguiente interseccion donde escoge de nuevo al azar y asi sucesivamente. Cual es la Probabilidad que una persona que parta de la interseccion i emerja eventualmente

Por el lado sur?



FIGURA

La solución a este problema es de la siguiente manera: supongamos que hay exactamente 9 intersecciones interiores, como se muestra. Sea P_1 la probabilidad que una persona que sale de la intersección 1 emerge en el lado sur. Sean P_2, \dots, P_9 definidas de manera similar. Suponiendo que en cada intersección a que llegue la persona hay tanta posibilidad que escoja una dirección como otra y habiendo llegado a una salida ha terminado su caminata, la teoría de la probabilidad ofrece las siguientes 9 ecuaciones para las P_k :

$$P_1 = 1/4(0 + 0 + P_2 + P_4)$$

$$P_5 = 1/4(P_2 + P_4 + P_6 + P_8)$$

$$P_2 = 1/4(0 + P_1 + P_3 + P_5)$$

$$P_6 = 1/4(P_3 + P_5 + 0 + P_9)$$

$$P_3 = 1/4(0 + P_2 + 0 + P_6)$$

$$P_7 = 1/4(P_4 + 0 + P_8 + 1)$$

$$P_4 = 1/4(P_1 + 0 + P_5 + P_7)$$

$$P_8 = 1/4(P_5 + P_7 + P_9 + 1)$$

$$P_9 = 1/4(P_6 + P_8 + 0 + 1)$$

Al correr este programa, se le dan los siguientes datos que pide:

DAR ORDEN DE LA MATRIZ Y MAXIMO NO. DE ITERACIONES

9 90
 DAR COMPONENTES DE LA MATRIZ
 -4 1 0 1 0 0 0 0 0
 1 -4 1 0 1 0 0 0 0
 0 1 -4 0 0 1 0 0 0
 1 0 0 -4 1 0 1 0 0
 0 1 0 1 -4 1 0 1 0
 0 0 1 0 1 -4 0 0 1
 0 0 0 1 0 0 -4 1 0
 0 0 0 0 1 0 1 -4 1
 0 0 0 0 0 1 0 1 -4

DAR VECTOR DE TERMINOS INDEPENDIENTES

0 0 0 0 0 0 -1 -1 -1

DAR SOLUCION INICIAL

0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5 0.5

LA SOLUCION DEL SISTEMA ES:

0.0714 0.0982 0.0714 0.1875 0.2500
 0.1875 0.4286 0.5262 0.4286

3.4.1) INTRODUCCION

En esta seccion consideraremos la solucion del sistema de ecuaciones lineales $Ax=b$, por medio de la eliminacion Gaussiana con pivoteo parcial, considerando las ventajas de trabajar con la representacion matricial. Se aclaran dos aspectos de la eliminacion Gaussiana que son la busqueda de los pivotes y la interpretacion de los efectos de errores de redondeo.

Por medio de un ejemplo, ilustraremos el metodo e iremos definiendo algunos terminos propios para la eliminacion. Sea el siguiente sistema de ecuaciones, en notacion matricial, de orden 3:

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

El primer paso consiste en usar la primera ecuacion para eliminar x_1 de las otras ecuaciones; esto se logra sumando 0.3 veces la primera ecuacion a la segunda ecuacion y sumando -0.5 veces la primera ecuacion a la tercera ecuacion. Las cantidades 0.3 y -0.5 se les llama multiplicadores. Estos multiplicadores son la expresion decimal de $-(-3/10)$ y $-(5/10)$ respectivamente. Despues de este paso el sistema queda:

$$\begin{bmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 2.5 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 6.1 \\ 2.5 \end{bmatrix}$$

El segundo paso consistirá en eliminar la x_2 de la tercera ecuación haciendo uso de la segunda ecuación. Sin embargo, el coeficiente de x_2 en la segunda ecuación es un número muy pequeño, -0.1. Por tal motivo, las dos últimas ecuaciones serán intercambiadas. Esto no es necesario en este ejemplo porque no hay errores de redondeo, pero es crucial en general.

$$\begin{bmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & -0.1 & 6 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 2.5 \\ 6.1 \end{bmatrix}$$

Ahora, la segunda ecuación puede usarse para eliminar x_2 de la tercera ecuación. Esto se logra sumando 0.04 veces la segunda ecuación a la tercera ecuación:

$$\begin{bmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ 2.5 \\ 6.2 \end{bmatrix}$$

La última ecuación es ahora:

$$6.2 x_3 = 6.2$$

Resolviendo: $x_3 = 1$; este valor se sustituye en la segunda ecuación

$$2.5 x_2 + (5)(1) = 2.5$$

$$\text{y nos da } x_2 = -1.$$

Finalmente los valores de x_2 y x_3 se sustituyen en la primera ecuación: $10 x_1 + (-7)(-1) = 7$, esto da $x_1 = 0$. La solución se puede

chechar facilmente usando el sistema original:

$$\begin{bmatrix} 10 & -7 & 0 \\ -3 & 2 & 6 \\ 5 & -1 & 5 \end{bmatrix} \begin{bmatrix} 0 \\ -1 \\ 1 \end{bmatrix} = \begin{bmatrix} 7 \\ 4 \\ 6 \end{bmatrix}$$

En general, la eliminacion Gaussiana involucra dos estados, la eliminacion "hacia adelante" y la sustitucion "hacia atras".

La eliminacion "hacia adelante" consiste de (n-1) pasos. En el k-esimo paso, multiples de la k-esima ecuacion son sustraídos de las ecuaciones restantes para eliminar la k-esima variable. Si el coeficiente de la x_k es "pequeno", se intercambian las ecuaciones antes de hacer esto. Es decir, despues de hacer el ultimo cambio mencionado, para eliminar la k-esima variable de las ecuaciones restantes, se procede así: (J-esima ecuacion) - $(a_{jk}/a_{kk}) * (k-esima ecuacion)$. ($J=k+1, \dots, n$).

La sustitucion "hacia atras" consiste en resolver la ultima ecuacion para x_n , posteriormente se resuelve para x_{n-1} de la penultima ecuacion, y así sucesivamente hasta que x_1 es calculada de la primera ecuacion.

El algoritmo completo puede ser expresado en notacion matricial. Para nuestro ejemplo tenemos:

$$\text{Sea } L_1 = \begin{bmatrix} 1 & 0 & 0 \\ 0.3 & 1 & 0 \\ -0.5 & 0 & 1 \end{bmatrix}$$

$$\text{entonces } L_1 A = \begin{bmatrix} 10 & -7 & 0 \\ 0 & -0.1 & 6 \\ 0 & 2.5 & 5 \end{bmatrix}, \quad L_1 b = \begin{bmatrix} 7 \\ 6.1 \\ 2.5 \end{bmatrix}$$

$$\text{Sea } P_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} \quad L_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & .04 & 1 \end{bmatrix}$$

$$\text{entonces } L_2 P_2 L_1 A = \begin{bmatrix} 10 & -7 & 0 \\ 0 & 2.5 & 5 \\ 0 & 0 & 6.2 \end{bmatrix}, \quad L_2 P_2 L_1 b = \begin{bmatrix} 7 \\ 2.5 \\ 6.2 \end{bmatrix}$$

El punto principal está en los cambios que han hecho las transformaciones L_1 , P_2 y L_2 ; el producto $U = L_2 P_2 L_1 A$ es una matriz triangular superior; esto es, todos los elementos diferentes de cero están en el lado superior derecho de la matriz. Con tal matriz, el sistema de ecuaciones

$$Ux = C$$

Es fácilmente resuelto usando sustitución "hacia atrás". Tomando $C = L_2 P_2 L_1 b$, el sistema $Ux = C$ tiene la misma solución que el sistema original $Ax=b$.

Una relación similar es válida en general. Sea P_k , ($k=1, \dots, n-1$), aquella matriz identidad, I , modificada por el intercambio de renglones de la misma manera que los renglones de A fueron intercambiados en el k -ésimo paso de la eliminación. Sea L_k aquella matriz identidad modificada por la inserción de los multiplicadores usados en el k -ésimo paso, abajo de la diagonal en la k -ésima columna. Cada matriz P_k es una matriz de permutación

y cada L es una simple matriz triangular inferior. Sean L y U los productos:

$$L = L_{n-1} P_{n-1} \dots L_2 P_2 L_1 P_1$$

$$U = LA$$

Entonces U es una matriz triangular superior, el sistema $Ux=Lb$ puede resolverse mas facilmente, y la solucion es la misma que para el sistema $Ax=b$. La matriz L no necesariamente es una matriz triangular inferior, pero es el producto de permutaciones de simples matrices triangulares inferiores. Consecuentemente la relacion $A = L^{-1}U$ es llamada algunas veces factorizacion triangular de A .

Es importante enfatizar que no se ha introducido nada nuevo; la factorizacion triangular es la simple eliminacion Gaussiana expresada en notacion matricial. Los elementos de la diagonal de U se les llama pivotes; el k -esimo pivote es el coeficiente de la k -esima variable en la k -esima ecuacion, en el k -esimo paso de la eliminacion. En el ejemplo expuesto los pivotes son 10, 2.5 y 6.2.

El computo de los multiplicadores y la sustitucion 'hacia atras' requieren de divisiones por los pivotes. Consecuentemente el algoritmo no puede llevarse a cabo si cualquiera de los pivotes es cero. Intuitivamente esto nos dice que seria una mala idea completar el computo, si alguno de los pivotes es cercano a cero (la causa de esta dificultad es el uso de multiplicadores muy grandes).

Se ve en general que si los multiplicadores son todos menores o iguales a uno, en magnitud, entonces el computo de la solución puede ser aproximado en cierto sentido. Una manera de resolver lo anterior es usar pivoteo parcial que consiste en lo siguiente: en el k -ésimo paso de la eliminación "hacia adelante", se toma como pivote al mayor elemento (en valor absoluto), en la parte considerada de la k -ésima columna. El renglón conteniendo este pivote es intercambiado con el k -ésimo renglón, trayendo el elemento pivote a la posición (k,k) . El mismo intercambio puede hacerse con los elementos del lado derecho del sistema, es decir, b . Las incógnitas en el vector de incógnitas, x , no son reordenadas porque las columnas de A no son intercambiadas.

3.4.2) PROGRAMA FACTORIZACION

OBJETIVO:

Resolver el sistema de ecuaciones $Ax=b$, para uno o varias matrices A y vectores, b , por el método de eliminación Gaussiana con pivoteo parcial.

DESCRIPCION

Este programa resuelve cualquier sistema de ecuaciones de la forma $Ax=b$, hasta de orden 15. En caso de requerir resolver sistemas de mayor orden, modificar la constante N del programa. La información que se le debe dar al programa es: orden de la matriz del sistema, componentes de la matriz y coeficientes del vector de términos independientes.

Este programa tiene dos subrutinas, propias de las etapas:

eliminación "hacia adelante" y sustitución "hacia atrás" que son TRIANGULARIZA y RESUELVE. TRIANGULARIZA puede usarse para calcular el valor de un determinante; además de la información de la eliminación Gaussiana queda almacenada en la misma matriz A, del sistema.

El determinante de la matriz A puede obtenerse, después de salir de la subrutina TRIANGULARIZA así: en el último componente de PIV, TRIANGULARIZA regresa el valor de +1 si un número par de intercambios fueron hechos; regresa -1 si un número impar de intercambios se efectuaron. El valor del determinante se obtiene multiplicando los elementos de la diagonal de la matriz de salida, A, multiplicada por PIVEN]. En el caso que TRIANGULARIZA detecte una matriz singular, PIVEN] será 0. Hay que hacer notar que la forma de obtener el determinante se debe gracias a las propiedades: la suma de un múltiplo de un renglón a otro, no cambia el valor del determinante; el intercambio de dos líneas (renglón o columna) cambia el signo del determinante; el determinante de una matriz triangular es igual al producto de los elementos de la diagonal.

La subrutina RESUELVE usa los resultados de TRIANGULARIZA para obtener la solución del sistema triangular o obtener. Esta subrutina puede resolver varios sistemas con la misma matriz A, obtenida de TRIANGULARIZA y varios vectores independientes, b.

ALGORITMO EN PSEUDOCODIGO:

PROGRAMA PRINCIPAL

VARIABLE	USO	TIPO	FUNCION
N1	-	Constante	Variable que indique el orden máximo de los sistemas, que resuelve

A	E	Arreglo	el programa (es modificable).
MC	E	Entero	Matriz del sistema de ecuaciones,
		de reales	Orden del sistema a resolverse, al final se transforma en una ma- triz triangular superior.
B	E/S	Arreglo	Vector de terminos independientes
		de reales	del sistema, finalmente contiene la solucion del sistema.
PIV	-	Arreglo	Vector conteniendo la informacion
		de reales	del pivoteo.

Pseudocodigo:

BEGIN

Dar el orden de la matriz A, MC.

REPEAT

CALL LEE MAT (A,MC,MC).

Lee el vector de terminos independientes.

CALL TRIANGULARIZA (A,PIV,MC).

IF PIVEMCJ <> 0 THEN

RESUELVE (A,PIV,MVC,B).

Escribe la solucion del sistema.

ELSE

Escribe letrero de indicacion.

Dar orden de la matriz A, MC.

UNTIL fin de archivo.

END.

SUBRUTINAS

1) LEE MAT (D,M,N)

(Lee una matriz de orden $M \times N$, ver sec. 2.2.2).

2) LEE VECT (D,M)

(Lee un vector de orden M, ver sec. 2.2.2)

3) ESC VECT (D,M)

(Escribe un vector de orden M).

4) TRIANGULARIZA (A,PIV,N)

Objetivo: transforma la matriz A en una matriz triangular superior, decir, efectua la eliminacion Gaussiana con pivoteo parcial dejando la informacion de los multiplicadores en la parte triangular inferior.

VARIABLE	USO	TIPO	FUNCION
A	E/S	Arreglo	Matriz del sistema de ecuaciones, de reales finalmente se transforma en una ma-

			triz triangular superior, con informacion en su parte inferior de los multiplicadores.
PIV	S	Arreglo de reales	Vector conteniendo la informacion del pivoteo para $k=1, \dots, n-1$. En PIV[N] se tendra cero, si el sistema es singular o $(-1)^k$ (numero de intercambios efectuados).
N	E	Entero	Orden de la matriz A.
I, J, K, M	L	Entero	Variables usadas como iteradores.
T	L	Real	Variable de localizacion temporal.

Pseudocodigo:

```

BEGIN
PIV[N] <--- 1.
FOR K := 1 TO N DO
  IF K <> N DO
    Busca el elemento menor de la columna
    K, en la k-esima iteracion, su posicion
    la deja en M.
    PIV[K] <--- M.
    IF M <> K THEN
      Habra intercambio de renglones, por tanto
      PIV[N] <--- -PIV[N].
      Se intercambia el menor elemento de la
      columna K a la posicion (K,K), nuevo pivote.
      IF el pivote diferente de cero THEN
        Divide la K-esima columna, a partir del k+1-esimo
        renglon entre  $(-1)^k$  pivote, obteniendo los
        multiplicadores.
        FOR J := K+1 TO N DO
          Se hace la permutacion necesaria para
          la columna J-esima.
          IF A[K,J] <> 0 THEN
            FOR I := K+1 TO N DO
              A[I,J] <--- A[I,J] + multiplicador(I,K)*A[K,J].
            ELSE
              PIV[N] <--- 0, A es singular.
          IF K = N THEN
            IF A[K,K] = 0 THEN PIV[N] <--- 0.
END.

```

5) RESUELVE (A,PIV,N,B)

Objetivo: Usar los resultados de triangulariza para obtener la solucion del sistema $Ax=b$.

VARIABLE	USO	TIPO	FUNCION
A	E	Arreglo de reales	Matriz conteniendo la informacion de la eliminacion Gaussiana y los multiplicadores.
PIV	E	Arreglo de reales	Vector que contiene la informacion del pivoteo.
N	E	Entero	Dimension del sistema a resolverse.
B	E/S	Arreglo	Vector de terminos independientes

de reales del sistema, finalmente se transforma en la solución del mismo.

KB, I, K L Entero Variables usadas como iteradores.
M, T L Real Variables de localización temporal.

Pseudocódigo:

```

BEGIN
IF N <> 1 THEN
  FOR K := 1 TO N-1 DO
    Hace la K-esima permutacion necesaria
    del vector B.
    FOR I := K+1 TO N DO
      BC[I] <--- BC[I] + multiplicador(I,K)*BK[K].
  FOR KB := N-1 DOWNT0 1 DO
    K <--- KB + 1.
    Obtiene la incognita K-esima en BK[K].
    Sustituye el valor de BK[K] en las primeras
    ecuaciones, es decir, de la primera a la K-1=KB.
    BC[1] <--- BC[1] / AC[1,1].
ELSE
  BC[1] <--- BC[1] / AC[1,1].
END.

```

3.4.3) LISTADO DEL PROGRAMA

```
PROGRAM FACTORIZACION (INPUT,OUTPUT);
```

```
CONST
```

```
  N1 = 15;
```

```
TYPE
```

```
  MATRIZ = ARRAY[1..N1,1..N1] OF REAL;
```

```
  MATRIZ1 = ARRAY[1..N1] OF REAL;
```

```
  MATRIZ2 = ARRAY[1..N1] OF INTEGER;
```

```
VAR
```

```
  NC : 1..N1;
```

```
  A : MATRIZ;
```

```
  B : MATRIZ1;
```

```
  PIV : MATRIZ2;
```

```
PROCEDURE TRIANGULARIZA (VAR A : MATRIZ; VAR PIV : MATRIZ2;
                        N : INTEGER);
```

```
VAR
```

```
  I,J,K,M : INTEGER;
```

```
  T : REAL;
```

```
BEGIN
```

```
  PIV[N] := 1;
```

```
  FOR K := 1 TO N DO
```

```
    BEGIN
```

```
      IF K <> N THEN
```

```
        BEGIN
```

```
          M := K;
```

```
          FOR I := K + 1 TO N DO
```

```
            IF ABS(A[I,K]) > ABS(A[M,K]) THEN
```

```
              M := I;
```

```
          PIV[K] := M;
```

```
          IF M <> K THEN
```

```
            PIV[N] := -PIV[N];
```

```
            T := A[M,K];
```

```
            A[M,K] := A[K,K];
```

```
            A[K,K] := T;
```

```
            IF T <> 0.0 THEN
```

```
              BEGIN
```

```
                FOR I := K + 1 TO N DO
```

```
                  A[I,K] := -A[I,K] / T;
```

```
                FOR J := K + 1 TO N DO
```

```
                  BEGIN
```

```
                    T := A[M,J];
```

```
                    A[M,J] := A[K,J];
```

```
                    A[K,J] := T;
```

```
                    IF T <> 0.0 THEN
```

```
                      FOR I := K + 1 TO N DO
```

```
                        A[I,J] := A[I,J] + A[I,K] * T;
```

```
                  END;
```

```
            END
```

```
          ELSE
```

```
            PIV[N] := 0;
```

```
        END;
```

```
      IF K = N THEN
```

```
        IF A[K,K] = 0.0 THEN
```

```
          PIV[N] := 0;
```

```

END(*FOR K *)
END(* DE TRIANGULARIZA*)

PROCEDURE RESUELVE (A : MATRIZ; PIV : MATRIZ2; N : INTEGER;
VAR B : MATRIZ1);
VAR
  KB,I,K,M      : INTEGER;
  T              : REAL;
BEGIN
  IF N <> 1 THEN
    BEGIN
      FOR K := 1 TO N - 1 DO
        BEGIN
          M := PIV(K);
          T := BCM(J);
          BCM(J) := BCK(J);
          BCK(J) := T;
          FOR I := K + 1 TO N DO
            BC(I) := BC(I) + AC(I,K) * T;
          END;
          FOR KB := N - 1 DOWNTO 1 DO
            BEGIN
              K := KB + 1;
              BCK(J) := BCK(J) / ACK(K);
              T := - BCK(J);
              FOR I := 1 TO KB DO
                BC(I) := BC(I) + AC(I,K) * T;
              END;
              BC(I) := BC(I) / AC(I,I);
            END;
          ELSE
            BC(I) := BC(I) / AC(I,I);
          END(* DE RESUELVE *)

```

(PROGRAMA PRINCIPAL)

```

BEGIN
  WRITELN (' DAR EL ORDEN DE LA MATRIZ');
  READ (MC);
  REPEAT
    WRITELN ('DAR COMPONENTES DE LA MATRIZ A');
    LEEMAT (A,MC,MC);
    WRITELN ('DAR COMPONENTES DEL VECTOR B');
    LEEVECT (B,MC);
    TRIANGULARIZA (A,PIV,MC);
    IF PIV(MC) <> 0 THEN
      BEGIN
        RESUELVE (A,PIV,MC,B);
        WRITELN (' LA SOLUCION DEL SISTEMA ES:');
        ESCVECT (B,MC);
      END
    ELSE
      WRITELN (' EL SISTEMA NO TIENE SOLUCION ');
    WRITELN;
    WRITELN ('DAR EL ORDEN DE LA MATRIZ');
    READ (MC);
  UNTIL EOF;
END.

```

3.5) COMPARACION DE METODOS

En este capitulo se han considerado basicamente 2 tecnicas para la solucion de un sistema de ecuaciones lineales simultaneas; estas tecnicas son la iterativa y la de eliminacion. Una pregunta muy natural es: Cual es el mejor?. La respuesta a esta pregunta la resumiremos asi:

a) Los metodos de eliminacion Gaussiana, como el metodo de Gauss-Jordan y el metodo de factorizacion triangular, tienen la ventaja de ser finitos, es decir, requieren un numero finito de operaciones para un sistema dado. Teoricamente la eliminacion Gaussiana trabaja con cualquier conjunto no singular de ecuaciones.

La desventaja de este tipo de metodos es que acumulan errores de redondeo, causando posibles errores en la solucion, aunque esto se minimiza con el pivoteo, es dificil manipular sistemas muy grandes.

De los dos metodos de eliminacion incluidos en este capitulo, el metodo de factorizacion triangular tiene ventajas por la forma tan eficiente de almacenar, en la misma matriz inicial del sistema, la informacion de los multiplicadores y el resultado de la eliminacion (en su parte triangular inferior y superior respectivamente) Esto ultimo permite obtener facilmente el determinante de una matriz, ademas se pueden resolver varios sistemas que tengan la misma matriz de coeficientes.

b) Los metodos iterativos tienen la desventaja de ser lentos y con pocas posibilidades de converger, es decir, convergen bajo

ciertas condiciones como el metodo de Gauss-Seidel. Requieren una gran cantidad de operaciones, sin embargo cuando la iteracion funciona es preferible. El trabajo requerido es proporcional a n^2 por iteracion (n^3 para eliminacion). El error por redondeo es menor, lo que a menudo justifica el esfuerzo adicional de la computadora.

En matrices dispersas o poco densas, es decir, que tienen una alta proporcion de ceros, es posible reducir el numero de operaciones, verificando coeficientes y no efectuando multiplicaciones por cero.

Sistemas demasiado grandes no solo no pueden resolverse con precision por eliminacion sino que ni siquiera caben dentro del almacenamiento de alta velocidad de la computadora. Si los coeficientes son generados por la computadora se evita la dificultad con iteracion, generando cada ecuacion conforme se necesita. Muchos problemas donde aparecen ecuaciones diferenciales parciales son resueltos por metodos iterativos.

IV) SOLUCION DE ECUACIONES.

4.1) INTRODUCCION.

Gran parte de los cursos de matemáticas en las escuelas medias y superiores se dedican al estudio de las ecuaciones, siendo la razón de esto la importancia que tienen para la aplicación práctica de las matemáticas, estando entre sus principales aplicaciones la solución de problemas físicos y geométricos.

Muchas de las ecuaciones utilizadas para representar fenómenos físicos tienen soluciones analíticas, como es el caso de la ecuación cuadrática, la ecuación cúbica y aun la de cuarto grado, sin embargo, muchos problemas de física, ingeniería, economía, etc. pueden conducir a ecuaciones para las cuales no existe fórmula analítica para resolverlas, aquí es conveniente recordar que no existe solución para una ecuación si sus raíces no pueden expresarse en términos de las magnitudes de la misma por medio de operaciones aritméticas o trascendentes. Es precisamente este tipo de ecuaciones las que se resuelven por medio de métodos aproximativos.

Existen varios métodos para resolver ecuaciones mediante aproximaciones, en este capítulo se tratarán dos de ellos: El método de Newton-Raphson y el método de Lin-Bairstow para ecuaciones polinomiales, además los métodos: bisección y secante.

4.2) METODO DE BISECCION Y METODO DE LA SECANTE

4.2.1) INTRODUCCION

El metodo de biseccion tiene la caracteristica de ser seguro pero muy lento. Este metodo tiene por objeto reducir el intervalo inicial $[a,b]$ donde se tiene la seguridad que de que este el cero de la funcion dada, hasta un tamano deseado. Para consesuir esto el metodo se basa en el signo de la funcion en los puntos de evaluacion. El metodo consiste en lo siguiente:

Dada la funcion f definida en el intervalo $[a,b]$ con un unico cero x' , en dicho intervalo, se encuentra un subintervalo que contenga al cero, x' , de la funcion f . Se toma un punto z en $[a,b]$ y se evalua la funcion en z , es decir, $f(z)$, de acuerdo al signo de $f(z)$ obtenemos que el intervalo (a,z) o (z,b) contiene al cero de la funcion. Una forma conveniente de escoger el punto z es que sea el punto medio del intervalo considerado, es decir $z = (a+b)/2$, para asi tener la reduccion maxima del intervalo. En general dados a y b con x' en $[a_k, b_k]$ entonces:

$$1) \text{ Se toma } z_k = \frac{a_k + b_k}{2}$$

$$2) \text{ Si } f(a_k) f(z_k) < 0 \text{ entonces}$$

$$a_{k+1} = a_k$$

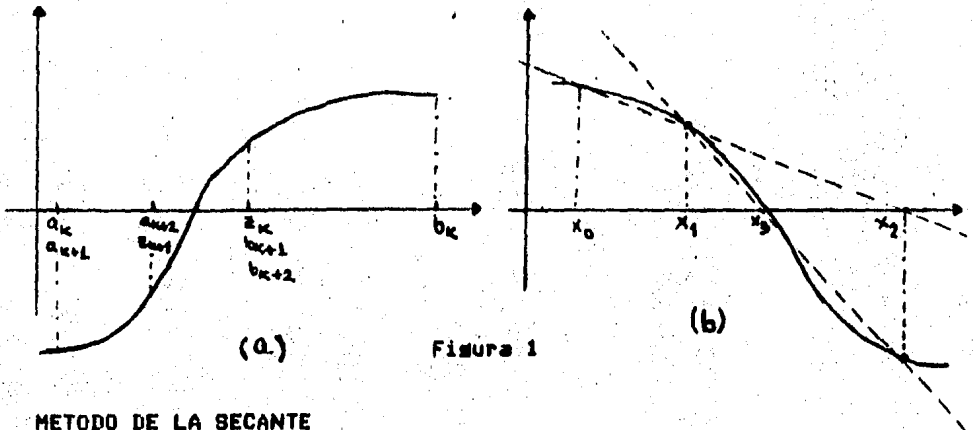
$$b_{k+1} = z_k$$

$$3) \text{ Si } f(a_k) f(z_k) > 0 \text{ entonces}$$

$$a_{k+1} = z_k$$

$$b_{k+1} = b_k$$

En la figura 1 se muestra el proceso.



METODO DE LA SECANTE

El método de la secante tiene la característica de ser un método bastante rápido pero no siempre converge. El método consiste en lo siguiente: Dada la función f con un único cero x' , en el intervalo $[a, b]$ se procede así: dados los puntos x_0, x_1 y x_2 el punto de intersección de la secante que pasa por $(x_0, f(x_0))$ y $(x_1, f(x_1))$ con el eje "X" x_3 es el punto de intersección de la secante que pasa por $(x_1, f(x_1))$ y $(x_2, f(x_2))$ con el eje "X", etc.; en general sea x_{k+1} el punto de intersección de la secante que pasa por $(x_{k-1}, f(x_{k-1}))$ y $(x_k, f(x_k))$ con el eje "X". Ver figura 1 (b).

Hay que notar que para ejecutar una nueva iteración con este método, necesitamos de las 2 últimas aproximaciones obtenidas sin importar el signo de la función en estos puntos. Como consecuencia de lo anterior este método no siempre converge a x' , pero cuando converge su convergencia es muy rápida.

La manera de obtener el punto x_{k+1} es muy sencilla; hay que recordar que la ecuación de la recta, conociendo un punto (x_0, y_0) y su pendiente, es: $y = y_0 + (x - x_0)m$. de esta manera la ecuación de la secante que pasa por los puntos $(x_{k-1}, f(x_{k-1}))$ y $(x_k, f(x_k))$ es:

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

igualando a cero y haciendo $x = x_{k+1}$

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

Debido a la sencillez de la programación de estos dos métodos no se programaron, pero teóricamente son útiles ya que combinando estos dos se obtienen métodos o algoritmos más eficientes conocidos como algoritmos híbridos; en la literatura aparecen el algoritmo de Dekker y el algoritmo de Brent.

4.3) Metodo de Newton-Raphson.

4.3.1) Introduccion

Este metodo proporciona la solucion a una ecuacion de la forma:

$$f(x) = 0$$

La interpretacion geometrica de este metodo se ilustra en la figura 1.

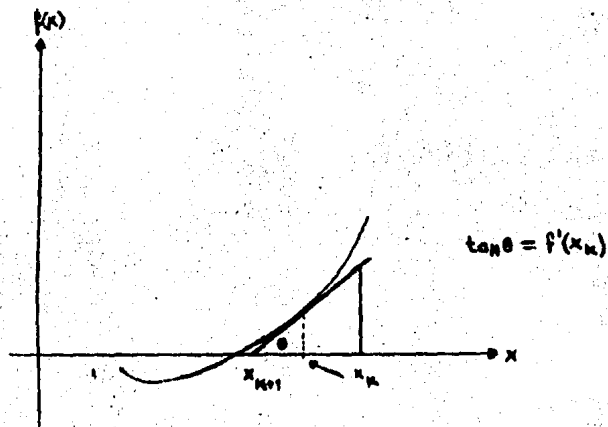


Figura 1

Primero se encuentra la derivada de la función $f(x)$ en el punto $x = x_k$. Siendo esta la tangente a la curva en el punto x_k , podemos encontrar la intersección de esta con el eje x (si la tangente no es horizontal) para encontrar una nueva aproximación de la raíz: x_{k+1} . Dicha aproximación está dada por

$$x_{k+1} = x_k - \frac{\Delta x_k}{k}$$

pero dado que

$$\tan \theta = f'(x_k) = \frac{f(x_k)}{\Delta x_k}$$

es decir

$$\Delta x_k = \frac{f(x_k)}{f'(x_k)}$$

entonces

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

Este método solo nos permite encontrar las raíces reales de la función; para garantizar la convergencia del método requerimos de las siguientes restricciones:

1) Que $f'(x)$ no sea cero o muy pequeña, ya que en ese caso Δx_k tomaría valores muy grandes y puede dar lugar a valores de x_{k+1} fuera de la zona de interés.

2) El valor inicial debe estar cerca de la raíz que se busca; de no ser así, la recta tangente a la curva puede intersectar al eje x cerca de otra raíz que no sea la de interés.

Derivacion de la formula de Newton por Serie de Taylor:

Por la formula de Taylor:

$$f(x) = f(x_{n-1}) + (x - x_{n-1})f'(x_{n-1}) + 1/2(x - x_{n-1})^2 f''(x_{n-1}) + \dots$$

reteniendo la parte lineal y si consideramos que r es una raiz, es decir, $f(r) = 0$ tenemos:

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

definiendo x_n colocandola en lugar de las r restantes:

$$0 = f(x_{n-1}) + (x_n - x_{n-1})f'(x_{n-1})$$

o sea:

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

Ahora consideremos el problema un poco mas general de resolver el sistema:

$$\begin{aligned} f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned} \quad (1)$$

Aplicemos el método de Newton para resolver este sistema (en general la metodología se puede extender para resolver sistemas de mayor orden).

Aproximando f y g por las partes lineales de su serie de Taylor para la vecindad de (x_{n-1}, y_{n-1}) tenemos:

$$\begin{aligned} f(x, y) &\approx f(x_{n-1}, y_{n-1}) + (x - x_{n-1}) f_x(x_{n-1}, y_{n-1}) + (y - y_{n-1}) f_y(x_{n-1}, y_{n-1}) \\ g(x, y) &\approx g(x_{n-1}, y_{n-1}) + (x - x_{n-1}) g_x(x_{n-1}, y_{n-1}) + (y - y_{n-1}) g_y(x_{n-1}, y_{n-1}) \end{aligned} \quad (2)$$

(donde f_x, f_y, g_x, g_y son las derivadas parciales).

Suponiendo que existen las derivadas consideradas.

Denotando (x, y) una solución exacta, ambos lados de (2) se anulan. Definidos $x = x_n, y = y_n$ como los números que anulan los miembros del lado derecho tenemos:

$$\begin{aligned} f(x_{n-1}, y_{n-1})h + f_x(x_{n-1}, y_{n-1})k &= -f(x_n, y_{n-1}) \\ g(x_{n-1}, y_{n-1})h + g_x(x_{n-1}, y_{n-1})k &= -g(x_n, y_{n-1}) \end{aligned} \quad (3)$$

donde h, k representan las formulas recursivas:

$$\begin{aligned} x_n &= x_{n-1} + h \\ y_n &= y_{n-1} + k \end{aligned} \quad (4)$$

En cada iteración se debe resolver el sistema (3) cuya solución es (h_n, k_n) y la nueva aproximación del sistema será (x_n, y_n) . El método se detiene cuando tanto h como k son suficientemente pequeños.

4.3.2) PROGRAMA NEWTON.

OBJETIVO:

Encontrar la solución de ecuaciones del tipo $f(x) = 0$.

DESCRIPCION.

Dado que para este programa se requieren en forma explícita tanto la función como su derivada, es conveniente declarar las funciones F y DERIVADA que calculen f y f' respectivamente. Así, si se desea cambiar la función, solo será necesario alterar las líneas del programa fuente en donde se encuentren F y DERIVADA. Así mismo, se considera una variable EPSILON que define el criterio de convergencia.

ALGORITMO EN PSEUDOCODIGO:

PROGRAMA PRINCIPAL (NEWTON)

OBJETIVO: Calcular la raiz de una funcion dada.

VARIABLES	USO	TIPO	FUNCION
CONT	S	Entero	Contador de iteraciones
XNUEVA	-	Real	Nueva aproximacion de la raiz
NUMITER	E/S	Entero	Numero de iteraciones
EPSILON	E/S	Real	Criterio de convergencia
XANTIGUA	E/S	Real	Antigua aproximacion de la raiz

PSEUDOCODIGO:

```

BEGIN
  Lee Num. de iteraciones, epsilon y primera aproximacion.
  imprime los valores leidos.
  IF numero de iteraciones > 0 THEN
    WHILE funcion valuada en aprox. antigua >= EPSILON DO
      Calcule nueva aproximacion de la raiz.
      aproximacion antigua <--- aproximacion nueva.
      incrementa contador de iteraciones.
    Imprime raiz obtenida y funcion valuada en dicha raiz.
    IF contador llevo al numero maximo (NUMITER) THEN
      manda mensaje.
  ELSE
    Manda un mensaje de error.
  END.

```

LISTADO DEL PROGRAMA:

```

PROGRAM NEWTON (INPUT, OUTPUT);
VAR
  NUMITER, CONT          : INTEGER;
  EPSILON, XANTIGUA, XNUEVA : REAL;
  { FUNCION CUYAS RAICES SE DESEAN ENCONTRAR }
  FUNCTION F (VAR X : REAL) : REAL;
  BEGIN
    F := COS(X)-2 * X * X * X;
  END;
  { DERIVADA DE LA FUNCION F }
  FUNCTION DERIVADAF (VAR X : REAL) : REAL;
  BEGIN
    DERIVADAF := - SIN(X) - 6 * X * X;
  END;
  { PROGRAMA PRINCIPAL }
  BEGIN
    XNUEVA := 0;
    CONT := 0;
    WRITELN ('NUM. DE ITERACIONES?, EPSILON?, X INICIAL?');
    READLN;
    READ (NUMITER, EPSILON, XANTIGUA);
    WRITELN;WRITELN;
    WRITELN (' :10, 'APROXIMACION INICIAL DE LA RAIZ = ', XANTIGUA:4);
    WRITELN (' :10, 'NUM. MAXIMO DE ITERACIONES = ', NUMITER);
    WRITELN (' :10, 'EPSILON = ', EPSILON);
    IF (NUMITER > 0) THEN
      BEGIN
        WHILE ((ABS (F (XANTIGUA))) >= EPSILON)
          AND (CONT < NUMITER) DO
          BEGIN
            { CALCULA NUEVA APROX. DE LA RAIZ }
            XNUEVA := XANTIGUA-F(XANTIGUA)/DERIVADAF(XANTIGUA);
            XANTIGUA := XNUEVA;
            CONT := CONT + 1;
          END;
          WRITELN;WRITELN;
          WRITELN (' :10, 'X = ', XANTIGUA);
          WRITELN (' :10, 'F (X) = ', F (XANTIGUA));
          WRITELN (' :10, 'NUM. DE ITERACIONES EFECTUADAS = ', CONT:4);
          IF NUMITER = CONT THEN
            BEGIN
              WRITELN;WRITELN;
              WRITELN(' :10, '*** NO SE ALCANZO LA PRECISION DESEADA ***');
              WRITELN(' :19, '*** CON EL NUM. DE ITERACIONES INDICADO ***');
            END
          END
        ELSE
          WRITELN (' ***** DATOS ERRONEOS *****');
        END.

```

EJEMPLO:

Supongamos que se quiere conocer la raíz de la ecuación:

$$f(x) = \cos(x) - x^3$$

o sea la función mostrada en el listado del programa. La exactitud requerida es de 0.0000001 y se propone un máximo de 50 iteraciones.

Para dar una primera aproximación de la raíz podemos observar que estando $\cos(x)$ entre -1 y 1 , el término cúbico debe estar también en ese orden para hacer cero la función. De esta forma se propone el valor 1 para la aproximación inicial y nuestros datos de entrada quedarían de la siguiente forma:

NUMITER = 50

EPSILON = 0.0000001

XANTIGUA = 1

por lo tanto la ejecución de este programa quedaría de la siguiente manera:

APROXIMACION INICIAL DE LA RAIZ = 1
 NUM. MAXIMO DE ITERACIONES = 50
 EPSILON = 1.0E-7

X = 0.72140603364
 F (X) = -3.67799657399E-9
 NUM. DE ITERACIONES EFECTUADAS = 4

4.4) METODO DE LIN-BAIRSTOW PARA POLINOMIOS

4.4.1) INTRODUCCION

Un polinomio es una funcion construida como combinacion lineal de potencias de x (suma de potencias de x multiplicadas por un escalar). Por ello, un polinomio $A(x)$ se escribe como

$$A(x) = a_N x^N + a_{N-1} x^{N-1} + \dots + a_1 x + a_0 \quad (1)$$

En este ejemplo N es el mayor exponente y denota el grado del polinomio. A los terminos a se les conoce como coeficientes del polinomio.

El metodo de Lin-Bairstow para encontrar las raices de un polinomio se basa en la posibilidad de expresarlo, segun un teorema del algebra, como producto de factores quadraticos (polinomios de grado dos), o sea, en la forma

$$A(x) = (x^2 + p_1 x + a_1) \dots (x^2 + p_{k-1} x + a_{k-1}) (x^2 + p_k x + a_k) (p x + a)$$

Si queremos conocer los numeros en donde el polinomio vale cero ($A(x)=0$) basta con encontrar las raices (reales o imaginarias) de cada factor cuadratico (resolviendo $(x^2 + p x + a = 0)$).

Si dividimos un polinomio entre otro cuadratico se obtiene como residuo un factor lineal (polinomio de grado uno). En otras palabras, si $A(x)$ es un polinomio cualquiera y $x^2 + p x + a$ un po-

polinomio cuadrático, al dividir el primero sobre el segundo se obtiene como cociente un polinomio $B(x)$ y un residuo $Rx + S$. En símbolos

$$A(x) = B(x)(x^2 + px + a) + Rx + S \quad (2)$$

Cuando R y S se hacen 0, entonces $x^2 + px + a$ es parte de los factores buscados. Por tanto, es necesario buscar los valores p y a donde R y S sean cero. Si en la ecuación (2) multiplicamos del lado derecho el polinomio obtenido por $B(x)$ e igualamos con los coeficientes de $A(x)$ se puede obtener una expresión de las funciones $R(p, a)$ y $S(p, a)$ que no se dan en forma explícita.

Existiendo estas funciones, se puede usar una extensión del método de Newton-Raphson para las funciones de dos variables $R(p, a)$ y $S(p, a)$. Desarrollando una serie de Taylor para las funciones R y S , tomando la parte lineal (i. e. eliminando términos de mayor orden) e igualándolas a cero se obtienen las ecuaciones

$$0 = R = R(p, a) + \frac{\partial R}{\partial p} \Delta p + \frac{\partial R}{\partial a} \Delta a$$

y

$$0 = S = S(p, a) + \frac{\partial S}{\partial p} \Delta p + \frac{\partial S}{\partial a} \Delta a$$

Conociendo $R(p, a)$, $S(p, a)$ y los valores de las derivadas parciales

$$\frac{\partial R}{\partial p}, \frac{\partial S}{\partial p}, \frac{\partial R}{\partial a} \text{ y } \frac{\partial S}{\partial a}$$

y eliminando los terminos de mayor orden se obtiene un sistema de dos ecuaciones lineales con las incognitas Δp y Δa .

Los valores R y S se encuentran con un algoritmo sencillo basado en la division de polinomios. Si escribimos el polinomio $B(x)$ resultante de la division entre $x^2 + px + a$ como

$$B(x) = b_{N-2} x^{N-2} + b_{N-3} x^{N-3} + \dots + b_1 x + b_0$$

cada b se obtiene de la formula

$$b_{N-2} = b_{N-2}(-a) + b_{N-1}(-p) + a$$

Esto sera valido para todas las b 's si definimos

$$b_N = b_{N-1} = 0$$

Ademas, los valores de R y S corresponden a los de

b_{-1} y b_{-2} , respectivamente, siguiendo el mismo algoritmo.

Las derivadas parciales $\partial R/\partial a$ y $\partial S/\partial a$ se obtienen de dividir $B(x)$ sobre $x^2 + px + a$ dado que si derivamos el polinomio original $A(x)$ con respecto a a se obtiene, de la ecuacion (2):

$$0 = \frac{\partial A(x)}{\partial a} = B(x) + \frac{\partial B(x)}{\partial a} (x^2 + px + a) + \frac{\partial R}{\partial a} + \frac{\partial S}{\partial a}$$

de donde

$$B(x) = - \frac{\partial B(x)}{\partial a} (x^2 + px + a) - \frac{\partial R}{\partial a} - \frac{\partial S}{\partial a}$$

Esto no es mas que una division de $B(x)$ sobre $x^2 + px + a$ con $\partial B(x) / \partial a$ como cociente y $-\partial R / \partial a$ y $-\partial S / \partial a$ como residuos por lo que es suficiente aplicar el mismo algoritmo a $B(x)$ y $x^2 + px + a$. $\partial R / \partial p$ y $\partial S / \partial a$ se pueden obtener de la misma forma, dividiendo $x B(x)$ sobre $x^2 + px + a$ dado que si derivamos $A(x)$ con respecto a p tenemos

$$0 = \frac{\partial A(x)}{\partial p} = x B(x) + \frac{\partial B(x)}{\partial p} (x^2 + px + a) + \frac{\partial R}{\partial p} + \frac{\partial S}{\partial p}$$

de donde

$$x B(x) = - \frac{\partial x B(x)}{\partial p} (x^2 + px + a) - \frac{\partial R}{\partial p} - \frac{\partial S}{\partial p}$$

De esta forma se completa el sistema de ecuaciones y se obtienen Δp y Δa . Estos valores son sumados a los p y a iniciales y el proceso se reinicia hasta que Δp y Δa sean tan pequenos como el criterio de convergencia lo requiera.

Una vez encontrados se toma el polinomio $B(x)$ y se procede de la misma forma hasta llegar a un polinomio cuadratico o a uno lineal. El metodo puede ser resumido en los siguientes pasos:

1) Division de $A(x)$ entre $x^2 + px + a$ para producir R , S y $B(x)$.

2) Division de $B(x)$ entre $x^2 + px + a$ para producir $\partial R / \partial a$ y $\partial S / \partial a$.

3) Division de $x B(x)$ entre $x^2 + px + a$ para producir $\partial R / \partial p$ y $\partial S / \partial a$.

- 4) Resolución de sistema de ecuaciones para obtener Δp y Δa .
- 5) Sumas $p' = p + \Delta p$ y $a' = a + \Delta a$.
- 6) Repetir el procedimiento 1-5 con los nuevos valores p' y a' hasta que Δp y Δa sean lo suficientemente pequeños.
- 7) Alcanzado el criterio de convergencia resolver polinomio cuadrático, encontrar raíces y repetir el procedimiento 1-6 tomando el polinomio reducido $B(x)$ como $A(x)$.

4.4.2) PROGRAMA RAICES

OBJETIVO:

Obtener las raíces de polinomios dados.

DESCRIPCION:

El programa tiene como entrada el grado N del polinomio y de aquí los N coeficientes que lo forman. En la salida se escribe el polinomio con sus coeficientes en orden decreciente de grado y a continuación los vectores de las raíces con su parte real y su parte imaginaria.

El programa está estructurado de tal manera que llama a una subrutina para encontrar las raíces del polinomio dado. En ella se aproximan los pares de valores p y a (a partir de valores iniciales dados en el programa) hasta que el grado se reduce a dos o a uno. Esta subrutina a su vez utiliza a otras dos. La primera encuentra las raíces de la ecuación $x^2 + px + a$ con los valores

datos y la segunda hace la division entre un polinomio y el termino cuadratico $x^2 + px + a$.

Tanto el criterio de convergencia como el numero maximo de iteraciones son dadas en el programa a traves de constantes. Esto ultimo se requiere dado que hay casos excepcionales en que el metodo no converge. Para esto el programa provee de cuatro pares de valores alternativos y en ultimo caso, da posibilidad de que el usuario de por si mismo los valores.

ALGORITMO EN PSEUDOCODIGO:

PROGRAMA RAICES

VARIABLES	USO	TIPO	FUNCION
LIMITE	-	Constante	Tamano del arreslo para coeficientes
ITERMAX	-	Constante	Numero maximo de iteraciones
AA	E	Arreslo	Guarda coeficientes del polinomio de Reales
SOLUCION	S	Arreslo	Guarda soluciones reales y complejas de Reales
M	E	Entero	Grado del polinomio
I	L	Entero	Contador

Pseudocodiso:

```

BEGIN
  Lee M
  WHILE NOT eof DO
    FOR I := 1 TO M DO
      Lee coeficiente del polinomio
      CALL RAICES (AA, M, SOLUCION)
      Escribe polinomio en orden decreciente (de grado)
    FOR I := 1 TO N DO
      Escribe raiz con parte real y parte compleja
    Lee M
  END

```

SUBROUTINAS

1) RAICES (A, N, V)

Objetivo: Dado un polinomio, encuentra sus raices reales y complejas.

VARIABLES	USO	TIPO	FUNCION
-----------	-----	------	---------

A	E	Arreglo de Reales	Coefficientes del Polinomio
N	E	Entero	Grado del polinomio
V	S	Arreglo de Reales	N vectores de raices con parte real y parte imaginaria
EPS	L	Constante	Criterio de convergencia para DELTAP y DELTAQ
ITERMAX	L	Constante	Numero maximo de iteraciones
PQCERO(1..4)	L	Constantes	Valores alternativos para PCERO y QCERO
J	L	Entero	Contador de coeficientes 0
N1	L	Entero	Guarda copia de N
T	L	Entero	Contador
B	L	Arreglo de Reales	Guarda coeficientes de B(x), resultados de division A(x)
BDES	L	Arreglo de Reales	Resultados de division B(x) y de xB(x)
BANDERA	L	Entero	Marca que opciones de PCERO y QCERO se utilizaron
DISCRIMIN	L	Real	Discriminante del sistema de ecuaciones
ITER	L	Entero	Marcador de numero de iteraciones
PP, QQ	L	Real	Guardan aproximaciones de p y q en $x^2 + px + q$
PCERO, QCERO	L	Reales	Valores iniciales de p y q
DELTAP	L	Real	Valor a aumentar a PP (Δp)
DELTAQ	L	Real	Valor a aumentar a QQ (Δq)
DRDP, DRDQ	L	Reales	Derivadas parciales de R sobre p y sobre q
DSDP, DSDQ	L	Reales	Derivadas parciales de S sobre p y sobre q

Pseudocodiso:

```

BEGIN
  Inicializa J
  N1 <--- N
  WHILE coeficientes de A(x) = 0 AND J <> N1 DO
    Raices reales y complejas <--- 0
    N <--- N - 1
    J <--- J + 1
  Recoloca los coeficientes de A(x)
  WHILE N > 2 DO
    PP <--- PQCERO1
    QQ <--- PQCERO2
    Inicializa DELTAP, DELTAQ e ITER
    BANDERA <--- 1
    REPEAT
      PP <--- PP + DELTAP
      QQ <--- QQ + DELTAQ (aproxima PP y QQ)
    IF numero maximo de iteraciones se alcanza THEN
      CASE BANDERA OF (marca que opciones de PCERO y QCERO
        ya se utilizaron)
        1: Inicializar PP y QQ con PQCERO1 y PQCERO2 respect.
        2: Inicializar PP y QQ con PQCERO2 y PQCERO1 respect.
        3: Inicializar PP y QQ con PQCERO3 ambas

```

```

4: Inicializar PP y QQ con PQCERO4 ambas
5: Leer nuevos valores iniciales para PP y QQ
CALL DIVISION (A, B, N, PP, QQ)
R <--- B[-1] (Toma termino lineal del residuo de la division)
S <--- B[-2] (Toma termino independiente del residuo)
CALL DIVISION (B, BDES, N - 2, PP, QQ)
DRDQ <--- -BDES[-1]
DSDQ <--- -BDES[-2] (Toma residuo de division de B(x))
FOR J := N - 2 DOWNTO 0 DO
    BCJ + 1] <--- BCJ]
BCOJ := 0 (Mueve los coeficientes de B(x) para hacer
    la division con xB(x))
CALL DIVISION (B, BDES, N - 1, PP, QQ)
DRDQ <--- -BDES[-1]
DSDQ <--- -BDES[-2] (Toma residuo de division de xB(x))
Resuelve sistema de dos ecuaciones al calcular el
    discriminante DISCRIMIN y de aqui DELTAP y DELTAQ
ITER <--- ITER + 1 (Actualiza contador de iteraciones)
UNTIL !DELTAP! < EPS AND !DELTAQ! < EPS (se cumpla criterio de
    de convergencia)
CALL CUADRATICO (1, PP, QQ, REN,0], REN-1,0], REN,1], REN-1,1])
(Manda encontrar raices de termino cuadratico)
FOR J := N - 1 DOWNTO 1 DO
    ACJ - 1] := BCJ] (Convierte a B(x) en un nuevo A(x))
    N <--- N - 2 (Baja el grado del polinomio)
IF N = 2 (Polinomio restante cuadratico) THEN
    CALL CUADRATICO (AC2],AC1],AC0],RC2,0],RC1,0],RC2,1],RC1,1])
ELSE
    IF N = 1 (Polinomio restante lineal) THEN
        Resuelve ecuacion lineal
END

```

1.1) CUADRATICO (AAA, BBB, CCC, RAIZ1R, RAIZ2R, RAIZ1I, RAIZ2I)

Objetivo: Encontrar las raices de una ecuacion cuadratica.

VARIABLES	USO	TIPO	FUNCION
AAA, BBB, CCC	E	Reales	Valores a,b,c de la ecuacion $ax^2 + bx + c = 0$
RAIZ1R, RAIZ2R	S	Reales	Partes reales de las raices
RAIZ1I, RAIZ2I	S	Reales	Partes inasimarias de las raices
DISCRIM	L	Real	Discriminante de la ecuacion

Pseudocodiso:

```

BEGIN
DISCRIM <--- (BBB BBB) - 4 AAA CCC
IF DISCRIM >= 0 THEN
    Parte inasimaria es 0
IF DISCRIM <= 0 THEN
    Parte real es - BBB / (2 AAA)
IF DISCRIM > 0 THEN

```

```

Parte real es (- BBB +/- DISCRIM ) / (2 * AAA)
IF DISCRIM < 0 THEN
    Parte imaginaria es +/- (DISCRIM ) / (2 * AAA)
    1/2
END

```

1.2) DIVISION (AP, BP, L, P, Q)

Objetivo: Dividir un polinomio entre otro cuadrático.

VARIABLES	USO	TIPO	FUNCION
AP	E	Arreglo de Reales	Polinomio a ser dividido
BP	S	Arreglo de Reales	Polinomio-cociente que incluye residuo
L	E	Entero	Grado del polinomio (a ser dividido)
P, Q	E		Valores a y b del polinomio $x^2 + ax + b$ a dividir
K	L	Entero	Contador

Pseudocódigo:

```

BEGIN
P <--- -P
Q <--- -Q (Cambio de signo a P y Q para mayor eficiencia)
BP[L] <--- 0
BP[L - 1] <--- 0
BP[-1] <--- 0 (Define valores para que cumplan el algoritmo)
FOR K := L - 2 DO (Algoritmo de division) UNTO -2 DO
    BPE[K] <--- BP[K + 2]*Q + BP[K+1]*P + AP[K + 2]
END

```

4.4.3) LISTADO DEL PROGRAMA

```
PROGRAM POLINOMIO (INPUT, OUTPUT);
CONST
```

```
  LIMITE = 100;
```

```
TYPE
```

```
  COEF = ARRAY[-2..LIMITE] OF REAL;
```

```
  RAIZ = ARRAY[1..LIMITE, 0..1] OF REAL;
```

```
VAR
```

```
  AA      : COEF;
```

```
  SOLUCION : RAIZ;
```

```
  I, M    : INTEGER;
```

```
PROCEDURE RAICES (A : COEF) N : INTEGER; VAR V : RAIZ);
```

```
CONST
```

```
  EPS      = 0.0000000001;
```

```
  ITERMAX  = 3000;
```

```
  PQCERO1  = -2;
```

```
  PQCERO2  = 2;
```

```
  PQCERO3  = 5;
```

```
  PQCERO4  = -5;
```

```
VAR
```

```
  J, N1, T      : INTEGER;
```

```
  B, BDES       : COEF;
```

```
  BANDERA       : INTEGER;
```

```
  DISCRIMIN     : REAL;
```

```
  ITER          : REAL;
```

```
  PCERO, QCERO  : REAL;
```

```
  PP, QQ        : REAL;
```

```
  DELTAP, DELTAQ : REAL;
```

```
  R, S          : REAL;
```

```
  DRDP, DRDQ    : REAL;
```

```
  DSDP, DSDQ    : REAL;
```

```
(ENCUENTRA RAICES DE UN POLINOMIO DE GRADO DOS)
```

```
PROCEDURE CUADRATICO (AAA, BBB, CCC : REAL; VAR
```

```
  RAIZ1R, RAIZ2R, RAIZ1I, RAIZ2I : REAL);
```

```
VAR
```

```
  DISCRIM : REAL;
```

```
BEGIN
```

```
  DISCRIM := (BBB * BBB) - (4 * AAA * CCC);
```

```
  IF DISCRIM >= 0 THEN
```

```
    BEGIN
```

```
      RAIZ1I := 0;
```

```
      RAIZ2I := 0;
```

```
    END (IF);
```

```
  IF DISCRIM <= 0 THEN
```

```
    BEGIN
```

```
      RAIZ1R := -BBB / (2 * AAA);
```

```
      RAIZ2R := RAIZ1R;
```

```
    END (IF);
```

```
  IF DISCRIM > 0 THEN
```

```
    BEGIN
```

```
      RAIZ1R := ((-BBB) + SQRT (DISCRIM)) / (2 * AAA);
```

```
      RAIZ2R := ((-BBB) - SQRT (DISCRIM)) / (2 * AAA);
```



```

END (IF);
IF DISCRIM < 0 THEN
BEGIN
  RAIZ1I := (SQRT (-DISCRIM)) / 2 * AAA;
  RAIZ2I := - RAIZ1I;
END (IF);
END (PROCEDURE);

```

```

{DIVIDE UN POLINOMIO ENTRE OTRO CUADRATICO}
PROCEDURE DIVISION (AP : COEF; VAR BP : COEF;
  L : INTEGER) P, Q : REAL);

```

```

VAR
  K : INTEGER;
BEGIN
  P := -P;
  Q := -Q;
  BPELJ := 0;
  BPEL - 1J := 0;
  BPC-1J := 0;
  FOR K := L - 2 DOWNT0 -2 DO
    BPEKJ := BPEK + 2J * Q + BPEK + 1J * P + APCK + 2J;
  END (PROCEDURE);

```

```

BEGIN
  J := 0;
  N1 := N;
  WHILE (ACJ = 0) AND (J < N1) DO
    BEGIN
      VCN, 0J := 0;
      VCN, 1J := 0;
      N := N - 1;
      J := J + 1;
    END (WHILE);

```

```

FOR T := 0 TO N1 - J DO
  ACTJ := ACJ + TJ;

```

```

WHILE N > 2 DO

```

```

  BEGIN
    PP := PQCERO1;
    QQ := PQCERO2;
    DELTAP := 0;
    DELTAQ := 0;
    ITER := 0;
    BANDERA := 1;
    REPEAT

```

```

      BEGIN
        PP := PP + DELTAP;
        QQ := QQ + DELTAQ;
        IF ITER > ITERMAX THEN
          BEGIN
            ITER := 0;
            CASE BANDERA OF
              1: BEGIN
                  PP := PQCERO1;
                  QQ := PQCERO2;
                  BANDERA := 2;
                END (1);
              2: BEGIN

```

```

PP := PQCERO2;
QQ := PQCERO1;
BANDERA := 3;
END (2);
3: BEGIN
PP := PQCERO3;
QQ := PQCERO3;
BANDERA := 4;
END (3);
4: BEGIN
PP := PQCERO4;
QQ := PQCERO4;
BANDERA := 5;
END (4);
5: BEGIN
WRITE ('NUMERO MAXIMO DE ITERACIONES');
WRITE ('. PROPONGA DOS VALORES ');
WRITELN ('PARA PP Y QQ');
READLN (PP, QQ);
END (5);
END (CASE);
END (IF);
DIVISION (A, B, N, PP, QQ);
R := BC-1;
S := BC-2;
DIVISION (B, BDES, N - 2, PP, QQ);
DRDQ := -BDESC-1;
DSDQ := -BDESC-2;
FOR J := N - 2 DOWNTO 0 DO
    BCJ + 1 := BCJJ;
    BC0 := 0;
    DIVISION (B, BDES, N - 1, PP, QQ);
    DRDP := -BDESC-1;
    DSDP := -BDESC-2;
    DISCRIMIN := DRDP * DSDQ - DRDQ * DSDP;
    DELTAP := (S * DRDQ - R * DSDQ) / DISCRIMIN;
    DELTAQ := (R * DSDP - S * DRDP) / DISCRIMIN;
    ITER := ITER + 1;
END (REPEAT);
UNTIL (ABS(DELTAP) < EPS) AND (ABS(DELTAQ) < EPS);
CUADRATICO (1, PP, QQ, VC1,0, VC1,0, VC1,1, VC1,1);
FOR J := N - 1 DOWNTO 1 DO
    ACJ - 1 := BCJJ;
    N := N - 2;
END (WHILE);
IF N = 2 THEN
    CUADRATICO (AC2, AC1, AC0, VC2,0, VC1,0, VC2,1, VC1,1);
ELSE
    IF N = 1 THEN
        BEGIN
            VC1,0 := -AC0/AC1;
            VC1,1 := 0;
        END (IF);
    END (PROCEDURE);

```

(PROGRAMA PRINCIPAL)

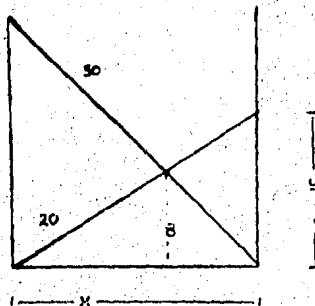
```

BEGIN
WRITELN ('GRADO DEL POLINOMIO?');
READ (M);
WHILE NOT EOF DO
  BEGIN
    WRITE ('COMPONENTES EN ORDEN DECRECIENTE? ');
    WRITELN (('INCLUYENDO CEROS?'));
    FOR I := M DOWNTD 0 DO
      READ (AACIJ);
    RAICES (AA, M, SOLUCION);
    WRITELN ('EL POLINOMIO:');
    WRITELN ('GRADO          COEFICIENTE');
    FOR I := M DOWNTD 0 DO
      WRITELN (I, AACIJ);
    WRITELN ('TIENE LAS RAICES:');
    WRITELN ('PARTE REAL          PARTE IMAGINARIA');
    FOR I := 1 TO M DO
      WRITELN (SOLUCIONCI,OJ, SOLUCIONCI,IJ);
    WRITELN ('GRADO DEL POLINOMIO?');
    READ (M);
  END (WHILE);
END.

```

4.4.4) EJEMPLO

Dos escaleras, una de 20 y otra de 30 metros estan apoyadas contra la pared de un callejon segun se muestra en el dibujo. Si el punto en que se cruzan las escaleras esta situado a 8 metros cual es el ancho del callejon?



$$x^2 = 400 - y^2$$

Las consideraciones anteriores nos llevan a la ecuacion

$$w^4 - 16w^3 + 500w^2 - 8000w + 32000 = 0$$

CORRIDA

GRADO DEL POLINOMIO?

4

COMPONENTES EN ORDEN DECRECIENTE (INCLUYENDO CEROS)

1 -16 500 -8000 32000

EL POLINOMIO:

GRADO	COEFICIENTE
4	1
3	-16
2	500
1	-8000
0	32000

TIENE LAS RAICES:

PARTE REAL	PARTE IMAGINARIA
-0.82821145118	-21.4228501959
-0.82821145118	21.4228501959
5.9445923916	0
11.7118305108	0

El problema tiene pues, dos soluciones reales posibles que son, aproximando 5.9446 y 11.7118.

4.5) COMENTARIOS DE LOS METODOS

A lo largo de este capítulo se han visto varias técnicas para resolver el problema $f(x)=0$, la pregunta natural es cuál es mejor? Para responder a esta pregunta diremos que, en general, existen dos técnicas para resolver el problema; dichas técnicas son: los métodos seguros y los métodos rápidos. Dentro de los métodos seguros están: el método de bisección y el método de la regla falsa (basicamente consisten en ir reduciendo el intervalo donde se encuentra la raíz). Dentro de los métodos rápidos están: el método de Newton y el método de la secante (consisten en hacer aproximaciones a la raíz por medio interpolaciones de polinomios; en este caso lineales).

Los métodos más eficientes tratan de ser seguros y rápidos; esto se logrará combinando adecuadamente dichas técnicas. Por esta razón los métodos que, en la práctica, tienen más utilidad son los métodos híbridos, que son el resultado de combinar dos o más técnicas.

El método de Newton y el método de la secante pueden usarse indistintamente, en el caso raro de saber que hay convergencia. El método de la secante tiene cierta ventaja con respecto al de Newton ya que no se requiere determinar la derivada; los dos métodos, si convergen, tienen una aceptable rapidez de convergencia.

V) VECTORES Y VALORES CARACTERISTICOS

5.1) INTRODUCCION

El analisis de algunos sistemas fisicos dan lugar a un conjunto de ecuaciones lineales algebraicas, las cuales pueden ser resueltas unicamente cuando el valor de un parametro dentro de las ecuaciones es conocido. Este parametro se le llama valor caracteristico, y la solucion asociada con cada valor caracteristico es llamado su vector caracteristico asociado.

Estos vectores caracteristicos describen configuraciones criticas o modos del sistema. Problemas relacionados con valores caracteristicos ocurren en el analisis de masas vibratorias, circuitos electricos, resistencia de materiales, etc.

La forma general del problema de valores caracteristicos en notacion matricial es:

$$AX = \lambda BX \quad X \neq 0 \quad (1)$$

Aqui A y B son matrices cuadradas de orden N . Es necesario determinar alguno o todos los n escalares, λ , los cuales satisfacen esta relacion, y los vectores caracteristicos, X , asociados a cada valor caracteristico. Si consideramos que nuestros sistemas involucran matrices simetricas y no singulares podemos hacer lo siguiente: premultiplicamos ambos lados de la ecuacion (1) por la inversa de B , es decir:

$$B^{-1}AX = \lambda B^{-1}BX \quad \text{o} \quad HX = \lambda X \quad (2)$$

$$\text{Donde } H = B^{-1}A$$

En la solucion formal del problema, la ecuacion (1) es es-

crita en la forma

$$(A - \lambda B)X = 0 \quad (3)$$

Para que X exista el determinante de $(A - \lambda B)$ deberá ser cero, dando una ecuación polinomial de grado N en λ llamada ecuación característica. Cada una de las raíces del polinomio característico $|A - \lambda B| = 0$, es un valor característico; sustituyendo cada raíz en la ecuación (3) y resolviendo este sistema, encontramos el correspondiente vector característico. En este trabajo nos limitaremos al caso en que $B = I$ en la ecuación (1).

5.2) METODO DE JACOBI

5.2.1) INTRODUCCION

El metodo de Jacobi es un proceso numerico de tipo iterativo que permite obtener el mayor o el menor valor caracteristico de una matriz dada, con su correspondiente vector caracteristico.

Por definicion se tiene que un valor caracteristico cumple con la relacion:

$$Ax = \lambda x; \quad x \text{ diferente de cero} \quad (1)$$

El metodo consiste en dar un vector inicial X_0 , efectuando el producto AX_0 obtenemos el vector C_1 , el cual se factoriza de la siguiente forma: $C_1 = \lambda_1 X_1$, donde λ_1 es la coordenada mayor de C_1 , por ejemplo si $C_1 = (3,5,6)$, factorizamos C_1 asi $C_1 = 6(3/6, 5/6, 1)$. λ_1 corresponde a la primera aproximacion del valor caracteristico, el vector X_1 es la nueva aproximacion del vector caracteristico. El procedimiento general es el siguiente:

$$\begin{array}{r} AX = C = \lambda X \\ \quad \quad \quad 1 \quad 1 \quad 1 \\ \\ AX = C = \lambda X \\ \quad \quad \quad 1 \quad 2 \quad 2 \quad 2 \\ \\ AX = C = \lambda X \\ \quad \quad \quad 2 \quad 3 \quad 3 \quad 3 \\ \\ \cdot \quad \cdot \quad \cdot \\ \\ AX = C = \lambda X \\ \quad \quad \quad n \quad n+1 \quad n+1 \quad n+1 \end{array} \quad (2)$$

El metodo se detiene cuando:

$$| \lambda_{n+1} - \lambda_n | < \text{epsilon}$$

Se demuestra que el método siempre converge al vector característico correspondiente al mayor valor característico. En algunos problemas, el mayor valor característico es más importante, mientras que en otros (problemas de vibración) el menor valor característico es el más interesante; podemos obtener otros valores característicos usando este método.

Se puede obtener el menor valor característico usando el método de Jacobi de la siguiente manera:

Si se premultiplica ambos lados de la ecuación (1) por la inversa de A y dividiendo por λ da:

$$\frac{1}{\lambda} A^{-1} A X = A^{-1} X$$

$$\frac{1}{\lambda} X = A^{-1} X \quad (3)$$

$$A^{-1} X = \frac{1}{\lambda} X$$

Esta ecuación tendrá el mayor valor característico ($1/\lambda$), el cual corresponde al menor valor característico de A .

Después de determinar el mayor (o menor) valor característico, es posible obtener valores característicos intermedios, empleando la ortogonalidad de los vectores característicos con respecto a la matriz B de la ecuación (1), sección 5.1; para esto se requiere que tanto A como B sean simétricas.

5.2.2) PROGRAMA JACOBI

OBJETIVO:

Obtener el mayor valor característico para una o varias matrices cuadradas, por el método de Jacobi.

ALGORITMO EN PSEUDOCODIGO

PROGRAMA PRINCIPAL

VARIABLE	USO	TIPO	FUNCION
EPS	-	Constante	Criterio de convergencia.
N1	-	Constante	Orden maximo posible para las matrices usadas en el programa (es modificable).
MC	E	Entero	Orden de la matriz A.
AMAX	-	Real	Variable que tendra el mayor valor característico.
MAXITER	E	Entero	Maximo numero de iteraciones.
A	E	Arreglo	Matriz a la que se busca el mayor valor característico.
B	E/S	Arreglo	Vector inicial para arrancar el metodo; finalmente corresponde al vector característico buscado.
DECIDE	-	Real	Variable de decision para la subrutina JACOBI (decide si el metodo ha convergido).

Pseudocodiso:

BEGIN

Escribe letrero de entrada.

Lee orden de la matriz y max. no. de iteraciones.

REPEAT

Leer la matriz A

Leer el vector característico inicial.

CALL JACOBI (A,B,MC,AMAX,DECIDE,MAXITER,EPS).

IF DECIDE <= EPS THEN

Escribe el mayor valor característico.

Escribe el vector característico.

ELSE

IF AMAX = 0.0 THEN

Escribe letrero de indicacion.

ELSE

Escribe letrero de no convergencia.

Lee orden de la matriz y max. no. de iteraciones.

UNTIL fin de archivo.

END.

SUBROUTINAS

1) LEEHAT (D,M,N)

(Lee una matriz de orden $M \times N$, ver sec 2.2.2).

2) LEEVECT (D,M,N)

(Lee un vector de orden M, ver sec. 3.2.2).

3) ESCVECT (D,M,N)

(Escribe un vector de orden M, ver sec. 3.2.2).

4) JACOBI (A,X,N,AMAX,DECIDE,M,EPS)

Objetivo: Obtener el menor valor característico de la matriz A por el método de Jacobi; este valor se da en AMAX.

VARIABLE	USO	TIPO	FUNCION
A	E	Arreglo	Matriz a la que se busca el menor valor característico.
X	E/S	Arreglo	Vector inicial para arrancar el método; finalmente contiene el vector característico correspondiente al menor val. caract.
N	E	Entero	Orden de la matriz A.
AMAX	S	Real	Variable que contiene el máximo valor característico.
DECIDE	S	Real	Variable usada como criterio de convergencia para saber si el método ha convergido.
M	E	Entero	Máximo número de iteraciones.
EPS	E	Real	Criterio de convergencia.
C	L	Arreglo	Vector que contiene el producto AX en cada iteración.
VALCAR	L	Real	Variable que contiene, en cada iteración, el valor actual del valor característico buscado.

Pseudocódigo:

BEGIN

K ← 1

VALCAR ← 0.0.

REPEAT

 Calcula el producto C ← AX

 AMAX ← máximo valor del vector C.

 IF AMAX < 0.0 THEN

 divide el vector C entre AMAX, y lo deposita en el vector X.

 DECIDE ← valor absoluto de (AMAX-VALCAR).

 IF DECIDE > EPS THEN

 VALCAR ← AMAX

 K ← K + 1

UNTIL (K > M) o (AMAX = 0.0) o (DECIDE ≤ EPS).

END.


```

LEEMAT (A,MC,MC);
WRITELN ('DAR EL VECTOR INICIAL DE ARRANQUE ');
LEEVECT (B,MC);
JACOBI (A,B,MC,AMAX,DECIDE,MAXITER,EPS);
IF DECIDE <= EPS THEN
  BEGIN
    WRITELN('EL MAXIMO VALOR CARACT. ES: ',AMAX);
    WRITELN('SU VECTOR CARACTERISTICO ES:');
    ESCVECT(B,MC);
  END
ELSE
  IF AMAX = 0.0 THEN
    WRITELN ('SELECCIONAR OTRO VECTOR INICIAL (AMAX=0.0)')
  ELSE
    BEGIN
      WRITELN ('NO CONVERGE EN',MAXITER,' ITERACIONES');
      WRITELN ('LA ULTIMA APROX. FUE',AMAX);
    END;
  WRITELN;
  WRITELN('DAR ORDEN DE LA MATRIZ Y MAX. NO. DE ITERACIONES');
  READ (MC,MAXITER);
UNTIL EOF;
END.

```

5.2.4) EJEMPLO

El siguiente sistema aparece en varias situaciones de Física, como en la vibración de un sistema de 3 masas conectadas por resortes.

$$\begin{array}{rcl}
 (2 - \lambda)X_1 - X_2 & & = 0 \\
 -X_1 + (2 - \lambda)X_2 - X_3 & & = 0 \\
 -X_2 + (2 - \lambda)X_3 & & = 0
 \end{array}$$

Hallar su máximo valor característico y su correspondiente vector característico asociado. Al correr el programa, pide los siguientes datos:

```

DAR ORDEN DE LA MATRIZ Y MAX. NO. DE ITERACIONES
      3  15
DAR COMPONENTES DE LA MATRIZ
      2  -1  0
     -1  2  -1
      0  -1  2

```

DAR VECTOR INICIAL PARA FORMAR EL SISTEMA
1 1 1

SU MAXIMO VALOR CARACTERISTICO ES: 3.41421357309

SU VECTOR CARACTERISTICO ASOCIADO ES:
-0.7071 1.0000 -0.7071

5.3) METODO DE KRYLOV

5.3.1) INTRODUCCION

Este metodo obtiene los coeficientes del polinomio caracteristico de la matriz A en forma numerica y posteriormente, dichos valores, se alimentan a la subrutina RAICES, descrita en el capitulo anterior para obtener los valores caracteristicos. Este metodo se basa en el siguiente teorema:

TEOREMA (Cayley-Hamilton).-

$$\text{si } P(\lambda) = 0 \implies P(A) = 0 \quad (1)$$

Donde $P(\lambda)$ es el polinomio caracteristico de la matriz A

a continuacion se describe el metodo:

sea $P(\lambda)$ de la siguiente forma:

$$P(\lambda) = b_n \lambda^n + b_{n-1} \lambda^{n-1} + \dots + b_1 \lambda + b_0 = 0 \quad (2)$$

Normalizando el coeficiente que da el grado del polinomio:

$$PN(\lambda) = \lambda^n + c_{n-1} \lambda^{n-1} + \dots + c_0 = 0 \quad (3)$$

donde:

$$c_{n-1} = \frac{b_{n-1}}{b_n} \quad (4)$$

Aplicando el teorema de Cayley-Hamilton a la ecuacion (3) se

Obtiene:

$$PN(A) = A^n + c_{N-1} A^{n-1} + \dots + c_1 A + c_0 I = 0 \quad (5)$$

se postmultiplica $PN(A)$ por un vector arbitrario $Y \neq 0$

$$PN(A)Y = A^n Y + c_{N-1} A^{n-1} Y + \dots + c_1 AY + c_0 Y = 0 \quad (6)$$

la ecuacion (6) representa un sistema de ecuaciones con incognitas $c_0, c_1, c_2, \dots, c_{n-1}$, las cuales representan

coeficientes de la ecuacion caracteristica (3). Dichos valores se sustituyen en (3) y se obtienen las raices del polinomio que representan los valores caracteristicos.

Para ejemplificar un poco el metodo, considerese que se requiere obtener los valores caracteristicos de la siguiente matriz:

$$A = \begin{bmatrix} 7 & 3 \\ 2 & 4 \end{bmatrix}$$

Se da ademas como vector arbitrario, para arrancar el metodo, al vector $Y = (1, 2)$, de esta manera $PN(A)$ queda:

$$\begin{aligned} PN(A)Y &= \begin{bmatrix} 7 & 3 \\ 2 & 4 \end{bmatrix}^2 \begin{bmatrix} 1 \\ 2 \end{bmatrix} + c_1 \begin{bmatrix} 7 & 3 \\ 2 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix} + c_0 \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ &= \begin{bmatrix} 121 \\ 6 \end{bmatrix} + c_1 \begin{bmatrix} 13 \\ 10 \end{bmatrix} + c_0 \begin{bmatrix} 1 \\ 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \end{aligned}$$

Esto nos da el sistema:

$$\begin{cases} 13 c_1 + c_0 = -121 \\ 10 c_1 + 2 c_0 = -6 \end{cases}$$

La solución de este sistema es $(-11, 22)$ y representan los coeficientes de la ecuación característica (3), en este caso resulta el polinomio:

$$\lambda^2 - 11\lambda + 22$$

Cuyas raíces son los valores característicos buscados.

5.3.2) PROGRAMA KRYLOV

OBJETIVO:

Obtener los valores característicos de una o varias matrices cuadradas por el método de Krylov.

ALGORITMO EN PSEUDOCODIGO:

PROGRAMA PRINCIPAL

VARIABLE	USO	TIPO	FUNCION
EPS	-	Constante	Criterio de convergencia para la subrutina GAUSJOR.
N1	-	Constante	Orden maximo de las matrices u vectores usados en el programa (es modificable).
MC, I	E	Real	Orden de la matriz en cuestion u variable de iteracion entera.
DET	-	Real	Variable para la subrutina GAUSJOR, para ver si el determinante de la matriz es nulo.
A	E	Arreglo	Matriz a la que se le buscan los valores caracteristicos.
D	-	Arreglo de Reales	Matriz de coeficientes del sistema de ecuaciones planteado.
B	E	Arreglo de reales	Vector arbitrario para formar el sistema de ecuaciones, se transforma en el vector de coeficientes del sistema (la solucion).
AA	-	Arreglo de reales	Vector en el que se dan los coeficientes del polinomio caracteristico, para la subrutina RAICES, que se obtuvieron en B.
RR	S	Arreglo	Matriz de orden $MC \times 2$ en la que se dan los valores caracteristicos encontrados, cada renglon es una raiz del pol. caract.

Pseudocodigo:

BEGIN

Leer el orden de la matriz, MC.

REPEAT

Leer la matriz A.

Leer el vector inicial, de arranque.

CALL SISTEMAKRYLOV (A,B,D,MC).

CALL GAUSSJORDAN (D,B,MC,EPS,DET).

IF DET > EPS THEN

Ponemos en el arreglo AA los coeficientes del polinomio caracteristico.

CALL RAICES (AA,MC,RR).

Se escriben los valores caracteristicos encontrados, parte real y parte imaginaria.

ELSE

Escribir letrero de indicacion.

Dar orden de la matriz, para otros datos.

END.

SUBROUTINAS

1) LEEHAT(D,M,N)

(Lee una matriz de orden $M \times N$; ver sec. 2.2.2).

2) LEEVECT (D,M)

(Lee un vector de orden M ; ver sec. 3.2.2).

3) GAUSJOR (A,B,N,EPS,DET)

(Resuelve el sistema $AX=B$; la solución la deja en B; ver sección 3.2.2).

4) RAICES (AA,N,RR)

(Obtiene las raíces de un polinomio de grado N ; los coeficientes se dan en AA; ver sec. 4.2.2).

5) SISTEMAKRYLOV (A,B,D,N)

OBJETIVO: Plantear el sistema de ecuaciones $PN(A)Y$, para obtener los coeficientes de la ec. caract. así $Y=B$.

VARIABLE	USO	TIPO	FUNCION
A	E	Arreglo de reales	Matriz a la que se le buscan los valores característicos.
B	E/S	Arreglo de reales	Vector inicial para arrancar el método; finalmente se transforma en la solución del sistema.
D	S	Arreglo de reales	Matriz del sistema que queda planteado.
N	E	Entero	Orden de la matriz A.
CO	L	Arreglo	Vector temporal del producto AB.

Pseudocódigo:

```
BEGIN
  FOR I := N DOWNT0 1 DO
    Coloca el vector B en la columna 'I'
    de la matriz D.
    efectua el producto CO <--- AB.
    B <--- CO (valores de CO pasan a B).
    Multiplica el vector B por (-1).
  END.
```

5.3.3) LISTADO DEL PROGRAMA

```

PROGRAM KRYLOV (INPUT,OUTPUT);
CONST
  EPS = 0.0000001;
  N1 = 15;
TYPE
  MATRIZ = ARRAY[1..N1,1..N1] OF REAL;
  VECTOR = ARRAY[1..N1] OF REAL;
  COEF = ARRAY[-2..N1] OF REAL;
  RAIZ = ARRAY[1..N1,0..1] OF REAL;
VAR
  MC : 1..N1;
  DET : REAL;
  A,D : MATRIZ;
  B : VECTOR;
  AA : COEF;
  RR : RAIZ;
  I : 0..N1;

PROCEDURE SISTEMAKRYLOV (A : MATRIZ) VAR B : VECTOR;
  VAR D : MATRIZ; N : INTEGER;
TYPE
  VECTOR = ARRAY[1..N1] OF REAL;
VAR
  CO : VECTOR;
  I,J,K : INTEGER;
BEGIN
  FOR I := N DOWNTO 1 DO
    BEGIN
      FOR J := 1 TO N DO
        DCJ,IJ := BCJJ;
      FOR K := 1 TO N DO
        BEGIN
          COCKJ := 0.0;
          FOR J := 1 TO N DO
            COCKJ := COCKJ + ACK,JJ * BCJJ;
          END;
          FOR J := 1 TO N DO
            BCJJ := COCJJ;
          END;
        END;
      FOR I := 1 TO N DO
        BCII := - BCII;
      END(* DE SISTEMAKRYLOV *);
    END;
  (* PROGRAMA PRINCIPAL *)
BEGIN
  WRITELN (' DAR EL ORDEN DE LA MATRIZ');
  READ (MC);
  REPEAT
    WRITELN ('DAR COMPONENTES DE LA MATRIZ A');
    LEEMAT (A,MC,MC);
    WRITELN ('DAR COMPONENTES DEL VECTOR B');
    LEEVECT (B,MC);
    SISTEMAKRYLOV (A,B,D,MC);
  UNTIL

```

```

GAUSJOR (D,B,MC,EPS,DET);
IF DET > EPS THEN
  BEGIN
    FOR I := 0 TO MC - 1 DO
      AACIJ := BMC - IJ;
      AACMCJ := 1;
      RAICES (AA,MC,RR);
      WRITELN ('LOS VALORES CARACTERISTICOS SON:');
      WRITELN (' PARTE REAL           PARTE IMAGINARIA');
      FOR I := 1 TO MC DO
        WRITELN (RRCI,0J,RRCI,1J);
      END
    ELSE
      WRITELN ('NO HAY SOLUCION, SELECCIONAR OTRO VECTOR INICIAL');
      WRITELN;
      WRITELN ('DAR EL ORDEN DE LA MATRIZ');
      READ (MC);
      UNTIL EOF;
  END.

```

5.3.4) EJEMPLO

Obtener los valores característicos del sistema:

$$\begin{array}{rclcl}
 (1 - \lambda)X_1 + X_2 + X_3 + X_4 & = & 0 \\
 X_1 + (2 - \lambda)X_2 + 3X_3 + 4X_4 & = & 0 \\
 X_1 + 3X_2 + (6 - \lambda)X_3 + 10X_4 & = & 0 \\
 X_1 + 4X_2 + 10X_3 + (20 - \lambda)X_4 & = & 0
 \end{array}$$

Al correr el programa pide los datos:

DAR ORDEN DE LA MATRIZ

4

DAR COMPONENTES DE LA MATRIZ

```

1 1 1 1
1 2 3 4
1 3 6 10
1 4 10 20

```

DAR VECTOR INICIAL DE ARRANQUE

```

1 1 1

```

LOS VALORES CARACTERISTICOS SON:

PARTE REAL	PARTE IMAGINARIA
0.453834494227	0
26.3047032671	0
0.0380135419109	0
2.20344616508	0

5.4) COMENTARIOS

En este capítulo se abordó el problema $Ax = \lambda x$. Se dieron dos técnicas de solución como son el método de potencias de Jacobi (que solo encuentra el mayor valor característico y su vector característico asociado) y el método de Krylov (que obtiene todos los valores característicos). Ambas técnicas podrán usarse de acuerdo a las características del problema a resolver.

Sin embargo, es importante mencionar que existen técnicas, como los métodos de Jacobi, para resolver este tipo de problema, es decir $Ax = \lambda x$; la manera de resolver el problema es haciendo uso de transformaciones de coordenadas, representadas por matrices T_1, T_2, \dots, T_m , transformando la matriz original a una forma diagonal; matricialmente esto sería:

$$T_m^{-1} T_{m-1}^{-1} \dots T_1^{-1} A T_1 T_2 \dots T_m = B, \text{ donde } B \text{ es matriz diagonal.}$$

La matriz $U = T_1 T_2 \dots T_m$, tendrá finalmente los vectores característicos en cada una de sus columnas.

En la literatura aparecen otros métodos que podrían ser más eficientes, tomando en cuenta las características de la matriz involucrada, o bien métodos que resuelven el problema más general $Ax = \lambda Bx$. Todos ellos son estudiados en materias como Álgebra lineal numérica.

CAPITULO VI) INTERPOLACION

6.1) INTRODUCCION

En muchos procesos estadísticos y experimentos de tipo científico, los resultados obtenidos son una serie de valores muestrales, para ciertas condiciones dadas (una salida para cada entrada). A partir de dichos valores muestrales, frecuentemente es necesario obtener la(s) respuesta(s) debida(s) a alguna(s) entrada(s) (variable independiente) diferente(s) de los valores de la muestra, en esa situación se habla de interpolación, si la entrada dada se encuentra dentro del rango de los valores muestrales.

Supongamos que tenemos un conjunto de abscisas x_1, x_2, \dots, x_n y sus correspondientes ordenadas y_1, y_2, \dots, y_n donde:

$$a) x_i < x_{i+1}$$

$$b) y_i(x_i) = y_i$$

Es decir, tenemos el conjunto: $\{(x_i, y_i(x_i))\}, i = 1, 2, \dots, n$

El problema de interpolación unidimensional es construir una función F tal que $F(x_i) = y_i$ para toda 'i', tal que $F(x)$ asuma valores razonables para toda x entre los puntos de datos. El criterio de razonable es muy variado de problema a problema y nunca puede ser entendido satisfactoriamente.

Si las ordenadas (y) vienen de una función matemática y tienen un error de redondeo mínimo, entonces podemos esperar que tenga una solución satisfactoria para un x dado. El lector estará familiarizado con la interpolación lineal en una tabla de losaritmós, por poner un ejemplo.

Si los puntos (x_i, y_i) vienen de muchas observaciones experimentales y precisas, pueden ser considerados libres de error y pueden ser interpolados con una función lisa o continua. Si por otro lado, provienen de experimentos relativamente crudos no podemos justificar o forzar una función de interpolación para los datos exactamente.

Los propósitos de interpolación son muchos pero se usan frecuentemente como un criterio para obtener valores $F(x)$ para x que no están en la tabla de datos $\{(x_i, y_i)\}$. Algunas veces se requiere obtener $F'(x)$ y $F''(x)$ en puntos intermedios, o también querer evaluar la integral de F sobre un intervalo de (x_1, x_n) .

El conjunto de datos puede ser interpolado por un número infinito de funciones diferentes y debemos tener algún criterio para escoger alguna de ellas. El criterio normal está en términos de simplicidad y suavidad o continuidad de la curva, es decir, F será analítica y el valor de $|F''(x)|$ estará en un intervalo tan pequeño como sea posible, o F será un polinomio de grado mínimo, etc.

Funciones de interpolación son construidas por combinaciones lineales de funciones elementales como: combinaciones lineales de funciones trigonométricas $(\cos kx, \sin kx)$, combinaciones lineales de (x^k) , etc.

En este capítulo veremos la interpolación polinomial de Lagrange.

6.2) METODO DE LAGRANGE

6.2.1) INTRODUCCION

El problema de interpolación polinomial puede plantearse, inicialmente, como el de encontrar los $n+1$ coeficientes de un polinomio de grado n :

$$P(X) = a_0 + a_1 x + \dots + a_n x^n = \sum_{i=0}^n a_i x^i \quad (1)$$

De tal manera que el conjunto de datos $\{(x_i, y_i)\}_{i=0, \dots, n}$

Sea satisfecho por el polinomio; esto nos lleva a plantear un sistema de $n+1$ ecuaciones con $n+1$ incógnitas; es decir:

$$y_i = \sum_{j=0}^n a_j x_i^j \quad (i=0, \dots, n) \quad (2)$$

Se demuestra que existe un único polinomio P_n para cualquier problema de interpolación con $n+1$ abscisas distintas. Sin embargo siguiendo este camino puede llevarnos a plantear un sistema bastante difícil de resolver en un cómputo normal.

Un camino satisfactorio para computar el polinomio que interpola los puntos $\{(X_i, Y_i)\}$ es usar los polinomios básicos de Lagrange asociados con el conjunto $\{X_i\}$.

Dichos polinomios $\{L_j(X)\}_{j=0, \dots, n}$ de grado n tienen la

propiedad

$$L_j(X_i) = \begin{cases} 1 & \text{si } i = j \\ 0 & \text{en otro caso} \end{cases}$$

Es facil ver que el polinomio de grado n:

$$L_J(X) = \frac{(X-X_0)(X-X_1)\dots(X-X_{J-1})(X-X_{J+1})\dots(X-X_n)}{(X-X_0)(X-X_1)\dots(X-X_{J-1})(X-X_{J+1})\dots(X-X_n)}$$

Satisface estas condiciones; ademas $L_J(X_i) = 0$ para alguna "i" diferente de "J".

Correspondientes factores en el denominador normalizan el resultado $L_J(X_J) = 1$

El polinomio $L_J(X)Y_J$ toma los valores Y_J para cada X_J y es cero en los puntos X_i (i diferente de J).

Asi la interpolacion polinomial de grado n; la cual consta de n+1 puntos $\{(X_i, Y_i)\}$ esta dada por:

$$Y(X) = \sum_{J=0}^n L_J(X)Y_J \quad (4)$$

Es claro que el numero de operaciones (y consecuentemente tiempo de ejecucion) es proporcional a $n \times n$. Es importante enfatizar que hay uno y solo un polinomio de grado n o menor que interpola n+1 datos.

Hay muchas diferentes formulas de interpolacion polinomial en la literatura para otras bases; el polinomio dado por esta aproximacion es el mismo que el encontrado en la solucion del sistema de ecuaciones planteado (1).

6.2.2) PROGRAMA LAGRANGE

OBJETIVO:

Obtener valores Y para X que no estan en la tabla de datos $\{(X_i, Y_i)\}$, mediante interpolacion polinomial de Lagrange, para varios paquetes de datos.

ALGORITMO EN PSEUDOCODIGO

PROGRAMA PRINCIPAL

VARIABLE	USO	TIPO	FUNCION
X	E	Arreglo	Vector conteniendo las variables independientes de los puntos muestrales.
Y	E	Arreglo	Vector conteniendo las variables dependientes de los puntos muestrales.
VX	E	Arreglo	Vector de variables independientes de las que se obtiene por interpolacion sus correspondientes variables dependientes.
N	E	Entero	Cantidad de puntos muestrales.
M	E	Entero	Numero de puntos a interpolar.

Pseudocodiso:

BEGIN

```

Escribir letrero de entrada
Leer no. de muestras y no. de puntos a interpolar
WHILE no sea fin de archivo DO
  Leer vector de abscisas a interpolar
  Leer las muestras en parejas (abscisa, ordenada)
  CALL LAGRANGE (X, Y, M, N, VX, FVX)
  Escribir letreros de salida
  Escribir los datos encontrados
  Pide mas datos

```

END.

SUBROUTINAS

1) LEEVECT (D, M)

(Lee un vector de orden M; ver sec. 2.2.2),

2) ESCVECT (M)

(Escribe un vector de orden M, ver sec. 3.2.2).

3) LAGRANGE (X,Y,M,N,UX,FX)

Objetivo: Interpolar un conjunto de datos dados en UX.

VARIABLE	USO	TIPO	FUNCION
X	E	Arreglo	Vector conteniendo las variables independientes de los puntos muestrales.
Y	E	Arreglo	Vector conteniendo las variables dependientes de los puntos muestrales.
M	E	Entero	Numero de puntos a interpolar.
N	E	Entero	Numero de puntos muestrales.
UX	E/S	Arreglo	Vector de variables independientes de las que se obtiene por interpolacion sus correspondientes variables dependiente.
FX	S	Arreglo	Vector en el que obtienen las variables interpoladas.
RNUM	L	Real	Variable que contiene el valor del numerador de la exp. de Lagrange.
DENOM	L	Real	Variable que contiene el valor del denominador de la exp. de Lagrange.

Pseudocodiso:

```

BEGIN
  FOR K := 1 TO M DO
    FVXCKJ <--- 0.0
  FOR I := 1 TO N DO
    RNUM <--- 1.0
    DENOM <--- 1.0
    FOR J := 1 TO N DO
      IF J <> I THEN
        RNUM <--- RNUM * (UXCKJ - X[J])
        DENOM <--- DENOM * (XC[I] - X[J])
    FVXCKJ <--- FVXCKJ + YC[I] * RNUM / DENOM
END.

```

5.2.3) LISTADO DEL PROGRAMA

```

PROGRAM LAGRANGE (INPUT,OUTPUT)
CONST
  MAXIMO = 30
TYPE
  VECTOR = ARRAY[1..MAXIMO] OF REAL
VAR
  X, Y, VX, FVX : VECTOR
  I, M, N : 1..MAXIMO
  VI      : REAL

PROCEDURE LAGRANGE (X, Y : VECTOR; M, N : INTEGER;
                   VAR VX, FVX : VECTOR)
VAR
  RNUM, DENOM : REAL
  I, J, K     : INTEGER
BEGIN
  FOR K := 1 TO M DO
    BEGIN
      FVX[K] := 0.0
      FOR I := 1 TO N DO
        BEGIN
          RNUM := 1.0
          DENOM := 1.0
          FOR J := 1 TO N DO
            IF J <> I THEN
              BEGIN
                RNUM := RNUM * (VX[K] - X[J])
                DENOM := DENOM * (X[I] - X[J])
              END
            FVX[K] := FVX[K] + Y[I] * RNUM / DENOM
          END
        END
      END
    END
  END
  Writeln
END(* DE LAGRANGE *)

(      PROGRAMA PRINCIPAL      )

BEGIN
  Writeln('DAR NO. DE MUESTRAS Y NO. DE ABSC. A INTERPOLAR')
  Read(N,M)
  While Not Eof Do
    BEGIN
      Writeln('DAR ABSCISAS DE PUNTOS A INTERPOLAR')
      LEEVect (VX,M)
      Writeln('DAR LAS MUESTRAS EN PAREJAS (ABSCI,ORDEN)')
      FOR I := 1 TO N DO
        READ (X[I],Y[I])
      LAGRANGE (X,Y,M,N,VX,FVX)
      Writeln(' ' ; 15, 'LA INTERPOLACION DE LOS PUNTOS QUEDA:')
      WRITE (' ' ; 10, 'VALOR DE LA ABSCISA')
      Writeln(' ' ; 12, ' VALOR DE ORDENADA')
      FOR I := 1 TO M DO
        BEGIN

```

```

WRITE (' ':10,UXCII:12:5);
WRITELN (' ':25,FVXCII:12:5);
END;
WRITELN;
WRITELN('DAR NO. DE MUESTRAS Y NO. DE ABSC. A INTERPOLAR');
READ (N,M);
END;
END.

```

6.2.4) EJEMPLO

Aplicar la formula de lasranse para interpolar los puntos 1.10, 1.50, 1.70, a partir de los siguientes valores de la funcion de error normal:

$$Y(X) = \frac{E^{-X^2/2}}{\sqrt{2\pi}}$$

X	1.00	1.20	1.40	1.60	1.80	2.00
K						
Y	0.2420	0.1942	0.1497	0.1109	0.0720	0.054

Al ejecutar el programa se introducen los datos:

DAR NO. DE MUESTRAS Y NO. DE PTOS. A INTERPOLAR

DAR ABCISAS DE PUNTOS A INTERPOLAR

1.10 1.50 1.70

DAR LAS MUESTRAS EN PAREJAS (ABSCISA,ORDENADA)

1.00 0.2420

1.20 0.1942

1.40 0.1497

1.60 0.1109

1.80 0.0720

2.00 0.054

LA INTERPOLACION DE LOS PUNTOS QUEDA:

VALOR DE LA ABSCISA

1.10000

1.50000

1.70000

VALOR DE ORDENADA

0.21911

0.13017

0.09119

6.3) COMENTARIOS

Existen varias formulas para el problema de la interpolacion polinomial, estas pueden clasificarse en dos grupos: aquellas que se aplican a puntos igualmente espaciados y formulas para puntos arbitrariamente espaciados. Las formulas mas comunes son las del segundo grupo que son la interpolacion de diferencias divididas de Newton y la interpolacion de Lagrange.

Las formulas del primer grupo se han desarrollado para simplificar la interpolacion en tablas de funciones con argumentos igualmente espaciados; todas pueden ser derivadas por algunas formulas del primer grupo.

En la literatura aparecen otros esquemas de interpolacion equivalentes a pasar polinomios de segundo, tercero o mayor orden por tres, cuatro o mas puntos consecutivos y utilizar el polinomio para evaluar Y en cualquier valor, x , comprendido entre los puntos.

La formula de interpolacion de Stirling es un ejemplo de los mencionados; consiste en lo siguiente: dados tres puntos de una curva (x_{i-1}, y_{i-1}) , (x_i, y_i) y (x_{i+1}, y_{i+1}) tal que $x_{i+1} - x_i = x_i - x_{i-1} = h$ escribimos $u = (x - x_i) / h$ la formula

$$S = y_i + u \frac{y_{i+1} - y_{i-1}}{2} + \frac{u^2}{2} (y_{i+1} - 2y_i + y_{i-1})$$

indica el valor interpolado de Y que corresponde a x , determinado pasando una ecuacion cuadratica por las tres ordenadas.

Sin embargo, existen tecnicas muy eficientes (interpolan los datos con mayor exactitud) llamados SPLINE; la idea de estas

técnicas es que dados un conjunto de puntos muestrales (nodos) $\{(x_i, y_i)\}$ ($i = 1, \dots, n$), se aproxima una función $S(x)$ tal que $S(x_i) = y_i$, en cada subintervalo $[x_{i-1}, x_i] = I$, donde $S(x)$ es un polinomio.

VII) APROXIMACION POLINDHIAL.

7.1) INTRODUCCION

El problema de la aproximación polinomial consiste en la representación de un conjunto de puntos por medio de un polinomio, aunque, a diferencia de la interpolación, dicho polinomio no tiene que coincidir necesariamente con la función representada por los puntos, sino que este se ajusta con una cierta desviación a dichos puntos. Esto hace de la aproximación polinomial un método particularmente útil para tratar puntos experimentales, ya que estos tienen intrínsecamente un cierto error y no es necesario que el polinomio pase exactamente por los puntos en cuestión.

En este capítulo se trata el método de aproximación por mínimos cuadrados.

7.2) MINIMOS CUADRADOS

7.2.1) INTRODUCCION

Este metodo consiste en aproximar un polinomio a un conjunto de puntos $(x_0, y_0), \dots, (x_N, y_N)$ de tal forma que la suma de los cuadrados de la diferencia (o desviacion) entre el polinomio seleccionado, $P(x)$, evaluado en cierta abscisa, y la ordenada correspondiente a dicha abscisa sea minimo (fig. 1), es decir:

$$\sum_{k=0}^N (P(x_k) - y_k)^2 = \text{minimo}$$

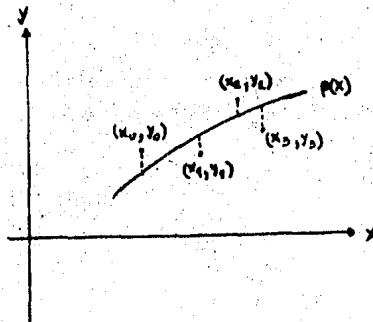


Figura 1.

Una vez escogido el grado del polinomio $P(x)$ que mejor se ajuste a los puntos experimentales, el problema se reduce a encontrar los coeficientes adecuados a_0, a_1, \dots, a_n del polinomio para los cuales se cumple la condicion anterior. Llamese d_k a las desviaciones $P(x_k) - y_k$ y definase F como la suma de los cuadrados

de dichas desviaciones, es decir:

$$F = \sum_{k=0}^N d_k^2 = \sum_{k=0}^N (P(x_k) - y_k)^2$$

Expresando F como función de los coeficientes del polinomio $P(x)$ y aplicando el criterio de mínimos cuadrados obtenemos:

$$F(a_0, a_1, \dots, a_n) = \sum_{k=0}^N (a_0 + a_1 x_k + a_2 x_k^2 + \dots + a_n x_k^n - y_k)^2 = \text{mínimo}$$

Y siendo F una función de $n+1$ variables, el problema de encontrar su valor mínimo se reduce a encontrar los valores de F para los cuales las $n+1$ derivadas parciales de la función con respecto a cada uno de los coeficientes sean simultáneamente cero (esto nos garantiza que encontraremos su valor mínimo porque una función que es la suma de los cuadrados de varias variables no tiene máximo, solamente mínimo) o sea:

$$\frac{\partial F}{\partial a_0} = 0, \quad \frac{\partial F}{\partial a_1} = 0, \quad \dots, \quad \frac{\partial F}{\partial a_n} = 0$$

Substituyendo la expresión para F en las parciales tenemos que:

$$\frac{\partial F}{\partial a_0} = \sum_{k=0}^N 2(a_0 + a_1 x + \dots + a_n x^n - y) = 0$$

$$\frac{\partial F}{\partial a_1} = \sum_{k=0}^N 2x(a_0 + a_1 x + \dots + a_n x^n - y) = 0$$

$$\frac{\partial F}{\partial a_2} = \sum_{k=0}^N 2x^2(a_0 + a_1 x + \dots + a_n x^n - y) = 0$$

⋮

$$\frac{\partial F}{\partial a_n} = \sum_{k=0}^N 2x^n(a_0 + a_1 x + \dots + a_n x^n - y) = 0$$

Eliminando el factor 2 de este sistema de ecuaciones y reordenando terminos obtenemos el sistema equivalente:

$$\begin{aligned} \sum_{J=0}^n \left(\sum_{k=0}^N x_k^J \right) a_J &= \sum_{k=0}^N y_k \\ \sum_{J=0}^n \left(\sum_{k=0}^N x_k^{J+1} \right) a_J &= \sum_{k=0}^N x_k y_k \\ &\vdots \\ \sum_{J=0}^n \left(\sum_{k=0}^N x_k^{J+2} \right) a_J &= \sum_{k=0}^N x_k^2 y_k \end{aligned}$$

O bien de forma mas compacta:

$$\sum_{J=0}^n \left(\sum_{k=0}^N x_k^{i+J} \right) a_J = \sum_{k=0}^N x_k^i y_k \quad i = 0, 1, \dots, n$$

O lo que es lo mismo:

$$\sum_{J=1}^{n+1} \left(\sum_{k=0}^N x_k^{i+J-2} \right) a_{J-1} = \sum_{k=0}^N x_k^{i-1} y_k \quad i = 1, 2, \dots, n, n+1 \quad (1)$$

En base a esta ultima expresion se puede desarrollar el algoritmo

para determinar los coeficientes a_0, a_1, \dots, a_n del polinomio ya que al desarrollarla obtenemos un sistema de ecuaciones lineales con las incógnitas $a_0, a_1, a_2, \dots, a_n$. Por ejemplo, al desarrollar la expresión (1) para $n=2$ obtenemos el siguiente sistema:

$$\begin{aligned} \left(\sum_{k=0}^N 1 \right) a_0 + \left(\sum_{k=0}^N x_k \right) a_1 + \left(\sum_{k=0}^N x_k^2 \right) a_2 &= \sum_{k=0}^N y_k \\ \left(\sum_{k=0}^N x_k \right) a_0 + \left(\sum_{k=0}^N x_k^2 \right) a_1 + \left(\sum_{k=0}^N x_k^3 \right) a_2 &= \sum_{k=0}^N x_k y_k \\ \left(\sum_{k=0}^N x_k^2 \right) a_0 + \left(\sum_{k=0}^N x_k^3 \right) a_1 + \left(\sum_{k=0}^N x_k^4 \right) a_2 &= \sum_{k=0}^N x_k^2 y_k \end{aligned}$$

El cual, una vez conocidos los términos entre parentesis, puede resolverse por el método de Gauss-Jordan (Capítulo 3) para a_0, a_1 y a_2 .

7.2.2) PROGRAMA MINIMOS CUADRADOS.

OBJETIVO:

Determinar, utilizando el criterio de mínimos cuadrados, los coeficientes de un polinomio de grado n que aproxime a un conjunto de puntos.

DESCRIPCION:

Este programa utilizara 4 subrutinas, la primera de ellas, la subrutina PLANTEASISTEMA, calculara los coeficientes del sis-

tema de ecuaciones lineales generado por la ecuación (1) así como los términos independientes de dicho sistema. La segunda será la subrutina GAUSJOR (desarrollada en el capítulo 3) que resolverá el sistema de ecuaciones planteado por la subrutina PLANTEASISTEMA y generará los coeficientes a_0, a_1, \dots, a_n del polinomio. Además se hará uso de las subrutinas LEEVECT Y ESCVECT (Capítulo 2) con el fin de facilitar la lectura de los datos y la escritura de los resultados.

El objeto de la subrutina planteasistema es calcular los elementos de la matriz asociada al sistema de ecuaciones generado por los datos, así como el vector de términos independientes. Para esto se consideran las variables MATRIZN y VECTORN (que son, respectivamente, arreglos de 2 y de una dimensión) como parámetros en la subrutina y son los que finalmente se darán como entrada a la subrutina GAUSJOR para resolver el sistema formado por MATRIZN y VECTORN.

Para calcular los valores de MATRIZN y VECTORN la subrutina utiliza la fórmula (1) dada en la introducción de la siguiente manera:

para calcular los elementos de MATRIZN:

$$\text{MATRIZN}(I, J) = \sum_{k=0}^N x_k^{i+J-2} = \sum_{k=0}^N x_k^{i-1} x_k^{J-1} \quad (2)$$

donde $i = 1, 2, \dots, n+1$, $J = 1, 2, \dots, n+1$

y para los elementos de VECTORN:

$$\text{VECTORN}(I) = \sum_{k=0}^{N} x_k^{i-1} y_k \quad i = 1, 2, \dots, n+1$$

Para facilitar los calculos de los elementos de MATRIZN se consideraron las variables auxiliares PROD1 y PROD2 que son definidas como:

$$\text{PROD1} = x_k^{i-1}$$

$$\text{PROD2} = x_k^{j-1}$$

Así, cada elemento de la matriz se genera de la sig. forma:

$$\text{MATRIZN}(I,J) = \text{MATRIZN}(I,J) + (\text{PROD1})(\text{PROD2})$$

con

$$\text{PROD1} = x_k^{i-1} \quad ; \quad \text{PROD2} = x_k^{j-1} \quad k = 0, 1, 2, \dots$$

Puesto que la matriz es simétrica, solo se calculan los elementos del triángulo inferior y la diagonal de la matriz para luego ser transferidos al triángulo superior.

Se hacen llamadas a las subrutinas GAUSJOR (Capítulo 3), LEEVECT (Capítulo 2) y ESCVECT (Capítulo 2).

ALGORITMO EN PSEUDOCODIGO:

PROGRAMA MINIMOSCUADRADOS

VARIABLES	USO	TIPO	FUNCION
EPS	-	Const.	Criterio para determinar si el determinante de la matriz es cero
MAXORDEN	-	Const.	Maximo orden del polinomio
MAXPUNTOS	-	Const.	Numero maximo de datos
DET	-	Real	Determinante de la matriz
MATRIZN	-	Arreglo de Reales	Elementos de la matriz asociada al sistema de ecuaciones.
GRADO	E	Entero	Grado del polinomio
NUMPUNTOS	E	Entero	Numero de puntos
X	E	Arreglo de Reales	Abscisas de los puntos
Y	E	Arreglo de Reales	Ordenadas de los puntos
VECTORN	S	Arreglo de Reales	Coefficientes del polinomio P(x)

Pseudocodigo:

```

BEGIN
Lee grado del polinomio y numero de puntos.
WHILE not eof DO
  Lee X, Y.
  CALL PLANTEASISTEMA (MATRIZN,VECTORN,X,Y,NUMPUNTOS,GRADO).
  CALL GAUSJOR (MATRIZN,VECTORN,GRADO+1,EPS,DET).
  IF DET > EPS THEN
    Escribe los coeficientes del polinomio.
  ELSE
    manda mensaje.
  Lee grado del polinomio y numero de puntos.
END.

```

SUBROUTINAS:

1) LEEVECT (D,M)

Objetivo: Leer un vector de orden M.

(Ver capitulo 2)

2) ESCVECT (D,M)

Objetivo: Escribe un vector de orden M.

(Ver capitulo 2)

3) GAUSJOR (A,B,N,EPS,DET)

Objetivo: Obtener la solución un sistema de ecuaciones.

(Ver capítulo 3)

4) PLANTEASISTEMA(MATRIZN,VECTORN,X,Y,NUMPUNTOS,GRADO)

Objetivo: Calcular los elementos de la matriz asociada al sistema de ecuaciones generado por el criterio de mínimos cuadrados.

VARIABLES	USO	TIPO	FUNCION
X	E	Arreglo de Reales	Abciscas de los puntos
Y	E	Arreglo de Reales	Ordenadas de los puntos
NUMPUNTOS	E	Entero	Numero de puntos
GRADO	E	Entero	Grado del polinomio
MATRIZN	S	Arreglo de Reales	Elementos de la matriz asociada al sistema de ecuaciones
VECTORN	S	Arreglo de Reales	Terminos independientes del sistema de ecuaciones
I	L	Entero	Indicador de renglones de la matriz
J	L	Entero	Indicador de columnas de la matriz
K	L	Entero	Contador de puntos
PROD1	L	Real	Guarda resultados parciales
PROD2	L	Real	" " " "

Pseudocodiso:

```

BEGIN
  Inicializar MATRIZN y VECTORN en ceros.
  FOR K = 1 TO NUMPUNTOS DO
    FOR I = 1 TO GRADO + 1 DO
      FOR J = 1 TO I DO
        Calcula el elemento i-J de la matriz.
        Calcula el i-esimo termino independiente.
      Transfiere los elementos del triangulo inferior de la matriz (ya
      calculados) al triangulo superior.
    END.
  END.

```

7.2.3) LISTADO DEL PROGRAMA:

```

PROGRAM MINIMOSCUADRADOS(INPUT, OUTPUT);
CONST
  EPS = 0.0000001;
  MAXORDEN = 6;
  MAXPUNTOS = 20;
TYPE
  MATRIZ = ARRAY[1..MAXORDEN, 1..MAXORDEN] OF REAL;
  VECTOR = ARRAY[1..MAXORDEN] OF REAL;
  COORDENADAS = ARRAY[1..MAXPUNTOS] OF REAL;
VAR
  DET          : REAL;
  GRADO, NUMPUNTOS : INTEGER;
  MATRIZN     : MATRIZ;
  VECTORN     : VECTOR;
  X, Y        : COORDENADAS;
PROCEDURE PLANTEASISTEMA(VAR MATRIZN : MATRIZ) VAR VECTORN : VECTOR;
  X, Y : COORDENADAS;
  NUMPUNTOS, GRADO : INTEGER);
VAR
  I, J, K      : INTEGER;
  PROD1, PROD2 : REAL;
BEGIN
  { INICIALIZACION DE LA MATRIZN }
  FOR I:= 1 TO GRADO + 1 DO
    BEGIN
      VECTORNCI := 0;
      FOR J:=1 TO I DO
        MATRIZNCI,J := 0;
      END;
    FOR K:=1 TO NUMPUNTOS DO
      BEGIN
        PROD1 := 1.0;
        FOR I := 1 TO GRADO + 1 DO
          BEGIN
            PROD2 := 1.0;
            FOR J := 1 TO I DO
              BEGIN
                MATRIZNCI,J := MATRIZNCI,J + PROD1 * PROD2;
                PROD2 := PROD2 * XCK;
              END;
            VECTORNCI := VECTORNCI + PROD1 * YCK;
            PROD1 := PROD1 * XCK;
          END;
        END;
      { TRANSFIERE LOS ELEMENTOS DEL TRIANGULO INFERIOR AL TRIANGULO SUP. }
      FOR I := 1 TO GRADO + 1 DO
        FOR J := 1 TO I DO
          MATRIZNCJ,I := MATRIZNCI,J;
        END;
      END;
  {          PROGRAMA PRINCIPAL          }
  BEGIN
    WRITELN('GRADO DEL POLINOMIO ?, NUM. DE PUNTOS ?');
    READLN(GRADO, NUMPUNTOS);
  REPEAT

```

```

WRITELN('DAR LAS ABCISAS DE LOS PUNTOS');
LEEVECT(X, NUMPUNTOS);
WRITELN('DAR LAS ORDENADAS DE LOS PUNTOS');
LEEVECT(Y, NUMPUNTOS);
PLANTEASISTEMA (MATRIZN, VECTORN, X, Y, NUMPUNTOS, GRADO);
GAUSJOR (MATRIZN, VECTORN, GRADO + 1, EPS, DET);
IF DET > EPS THEN
  BEGIN
    WRITELN(' COEFICIENTES DEL POLINOMIO (EN ORDEN CRECIENTE):');
    WRITELN;
    ESCVECT (VECTORN, GRADO + 1);
    END
ELSE
  WRITELN('EL SISTEMA PLANTEADO NO TIENE SOLUCION');
  WRITELN('GRADO DEL POLINOMIO ?, NUM. DE PUNTOS ?');
  READLN(GRADO, NUMPUNTOS);
UNTIL EOF;
END.

```

7.1.4) EJEMPLO:

Los valores para el voltaje y la intensidad de la corriente en un circuito simple de corriente alterna con un capacitor de .47 microfaradios son los siguientes:

V (Volts)	I (Microamperios)
10	1
20	3
30	5
40	7
50	9
60	11.2
70	13.5
80	15.2
90	17
100	19
110	21.5
120	23.5

Sabiendo que en un circuito de este tipo la relacion Tension - Intensidad esta dada por:

$$I = (1/R)V$$

Entonces el inverso de la capacitancia se obtiene calculando la pendiente del polinomio de grado 1 ajustado a los puntos dados.

Para encontrar dicho polinomio, se ejecuta el programa MINIMOS-CUADRADOS de la siguiente forma:

GRADO DEL POLINOMIO?, NUM. DE PUNTOS?

1
12

DAR LAS ABCISAS DE LOS PUNTOS

10 20 30 40 50 60 70 80 90 100 110 120

DAR LAS ORDENADAS DE LOS PUNTOS

1 3 5 7 9 11.2 13.5 15.2 17 19 21.5 23.5

COEFICIENTES DEL POLINOMIO (EN ORDEN CRECIENTE):

-1.0894 0.2038

7.3) COMENTARIOS

En la seccion anterior se describio la tecnica de los minimos cuadrados para el caso de aproximacion polinomial. De esta tecnica resulto un sistema de ecuaciones llamado en la literatura ecuacion normal. Desafortunadamente el metodo tiene deficiencias, es decir, resolver el sistema o ecuacion normal, debido a que el sistema resulta ser incondicionado, sobre todo cuando el grado (m) del polinomio requerido sea muy grande. Otra desventaja es cuando el valor de m es cambiado, se deben calcular un conjunto de nuevos coeficientes.

La ecuacion normal puede resolverse por cualquiera de las subrutinas vistas en este capitulo, pero se corre el riesgo de introducir graves errores de redondeo, ademas, dadas las caracteristicas de la ecuacion normal podria resolverse por metodos mas eficientes, es decir que requieran menor numero de operaciones. Los errores de redondeo introducidos, usando eliminacion Gaussiana, tienen su origen en la formacion de las sumas de la ecuacion normal y en la eliminacion propiamente; esto se debe a que al trabajarse con cantidades muy grandes hay perdida de digitos, debido al almacenamiento finito de la maquina.

Las dificultades mencionadas arriba pueden evadirse haciendo uso de los polinomios ortogonales (se habla un poco de ellos en el capitulo ix). Usando estos polinomios se tienen soluciones mas precisas y algunas veces se permite la solucion por Gauss-Seidel. Un ejemplo de estos polinomios son los polinomios de Chebychev definidos asi: $T_n(x) = \cos n\theta$, con $x = \cos\theta$. Estos polinomios aproximan los datos a expresiones de la forma:

$$Y_m(x) = C_0 T_0(x) + C_1 T_1(x) + \dots + C_m T_m(x)$$

Donde cada T_j es un polinomio de Chebuehev de orden J . Nuestro problema inmediato es derivar una expresion para cada uno de los coeficientes c_j ($J=0, \dots, m$). Esto no se hace aqui ya que se requiere tener conocimiento de ciertas propiedades de los polinomios ortogonales; pero cabe mencionar que esta tecnica evita resolver un sistema de ecuaciones; ya que la matriz resultante es diagonal. La convergencia es mas rapida que la serie de Taylor y

a) se reduce el tiempo de maquina.

b) la estimacion del error es mas aproximada.

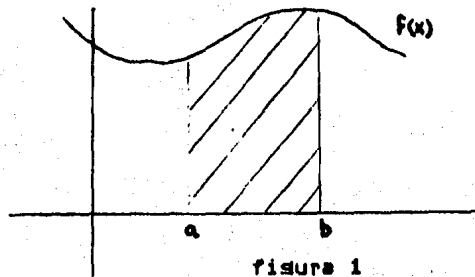
Esta economia hace que las aproximaciones en series de Chebuehev se use en muchas computadoras para evaluar funciones.

VIII) INTEGRACION NUMERICA

8.1) REGLA DEL TRAPECIO Y REGLAS DE SIMPSON

8.1.1) INTRODUCCION

Recordemos que la integral de una función $f(x)$ representa el área limitada por la gráfica de una función, el eje de las X's y los límites de integración $[a,b]$ (ver figura 1).



Si bien muchas funciones se pueden integrar al obtener su primitiva, es decir una función $F(x)$ que al derivarla nos da $f(x)$ ($F'(x) = f(x)$), existen otras tales que carecen de esta, o sea, su integral no tiene una expresión explícita. Por otro lado hay funciones cuya primitiva u/o la forma de obtenerla son tan complicadas que es preferible aplicar otro método para integrarlas. Ejemplos del primer tipo de funciones son

$$f(x) = e^{-x^2} \quad f(x) = \sqrt{1+x^3} \quad f(x) = \frac{\sin x}{x}$$

y del segundo tipo las funciones racionales

$$f(x) = \frac{12x^4 - 11x^3 + 134}{x^7 + 8x^6 - 13x^4 + 3x^3 - 6} \quad f(x) = \frac{-x^5 + 19x^3 - 12x^2 + 4}{x^{12} - 7x^8 + 5x^7 + 12x^4}$$

Para estos casos se utilizan métodos de integración numérica que logran acercarse al valor de la integral con un grado de exactitud tan grande como se quiera.

Los métodos de integración numérica se basan en dividir el intervalo a integrarse en un conjunto de partes pequeñas, calcular el área de cada una por un método aproximativo y sumar todas estas áreas. En general hay dos tipos de métodos: Aquellos en que el intervalo se divide en partes iguales y aquellos en que el tamaño de las partes es arbitrario y depende de la función. Aquí se tratarán métodos del primer tipo. Es claro que entre más divisiones se tengan más cercano será el valor del área aproximada al valor de la integral en una división y entonces, la suma de estas pequeñas áreas, se acercará más al valor de la integral en todo el intervalo. Esto se representa en la figura 2.

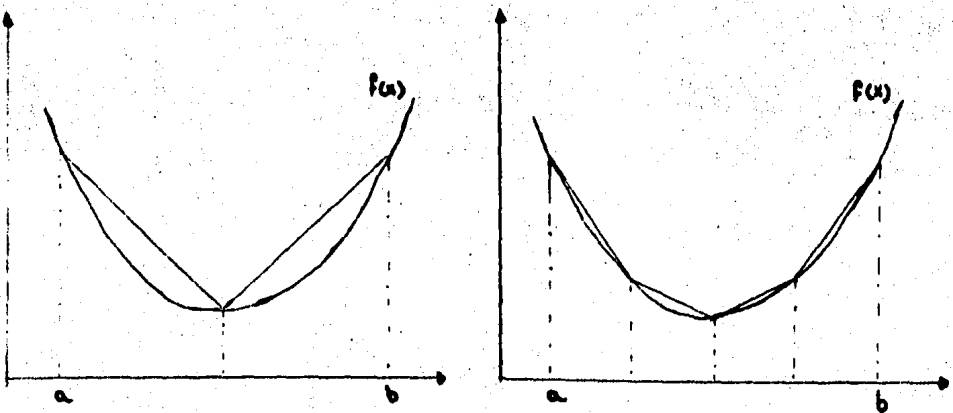


figura 2

Es a partir de polinomios que las áreas de las divisiones de los intervalos se aproximan como se verá más adelante. A continuación se presentan de manera ordenada formas de obtener esta

aproximación.

1) Regla trapecial.

Esta es la más sencilla y a la vez la menos exacta de las reglas existentes; aunque puede dar una buena idea acerca de la integración numérica estudiada. Supongamos que dividimos el intervalo $[a, b]$ en N partes iguales. Si h es el tamaño de cada parte, entonces h está dada por:

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

Los N puntos x que separan cada parte serán entonces

$$\begin{aligned} x_0 &= a \\ x_1 &= a + h \\ x_2 &= a + 2h \\ &\vdots \\ x_{N-1} &= a + (N-1)h \\ x_N &= b \end{aligned}$$

Si queremos saber un valor aproximado de la integral en alguna de las partes, digamos entre los puntos x_i y x_{i+1} , podemos construir un trapecio cuya área se puede calcular fácilmente y se parece a la de la función si los puntos están lo suficientemente cercanos. (Ver figura 3).

El área A del trapecio en el intervalo $[x_i, x_{i+1}]$ es (áreas del rectángulo más la del triángulo):

$$A = f(x_i)h + \left[\frac{f(x_i) + f(x_{i+1})}{2} \right] (h/2) = (1/2) h [f(x_i) + f(x_{i+1})]$$

Sumando todas las áreas A de todas las partes (del intervalo $[a, b]$) obtenemos el valor aproximado a la integral (el dos aparece puesto que se suman los valores $f(x_i)$ para dos divisiones contiguas)

$$\begin{aligned} \int_b^a f(x) dx &\approx A_1 + \dots + A_{N-1} + A_N = \sum_{i=1}^N A_i = \\ &= \frac{h}{2} [f(a) + 2f(x_1) + 2f(x_2) + \dots + 2f(x_{N-1}) + f(b)] \end{aligned} \quad (1)$$

Otra manera de tratar el problema en que se lleve a la misma solución es considerar la función lineal $f_1(x) = ax + b$ que pasa por los puntos $(x_i, f(x_i))$ y $(x_{i+1}, f(x_{i+1}))$ (ver figura 3). La integral de esta función será una buena aproximación a la función $f(x)$ en ese intervalo (dado por $[x_i, x_{i+1}]$). Al igual que antes, sumando todas las integrales de cada parte, obtendremos un valor cercano a la integral. Una de las formas de obtener tal función lineal es la siguiente:

Llamamos A_i al área bajo la curva en el intervalo $[x_i, x_{i+1}] = [x_i, x_i + h]$. Entonces

$$\begin{aligned}
 A_i &= \int_{x_i}^{x_i+h} f(x) dx \sim \int_{x_i}^{x_i+h} (ax + b) dx = \\
 &= \frac{a(x_i + h)^2}{2} - \frac{ax_i^2}{2} + b(x_i + h) - bx_i
 \end{aligned}$$

Esto puede ser escrito como

$$= h \left[\frac{(ax_i + b) + (a(x_i + h) + b)}{2} \right]$$

pero sabiendo que los valores de la función $f(x)$ y $f_1(x)$ en los puntos x_i y $x_i + h$ coinciden, o sea

$$f(x_i) = f_1(x_i) = ax_i + b$$

y

$$f(x_{i+1}) = f_1(x_{i+1}) = f(x_i + h) = a(x_i + h) + b$$

se deduce entonces que esto se puede convertir en

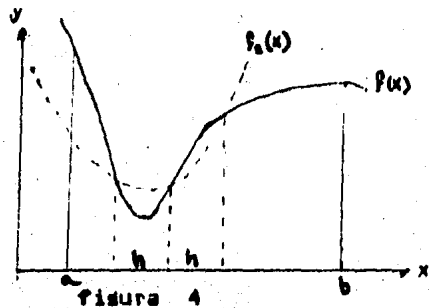
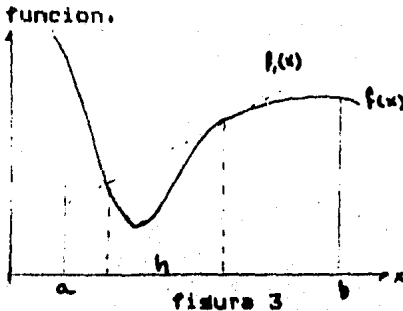
$$A = \frac{1}{2} h [f(x_i) + f(x_{i+1})]$$

Por tanto hemos llegado analíticamente a la deducción geométrica anterior. Evidentemente, la aproximación de la integral estará dada en igual forma que en la ecuación (1).

2) Regla de Simpson (1/3)

Esta es una de las más usadas reglas de integración. A dife-

rencia del metodo anterior aqui se trata de aproximar una parabolica que se ajuste a la funcion en una division $[x_L, x_{L+1}]$. De la figura 4 se puede deducir que el metodo es preferible que la regla del trapecio, dado que se acerca mejor a la integral de la funcion.



La regla de Simpson da precisamente la integral de la función cuadrática que pasa por tres puntos. Siguiendo el ejemplo de la regla del trapecio, dividimos el intervalo $[a, b]$ en N partes de tres puntos ($h = (b - a) / 2N$). Tomamos la parte $[x_L - h, x_L + h]$ del intervalo y tratamos de obtener la integral de la función cuadrática $f_2(x) = ax^2 + bx + c$ que pasa por los tres puntos $x_L - h$, x_L y $x_L + h$. Esta área será una aproximación a la integral en esa parte del intervalo. Sumando estas áreas obtendremos un valor aproximado de la integral.

De acuerdo a lo anterior, si $[x_L - h, x_L + h]$ es el intervalo a integrarse, el área A_L bajo la curva de la función $f(x)$ estará aproximada por:

$$A_L = \int_{x_L-h}^{x_L+h} f(x) dx \approx \int_{x_L-h}^{x_L+h} f_2(x) dx = \int_{x_L-h}^{x_L+h} (ax^2 + bx + c) dx =$$

(Resolviendo la integral y agrupando terminos)

$$= \frac{1}{3} h [f(x_{i-h}) + 4f(x_i) + f(x_{i+h})]$$

El area total es pues:

$$\int_b^a f(x) dx \approx A_1 + \dots + A_{N-1} + A_N = \sum_{i=1}^N A_i =$$

$$= \frac{h}{3} [f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \dots$$

$$\dots + 2f(x_{N-4}) + 4f(x_{N-3}) + 2f(x_{N-2}) + 4f(x_{N-1}) + f(x_N)]$$

Notese que aqui nuevamente el dos se ha agregado puesto que se ha sumado valores de la funcion $f(x)$ en puntos x que pertenecen a dos divisiones (contiguas) al mismo tiempo.

3) Caso general.

Hasta ahora, hemos aproximado a la integral de la funcion integrales de rectas, funciones lineales o de primer grado, dados dos puntos de una particion y funciones quadraticas o de segundo grado, dados tres puntos de una particion. Podemos ahora generalizar lo que se ha hecho al afirmar que de igual forma se podra ajustar una funcion cuadratica, una funcion cubica o en general una funcion de grado N , dados tres, cuatro o $N+1$ puntos, respectivamente en una particion. A continuacion presentamos una tabla con las reglas que aproximan la integral de una funcion con integrales de polinomios ajustados, evaluados en el intervalo $[x_1, x_{N+1}]$, dados los $N+1$ puntos necesarios. Se repiten las va ob-

tenidas. En general, estas se pueden obtener aplicando el metodo analitico usado para los dos anteriores ya tratadas.

Grado	Polinomio
1	$(h/2) [f(x_1) + f(x_2)]$
2	$(h/3) [f(x_1) + 4f(x_2) + f(x_3)]$
3	$(3h/8) [f(x_1) + 3f(x_2) + 3f(x_3) + f(x_4)]$
4	$(2h/45) [7f(x_1) + 32f(x_2) + 12f(x_3) + 32f(x_4) + 7f(x_5)]$
5	$(5h/288) [19f(x_1) + 75f(x_2) + 50f(x_3) + 50f(x_4) + 75f(x_5) + 19f(x_6)]$

De estas reglas escogimos la de grado tres para programarla. Esta se conoce como la regla de Simpson (3/8). Dado que lo representado arriba es aproximacion de la integral unicamente para una parte del intervalo, la regla de integracion para todo el intervalo para un polinomio de grado tres es:

$$\int_b^a f(x) dx \sim A_1 + \dots + A_{N-1} + A_N = \sum_{i=1}^N A_i =$$

$$= \frac{3h}{8} [f(a) + 3f(x_1) + 3f(x_2) + 2f(x_3) +$$

$$3f(x_4) + 3f(x_5) + 2f(x_6) + \dots$$

$$\dots + 3f(x_{N-4}) + 3f(x_{N-3}) + 2f(x_{N-2}) + 3f(x_{N-1}) + f(b)]$$

(2)

8.2.2) PROGRAMA INTEGRAL

OBJETIVO:

Obtener las integrales de una función en intervalos dados.

DESCRIPCION:

El programa que se tiene calcula la integral usando la regla de aproximación de un polinomio de tercer grado. El algoritmo del programa es sencillo y puede ser fácilmente deducido de la ecuación (2) escrita arriba. El programa tiene como entrada el intervalo a integrarse dado que la función está declarada internamente como subrutina (de tipo función). El programa consta de una subrutina principal que calcula la integral y que tiene como parámetros los límites de integración y además la función a integrarse. Hay que hacer notar aquí que el programa aprovecha la ventaja de Pascal que posibilita pasar una función como parámetro.

El algoritmo aumenta el número de particiones y calcula la integral para cada partición hasta que el error en las integrales calculado como error relativo (toma el valor absoluto de la diferencia entre la integral calculada anteriormente y la actual, dividida entre la segunda) es lo suficientemente pequeño, según un parámetro marcado declarada por el usuario.

El algoritmo es eficiente puesto que aprovecha la suma de valores de la función obtenidos en la división anterior. Según la ecuación (2) hay que sumar valores de la función en ciertos puntos y multiplicarlos por dos, sumar otros valores y multiplicarlos por tres y finalmente, sumar los valores de la función evaluada en a y en b . El algoritmo obtiene la nueva división tomando la

anterior y fraccionandola en tres. De esta manera solo es necesario evaluar los puntos intermedios (dado que los extremos fueron evaluados anteriormente).

ALGORITMO EN PSEUDOCODIGO:

PROGRAMA INTEGRACION

VARIABLES	USO	TIPO	FUNCION
AA	E	Real	Limite inferior de integracion
BB	E	Real	Limite superior de integracion
AREAP	S	Real	Integral de la funcion en [AA, BB]

Pseudocodizo:

```

BEGIN
Lee AA y BB
WHILE NOT fin de archivo DO
    CALL INTEGRAL (AA, BB, AREAP, FUNCION)
    Escribe valor de la integral
    Lee AA Y BB
END
  
```

SUBROUTINAS

1)FUNCTION FUNCION (X)

Objetivo: Calcula el valor de la funcion en el punto x. Esta funcion es la que el usuario desea integrar y es modificable por el usuario.

2)INTEGRAL (A, B, AREA)

Objetivo: Calcula la integral dados el intervalo [a,b] y la funcion f(x).

VARIABLES	USO	TIPO	FUNCION
A, B	E	Reales	Limites del intervalo ([a,b])
AREA	S	Real	Valor de la integral
FUN	E	Funcion	Funcion a integrar
ERROR	L	Constante	Criterio error relativo en la integral
ITERMAX	L	Constante	Numero maximo de iteraciones
I	L	Entero	Contador
N	L	Entero	Numero de divisiones del intervalo
SUMA	L	Real	Suma total de las evaluaciones de la funcion en los puntos de la particion (Multiplicados por 2 o 3 segun el punto)
SUMA1	L	Real	Suma de las evaluaciones de la funcion de los puntos nuevos
SUMA2	L	Real	Suma de las evaluaciones de la funcion, en la particion anterior

PUNTO (1..2)	L Real	X's en una division a ser evaluadas
INTERVALO	L Real	Tamano del intervalo [a,b]
DIVISION	L Real	Tamano de la n-esima parte en que se dividio el intervalo (h)
AREA0	L Real	Guarda el area evaluada con la particion anterior

Pseudocodiso:

```

BEGIN
N <--- 1
INTERVALO <--- B - A
SUMA <--- FUN (A) + FUN (B) (Inicializa SUMA asi puesto que estos
valores seran siempre parte de la particion)
Inicializa AREA y SUMA2
REPEAT
N <--- 3 * N (Incrementa division de particion en el valor optimo)
DIVISION <--- INTERVALO / N
SUMA1 <--- 0
AREA0 <--- AREA
FOR I <--- 1 TO N DO
  Calcula valor del primer punto (PUNTO1) en la division I
  Incrementa I
  Calcula valor del segundo punto (PUNTO2)
  Incrementa I
  Evalua la funcion en ambos puntos y lo agrega a SUMA1
SUMA <--- SUMA + (3 * SUMA1) - SUMA2
(AI restar SUMA2 y sumar SUMA (anterior) esto equivale a
sumar dos veces las funciones evaluadas en los puntos limite
de las divisiones)
AREA <--- (3/8) * PARTICION * SUMA
SUMA2 <--- SUMA1
UNTIL (El error relativo es menor que el deseado) OR
(Se lleo al maximo de iteraciones)
IF se alcanzo el maximo de iteraciones THEN
  Escribir mensaje
END

```

8.2.3) LISTADO DEL PROGRAMA

```
PROGRAM INTEGRACION (INPUT, OUTPUT)
```

```
VAR
```

```
AA, BB, AREAP : REAL;
```

```
FUNCTION FUNCION (X : REAL) : REAL;
```

```
CONST
```

```
K1 = 6.6710543E-10;
```

```
K2 = -4.4375127E-7;
```

```
VAR
```

```
XX : REAL;
```

```
BEGIN
```

```
XX := X * X;
```

```
FUNCION := K1 * XX * EXP (K2 * XX);
```

```
END (FUNCION);
```

```
PROCEDURE INTEGRAL (A, B : REAL; VAR AREA : REAL;
FUNCTION FUN (X : REAL) : REAL);
```

```
CONST
```

```
ERROR = 1E-9;
```

```
ITERAMAX = 5000000;
```

```
VAR
```

```
I, N : INTEGER;
```

```
SUMA : REAL;
```

```
SUMA1, SUMA2 : REAL;
```

```
PUNTO1, PUNTO2 : REAL;
```

```
INTERVALO, DIVISION : REAL;
```

```
AREA0 : REAL;
```

```
BEGIN
```

```
N := 1;
```

```
INTERVALO := B - A;
```

```
SUMA := FUN (A) + FUN (B);
```

```
AREA := 0;
```

```
SUMA2 := 0;
```

```
REPEAT
```

```
  BEGIN
```

```
    N := 3 * N;
```

```
    DIVISION := INTERVALO / N;
```

```
    SUMA1 := 0;
```

```
    AREA0 := AREA;
```

```
    FOR I := 0 TO N - 1 DO
```

```
      BEGIN
```

```
        I := I + 1;
```

```
        PUNTO1 := A + (I / N) * INTERVALO;
```

```
        I := I + 1;
```

```
        PUNTO2 := A + (I / N) * INTERVALO;
```

```
        SUMA1 := SUMA1 + FUN (PUNTO1) + FUN (PUNTO2);
```

```
      END (FOR);
```

```
    SUMA := SUMA + 3 * SUMA1 - SUMA2;
```

```
    AREA := (3 / 8) * DIVISION * SUMA1;
```

```
    SUMA2 := SUMA1;
```

```
  END (REPEAT);
```

```
UNTIL (ABS ((AREA - AREA0) / AREA) < ERROR)
```

```
  OR (N > ITERAMAX);
```

```
IF N > ITERAMAX THEN
```

```

WRITELN ('NUMERO MAXIMO DE ITERACIONES');
END <PROCEDURE>;

```

```

< PROGRAMA PRINCIPAL >

```

```

BEGIN
WRITELN ('LIMITES DEL INTERVALO (DE MENOR A MAYOR)');
READ (AA, BB);
WHILE NOT EOF DO
  BEGIN
    INTEGRAL (AA, BB, AREAP, FUNCION);
    WRITELN ('EL VALOR DE LA INTEGRAL ES ', AREAP);
    WRITELN ('LIMITES DEL INTERVALO (DE MENOR A MAYOR)');
    READ (AA, BB);
  END <WHILE>;
END.

```

8.2.4) EJEMPLO

La formula de distribucion de velocidades de las moleculas de un gas de Maxwell y Boltzmann da la fraccion de moleculas cuya rapidez se encuentra entre v y $v + dv$. La formula esta dada por:

$$f(v) = 4 \text{ PI } \left[\frac{m}{2\text{PI}kT} \right]^{3/2} \frac{1}{v} e^{[-(m/2kT)v^2]}$$

donde k es la constante de Boltzmann, m la masa de la molecula y T la temperatura en escala absoluta. Para obtener la fraccion de moleculas que se encuentran entre dos velocidades es necesario integrar la funcion con estas velocidades como limites. El problema es encontrar la fraccion de moleculas cuyas velocidades se encuentran entre 0 y 1000 metros/segundo, 1000 y 2000 m/s y 2000 y 1000 m/s de gas hidrogeno a 273 grados Kelvin (aproximadamente cero grados centisrados). Esta formula lleva a otra para la cual no existe primitiva. Por ello aplicamos el programa de integracion numerica. Reducimos la formula basandonos en los siguientes

valores:

$$P_i - P_{i+1} = 3.141592654$$

$$k - \text{Constante de Boltzmann} = 1.381 \times 10^{-23} \text{ Joules/grados}$$

$$m - \text{Masa de la molecula} = 3.346 \times 10^{-27} \text{ kilogramos}$$

quedando entonces

$$f(v) = (6.715430 \times 10^{-10})^2 \left[(-4.4375127 \times 10^{-7}) v^2 \right] v e^{-v^2}$$

CORRIDA

LIMITES DEL INTERVALO? (DE MENOR A MAYOR)
0 1000

EL VALOR DE LA INTEGRAL ES 0.171557364167

LIMITES DEL INTERVALO? (DE MENOR A MAYOR)
1000 2000

EL VALOR DE LA INTEGRAL ES 0.514108987794

LIMITES DEL INTERVALO? (DE MENOR A MAYOR)
2000 10000

EL VALOR DE LA INTEGRAL ES 0.314333664588

Por tanto las moleculas estan distribuidas aproximadamente segun

Intervalo de velocidad (m/s)	Porcentaje (%)
0 - 1000	17.16
1000 - 2000	51.14
2000 - 10000	31.43

8.3) COMPARACION DE METODOS

En este capítulo se consideraron básicamente las fórmulas de la regla del trapecio y las fórmulas de Simpson que son casos particulares de las fórmulas de Newton-Cotes. Estos métodos tienen la característica de que el analista tiene la libertad de seleccionar la magnitud del intervalo; de hecho, siempre se han escogido intervalos iguales, la técnica es pasar polinomios de algún orden a través de grupos de ordenadas.

Existen, sin embargo, métodos que optimizan el error de truncamiento dando la libertad de seleccionar la magnitud de los intervalos de integración y haciendo uso de los polinomios ortogonales. Dichos métodos se les conoce como cuadratura Gaussiana.

Todas las fórmulas de integración desarrolladas en secciones precedentes son de la forma:

$$\int_a^b f(x) dx \approx \sum_{i=0}^n w_i f(x_i)$$

donde los $n+1$ valores w_i son los pesos que se les da a los $n+1$ valores funcionales $f(x_i)$.

Se pueden desarrollar diferentes fórmulas de cuadratura Gaussiana haciendo uso de las propiedades de los polinomios ortogonales como los polinomios de Legendre, Laguerre, Chebyshev, Hermite. El desarrollo de dichas fórmulas es complicado, pero de una manera sencilla consisten en lo siguiente: se cambia el intervalo de integración por un cambio de variable apropiado, por ejemplo

$$\int_a^b f(x) dx = \int_{-1}^1 \phi(u) du = \sum_{i=0}^n w_i \phi(u_i)$$

donde los w_i son los pesos; con $n+1$ puntos se obtiene una fórmula exacta para un polinomio de grado $2n+1$. Los u_i resultan, en este caso, las raíces del polinomio de Legendre de grado n .

Resumiendo, la cuadratura Gaussiana de mayor precisión que las reglas de Simpson para el mismo número de ordenadas a expensas de una completa falta de control en la localización de los puntos. Como una última observación diremos que la cuadratura Gaussiana requiere aproximadamente la mitad del trabajo requerido por la regla de Simpson para obtener la misma precisión.

Finalmente, como un brevísimo cultural, diremos que son los polinomios ortogonales.

dos funciones $g_n(x)$ y $g_m(x)$ seleccionadas de una familia de funciones $g_k(x)$ se dice que son ortogonales con respecto a una función de peso $w(x)$ en el intervalo $[a,b]$.

$$\int_a^b w(x) g_n(x) g_m(x) dx = 0 \quad n \neq m$$

$$\int_a^b w(x) [g_n(x)]^2 dx = c(n) \neq 0$$

en general c depende de n . Si estas relaciones suceden para toda n entonces $\{g_n(x)\}$ constituye un conjunto de funciones ortogonales.

Si el conjunto $\{g_n(x)\}$ son polinomios ortogonales, se pueden encontrar fórmulas de cuadratura Gaussiana para obtener integrales como:

$$\int_{-1}^1 \frac{1}{\sqrt{1-x^2}} f(x) dx \approx \sum_{i=0}^n w_i f(x_i) \quad (\text{Chebyshev})$$

$$\int_0^{\infty} e^{-x} dx \approx \sum_{i=0}^n w_i F(x_i) \quad (\text{Laguerre}), \text{ etc.}$$

IX) SOLUCION DE ECUACIONES DIFERENCIALES ORDINARIAS

9.1) INTRODUCCION

Hay una gran variedad de aplicaciones en que aparecen las ecuaciones diferenciales en forma natural en areas Humanisticas como Cientificas, entre algunos de los problemas en que aparecen las ecuaciones diferenciales estan: la deteccion de falsificaciones de arte, diagnostico de la diabetes, diseminacion de la sonorra analisis de poblacion, sistemas dinamicos, etc.

Una ecuacion diferencial es una relacion entre una funcion del tiempo y sus derivadas. Las ecuaciones

$$i) \frac{dy}{dt} = 3y \sin(t+y) \quad \text{y} \quad ii) \frac{d^3 y}{dt^3} = e^{-y} + t + \frac{d^2 y}{dt^2}$$

Son, ambas, ejemplos de ecuaciones diferenciales. El orden de una ecuacion diferencial es el orden de la derivada mas alta de la funcion y que aparece en la ecuacion. Entonces (i) es una ecuacion diferencial de primer orden y (ii) es una ecuacion diferencial de tercer orden. Por solucion de una ecuacion diferencial entenderemos una funcion continua $y(t)$ tal que, Junto con sus derivadas, satisfacen la relacion. Por ejemplo la funcion:

$$y(t) = 2 \sin t - 1/3 \cos 2t$$

Es una solucion de la ecuacion diferencial de segundo orden

$$\frac{d^2 y}{dt^2} + y = \cos 2t$$

La forma mas general de una ecuacion diferencial ordinaria

de orden n es:

$$y^{(n)} = f(t, y, y', \dots, y^{(n-1)})$$

Donde:

$$y^{(n)} = \frac{d^n y}{dt^n}$$

Una ecuación diferencial del tipo anterior puede ser lineal o no lineal. Una ecuación diferencial ordinaria es lineal si puede ser expresada como combinación lineal de la variable dependiente y de todas sus derivadas; la no lineal es aquella que no cumple esta condición.

Para que la solución de una ecuación diferencial exista y sea única, se requieren especificar tantas condiciones iniciales o valores en la frontera como el orden de la ecuación diferencial. Las condiciones de existencia y unicidad se expresan en el teorema:

TEOREMA.-

Sean f y $\partial f / \partial y$ continuas en el rectángulo

$$R : t_0 \leq t \leq t_0 + a, \quad |y - y_0| \leq b$$

calculamos $M = \max_{(t,y) \text{ en } R} |f(t,y)|$,

y fijamos

$$\alpha = \min(a, b/M).$$

Entonces, el problema con valor inicial

$$y' = f(t,y), \quad y(t_0) = y_0$$

Tiene una solución única $y(t)$ en el intervalo

$$t_0 \leq t \leq t_0 + \alpha$$

Existe una gran variedad de ecuaciones diferenciales que no pueden ser resueltas analíticamente, de ahí la necesidad de contar con métodos de tipo numérico que ofrezcan un camino alternativo de solución.

El problema clásico del valor inicial es hallar una función $f(x)$ que satisfaga a la ecuación diferencial de primer orden $y' = f(t, y)$ y tome el valor inicial $y(x_0) = y_0$. Se ha diseñado una amplia variedad de métodos para la solución aproximada de este problema clásico, la mayor parte de los cuales han sido generalizados para tratar también problemas de orden superior. (dado que cualquier ecuación diferencial de orden n se puede descomponer en un sistema de n ecuaciones diferenciales de primer orden).

En este capítulo abordaremos la solución de ecuaciones diferenciales de primer orden con valor inicial usando los métodos: Euler, Euler mejorado (aquí se hace una introducción a los métodos predictor-corrector), Runse-Kutta, y método de Milne.

9.2) METODO DE EULER Y EULER MEJORADO

9.2.1) INTRODUCCION

En esta seccion abordaremos el metodo de Euler y Euler mejorado, dando una introduccion a metodos mas generales como lo son los metodos Predictor-Corrector, para el problema con valor inicial:

$$y' = f(t, y), \quad y(t_0) = y_0 \quad (1)$$

Una computadora obviamente no puede aproximar una funcion sobre un intervalo completo $t_0 \leq t \leq t_0 + a$ ya que requeriria una gran cantidad de informacion. A lo mas puede calcular valores aproximados y_1, y_2, \dots, y_n de $y(t)$ en un numero finito de puntos t_1, t_2, \dots, t_n . Sin embargo esto es suficiente para nuestro proposito puesto que podemos usar los numeros y_1, y_2, \dots, y_n para obtener una aproximacion sobre el intervalo $t_0 \leq t \leq t_0 + a$. Una forma en que se podria hacer esto se muestra en la figura 1:

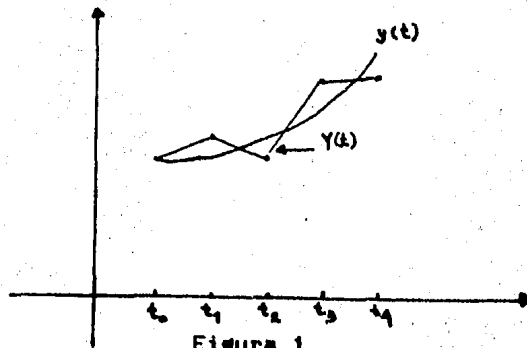


Figura 1

Es decir, uniendo los puntos (t_j, y_j) y (t_{j+1}, y_{j+1}) por una recta, de esta manera $Y(t) \approx y(t)$.

Para simplificar el trabajo pediremos que t_1, \dots, t_n estén igualmente espaciados. Ahora bien, lo único que conocemos acerca de $w(t)$ es que satisface una ecuación diferencial y que su valor en $t=t_0$ es w_0 , usando esta información para calcular un valor aproximado w_1 de $w(t_1)$, donde $t_1 = t_0 + h$ luego usaremos este valor aproximado de w_1 para calcular un valor aproximado de w_2 de $w(t_2)$, donde $t_2 = t_1 + h$, y así sucesivamente. Para lograr esto nos basamos en el teorema de Taylor que permite calcular el valor de $w(t)$ en $t_k + h$ a partir del conocimiento de $w(t)$ en $t = t_k$. El teorema establece que:

$$w(t_k + h) = w(t_k) + h \frac{dw(t_k)}{dt} + \frac{h^2}{2!} \frac{d^2 w(t_k)}{dt^2} + \dots \quad (2)$$

Por tanto si conocemos el valor de $w(t)$ en $t=t_k$ podemos calcular el valor de $w(t)$ en $t=t_k + h$. Ahora $w(t)$ es solución del problema con valor inicial (1), en consecuencia su primera derivada en $t=t_k$ será:

$$f(t_k, w(t_k)) \quad (3)$$

Por otro lado aplicando sucesivamente la regla de la cadena para la diferenciación parcial podemos evaluar:

$$\begin{aligned} \frac{d^2 w(t_k)}{dt^2} &= \left[\frac{\partial f}{\partial t} + f \frac{\partial f}{\partial w} \right] (t_k, w(t_k)) \\ &= \left[f_t + f f_w \right] (t_k, w(t_k)) \end{aligned} \quad (4)$$

Y todas las demas derivadas de orden superior de $y(t)$ en $t=t_k$.

De esta manera podemos reescribir la ecuacion (2)

$$y(t_{k+1}) = f(t_k) + hf(t_k, y(t_k)) + \frac{h^2}{2!} \left[f_t + ff_y \right] (t_k, y(t_k)) + \dots \quad (5)$$

La expresion mas simple de $y(t_{k+1})$ se obtiene eliminando de la serie (5) todos los terminos, excepto los dos primeros, esto da el siguiente metodo numerico:

$$y_1 = y_0 + hf(t_0, y_0) \quad , \quad y_2 = y_1 + hf(t_1, y_1)$$

y en general:

$$y_{k+1} = y_k + hf(t_k, y_k) \quad , \quad y_0 = y(t_0) \quad (6)$$

La ecuacion (6) es conocida como el metodo de Euler. El metodo es el mas simple y por supuesto es el menos preciso (el error que produce es del orden h^2). Sin embargo su interes es a nivel introductorio para metodos mas complicados.

EULER MODIFICADO

El metodo de Euler fue obtenido cortando la serie de Taylor (5) despues del segundo termino. El camino mas obvio para la obtencion de mejores metodos numericos, es la retencion de mas terminos en la ecuacion (5); no obstante esto tiene la serie desventajas de obligarnos a calcular las derivadas parciales de $f(t,y)$, y esto puede ser bastante dificil si la funcion $f(t,y)$ es muy complicada. Por esta razon se han deducido metodos numericos que no obligan a calcular las derivadas de $f(t,y)$.

Una aproximación a este problema consiste en integrar ambos lados de la ecuación diferencial $y' = f(t, y)$ entre t_k y t_{k+1} para obtener:

$$y(t_{k+1}) = y(t_k) + \int_{t_k}^{t_{k+1}} f(t, y(t)) dt$$

Esto reduce el problema de encontrar un valor aproximado de $y(t_{k+1})$ al problema de aproximar el área bajo la curva $f(t, y(t))$ entre t_k y t_{k+1} .

Una burda aproximación a esta área es $h f(t_k, y(t_k))$ que es el área del rectángulo en la figura 2(a).

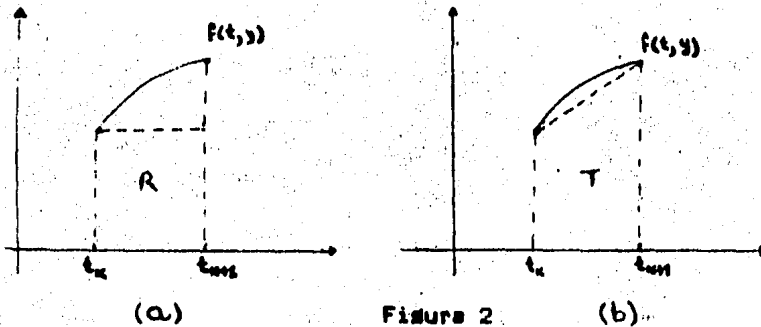


Figura 2

Que es por supuesto el método de Euler. Una mucho mejor aproximación a esta área es

$$\frac{h}{2} \left[f(t_k, y(t_k)) + f(t_{k+1}, y(t_{k+1})) \right]$$

que representa el área del trapecioide T de la figura 2(b).

Esto da origen al metodo numerico

$$y_{k+1} = y_k + \frac{h}{2} (f(t_k, y_k) + f(t_{k+1}, y_{k+1})) \quad (7)$$

Sin embargo no podemos usar este metodo para determinar y_{k+1} a partir de y_k , puesto que y_{k+1} tambien aparece del lado derecho de (7). Una forma muy inteligente de salvar esta dificultad es reemplazar y_{k+1} en la parte derecha de (7) por $y_k + hf(t_k, y_k)$ dada por el metodo de Euler; esto da origen al metodo numerico:

$$y_{k+1} = y_k + \frac{h}{2} (f(t_k, y_k) + f(t_{k+1}, y_k + hf(t_k, y_k))) \quad (8)$$

$$y = y(t)$$

0 0

La ecuacion (8) se conoce como metodo de Euler mejorado. Este metodo nos da la misma exactitud que el metodo de la serie de Taylor con 3 terminos sin tener que calcular las derivadas parciales.

La pareja:

$$(9) \begin{cases} y_{k+1} = y_k + hf(t_k, y_k) \\ y_{k+1} = y_k + \frac{h}{2} (f(t_k, y_k) + f(t_{k+1}, y_k + hf(t_k, y_k))) \end{cases}$$

Pertenece a los metodos del tipo Predictor-corrector que consisten en el uso de una formula para hacer una primera prediccion del siguiente valor w_k , seguida de la aplicacion de una formula correctora mas aproximada que proporciona entonces mejoramientos sucesivos. En la pareja mencionada anteriormente la formula de Euler es la predictora y el de Euler mejorada es el corrector. El predictor estima primero w_{k+1} , este estimativo conduce a un valor w'_{k+1} luego a uno w_{k+1} corregido. Se pueden hacer correcciones amplias de w'_{k+1} y w_{k+1} sucesivamente hasta que se obtenga un resultado satisfactorio empleando la pareja (9), es decir hasta que

$$\left| w_{k+1}^{\text{corr. J}} - w_{k+1}^{\text{corr. J-1}} \right| < \text{EPS}$$

9.2.2) PROGRAMA EULER

OBJETIVO:

Obtener la solucion de la ecuacion diferencial de primer orden de tipo $w' = f(t, w)$, $w(t_0) = w_0$ por el metodo de Euler y Euler mejorado, para una o varias funciones $f(t)$ con sus respectivas condiciones iniciales.

ALGORITMO EN PSEUDOCODIGO

PROGRAMA PRINCIPAL

VARIABLE	USO	TIPO	FUNCION
LIMITE	-	Constante	Dimension maxima para los vectores usada en el programa (es modificable).
EPB	-	Constante	Criterio de convergencia para el metodo de Euler mejorado.
X, Y	E/B	Arreglo	Vector de abscisas y vector solucion por Euler, inicialmente tienen los valores - iniciales en X[1], Y[1] resp.
Z	S	Arreglo	Vector que contiene la solucion por Euler mejorado.
NUMPUNTOS	S	Entero	Numero de puntos en que se subdivide el intervalo de integracion.
INTERVALO	E	Real	Espaciamento entre los puntos muestrales.
I	-	Entero	Variable usada como iterador.

Pseudocodigo:

BEGIN

 Dar valores iniciales (X0,Y0).

REPEAT

 Dar cantidad de puntos y espaciamento.

 CALL EULER (X,Y,NUMPUNTOS,INTERVALO,ECDIFER).

 Escribir letrero de salida.

 Escribir el vector de abscisas con la solucion por Euler y Euler mejorado.

 Dar otros valores iniciales.

UNTIL fin de archivo.

END.

SUBROUTINAS

1) FUNCTION ECDIFER (T,Y)

(Esta funcion tiene la parte derecha de la ecuacion diferencial que se desea evaluar; esta la introduce el usuario).

2) EULER (X,Y,Z,NUMPUNTOS,INTERVALO,DERIV)

Objetivo: obtiene la solucion numerica de la ecuacion diferencial con valor inicial $y' = f(t,y)$ $y(t_0)=y_0$ por los metodos de Euler y Euler mejorado (predictor-corrector).

VARIABLE	USO	TIPO	FUNCION
X	E	Arreglo	Vector conteniendo las abscisas

Y, Z	E/S	De Reales Arreglo De Reales	consideradas. En X[I] entra la condicion inicial X0. Vectores que contienen la solucion inicial y0, respectivamente; en y[I] entra la condicion inicial y0.
NUMPUNTOS	E	Entero	Cantidad de puntos en que se subdivide el intervalo de integracion.
INTERVALO	E	Real	Espaciamento entre los puntos muestrales.
ECDIFER	E	Funcion	Funcion que corresponde a la parte derecha de la ecuacion diferencial
L, I	L	Entero	Variables usadas para iteracion.
V, TEMP	L	Real	Variables temporales para la evaluacion del metodo de Euler mejorado.

Pseudocodiso:

```

BEGIN
ZCII <--- YCII,
FOR I := 2 TO NUMPUNTOS DO
  XCII <--- XCI-1] + INTERVALO.
  Calcula YCII por metodo de Euler.
  Calcula zCII por Euler modificado.
  Hace las correcciones para mejorar ZCII
  hasta que dos correcciones sucesivas
  sean menor que EPS.
END.

```

9.2.3) LISTADO DEL PROGRAMA

```

PROGRAM EULER (INPUT,OUTPUT);
CONST
  LIMITE = 100;
  EPS     = 0.001;
TYPE
  MATRIZ = ARRAY[1..LIMITE,1..6] OF REAL;
  VECTOR = ARRAY[1..LIMITE] OF REAL;
VAR
  X, Y, Z      : VECTOR;
  A            : MATRIZ;
  I, NUMPUNTOS : INTEGER;
  INTERVALO    : REAL;

FUNCTION ECDIFER (T,Y : REAL) : REAL;
BEGIN
  ECDIFER := -0.005 * Y - 0.005 * (100 * (T - 4));
END;

PROCEDURE EULER (VAR X, Y, Z : VECTOR; NUMPUNTOS : INTEGER;
  INTERVALO : REAL; FUNCTION DERIV (T,Y : REAL) : REAL);
VAR
  L, I : INTEGER;
  U, TEMD : REAL;
BEGIN
  Z[I] := Y[I];
  FOR I := 2 TO NUMPUNTOS DO
    BEGIN
      XCI := XCI - 1] + INTERVALO;
      YCI := YCI - 1] + INTERVALO * DERIV (XCI-1],YCI-1]);
      U   := ZCI - 1] + INTERVALO * DERIV (XCI-1],ZCI-1]);
      TEMD := DERIV (XCI-1],ZCI-1]);
      ZCI := ZCI - 1] + ((TEMD + DERIV (XCI,U)) * INTERVALO) / 2.0;
      L := 0;
      REPEAT
        U := ZCI;
        ZCI := ZCI - 1] + ((TEMD + DERIV (XCI,U)) * INTERVALO) / 2.0;
        L := L + 1;
      UNTIL (ABS(ABS(U) - ABS(ZCI)) <= EPS) OR (L = 50);
    END (*FOR*);
  END (* EULER *);

  ( PROGRAMA PRINCIPAL )

BEGIN
  WRITELN ('DAR VALORES INICIALES (X0,Y0)');
  READ (XCI,YCI);
  REPEAT
    WRITELN ('DAR CANTIDAD DE PUNTOS Y ESPACIAMIENTO');
    READ (NUMPUNTOS, INTERVALO);
    EULER (X,Y,Z,NUMPUNTOS,INTERVALO,ECDIFER);
    WRITE (' :5, 'X' Euler');
    WRITELN (' :10, 'EULER MEJORADO');
  FOR I := 1 TO NUMPUNTOS DO
    BEGIN

```



```

WRITE (XC1J : 10 : 5, ' ', YC1J : 10 : 5)
WRITELN (' :B,ZC1J : 10 : 5)
END!
WRITELN ('DAR VALORES INICIALES (X0,Y0)')
READ (XC1J,YC1J)
UNTIL EOF!
END.

```

9.2.4) EJEMPLO

Sea Y la temperatura en el tiempo t de un cuerpo sumergido en un medio donde la temperatura se describe mediante la expresion:

$$G(t) = -100(t - 4)$$

La temperatura satisface la ecuacion diferencial

$$dY/dt + kY = kG(t),$$

Donde k es una constante proporcionalmente igual a 0.005

y la condicion inicial es

$$t = 0, \quad Y = 100 \text{ grados F.}$$

Obtener la relacion $Y-t$ por el metodo de Euler para el intervalo $(0,2.5)$

La solucion de este problema, introduciendo los datos al correrse el Programa es:

DAR VALORES INICIALES (X₀,Y₀)

0 1000

DAR CANTIDAD DE PUNTOS Y ESPACIAMIENTO

100 0.025

X	EULER	EULER MEJORADO
0.00000	1000.00000	1000.00000
0.02500	999.92500	999.92485
0.05000	999.84970	999.84939
0.07500	999.77409	999.77364
0.10000	999.69818	999.69758
0.12500	999.62197	999.62121
0.15000	999.54545	999.54455
0.17500	999.46864	999.46758
0.20000	999.39151	999.39030
0.22500	999.31409	999.31273
0.25000	999.23636	999.23485
0.27500	999.15833	999.15667
0.30000	999.08000	999.07819
.	.	.
2.10000	992.64691	992.63431
2.12500	992.54658	992.53383
2.15000	992.44595	992.43305
2.17500	992.34502	992.33197
2.20000	992.24378	992.23059
2.22500	992.14225	992.12891
2.25000	992.04042	992.02693
2.27500	991.93829	991.92466
2.30000	991.83586	991.82208
2.32500	991.73313	991.71920
2.35000	991.63011	991.61602
2.37500	991.52678	991.51255
2.40000	991.42315	991.40877
2.42500	991.31922	991.30469
2.45000	991.21499	991.20032
2.47500	991.11047	991.09564

9.3) METODO DE RUNGE-KUTTA.

9.3.1) INTRODUCCION

Este metodo es uno de los mas frecuentemente usados debido a que introduce gran facilidad en la programacion.

Las tres propiedades que caracterizan a este metodo son las siguientes:

1) Para obtener el punto y_{m+1} solo se utiliza la informacion suministrada por el punto anterior (x_m, y_m) .

2) El metodo usado puede ser de varios ordenes. Al metodo usado se le llama de orden p , donde p es el grado del termino de la serie de Taylor alcanzado por el metodo.

3) Solo requiere la evaluacion de la funcion $f(x, y)$ y no de sus derivadas, aunque la funcion f se evalua para mas de un punto en cada iteracion.

De la segunda propiedad se concluye que el metodo de Euler es en realidad un metodo de Runge-Kutta de orden 1. La formula para este metodo que ya se ha encontrado en la seccion 9.2 es:

$$y_{n+1} = y_n + k_1 \Delta t$$

donde $k_1 = y'_n = f(t_n, y_n)$. Asi, en el metodo de Euler la solucion aproximada y_{n+1} depende de y_n y de la pendiente k_1 en el punto (t_n, y_n) . De la misma forma, se puede ver que en el metodo de Euler modificado, que se puede considerar como un metodo de

Runse-Kutta de segundo orden, la solución y_{n+1} depende de y_n y de las pendientes k_1 y k_2 donde k_2 es una pendiente en un punto distinto de (t_n, y_n) .

El método que trataremos aquí es el de cuarto orden, en el cual la solución y_{n+1} dependerá de tres pendientes, k_2, k_3 y k_4 , además de k_1 , y por supuesto de y_n . Este método es el más usado para la solución de ecuaciones diferenciales con valores iniciales y es conocido en la literatura simplemente como "método de Runse-Kutta" sin hacer ninguna referencia al orden.

El método de Runse-Kutta se basa en la fórmula:

$$y_{i+1} = y_i + a_1 K_{11} + a_2 K_{22} + \dots + a_n K_{nn} \quad (1)$$

con:

$$K_1 = f(t_i, y_i) \Delta t$$

$$K_2 = f(t_i + p_1 \Delta t, y_i + a_{11} K_1) \Delta t$$

$$K_3 = f(t_i + p_2 \Delta t, y_i + a_{21} K_1 + a_{22} K_2) \Delta t$$

⋮

$$K_n = f(t_i + p_{n-1} \Delta t, y_i + a_{n-1,1} K_1 + a_{n-1,2} K_2 + \dots + a_{n-1,n-1} K_{n-1}) \Delta t$$

Y tomando $n=4$ en las ecuaciones anteriores se obtiene la fórmula de Runse-Kutta de orden 4, es decir:

$$y_{i+1} = y_i + a_1 K_1 + a_2 K_2 + a_n K_n \quad (2)$$

con:

$$K_1 = f(t_i, y_i) \Delta t$$

$$K_2 = f(t_i + p \Delta t, y_i + a K_1) \Delta t$$

$$K_3 = f(t_i + p \Delta t, y_i + a K_1 + a K_2) \Delta t$$

$$K_4 = f(t_i + p \Delta t, y_i + a K_1 + a K_2 + a K_3) \Delta t$$

Los valores de las constantes a , p y a se obtienen de igualar la ecuación (1) con los términos hasta de orden 4 del desarrollo por serie de Taylor de la variable y_{i+1} . Dicho desarrollo es:

$$y_{i+1} = y_i + y'_i(\Delta t) + \frac{y''_i(\Delta t)^2}{2!} + \frac{y'''_i(\Delta t)^3}{3!} + \frac{y''''_i(\Delta t)^4}{4!} \quad (3)$$

Como podrá notarse, en la ecuación (2) no aparece ninguna de las derivadas de la función f ; únicamente es necesario evaluar dicha función en cuatro puntos distintos para obtener K_1 , K_2 , K_3 y K_4 . Resolviendo el sistema de ecuaciones formado al igualar las ecuaciones (2) y (3), se obtiene un sistema de 11 ecuaciones con 13 incógnitas (es decir las 13 constantes buscadas) por lo que se tendrán que asignar valores arbitrarios a dos de estas incógnitas para convertir el sistema anterior en un sistema de 11 ecuaciones con 11 incógnitas, dependiendo de los valores que se

le asignen a esas incógnitas se pueden obtener tres variantes del método de Runse-Kutta de cuarto orden. El algoritmo que se desarrollara en esta sección está basado en el método de Runse-Kutta usando coeficientes de Runse, que consiste en asignar a las incógnitas a_3 y a_2 el valor de $1/3$ y resolviendo el sistema de 11 ecuaciones con 11 incógnitas así formado, obteniendo los siguientes valores para las 11 constantes restantes.

$$a_1 = 1/6$$

$$a_4 = 1/6$$

$$p_1 = 1/2$$

$$p_2 = 1/2$$

$$p_3 = 1$$

$$a_{11} = 1/2$$

$$a_{21} = 0$$

$$a_{31} = 0$$

$$a_{22} = 0$$

$$a_{32} = 0$$

$$a_{33} = 1$$

Sustituyendo estos valores en la ecuación (2) se obtiene la fórmula de Runse-Kutta con coeficientes de Runse:

$$y_{i+1} = y_i + \frac{\Delta t}{6} (K_1 + 2K_2 + 2K_3 + K_4) \quad (4)$$

con:

$$K_1 = f(t_i, y_i)$$

$$K_2 = f\left(t_i + \frac{\Delta t}{2}, y_i + \frac{K_1}{2}\right)$$

$$K_3 = f\left(t_i + \frac{\Delta t}{2}, y_i + \frac{K_2}{2}\right)$$

$$K_4 = f(t_i + \Delta t, y_i + K_3)$$

Los valores K_1 , K_2 , K_3 y K_4 se interpretan geométricamente como las pendientes de f en varios puntos. El valor K_1 es la pendiente en el punto (t_i, y_i) ; K_2 y K_3 son dos pendientes consideradas en el punto medio entre t_i y t_{i+1} , pero con distintas ordenadas cada una; finalmente, K_4 es la pendiente en el punto $t_{i+1} = t_i + \Delta t$ y cuya ordenada es $y_i + K_3 \Delta t$.

9.3.2) PROGRAMA RUNGE-KUTTA.

OBJETIVO:

Resolver una ecuación diferencial de primer orden del tipo $y' = f(t, y)$.

DESCRIPCION:

Este programa tiene como entrada el número de puntos en que se subdivide el intervalo, el espaciamiento entre las abscisas y los valores iniciales (tanto de la variable independiente como de

la dependiente). Como salida, dara la solucion de la ecuacion diferencial (en el intervalo indicado) en forma grafica y tabular.

Se utilizaran 2 subrutinas, una de ellas, la subrutina RUNKUT, calculara los valores de la solucion de la ecuacion diferencial en el intervalo indicado; la otra es la subrutina GRAFICA que pondra en forma grafica los resultados (esta subrutina no ha sido desarrollada en el capitulo I).

Se declaran en el programa 2 arreglos, X y Y de numeros reales que guardan las abscisas y las ordenadas de la solucion, respectivamente. Si el numero de puntos en que se subdivide el intervalo es mayor que 101, se debere aumentar en el programa el valor de la constante MAXPUNTOS.

ALGORITMO EN PSEUDOCODIGO:

PROGRAMA PRINCIPAL.

VARIABLES	USO	TIPO	FUNCION
MAXORD	-	Const.	Maximo numero de funciones a graficar.
CAMPO	-	Const.	Maximo numero de puntos a graficar.
NUMPUNTOS	E	Entero	Numero de puntos en que se subdivide el intervalo.
NUMFUNC	-	Entero	Numero de funciones a graficar por la subrutina GRAFICA (en este programa vale 1).
NUMABS	-	Entero	Numero de puntos a graficar por la subrutina GRAFICA.
INC	E	Real	Incremento (Espaciamiento entre las abscisas).
X	S	Arreglo de Reales	Vector que guarda las abscisas de los puntos solucion.
Y	S	Arreglo de Reales	Vector que guarda las ordenadas de los puntos solucion.
A	-	Arreglo de Reales	Matriz para graficar.

Pseudocódigo:

BEGIN

Lee NUMPUNTOS, INC, XE1J, YE1J.
 CALL RUNKUT (X, Y, NUMPUNTOS, INC, ECDIF).
 Imprime espaciamento y valores iniciales.
 Imprime la solución en forma de tabla.
 Se llena la matriz para graficar A con la solución de la
 ecuación diferencial obtenida por RUNKUT.
 NUMFUNC \leftarrow 1.
 NUMABS \leftarrow NUMPUNTOS.
 CALL GRAFICA (A, NUMABS, NUMFUNC).

END.

SUBROUTINAS:

1) FUNCTION ECDIF (T,Y)

(Contiene la parte derecha de la ecuación diferencial).

2) GRAFICA (A, NUMABS, NUMFUNC)

OBJETIVO: Graficar los puntos contenidos en el arreglo A.

(Ver capítulo 1)

3) RUNKUT (X, Y, NUMPUNTOS, INC, ECDIF)

OBJETIVO: Obtiene la solución numérica de la ecuación

diferencial $w' = f(t, w)$ con valor inicial $w(t_0) = w_0$
 por el método de Runge-Kutta.

VARIABLES	USO	TIPO	FUNCION
X	E	Arreglo de Reales	Vector conteniendo las abscisas consideradas. En XE1J entra la condición inicial x_0 .
Y	E/S	Arreglo de Reales	Vector que contiene la solución; en YE1J entra la condición inicial w_0 .
NUMPUNTOS	E	Entero	Número de puntos en que se subdivide el intervalo considerado.
INC	E	Real	Espaciamento entre las abscisas.
F	E	Funcion de Reales	Funcion que corresponde a la parte derecha de la ecuación diferencial.
K1,K2,K3,K4	L	Real	Parámetros usados por el método.
TI, YI	L	Real	Variables temporales.
I	G	Entero	Contador.

Pseudocódigo:

BEGIN

FOR I = 2 TO NUMPUNTOS DO

 XCIJ <--- XCI-1J + INC.

 Calcula YCIJ por el método de Runse-Kutta.

END.

9.3.3) LISTADO DEL PROGRAMA.

```

PROGRAM RUNGEKUTTA4 (INPUT, OUTPUT);
CONST
  MAXORD = 6;
  CAMPO = 101;
TYPE
  VECTORES = ARRAY [1..CAMPO] OF REAL;
  MATRICES = ARRAY [1..CAMPO, 1..6] OF REAL;
VAR
  NUMPUNTOS, I      : INTEGER;
  NUMFUNC           : INTEGER;
  NUMABS           : INTEGER;
  INC              : REAL;
  X, Y             : VECTORES;
  A               : MATRICES;

FUNCTION ECDIF (T, Y : REAL) : REAL;
BEGIN
  ECDIF := 10 * EXP(-2 * T) - 10 * Y;
END;

PROCEDURE RUNKUT (VAR X, Y : VECTORES; NUMPUNTOS, : INTEGER;
                 INC, : REAL;
                 FUNCTION F (T, Y : REAL) : REAL);
VAR
  K1, K2, K3, K4, TI, YI : REAL;
BEGIN
  FOR I := 2 TO NUMPUNTOS DO
    BEGIN
      XCIJ := XCI-1J + INC;
      TI := XCI-1J;
      YI := YCI-1J;
      K1 := F (TI, YI);
      TI := XCI-1J + INC/2;
      YI := YCI-1J + (INC * K1)/2;
      K2 := F (TI, YI);
      YI := YCI-1J + (INC * K2)/2;
      K3 := F (TI, YI);
      TI := XCI-1J + INC;
      YI := YCI-1J + (INC * K3);
      K4 := F (TI, YI);
      YCIJ := YCI-1J + (INC * (K1 + (2 * K2) + (2 * K3) + K4)) /
    END;
  END ( RUNKUT );

```

PROGRAMA PRINCIPAL

```

BEGIN
  WRITELN ('NUM. DE PUNTOS?, ESPACIAMIENTO?, XO ?, YO ?');
  READ(NUMPUNTOS, INC, XCIJ, YCIJ);
  RUNKUT (X, Y, NUMPUNTOS, INC, ECDIF);
  WRITELN ('ESPACIAMIENTO = ', INC);
  WRITELN;
  WRITELN ('XO = ', XCIJ;8:4, ' ', ':6, 'YO = ', YCIJ;8:4);

```

```

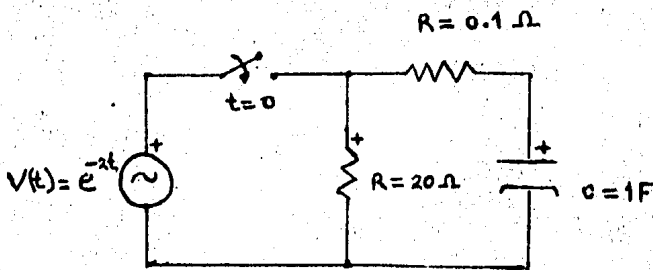
WRITELN; WRITELN;
WRITELN (' '13, 'X', ' '16, 'Y');
WRITELN;
FOR I := 1 TO NUMPUNTOS DO
  WRITELN (XCII:6:3, ' '18, YCII:10:6);
WRITELN;
{SE LLENA EL ARREGLO A PARA GRAFICAR}
FOR I := 1 TO NUMPUNTOS DO
  BEGIN
    AII,1J := XCII;
    AII,2J := YCII;
  END;
NUMFUNC := 1;
NUMABS := NUMPUNTOS;
GRAFICA (A, NUMABS, NUMFUNC);
END.

```

9.3.4) EJEMPLO:

La ecuación que caracteriza el voltaje del capacitor del circuito eléctrico mostrado en la figura es:

$$\frac{d V_c}{d t} = 10e^{-2t} - 10V_c$$



Los datos de entrada para este problema serian:

NUMPUNTOS = 50

INC = 0.02

X[1] = 0.0

Y[1] = 0.2

En la siguiente pagina se muestra como quedaria la ejecucion del programa RUNGEKUTTA4 para este problema.

RUN

#RUNNING 4400

NUM. DE PUNTOS?, ESPACIAMIENTO?, XO ?, YO ?

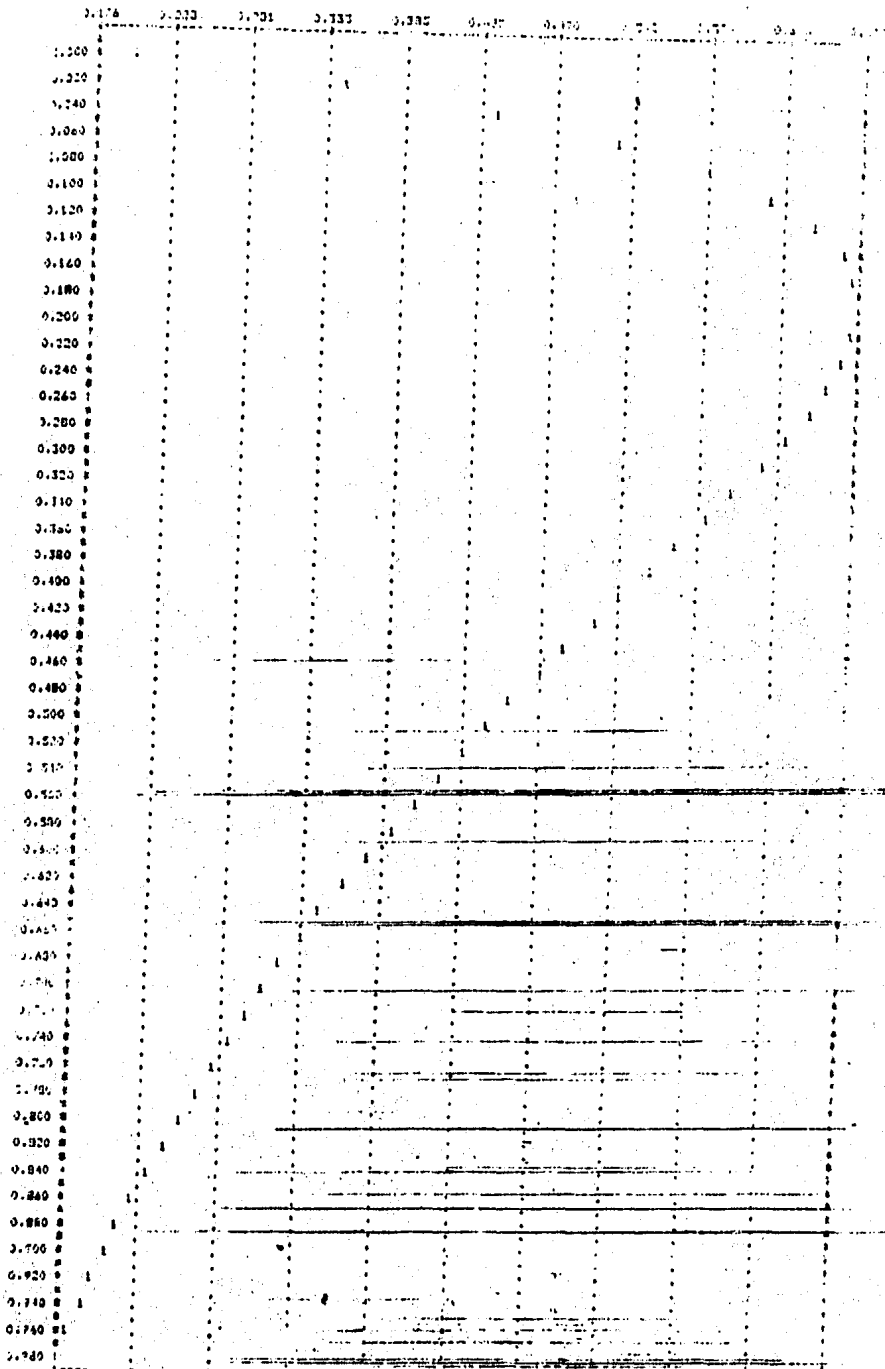
#?

50 0.02 0.0 0.2

ESPACIAMIENTO = 0.02

XO = 0.0000 YO = 0.2000

X	Y
0.000	0.200000
0.020	-0.314272
0.040	-1.043067
0.060	-0.770835
0.080	0.196442
0.100	0.493535
0.120	-0.256287
0.140	-0.874377
0.160	-0.389468
0.180	0.509705
0.200	0.507274
0.220	-0.368045
0.240	-0.784951
0.260	-0.102326
0.280	0.671373
0.300	0.388814
0.320	-0.517651
0.340	-0.683057
0.360	0.146472
0.380	0.733796
0.400	0.203553
0.420	-0.646835
0.440	-0.541149
0.460	0.364622
0.480	0.715967
0.500	-0.011722
0.520	-0.727496
0.540	-0.358476
0.560	0.543575
0.580	0.628538
0.600	-0.230256
0.620	-0.746321
0.640	-0.146484
0.660	0.671319
0.680	0.482696
0.700	-0.429687
0.720	-0.699386
0.740	0.077607
0.760	0.737873
0.780	0.292770
0.800	-0.591093
0.820	-0.590087
0.840	0.294333
0.860	0.737807
0.880	0.076275
0.900	-0.699603
0.920	-0.427943
0.940	0.484469
0.960	0.671329
0.980	0.147130



9.4) METODO DE MILNE

9.4.1) INTRODUCCION

El metodo de Milne pertenece a los metodos del tipo predictor corrector y tiene la ventaja de ser mas rapido que el metodo de Runge-Kutta. El metodo consiste en lo siguiente: Se subdivide cada subintervalo de integracion en cinco puntos igualmente espaciados y se aproxima la curva (grafica de la ecuacion diferencial) mediante una parabola de segundo grado que pasa por tres de esos puntos. El area en cada subintervalo se aproxima por el area debajo de la parabola; el area total es igual a la suma de las areas de cada subintervalo; ver figura 1.

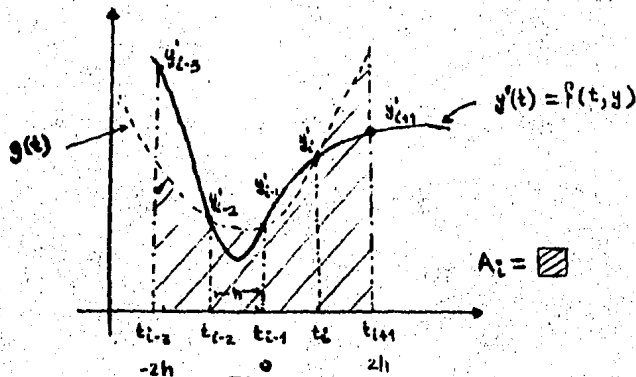


Figura 1

$$\text{Por tanto } u'(t) = f(t, u(t)) \approx m(t) \quad (1)$$

$$A_i = \int_{t_{i-3}}^{t_{i+1}} m(t) dt \quad (2)$$

$$m \text{ es de la forma } m(t) = at^2 + bt + c \quad (3)$$

de esta forma, sustituyendo la ecuación (3) en (2):

$$A_i = \int_{i-3}^{i+1} (at^2 + bt + c) dt \quad (4)$$

haciendo el siguiente cambio en los límites de integración:

$$t_{i+1} = 2h$$

$$t_{i-3} = -2h$$

obtenemos: $A_i = \frac{16}{3} ah^3 + 4ch$ (6)

Para evaluar a , b , c pedimos que la curva (3) pase por los puntos $(i-2, w'_{i-2})$, $(i-1, w'_{i-1})$, (i, w'_i) de lo que obtenemos:

$$a = \frac{w'_{i-2} - 2w'_{i-1} + w'_i}{2h^2} \quad (7)$$

$$c = w'_{i-1}$$

sustituyendo (7) en (6):

$$A_i = \frac{4}{3} h (2w'_{i-2} - w'_{i-1} + 2w'_i) \quad (8)$$

de la gráfica se observa que:

$$w_{i+1} = w_{i-3} + A_i \quad (9)$$

$$u_{i+1} = u_{i-3} + \frac{4}{3} h(2u'_{i-2} - u'_{i-1} + 2u'_i) \quad (10)$$

La ecuación (10) da la primera aproximación de la solución por método de Milne. El valor obtenido se mejora empleando un procedimiento similar al método de Euler mejorado; en este caso se emplea la fórmula de Simpson 1/3 para efectuar la corrección utilizando los puntos t_{i-1} , t_i , t_{i+1} ; el proceso se repite para cada u_i hasta que dos correcciones sucesivas de la variable dependiente sean aproximadamente iguales; los pasos a seguir para cada u_i se resumen en los puntos:

$$1) \text{ pred. } u_{i+1} = u_{i-3} + \frac{4}{3} h(2u'_{i-2} - u'_{i-1} + 2u'_i)$$

$$2) \text{ pred. } u'_{i+1} = f(t_{i+1}, u_{i+1}^{\text{pred.}})$$

$$3) \text{ corr.1 } u_{i+1} = u_{i-1} + \frac{h}{3} (u'_{i-1} + 4u'_i + u'_{i+1}^{\text{pred.}})$$

$$4) \text{ corr.1 } u'_{i+1} = f(t_{i+1}, u_{i+1}^{\text{corr.1}})$$

5) seguir con (3) y (4) hasta que

$$| u'_{i+1}^{\text{corr.J}} - u'_{i+1}^{\text{corr.(J-1)}} | < \text{EPS}$$

El error producido por el método es del orden de h^5 , igual que el producido por el método de Runge-Kutta, pero con la ventaja de ser más rápido. Su desventaja, como se ve en el paso (1), es que para arrancar requiere que se conozcan los valores u_0 , u'_1

, w'_1 , w'_3 . Estos valores se pueden obtener mediante expansion en series de Taylor o empleando el metodo de Runge-Kutta (Ara arrancar) esto ultimo es lo que se hace en este programa.

9.3.2) PROGRAMA MILNE

OBJETIVO:

Obtener la solucion de la ecuacion diferencial $w' = f(t,w)$, $w_0 = w(t_0)$, por el metodo de Milne, para una o varias funciones, f , definidas en la FUNCTION ECDIFER.

DESCRIPCION

La solucion que da este programa es en, a lo mas, 100 puntos. En caso de querer una solucion en un mayor numero de puntos modificar la constante LIMITE; si se quiere otro grado de convergencia modificar la constante EPS. La ecuacion diferencial a resolverse se dedera definir en la FUNCTION ECDIFER. La informacion que requiere el programa es: condiciones iniciales, cantidad de puntos en los que se quiere la solucion y el espaciamiento entre ellos; estos datos se alimentan a la subrutina MILNES que da la solucion por el metodo de Milne.

ALGORITMO EN PSEUDOCODIGO:

PROGRAMA PRINCIPAL

VARIABLE	USO	TIPO	FUNCION
LIMITE	-	Constante	Dimension maxima para los vectores usados en el programa (es modifiable).
EPS	-	Real	criterio de convergencia.
X	E	Arreglo de reales	Vector de abscisas; en X[1] se da la condicion inicial (X_0).
MILNE	S	Arreglo de reales	Vector solucion de la ecuacion diferencial; en MILNE[1] se da la condicion inicial (w_0).

I, NUMPUNTOS - Entero Iterador y numero de puntos en que se subdivide el intervalo de integracion.
 INTERVALO - Real Espaciamiento entre los puntos muestrales.

Pseudocodiso:

```
BEGIN
Dar valores iniciales (Xo, Yo).
REPEAT
  Dar cantidad de puntos y espaciamiento.
  CALL MILNES (X, MILNE, NUMPUNTOS, INTERVALO, ECDIFER).
  Escribir el vector de abscisas con su correspondiente
  solucion de la ec. diferencial, por MILNE.
  Dar valores iniciales para otro intervalo o prob.
UNTIL fin de archivo.
END.
```

SUBROUTINAS

1) FUNCTION ECDIFER (T, Y)

(Aqui se da la ecuacion diferencial a resolver).

2) MILNES (X, MILNE, NUMPUNTOS, INTERVALO, DERIV)

Objetivo: Obtiene la solucion de la ecuacion diferencial $y' = f(t, y)$, $y(t_0) = y_0$, de la funcion DERIV, es decir, aqui $f = \text{DERIV}$.

VARIABLE	USO	TIPO	FUNCION
X	E	Arreglo de reales	Vector de abscisas, en X[i] se da la condicion inicial (Xo).
MILNE	B	Arreglo de reales	Vector solucion de la ecuacion diferencial.
NUMPUNTOS	E	Entero	Numero de puntos en que se subdivide el intervalo de integracion.
INTERVALO	E	Real	Espaciamiento entre los puntos.
DERIV	E	FUNCTION	Funcion en la que se define la parte derecha de la ecuacion dif.
I, J	L	Entero	Variables usadas como iteraciones.

Pseudocodiso:

```
BEGIN
Obtiene los 4 primeros puntos iniciales por el metodo de Runge-Kutta.
FOR I := 5 TO NUMPUNTOS DO
  X[I] <--- X[I-1] + INTERVALO.
  Obtiene la solucion predictora para MILNE[I].
  Aplica la pareja predictora-correctora hasta que 2 aproximaciones sucesivas son menores
```

que EPS, en a lo mas 10 iteraciones.
En MILNECII queda finalmente la solucion
para XCII.

END.

9.4.3) LISTADO DEL PROGRAMA

```

PROGRAM MILNE (INPUT,OUTPUT)
CONST
  LIMITE = 100;
  EPS    = 0.001;
TYPE
  MATRIZ = ARRAY[1..LIMITE,1..6] OF REAL;
  VECTOR = ARRAY[1..LIMITE] OF REAL;
VAR
  X, MILNE      : VECTOR;
  A             : MATRIZ;
  I, NUMPUNTOS : INTEGER;
  INTERVALO    : REAL;

FUNCTION ECDIFER (T,Y : REAL) : REAL;
BEGIN
  ECDIFER :=
END;

PROCEDURE MILNES (VAR X, MILNE : VECTOR; NUMPUNTOS : INTEGER;
  INTERVALO : REAL; FUNCTION DERIV (T,Y : REAL) : REAL);
VAR
  I,J           : INTEGER;
  PRUE, C, D   : REAL;
  K1, K2, K3, K4 : REAL;
  G1, G2, G3   : REAL;
  PRE, DIF, CORR : VECTOR;
BEGIN
  FOR I := 2 TO 4 DO
    BEGIN
      X[I] := X[I-1] + INTERVALO;
      K1 := DERIV (X[I-1],MILNE[I-1]);
      C := X[I-1] + 0.5 * INTERVALO;
      D := MILNE[I-1] + 0.5 * INTERVALO * K1;
      K2 := DERIV (C,D);
      D := MILNE[I-1] + 0.5 * INTERVALO * K2;
      K3 := DERIV (C,D);
      C := X[I-1] + INTERVALO;
      D := MILNE[I-1] + INTERVALO * K3;
      K4 := DERIV (C,D);
      MILNE[I] := MILNE[I-1] + (INTERVALO * (K1 + 2.0 * K2 +
        2.0 * K3 + K4)) / 6.0;
    END(* FOR *)
  FOR I := 5 TO NUMPUNTOS DO
    BEGIN
      X[I] := X[I-1] + INTERVALO;
      G1 := DERIV (X[I-1],MILNE[I-1]);
      G2 := DERIV (X[I-2],MILNE[I-2]);
      G3 := DERIV (X[I-3],MILNE[I-3]);
      PRE[I] := MILNE[I-4] + (4.0 * INTERVALO * (2.0 * G1 - G2 +
        2.0 * G3)) / 3.0;
      C := X[I];
      D := PRE[I];
      DIF[I] := DERIV (X[I],PRE[I]);
      CORR[I] := MILNE[I-2] + (INTERVALO * (G2 + 4.0 * G1 +

```

```

                                DIF(I)) / 3.0)
J := 0)
REPEAT
  PRUE := CORR(I);
  DIF(I) := DERIV (C,CORR(I));
  CORR(I) := MILNE(I-2) + (INTERVALO * (G2 + 4.0 * G1 +
                                DIF(I)) / 3.0)
  J := J + 1)
UNTIL (ABS(ABS(PRUE) - ABS(CORR(I))) <= EPS) OR (J = 10))
MILNE(I) := CORR(I)
END(* FOR *)
END(* MILNE*)

```

(PROGRAMA PRINCIPAL)

```

BEGIN
WRITELN ('DAR VALORES INICIALES (X0,Y0)');
READ (XC(I),MILNE(I));
REPEAT
  WRITELN ('DAR CANTIDAD DE PUNTOS Y ESPACIAMIENTO');
  READ (NUNPUNTOS, INTERVALO);
  MILNE(X,MILNE,NUNPUNTOS,INTERVALO,ECDIFER);
  WRITELN (' :5, 'X          MILNE')
  FOR I := 1 TO NUNPUNTOS DO
    WRITELN (XC(I) : 10 : 5, '          ',MILNE(I) : 10 : 5);
  WRITELN ('DAR VALORES INICIALES (X0,Y0)');
  READ (XC(I),MILNE(I));
UNTIL EOF;
END.

```

9.5) COMPARACION DE METODOS

A lo largo de este capítulo se vieron básicamente 2 tipos de técnicas para resolver la ecuación diferencial con valor inicial $y' = f(t, y)$, $y(t_0) = y_0$; dichas técnicas son los métodos de un paso, como los métodos de Runge-Kutta y los métodos de pasos múltiples, como los métodos predictor corrector. A continuación resumimos sus características.

A) Métodos de Runge-Kutta.

- Arrancan por sí solos ya que no utilizan información de puntos previamente calculados.
- Debido a lo anterior requieren varias evaluaciones de la función $f(t, y)$, y esto consume tiempo.
- Permiten cambiar fácilmente el tamaño del intervalo gracias a que arrancan por sí solos.
- Coinciden con la serie de Taylor hasta los términos de orden K y se evitan el trabajo de evaluar las derivadas.
- No proporcionan un estimativo para el error.

B) Métodos de predictor corrector. Sus características son complementarias a las de los métodos de Runge-Kutta.

- No arrancan por sí solos ya que utilizan información de puntos previos.
- Sustituyen la información referente a puntos previos por las valuaciones repetidas de $f(t, y)$ y por tanto son más eficientes.

cientes.

- Proporcionan una buena estimacion del error.

En la practica lo que se debe hacer es combinar adecuadamente estas dos tecnicas, para producir algoritmos mas eficientes, dado que son complementarias.

BIBLIOGRAFIA

- KUD S. SHAN, "Computer Applications of Numerical Methods", Reading Mass.: Addison Wesley Co., 1972.
- CARNAHAN B., LUTHER H., WILKES J., "Applied Numerical Methods", New York: John Wiley and Sons Inc., 1969.
- HAMMING RICHARD, "Numerical Methods for Scientist and Engineers", New York: Mc. Graw Hill Book Co. 1962.
- JOSE ARMANDO TORRES F., "Proqramoteca para la solucion de problemas de Ingenieria mecanica electrica mediante el empleo de computadora digital", tesis profesional, Facultad de Ingenieria.
- FORSYTHE GEORGE E., MALCOM MICHAEL A., MOLER CLAVE B. "Computer Methods for Mathematical Computations".
- D.D. Mc CRACKEN, W.S. DORN, "Metodos numericos y programacion Fortran con aplicaciones en Insienieria y Ciencias". Editorial Limusa. Mexico 1980.
- N. YA VILENKIN, "Metodo de aproximaciones sucesivas". Lecciones populares de matematicas, editorial MIR-Moscu 1978.
- SERGE LANG, "Algebra Lineal".
- MARTIN BRAUN, "Ecuaciones Diferenciales".
"Solucion de ecuaciones diferenciales mediante computadora digital", Tesis Profesional, Facultad de Ciencias.
- MURRAY R. SPIEGEL, "Estadistica". Schaum.
- FRANCIS SCHEID, "Analisis Numerico". Schaum .

- BRUCE W. ARDEN, KENNETH N. ASTILL, "Numerical algorithms: origins and applications."

- SCHNEIDER, WEINGART PERLMAN, "An introduction to programming and problem solving with Pascal".

- JEAN PAUL TREMBLAY, RICHARD B. BUNT, LYLE M. OPSETH, "Structured Pascal".