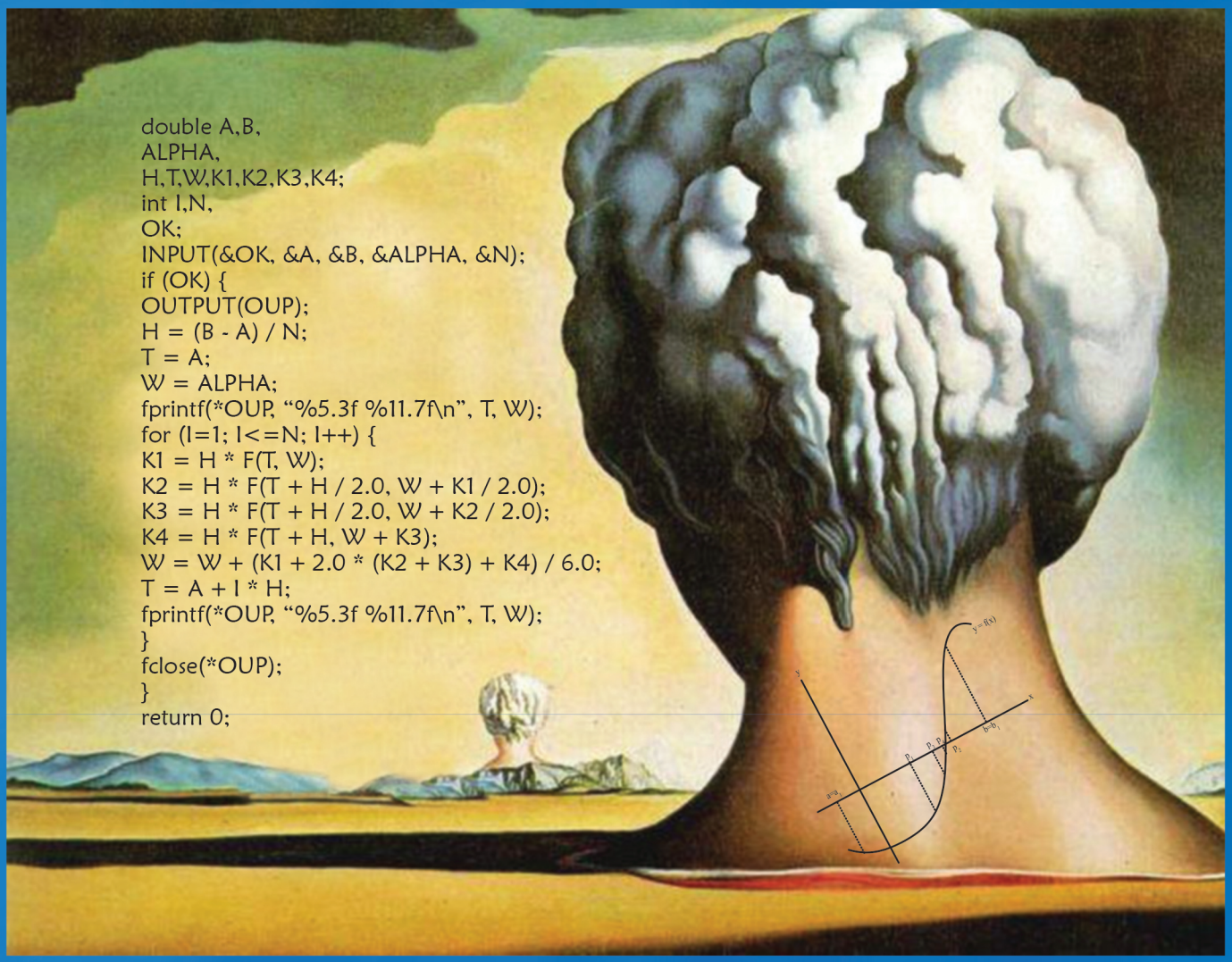
Algoritmos y códigos para cálculos numéricos

Fausto Cervantes Ortiz



Algoritmos y códigos para cálculos numéricos

Algoritmos y códigos para cálculos numéricos

Fausto Cervantes Ortiz

UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

Biblioteca del Estudiante

Cervantes Ortiz, Fausto Algoritmos y códigos para cálculos numéricos / Fausto Cervantes Ortiz. – México, D.F.: UACM, Coordinación Académica, Colegio de Ciencia y Tecnología, Academia de Matemáticas, Ciclo Básico para Ingenierías, 2016.

viii, 95 p. : il. ; 27 cm. – (Biblioteca del estudiante)

"Distribución gratuita para los estudiantes de la UACM" – Reverso de la portada
Bibliografía: p. 95 1. Ecuaciones diferenciales no lineales

QA372 C47

Algoritmos y códigos para cálculos numéricos
primera edición, 2016

ISBN: 978-607-9465-22-3

Biblioteca del Estudiante: http://portal.uacm.edu.mx/Estudiantes/BibliotecadelEstudiante/tabid/276/Default.aspx

Material educativo universitario de distribución gratuita para estudiantes de la UACM.
Prohibida su venta

Hecho e impreso en México

La Ley de la Universidad Autónoma de la Ciudad de México, en su Exposición de motivos, establece:

> *7. Contribuir al desarrollo cultural, profesional y personal de los estudiantes el:*
> [...] El empeño de la Universidad Autónoma de la Ciudad de México deberá ser que todos los estudiantes que a ella ingresen concluyan con éxito sus estudios. Para ello deberá construir los sistemas y servicios que éstos necesiten para alcanzar este propósito de acuerdo con su condición de vida y preparación previa.[1]

De igual manera, en su título primero, capítulo II, artículo 6, fracción IV, dice:

> Concebida como una institución de servicio, la Universidad brindará a los estudiantes los apoyos académicos necesarios para que tengan éxito en sus estudios.[2]

Atendiendo este mandato, los profesores–investigadores de la UACM preparan materiales educativos como herramienta de aprendizaje para los estudiantes de los cursos correspondientes, respondiendo así al principio de nuestra casa de estudios de proporcionarles los soportes necesarios para su avance a lo largo de la licenciatura.

Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

1 Ley de la Universidad Autónoma de la Ciudad de México, publicada en la *Gaceta Oficial del Distrito Federal* el 5 de enero de 2005, reproducida en el Taller de Impresión de la UACM, p. 14.
2 Ídem., p. 18.

Índice

Introducción

El curso de métodos numéricos de la UACM consta de dos partes esenciales: primera, exponer los fundamentos matemáticos de los métodos, y segunda, desarrollar códigos que permitan al alumno realizar los cálculos que se le solicitan usando las ventajas de la computadora.

Con respecto al desarrollo de los programas, es frecuente que se ignore el hecho de que el alumno tomó un curso de programación en C, y se le obligue a adoptar el programa de cómputo que cada profesor preveré, llámese Matlab, Scilab, etcétera. Sin embargo, esto puede provocar que el alumno tenga problemas para desarrollar los códigos, pues su preparación previa no es la adecuada. Como consecuencia, el alumno termina considerando el curso de métodos numéricos como dos cursos diferentes: uno de matemáticas y uno de computación. A consecuencia de ello, si el alumno no aprueba alguna de las dos partes, reprobara el curso completamente.

Con lo anterior en mente, el autor preparo el presente manual de algoritmos, que incluye códigos escritos en lenguaje C, a fin de que el alumno no se distraiga del contenido matemático del curso, y pueda aplicar los presentes algoritmos y códigos a sus problemas sin que esto último sonique un trabajo doble. De este modo, el alumno deja de tener el problema de entender el método sin poder aplicarlo al tener problemas con el programa de cómputo. Si entendió el método, no le será difícil adaptar los códigos dados como ejemplos para resolver los ejercicios que le sean asignados.

El capítulo 1 expone los algoritmos y códigos para la solución de ecuaciones algebraicas no lineales y trascendentes, con una incógnita. Se revisan los métodos de bisección, iteración de Punto fijo, de las tangentes y de las secantes, así como el de la posición falsa.

El capítulo 2 contiene los algoritmos y códigos para interpolación de polinomios a conjuntos de datos. Se exponen los métodos de Lagrange, Newton, así como los *splines* libres y sujetos.

El capítulo 3 trata los algoritmos y códigos para integración y derivación numérica. Se revisan los métodos de los rectángulos, trapecios y parábolas, así como integrales dobles. También la derivación numérica para funciones dadas en forma tabular.

El capítulo 4 se dedica a los algoritmos y códigos para integrar ecuaciones diferenciales ordinarias. Los métodos que abarca son los de Euler, Runge-Kutta, Fehlberg y Adams, así como sistemas de ecuaciones.

El capítulo 5 contiene los algoritmos y códigos para resolver numéricamente sistemas de ecuaciones, lineales o no lineales. Los métodos cubiertos son el de eliminación, iteración de punto fijo, así como el de las tangentes y el de las secantes, extrapolados a varias variables.

Cada capítulo contiene ejercicios al final de sección. En trabajos previos, el autor ha acostumbrado poner la solución de cada ejercicio propuesto. Por la naturaleza del tema de este libro, eso no es posible o recomendable en algunos de los ejercicios aquí planteados. Específicamente, en los capítulos de interpolación y ecuaciones diferenciales, escribir las respuestas haría que el volumen de este libro aumentara en forma considerable. En lugar de ello, se ha optado por dejar sin respuesta cada ejercicio de esos capítulos. El alumno no debería tener problema en verificar por sí mismo la certeza de sus resultados.

Este manual no pretende ser exhaustivo en modo alguno. Los métodos numéricos aquí abordados son los más usuales, sin ser todos los que comprende un curso de métodos numéricos.

Cada uno de los códigos dados se probó para verificar su funcionamiento.[1] A pesar de ello, no se descarta la posibilidad de errores durante la edición del libro. Se agradecerá a los lectores que se sirvan señalarlos para su corrección en futuras ediciones. Por otro lado, habrá que tener siempre presente la posibilidad de que los códigos dados no sean compatibles con la versión del compilador o sistema operativo usado por cada quien.

Se agradece el apoyo de la Academia de Matemáticas para que el presente libro pudiera publicarse. En particular, vayan agradecimientos a Fausto Jarquín Zárate y a Miguel Ángel Mendoza Reyes por sus comentarios y sugerencias útiles. Por otro lado, a Diana Aurora Cruz Hernández, de la Academia de Informática, cuya ayuda para revisar los códigos en C fue determinante. Asimismo, el autor agradece a Irma Irian García Salazar, del *Algonquin College* (en Ottawa, Canadá) por su revisión y las correcciones propuestas.

El presente libro se escribió durante el año sabático que el autor disfrutó de agosto de 2011 a julio de 2012. Por ello agradece a la Universidad Autónoma de la Ciudad de México el apoyo brindado para que este proyecto pudiera realizarse.

Nada humano me es ajeno
Ottawa, Ontario

1 Para ello se usó una computadora DELL Optiplex GX620, con sistema operativo Windows XP, usando el compilador Dev-C++ 4.9.9.2

Capítulo 1

Ecuaciones no lineales

En este capítulo trataremos un problema básico del cálculo numérico: el problema de aproximar raíces para ecuaciones cuya solución en forma analítica es imposible de encontrar. La idea básica es aproximar una raíz, es decir, una solución de una ecuación de la forma $f(x) = 0$, para una función dada $f(x)$, continua sobre algún intervalo $[a, b]$. Los métodos que se exponen son los siguientes: bisección, iteración de punto fijo, Newton-Raphson, secantes y posición falsa. Cada uno de estos métodos resulta ser más conveniente que los otros para diferentes casos, dependiendo eso tanto de la ecuación a resolver como de la facilidad para codificarlo.

1.1. El método de bisección

La primera técnica que se presenta, se conoce con el nombre de *método de bisección*. Sea $f(x)$ una función continua en el intervalo $[a, b]$ con $f(a) \cdot f(b) < 0$. Entonces existe al menos un número p en (a, b) tal que $f(p) = 0$. El método requiere dividir varias veces a la mitad los subintervalos de $[a, b]$ y, en cada paso, localizar la mitad que contenga a p.

Para ello, supongamos que $a_1 = a$ y $b_1 = b$, y sea p_1 el punto medio de $[a, b]$; es decir

$$p_1 = a_1 + \frac{b_1 - a_1}{2} = \frac{a_1 + b_1}{2}. \tag{1.1}$$

Si $f(p_1) = 0$, entonces $p = p_1$, de no ser así, entonces $f(p_1)$ tiene el mismo signo que $f(a)$ o $f(b)$. Si $f(a_1)$ y $f(p_1)$ tienen el mismo signo, entonces $p \in (p_1, b_1)$ y tomamos $a_2 = p_1$ y $b_2 = b_1$. Si $f(p_1)$ y $f(a)$ tienen signos opuestos, entonces $p \in (a_1, p_1)$ y tomamos $a_2 = a_1$ y $b_2 = p_1$. Después volvemos a aplicar el proceso al intervalo $[a_2, b_2]$, y así sucesivamente. Esto se muestra gráficamente en la figura 1.1.

En la práctica, cuando se presenta una ecuación que resolver, es necesario determinar los mejores puntos a y b con los que deberán empezarse los cálculos. Para ello, es muy recomendable generar una gráfica de la función,[1] lo que por inspección nos dará los puntos iniciales.

Enseguida se da el algoritmo general.

Algoritmo de bisección

> Aproxima una solución de $f(x) = 0$ dada la función continua $f(x)$ en el intervalo $[a, b]$, donde $f(a)$ y $f(b)$ tienen signos opuestos.

[1] Para ello se recomienda usar el programa *Graph*, que se puede descargar gratis de la dirección <http://www.padowan.dk>

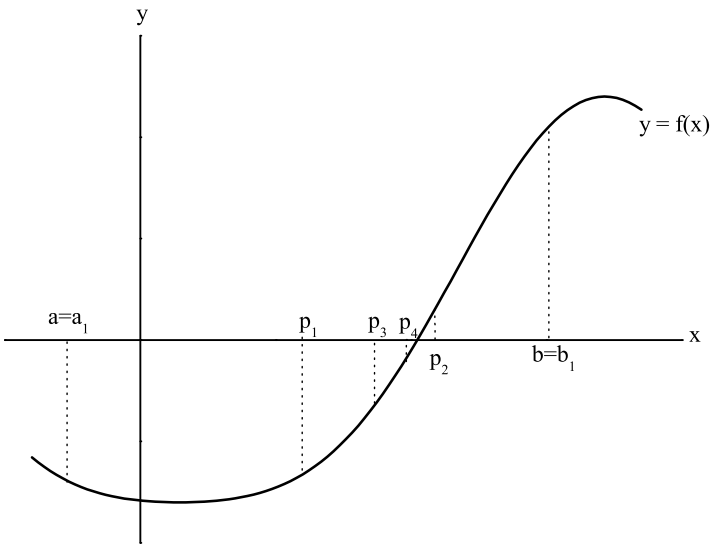


Figura 1.1: Método de bisección.

ENTRADAS

función $f(x)$

extremos a, b

tolerancia TOL

número máximo de iteraciones N_0.

SALIDAS

Solución aproximada p o mensaje de error.

Paso 1 Hacer $i = 1$

Hacer $FA = f(a)$

Paso 2 Mientras $i \leq N_0$, hacer pasos 3 al 6

Paso 3 Hacer $p = a + (b - a)/2$

Hacer $FP = f(p)$

Paso 4 Si $FP = 0$ o $(b - a)/2 < TOL$ entonces

SALIDA(La solución es p)

TERMINAR

Paso 5 Tomar $i = i + 1$

Paso 6 Si $FA \cdot FP > O$ entonces

Hacer $a = p$

Hacer $FA = FP$

Si no

Hacer $b = p$

Paso 7 SALIDA (El método no converge)

TERMINAR

Código para el método de bisección

Enseguida se da un ejemplo de código para el método de bisección, usando

$$f(x) = x^3 + 4x^2 - 10.$$

```
#include<stdio.h>
#include<math.h>
   double F(double);
   void INPUT(int *, double *,  double *, double *, double *, double *, int *);
main() {
   double A,FA,B,FB,C,P,FP,TOL;
   int I,NO,OK;
   INPUT(&OK, &A, &B, &FA, &FB, &TOL, &NO);
   if (OK) {
      I = 1;
      OK = true;
      while ((I<=NO) && OK) {
     C = (B - A) / 2.0;
     P = A + C;
     FP = F(P);
     printf("%3d   %15.8e   %15.7e \n",I,P,FP);
     if ((abs(FP)<0) || (C<TOL)) {
        printf("\nSolucion aproximada P = %11.8f \n",P);
        OK = false;
     }
     else {
        I++;
        if ((FA*FP) > 0.0) {
           A = P; FA = FP;
        }
        else {
           B = P; FB = FP;
        }
     }
      }
      if (OK) {
     printf("\nLa iteracion %3d",NO);
     printf(" da la aproximacion %12.8f\n",P);
     printf("fuera de la tolerancia aceptada\n");
      }
   }
   getchar();
   getchar();
}
double F(double X)
```

```
{
   double f;
   f = ( X + 4.0 ) * X * X - 10.0;
   return f;
}
void INPUT(int *OK, double *A, double *B, double *FA, double *FB, double *TOL, int *NO)
{
   double X;
      *OK = false;
      while (!(*OK)) {
     printf("Dar el valor de A\n");
     scanf("%lf", A);
     printf("Dar el valor de B\n");
     scanf("%lf", B);
     if (*A > *B) {
        X = *A; *A = *B; *B = X;
     }
     if (*A == *B) printf("A debe ser diferente de B\n");
     else {
        *FA = F(*A);
        *FB = F(*B);
        if (*FA*(*FB) > 0.0) printf("F(A) y F(B) tienen el mismo signo\n");
        else *OK = true;
     }
      }
      *OK = false;
      while(!(*OK)) {
     printf("Dar la tolerancia\n");
     scanf("%lf", TOL);
     if (*TOL <= 0.0) printf("La tolerancia debe ser positiva\n");
     else *OK = true;
      }
      *OK = false;
      while (!(*OK)) {
     printf("Dar el numero de iteraciones maximo\n");
     scanf("%d", NO);
     if (*NO <= 0) printf("Debe ser un numero natural\n");
     else *OK = true;
      }
}
```

1.2. Iteración de punto fijo

Un punto fijo de una función continua g es un número p para el cual $g(p) = p$. El método de aproximación de punto fijo consiste en partir de un punto p_0 (aproximación inicial) y generar puntos sucesivos $p_n = g(p_{n-1})$ hasta que se obtenga $p = g(p)$ con el grado de precisión deseado. Por supuesto que la sucesión podría no converger, en cuyo caso se debería probar otro método.

Algoritmo de iteración de punto fijo

Genera una solución a $p = g(p)$ dada una aproximación inicial p_0.

ENTRADA función $g(p)$ aproximación inicial p_0; tolerancia TOL; número máximo de iteraciones N_0.

SALIDA solución aproximada p o mensaje de error.

Paso 1 Hacer $i = 1$

Paso 2 Mientras $i < N_0$ hacer pasos 3 al 6

Paso 3 Tomar $p = g(p_0)$

Paso 4 Si $|p - p_0| < TOL$ entonces

SALIDA (La solución es p)

TERMINAR

Paso 5 Tomar $i = i + 1$

Paso 6 Tomar $p_0 = p$ (definir de nuevo p_0)

Paso 7 SALIDA (El método fracasó después de N_0 iteraciones)

TERMINAR

Este procedimiento se muestra gráficamente en la figura 1.2.

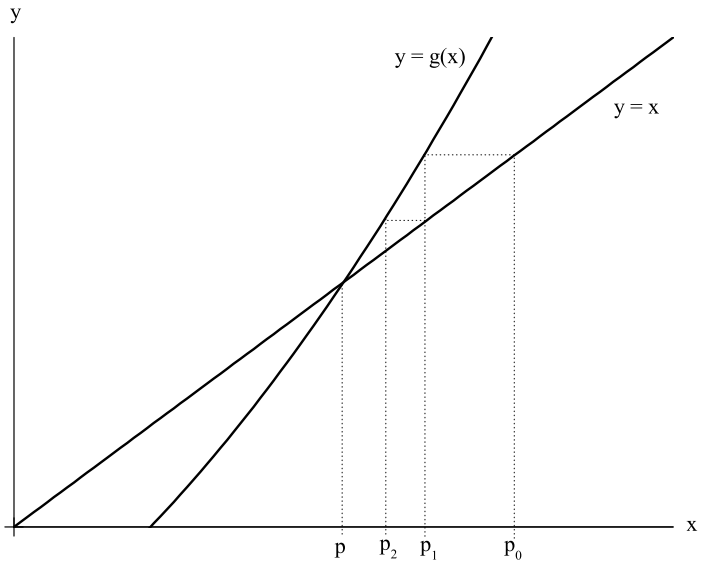


Figura 1.2: Iteración de punto fijo.

Código de iteración de punto fijo

El siguiente ejemplo usa

$$g(x) = \sqrt{\frac{10}{x+4}}.$$

```
#include<stdio.h>
#include<math.h>
   void INPUT(int *, double *, double *, int *);
   void OUTPUT(FILE **, int *);
   double G(double );
main() {
   double TOL,P0,P;
   int I,NO,OK;
   INPUT(&OK, &P0, &TOL, &NO);
   if (OK) {
      I = 1; OK = true;
      while((I<=NO) && OK) {
         P = G(P0);
         printf("%3d   %15.8e\n", I, P);
         if (abs(P-P0) < TOL) {
            printf("\nSolucion aproximada P = %12.8f\n", P);
            OK = false;
         }
         else {
            I++;
            P0 = P;
         }
      }
      if (OK) {
         printf("\nLa iteracion numero %3d", NO);
         printf(" da la aproximacion %12.8f\n", P);
         printf("fuera de la tolerancia aceptada\n");
      }
   }
   getchar();
   getchar();
}
double G(double X)
{
   double g;
   g = sqrt(10.0 / (4.0 + X));
   return g;
}
void INPUT(int *OK, double *P0, double *TOL, int *NO)
{
      *OK = false;
      printf("Dar la aproximacion inicial\n");
      scanf("%lf",P0);
      while(!(*OK)) {
         printf("Dar la tolerancia\n");
         scanf("%lf", TOL);
         if (*TOL <= 0.0) printf("La tolerancia debe ser positiva\n");
         else *OK = true;
      }
      *OK = false;
      while (!(*OK)) {
         printf("Dar el numero de iteraciones maximo\n");
         scanf("%d", NO);
         if (*NO <= 0) printf("Debe ser un numero natural\n");
```

```
         else *OK = true;
      }
}
```

1.3. El método de Newton-Raphson

El método de Newton-Raphson sirve para resolver una ecuación $f(x) = 0$ dada una aproximación inicial p_0. A partir de ella se generan valores sucesivos de p_n $(n \geq 1)$ haciendo

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}. \tag{1.2}$$

La ecuación (1.2) define un recta tangente a $f(x)$ en el punto $(p_{n-1}, f(p_{n-1}))$, y ésta cruza al eje x en p_n, que se toma como referencia para construir la siguiente aproximación.

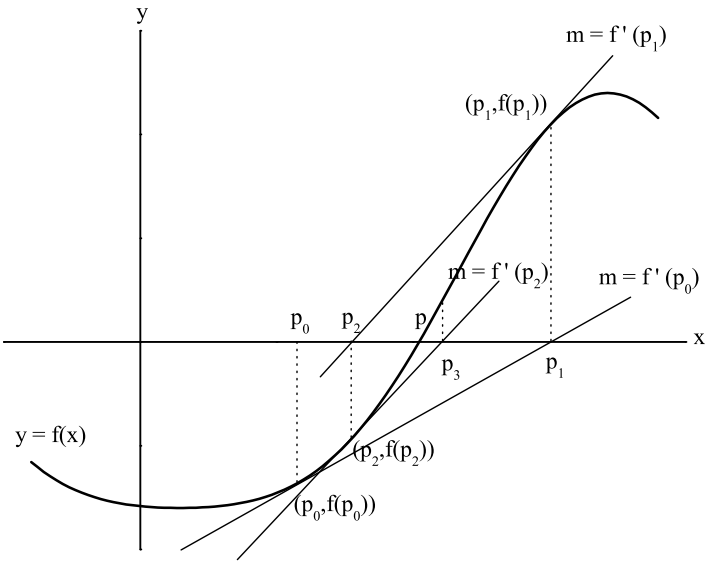


Figura 1.3: Método de Newton-Raphson

Como puede verse en la figura 1.3, la recta tangente acelera el proceso de convergencia a la raíz. Esto nos permite establecer un número máximo de iteraciones menor que en otros métodos, como el método de bisección o el del punto fijo.

Algoritmo de Newton-Raphson

Genera una solución a $f(p) = 0$ dada una aproximación inicial p_0.

ENTRADA $f(x)$ y su derivada $f'(x)$, aproximación inicial p_0, tolerancia TOL, número de iteraciones máximo N_0.

SALIDA Solución aproximada p o mensaje de error.

Paso 1 Hacer $i = 1$

Paso 2 Mientras $i \leq N_0$, hacer pasos 3 al 6

Paso 3 Hacer $p = p_0 - f(p_0)/f'(p_0)$

Paso 4 Si $|p - p_0| < TOL$

SALIDA (La solución es p)

TERMINAR

Paso 5 Hacer $i = i + 1$

Paso 6 Hacer $p_0 = p$

Paso 7 SALIDA (El método fracasó después de N_0 iteraciones)

TERMINAR

Código de Newton-Raphson

Este ejemplo utiliza

$$f(x) = \cos x - x.$$

```
#include<stdio.h>
#include<math.h>
   double F(double);
   double FP(double);
   void INPUT(int *, double *, double *, int *);
main() {
   double TOL,P0,D,F0,FP0;
   int OK,I,NO;
   INPUT(&OK, &P0, &TOL, &NO);
   if (OK) {
      F0 = F(P0);
      I = 1;
      OK = true;
      while ((I<=NO) && OK) {
     FP0 = FP(P0);
     D = F0/FP0;
     P0 = P0 - D;
     F0 = F(P0);
     printf("%3d   %14.8e   %14.7e\n",I,P0,F0);
     if (abs(D) < TOL) {
        printf("\nSolucion aproximada = %.10e\n",P0);
        OK = false;
  }
     else I++;
      }
      if (OK) {
     printf("\nLa iteracion %d",NO);
     printf(" da la aproximacion %.10e\n",P0);
     printf("fuera de la tolerancia aceptada\n");
      }
   }
   getchar();
   getchar();
```

```
}
double F(double X)
{
   double f;
   f = cos(X) - X;
   return f;
}
double FP(double X)
{
   double fp;
   fp = -sin(X) - 1;
   return fp;
}
void INPUT(int *OK, double *P0, double *TOL, int *NO)
{
      *OK = false;
      printf("Dar aproximacion inicial\n");
      scanf("%lf", P0);
      while(!(*OK)) {
     printf("Dar la tolerancia\n");
     scanf("%lf", TOL);
     if (*TOL <= 0.0) printf("La tolerancia debe ser positiva\n");
     else *OK = true;
      }
      *OK = false;
      while (!(*OK)) {
     printf("Dar el numero de iteraciones maximo\n");
     scanf("%d", NO);
     if (*NO <= 0) printf("Debe ser un entero positivo\n");
     else *OK = true;
      }
}
```

1.4. El método de las secantes

Como en el método de Newton-Raphson interviene $f'(x)$, a veces es preferible evitar su cálculo y en su lugar aproximar la recta tangente con la ecuación de una secante. En este caso se necesitan dos aproximaciones iniciales y a partir de la tercera se generan usando

$$p_n = p_{n-1} - \frac{f(p_{n-1})(p_{n-1} - p_{n-2})}{f(p_{n-1}) - f(p_{n-2})}. \tag{1.3}$$

La rapidez de convergencia de este método es menor que la del método de Newton-Raphson, pero mayor que la del método de bisección o del punto fijo. La figura 1.4 muestra gráficamente el método.

Algoritmo de las secantes

Aproxima una solución de la ecuación $f(x) = 0$ dadas dos aproximaciones iniciales.

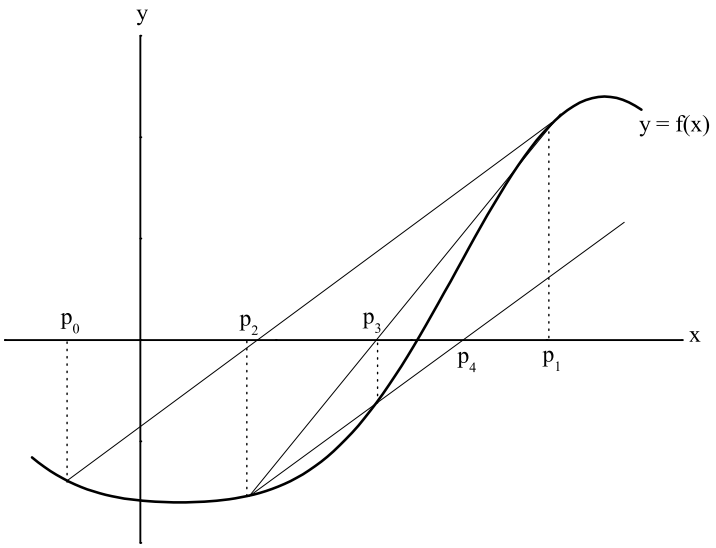


Figura 1.4: Método de las secantes

ENTRADA $f(x)$, aproximaciones iniciales p_0 y p_1, tolerancia TOL, número de iteraciones máximo N_0.

SALIDA Solución aproximada p o mensaje de error.

Paso 1 Hacer $i = 2$, $q_0 = f(p_0)$, $q_1 = f(p_1)$

Paso 2 Mientras $i \leq N_0$ hacer pasos 3 al 6

Paso 3 Hacer $p = p_1 - q_1(p_1 - p_0)/(q_1 - q_0)$

Paso 4 Si $|p - p_1| < TOL$ entonces

SALIDA (La solucion es p)

TERMINAR

Paso 5 Hacer $i = i + 1$

Paso 6 Hacer $p_0 = p_1$, $q_0 = q_1$, $p_1 = p$, $q_1 = f(p)$

Paso 7 SALIDA (El método fracasó después de N_0 iteraciones)

TERMINAR

Código de las secantes

En el siguiente ejemplo se usa $f(x) = \operatorname{sen} x + x$.

```
#include<stdio.h>
#include<math.h>
   double F(double);
   void INPUT(int *, double *, double *, double *, int *);
   void OUTPUT(FILE **, int *);
```

```
main() {
   double P0,F0,P1,F1,P,FP,TOL;
   int I,NO,OK;
   INPUT(&OK, &P0, &P1, &TOL, &NO);
   if (OK) {
      I = 2;
      F0 = F(P0);
      F1 = F(P1);
      OK = true;
      while ((I<=NO) && OK) {
     P = P1 - F1 * (P1 - P0) / (F1 - F0);
     FP = F(P);
     printf("%3d   %15.8e   %15.8e \n",I,P,FP);
     if (abs(P - P1) < TOL) {
        printf("\nSolucion aproximada P = %12.8f\n",P);
        OK = false;
     }
     else {
        I++;
        P0 = P1;
        F0 = F1;
        P1 = P;
        F1 = FP;
     }
      }
      if (OK) {
     printf("\nLa iteracion %3d",NO);
     printf(" da la aproximacion %12.8f\n",P);
     printf("fuera de la tolerancia aceptada\n");
      }
   }
   getchar();
   getchar();
}
double F(double X)
{
   double f;
   f = sin(X) + X;
   return f;
}
void INPUT(int *OK, double *P0, double *P1, double *TOL, int *NO)
{
      *OK = false;
      while (!(*OK)) {
     printf("Dar la aproximacion inicial P0\n");
     scanf("%lf", P0);
     printf("Dar la otra aproximacion inicial P1\n");
     scanf("%lf", P1);
     if (*P0 == *P1) printf("P0 debe ser diferente a P1\n");
     else *OK = true;
      }
      *OK = false;
      while(!(*OK)) {
     printf("Dar la tolerancia\n");
```

```
      scanf("%lf", TOL);
      if (*TOL <= 0.0) printf("La tolerancia debe ser positiva\n");
      else *OK = true;
       }
       *OK = false;
       while (!(*OK)) {
      printf("Dar el numero de iteraciones maximo\n");
      scanf("%d", NO);
      if (*NO <= 0) printf("Debe ser un numero natural\n");
      else *OK = true;
       }
}
```

1.5. El método de la posición falsa

El método de la posición falsa genera aproximaciones del mismo tipo que el de la secante, pero añade una prueba para asegurar que la raíz queda entre dos iteraciones sucesivas. Primero se eligen dos aproximaciones iniciales p_0 y p_1, con $f(p_0) \cdot f(p_1) < 0$. La aproximación p_2 se escoge de manera similar al a utilizada en el método de la secante, es decir, es la intersección en x de la recta que une a $(p_0, f(p_0))$ y $(p_1, f(p_1))$. Para decidir con cuál secante se calcula p_3, antes se verifica si $f(p_2) \cdot f(p_1) < 0$ y se elige p_3 como la intersección con x de la línea que une $(p_1, f(p_1))$ y $(p_2, f(p_2))$. En otro caso, se elige p_3 como la intersección con x de la línea que une $(p_0, f(p_0))$ y $(p_2, f(p_2))$, y se intercambian los índices de p_0 y p_1. Esto se repite en cada nueva iteración. La figura 1.5 muestra gráficamente este método.

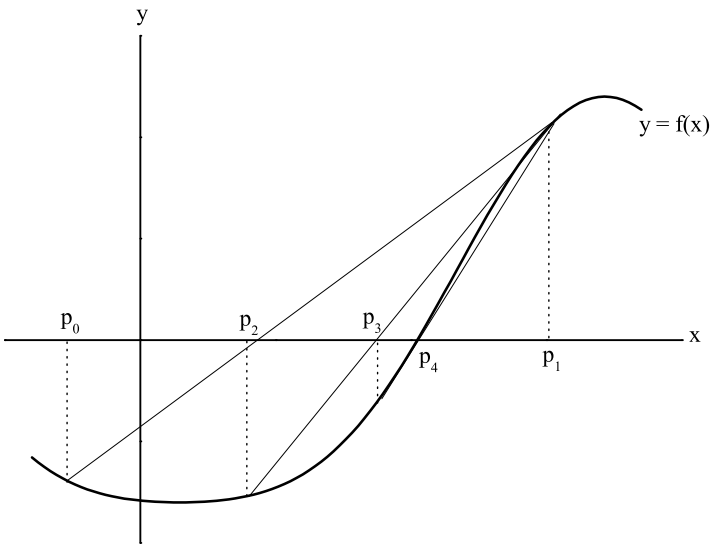


Figura 1.5: Método de la posición falsa

Algoritmo de la posición falsa

Aproxima una solución para la ecuación $f(x) = 0$ dadas dos aproximaciones iniciales.

ENTRADA $f(x)$, aproximaciones iniciales $p = 0$, $p = 1$, que cumplan $f(p_0) \cdot f(p_1) < 0$, tolerancia TOL, número de iteraciones máximo N_0.

SALIDA Solución aproximada p o mensaje de error.

Paso 1 Hacer $i = 2$, $q_0 = f(p_0)$, $q_1 = f(p_1)$

Paso 2 Mientras $i \leq N_0$ hacer pasos 3 al 7

Paso 3 Hacer $p = p_1 - q_1(p_1 - p_0)/(q_1 - q_0)$

Paso 4 Si $|p - p_1| < TOL$ entonces

SALIDA (La solución es p)

TERMINAR

Paso 5 Hacer $i = i + 1$, $q = f(p)$

Paso 6 Si $q \cdot q_1 < 0$, entonces hacer $p_0 = p_1$, $q_0 = q_1$

Paso 7 Hacer $p_1 = p$, $q_1 = q$

Paso 8 SALIDA (El método fracasó después de N_0 iteraciones)

TERMINAR

Código de la posición falsa

El siguiente ejemplo usa $f(x) = \tan(x) - x$.

```
#include<stdio.h>
#include<math.h>
   double F(double);
   void INPUT(int *, double *, double *, double *, double *, double *, int *);
main() {
   double Q,P0,Q0,P1,Q1,C,P,FP,TOL;
   int I,NO,OK;
   INPUT(&OK, &P0, &P1, &Q0, &Q1, &TOL, &NO);
   if (OK) {
      I = 2;
      OK = true;
      Q0 = F(P0);
      Q1 = F(P1);
      while ((I<=NO) && OK) {
     P = P1 - Q1 * (P1 - P0) / (Q1 - Q0);
     Q = F(P);
     printf("%3d   %15.8e   %15.8e\n",I,P,Q);
     if (abs(P-P1) < TOL) {
        printf("\nSolucion aproximada P = %12.8f\n",P);
        OK = false;
     }
     else {
```

```
         I++;
         if ((Q*Q1) < 0.0) {
            P0 = P1; Q0 = Q1;
         }
         P1 = P; Q1 = Q;
      }
       }
       if (OK) {
      printf("\nLa iteracion numero %3d",NO);
      printf(" da la aproximacion %12.8f\n",P);
      printf("fuera de la tolerancia\n");
       }
   }
   getchar();
   getchar();
}
double F(double X)
{
   double f;
   f = tan(X) - X;
   return f;
}
void INPUT(int *OK, double *P0, double *P1, double *Q0, double *Q1, double
*TOL, int *NO)
{
   double X;
      *OK = false;
      while (!(*OK)) {
     printf("Dar el valor de P0\n");
     scanf("%lf", P0);
     printf("Dar el valor de P1\n");
     scanf("%lf", P1);
     if (*P0 > *P1) {
        X = *P0; *P0 = *P1; *P1 = X;
     }
     if (*P0 == *P1) printf("P0 no debe ser igual a P1.\n");
     else {
        *Q0 = F(*P0);
        *Q1 = F(*P1);
        if (*Q0*(*Q1) > 0.0) printf("F(P0) y F(P1) tiene el mismo signo\n");
        else *OK = true;
     }
      *OK = false;
      while(!(*OK)) {
     printf("Dar la tolerancia\n");
     scanf("%lf", TOL);
     if (*TOL <= 0.0) printf("La tolerancia debe ser positiva\n");
     else *OK = true;
      }
      *OK = false;
      while (!(*OK)) {
     printf("Dar el numero de iteraciones maximo\n");
     scanf("%d", NO);
     if (*NO <= 0) printf("Debe ser un entero positivo\n");
```

```
      else *OK = true;
       }
   }
}
```

Ejercicios

Usar cada uno de los métodos estudiados en este capítulo para encontrar las raíces de las siguientes ecuaciones. Tomar una tolerancia de 0.0001. Se recomienda utilizar un programa de graficación para encontrar los puntos iniciales donde no se especifica intervalo alguno. Tomar en cuenta que algunas ecuaciones tienen más de una raíz, por lo que habrá que repetir el método para varios puntos iniciales diferentes. En el caso de funciones periódicas donde hay infinidad de soluciones, será suficiente con encontrar las cinco raíces más cercanas al origen. Comparar la eficiencia de cada método por medio del número de iteraciones que cada uno requirió en una misma ecuación.

1. $\sqrt{x} = \cos x, [0, 1]$ R: 0.625
2. $x^3 - 7x^2 + 14x - 6 = 0$ R: 0.5859, 3.002, 3.419
3. $x = \tan x, [4, 4.45]$ R: 4.4932
4. $x - 2^{-x} = 0, [0, 1]$ R: 0.641182
5. $e^x - x^2 + 3x - 2 = 0, [0, 1]$ R: 0.257530
6. $2 \operatorname{sen} \pi x + x = 0, [1, 2]$ R: 1.683855
7. $x^3 - 2x^2 - 5 = 0, [1, 4]$ R: 2.69065
8. $x^3 + 3x^2 - 1 = 0, [-3, -2]$ R: -2.87939
9. $x - \cos x = 0, [0, \pi/2]$ R: 0.73909
10. $x - 0.8 - .02 \operatorname{sen} x = 0, [0, \pi/2]$ R: 0.96434
11. $x^2 - 2xe^{-x} + e^{-2x} = 0, [0, 1]$ R: 0.567135
12. $\cos(x + \sqrt{2}) + x(x/2 + \sqrt{2}) = 0, [-2, -1]$ R: -1.414325
13. $x^3 - 3x^2(2^{-x}) + 3x(4^{-x}) - 8^{-x} = 0, [0, 1]$ R: 0.641166
14. $e^{6x} + 3(\ln 2)^2 e^{2x} - (\ln 8)e^{4x} - (\ln 2)^3 = 0, [-1, 0]$ R: -0.183274
15. $2x \cos x - (x + 1)^2 = 0, [-3, -2], [-1, 0]$ R: $-2.191307, -0.798164$
16. $x \cos x - 2x^2 + 3x - 1 = 0$ R: 0.297528, 1.256622
17. $3x = 2 - e^x + x^2$ R: 0.257531
18. $x = 5/x^2 + 2$ R: 2.690650
19. $x = \sqrt{e^3/3}$ R: 0.909999
20. $x = 5^{-x}$ R: 0.469625
21. $x = e^{-x}$ R: 0.448059
22. $x = 0.5(\operatorname{sen} x + \cos x)$ R: 0.704812
23. $x^2 + 10 \cos x = 0$ R: $\pm 3.16193, \pm 1.96882$
24. $x^5 - x^4 + 2x^3 - 3x^2 + x - 4 = 0$ R: 1.49819
25. $\ln(x^2 + 1) = e^{0.4x} \cos \pi x$ R: $-0.4341431, 0.4506567, 1.7447381, 2.2383198, 3.7090412, 24.4998870$

26. Se dispara una bala de $M = 2$ g en dirección vertical hacia arriba. Después desciende a su velocidad terminal, que está dada por

$$\frac{(2)(9.8)}{1000} = 1.4 \times 10^{-5} v^{1.5} + 1.15 \times 10^{-5} v^2,$$

con v la velocidad terminal en m/s. Determinar la velocidad terminal de la bala con una tolerancia de 0.01 m/s. R: 37.73

27. Un objeto que cae verticalmente en el aire está sujeto a una resistencia viscosa, además de a la fuerza de gravedad. Suponiendo que al dejar caer un objeto de masa $m = 0.1$ kg desde una altura $h_0 = 90$ m, la altura del objeto t segundos después es

$$h(t) = h_0 - \frac{mg}{k}t + \frac{m^2 g}{k^2}\left(1 - e^{-\frac{kt}{m}}\right),$$

donde g es la constante gravitacional en la superficie terrestre, generalmente aceptada como 9.8 m/s^2 y k representa el coeficiente de resistencia del aire en Ns/m. Suponiendo $k = 0.15$ Ns/m, calcular con una precisión de 0.01 s el tiempo que tarda este objeto en caer al suelo. R: 6.0028 s

28. Un kg mol de gas de CO está confinado en un recipiente a $T = 215$ K y $p = 70$ bar. Calcular el volumen del gas usando la ecuación de estado de Van der Waals

$$\left(p + \frac{a}{v^2}\right)(v - b) = RT,$$

donde $R = 0.08314$ bar m^3/(kg mol K), $a = 1.463$ bar m^6/(kg mol)^2 y $b = 0.0394$ m^3/kg. Comparar con el resultado que se obtiene al utilizar la ecuación de estado de un gas ideal $pv = RT$.

Capítulo 2

Interpolación

El problema de la interpolación surge de la acumulación de información en ciencias experimentales. Al formularse leyes físicas, por ejemplo, se hacen en términos de funciones continuas. Sin embargo, al realizar observaciones se tienen fundamentalmente conjuntos discretos. Entonces, se busca hacer que pase una función continua a través de tal conjunto de puntos. Frecuentemente, lo más fácil es ajustar algún polinomio de grado n a algún conjunto de $n+1$ puntos. Esto puede hacerse por medio de varios métodos diferentes. En este capítulo se cubren los métodos de Lagrange y de Newton, así como los trazadores cúbicos, o *splines*, tanto libres como sujetos.

2.1. Interpolación de Lagrange

Consiste en determinar un polinomio de grado n que pasa por $n+1$ puntos dados (x_0, y_0), (x_1, y_1), . . . , (x_n, y_n). El polinomio está dado por la fórmula

$$P_n(x) = \frac{(x-x_1)(x-x_2)\cdots(x-x_n)}{(x_0-x_1)(x_0-x_2)\cdots(x_0-x_n)}y_0 + \frac{(x-x_0)(x-x_2)\cdots(x-x_n)}{(x_1-x_0)(x_1-x_2)\cdots(x_1-x_n)}y_1 + \cdots$$

$$+\frac{(x-x_0)(x-x_1)\cdots(x-x_{n-1})}{(x_n-x_0)(x_n-x_1)\cdots(x_n-x_{n-1})}y_n. \tag{2.1}$$

Algoritmo de interpolación de Lagrange

Obtiene el polinomio interpolante de Lagrange

$$P_n(x) = \sum_{i=0}^{n} f(x_i)\left(\prod_{j=0, j\neq i}^{n}\frac{x-x_j}{x_i-x_j}\right)$$

y evalúa en x dada.

ENTRADA Número de puntos, valores de x_i, $f(x_i)$.

SALIDA Polinomio interpolante $P_n(x)$.

Paso 1 Hacer $p = 0$

Paso 2 Para $i = 0, n$ hacer pasos 3 al 6

Paso 3 Hacer $z = f(x_i)$

Paso 4 Para $j = 0, n$ hacer paso 5

Paso 5 Si $i \neq j$, hacer $z = z \cdot \left(\frac{x - x_j}{x_i - x_j}\right)$

Paso 6 Hacer $p = p + z$

Paso 7 SALIDA (El valor del polinomio en x es p)

TERMINAR

Código de interpolación de Lagrange

El siguiente ejemplo interpola puntos dados, usando como entrada un conjunto de n datos en un archivo de texto, que debe tener como formato en cada renglón: "x[tabulador]y".

```
#include<stdio.h>
#include<math.h>
void INPUT(int *, double *, double [][26], double *, int *);
void OUTPUT(int, double, double *, double [][26]);
main()
{
   double Q[26][26],XX[26],D[26];
   double X;
   int I,J,N,OK;
   INPUT(&OK, XX, Q, &X, &N);
   if (OK) {
      D[0] = X - XX[0];
      for (I=1; I<=N; I++) {
         D[I] = X - XX[I];
         for (J=1; J<=I; J++)
            Q[I][J] = (D[I] * Q[I-1][J-1] - D[I-J] *
                      Q[I][J-1]) / (D[I] - D[I-J]);
      }
      OUTPUT(N, X, XX, Q);
   }
   return 0;
} void INPUT(int *OK, double *XX, double Q[][26], double *X, int *N) {
   int I;
   char NAME[30];
   FILE *INP;
   printf("Dar el nombre y la extension del archivo de entrada en el formato ");
   printf("nombre.txt\n");
   scanf("%s", NAME);
   INP = fopen(NAME, "r");
   *OK = false;
   while (!(*OK)) {
      printf("Dar valor de n\n");
      scanf("%d", N);
      if (*N > 0) {
          for (I=0; I<=*N; I++)
            fscanf(INP, "%lf %lf", &XX[I], &Q[I][0]);
            fclose(INP);
```

```
            *OK = true;
         }
         else printf("Debe ser un entero positivo\n");
   }
   if (*OK) {
      printf("Dar el punto en que se va a evaluar el polinomio ");
      scanf("%lf", X);
            }
} void OUTPUT(int N, double X, double *XX, double Q[][26]) {
   int I, J;
   for (I=0; I<=N; I++) {
      printf("%11.6f ", XX[I]);
      for (J=0; J<=I-1; J++) printf("%11.8f ", Q[I][J]);
      printf("\n");
   }
}
```

2.2. Interpolación de Newton

Dado un conjunto de $n+1$ puntos cuyas diferencias en las abscisas son constantes, esto es, $x_i - x_{i-1} = \text{cte} = h$, el polinomio interpolante es de la forma

$$P_n(x) = y_0 + \Delta y_0 \frac{x-x_0}{h} + \frac{\Delta^2 y_0}{1\cdot 2}\cdot\frac{x-x_0}{h}\left(\frac{x-x_0}{h}-1\right) + \frac{\Delta^3 y_0}{1\cdot 2\cdot 3}\cdot\frac{x-x_0}{h}\left(\frac{x-x_0}{h}-1\right)\left(\frac{x-x_0}{h}-2\right) +$$

$$\frac{\Delta^n y_0}{n!}\cdot\frac{x-x_0}{h}\left(\frac{x-x_0}{h}-1\right)\left(\frac{x-x_0}{h}-2\right)\cdots\left(\frac{x-x_0}{h}-(n-1)\right). \tag{2.2}$$

El polinomio interpolante de Newton es igual al que se obtendría si se usara la fórmula de Lagrange. La ventaja de usar el polinomio de Newton es que si se agrega un nuevo punto, sólo hay que calcular un término nuevo, mientras que para Lagrange es necesario repetir todos los cálculos.

Si las diferencias no son constantes, el algoritmo tiene que tomar en cuenta la diferencia a cada paso. Para ello se definen las diferencias divididas como sigue:

Diferencia dividida cero:

$$f[x_i] = f(x_i), \tag{2.3}$$

1a diferencia dividida

$$f[x_i, x_{i+1}] = \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i}, \tag{2.4}$$

2a diferencia dividida

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}, \tag{2.5}$$

k-ésima diferencia dividida

$$f[x_i, x_{i+1}, ..., x_{i+k}] = \frac{f[x_{i+1}, x_{i+2}, ..., x_{i+k}] - f[x_i, x_{i+1}, ..., x_{i+k-1}]}{x_{i+k} - x_i}. \tag{2.6}$$

Con las diferencias divididas definidas así, el polinomio interpolante es de la forma

$$P_n(x) = f[x_0] + \sum_{k=1}^{n} f[x_0, x_1, ..., x_k](x - x_0) \cdots (x - x_{k-1}), \tag{2.7}$$

o bien

$$P(x) = \sum_{i=0}^{n} F_i \prod_{j=0}^{i-1} (x - x_j), \tag{2.8}$$

donde

$$F_i = f[x_0, x_1, ..., x_i]. \tag{2.9}$$

Enseguida se da el algoritmo para construir el polinomio interpolante de Newton.

Algoritmo de interpolación de Newton

Obtiene los coeficientes del polinomio interpolante P en $n + 1$ valores de f, en los puntos x_0, x_1, x_2, . . . , x_n.

ENTRADA Valores de x_0, x_1, . . . , x_n.

Valores de $f(x_1)$, $f(x_2)$, . . . , $f(x_n)$, que se identificarán como $F_{0,0}$, $F_{1,0}$,..., $F_{n,0}$.

SALIDA Coeficientes $F_i = F_{i,i}$ del polinomio

$$P(x) = \sum_{i=0}^{n} F_i \prod_{j=0}^{i-1} (x - x_j). \tag{2.10}$$

Paso 1 Para $i = 1, 2, ..., n$ hacer paso 2

Paso 2 Para $j = 1, 2, ..., i$ Hacer paso 3

Paso 3 Hacer $F_{i,j} = \frac{F_{i,j-1} - F_{i-1,j-1}}{x_i - x_{i-j}}$

Paso 4 Salida (Los coeficientes son: $F_i = F_{1,1}$)

TERMINAR

Código de interpolación de Newton

El siguiente ejemplo calcula los coeficientes del polinomio interpolante usando como entrada un conjunto de n datos en un archivo de texto, que debe tener como formato en cada renglón: "x[tabulador]y". La salida la escribe en un archivo de texto.

```
#include<stdio.h>
#include<math.h>
void INPUT(int *, double *, double [][26], int *);
void OUTPUT(FILE **);
main()
{
   double Q[26][26],X[26];
   int I,J,N,OK;
   FILE *OUP[1];
   INPUT(&OK, X, Q, &N);
   if (OK) {
      OUTPUT(OUP);
      for (I=1; I<=N; I++)
         for (J=1; J<=I; J++)
            Q[I][J] = (Q[I][J-1] - Q[I-1][J-1]) / (X[I] - X[I-J]);
      fprintf(*OUP, "Datos de entrada:\n");
      for (I=0; I<=N; I++)
        fprintf(*OUP, "X(%d) = %12.8f F(X(%d)) = %12.8f\n", I, X[I], I, Q[I][0]);
      fprintf(*OUP, "\nLos coeficientes Q(0,0), ..., Q(N,N) son:\n");
      for (I=0; I<=N; I++) fprintf(*OUP, "%12.8f\n", Q[I][I]);
   }
   return 0;
}
void INPUT(int *OK, double *X, double Q[][26], int *N)
{
   int I;
   char NAME[30];
   FILE *INP;
   printf("Dar el nombre del archivo de entrada en el formato:");
   printf("nombre.txt\n");
   scanf("%s", NAME);
   INP = fopen(NAME, "r");
   *OK = false;
   while (!(*OK)) {
      printf("Dar valor de n\n");
      scanf("%d", N);
      if (*N > 0) {
         for (I=0; I<=*N; I++)
            fscanf(INP, "%lf %lf", &X[I], &Q[I][0]);
         fclose(INP);
         *OK = true;
      }
      else printf("Debe ser un entero positivo\n");
   }
}
void OUTPUT(FILE **OUP)
{
   char NAME[30];
   printf("Dar el nombre del archivo de salida en el formato\n");
   printf("nombre.txt\n");
   scanf("%s", NAME);
   *OUP = fopen(NAME, "w");
}
```

2.3. Interpolación con splines cúbicos

Al realizar interpolación con polinomios de grado muy grande, se corre el riesgo de obtener curvas demasiado oscilantes que, si bien pasan por todos los puntos dados, no dan una representación simple del conjunto de datos. Esto se ilustra en la figura 2.1.

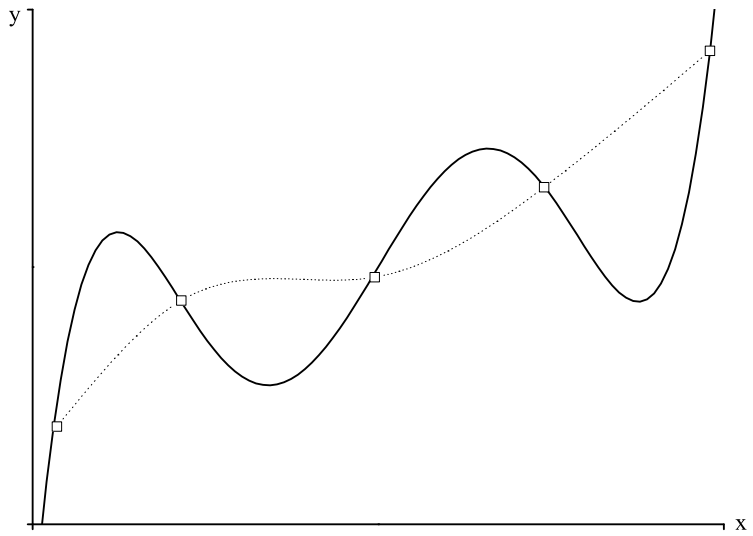


Figura 2.1: La línea continua muestra el polinomio interpolante que pasa por los puntos dados (marcados por cuadros), mientras que la línea punteada muestra una curva más suave, que representaría mejor a la función

Una alternativa para evitar esto es aproximar por fragmentos de cuatro puntos usando un polinomio cúbico para cada fragmento. A tales polinomios se les llama *splines* (palabra inglesa que se puede traducir en términos prácticos como *trazadores*). Para poder interpolar un conjunto de *splines* $S_j(x)$ a una serie de datos se buscará que se cumplan los siguientes requisitos:

1. La derivada del *spline* en cada extremo coincide con la derivada en el mismo extremo de los *splines* adyacentes, esto es

$$S'_{j-1}(x_i) = S'_j(x_i), \quad S'_j(x_{i+4}) = S'_{j+1}(x_{i+4}).$$

2. La segunda derivada también coincide en los extremos

$$S''_{j-1}(x_i) = S''_j(x_i), \quad S''_j(x_{i+4}) = S''_{j+1}(x_{i+4}).$$

3. El primer y último *splines* deben cumplir en los extremos

   a) *Splines* libres

$$S''(x_0) = S''(x_n) = 0,$$

   o bien

b) *Splines* sujetos

$$S'(x_0) = f'(x_0) \quad \text{y} \quad S'(x_n) = f'(x_n).$$

Los *splines* libres se usan a falta de una mejor estimación del valor de la derivada en los extremos del intervalo. Sin embargo, si se puede hacer una estimación, se obtiene un resultado mejor: la forma de la curva de interpolación es la que se tendría al hacer pasar una varilla flexible por todos los puntos, incluyendo a los extremos. En el caso de splines libres, forzar la derivada a cero puede crear una torsión innecesaria en los bordes. No obstante, cuando no se tiene la información necesaria, es lo mejor que se puede hacer para interpolar.

Algoritmo de interpolación con splines libres

Obtiene los coeficientes de los polinomios interpolantes $S_i(x)$ de la función f definida en los puntos x_0, x_1, . . . ,x_n, con $S''(x_0) = S''(x_n) = 0$.

ENTRADA n, x_i, $f(x_i)$ identificados por a_i.

SALIDA a_j, b_j, c_j, d_j, coeficientes de los polinomios

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$$

definidos en $x_j \leq x \leq x_{j+1}$.

Paso 1 Para $i = 0, ..., n-1$ hacer $h_i = x_{i+1} - x_i$

Paso 2 Para $i = 1, ..., n-1$ hacer paso 3

Paso 3
$$\alpha_i = \frac{3}{h_i}(a_{i+1} - a_i) - \frac{3}{h_{i-1}}(a_i - a_{i-1})$$

Paso 4 Hacer $\ell_0 = 1$, $\mu_0 = 0$, $z_0 = 0$

Paso 5 Para $i = 1, ..., n$ hacer pasos 6 al 8

Paso 6 Hacer $\ell_i = 2(x_{i+1} - x_{i-1}) - h_{i-1}\mu_{i-1}$

Paso 7 Hacer $\mu_i = \frac{h_i}{\ell_i}$

Paso 8 Hacer $z_i = \frac{\alpha_i - h_{i-1}z_{i-1}}{\ell_i}$

Paso 9 Hacer $\ell_n = 1$

Paso 10 Hacer $z_n = 0$

Paso 11 Hacer $c_n = 0$

Paso 12 Para $j = n-1, ..., 0$ hacer pasos 13 al 15

Paso 13 Hacer $c_j = z_j - \mu_j c_{j+1}$

Paso 14 $b_j = \frac{a_{j+1} - a_j}{h_j} - \frac{h_j(c_{j+1} + 2c_j)}{3}$

Paso 15 $d_j = \frac{c_{j+1} - c_j}{3h_j}$

Paso 16 SALIDA (Los coeficientes son a_j, b_j, c_j, d_j)

TERMINAR

Código de interpolación con splines libres

El siguiente ejemplo calcula los coeficientes de los *splines* cúbicos, usando como entrada un conjunto de n datos en un archivo de texto, que debe tener como formato en cada renglón: "x[tabulador]y". Los resultados los escribe en un archivo de texto.

```
#include<stdio.h>
#include<math.h>
void INPUT(int *, double *, double *, int *);
void OUTPUT(int, int, double *, double *, double *, double *, double *);
main()
{
   double X[26],A[26],B[26],C[26],D[26],H[26],XA[26],XL[26],XU[26],XZ[26];
   double XX,S,Y;
   int I,J,N,M,OK;
   INPUT(&OK, X, A, &N);
   if (OK) {
      M = N - 1;
      for (I=0; I<=M; I++) H[I] = X[I+1] - X[I];
      for (I=1; I<=M; I++)
         XA[I] = 3.0 * (A[I+1] * H[I-1] - A[I] *
                  (X[I+1] - X[I-1]) + A[I-1] * H[I])  /
                  (H[I] * H[I-1]);
      XL[0] = 1.0; XU[0] = 0.0; XZ[0] = 0.0;
      for (I=1; I<=M; I++) {
         XL[I] = 2.0 * (X[I+1] - X[I-1]) - H[I-1] * XU[I-1];
         XU[I] = H[I] / XL[I];
         XZ[I] = (XA[I] - H[I-1] * XZ[I-1]) / XL[I];
      }
      XL[N] = 1.0; XZ[N] = 0.0; C[N] = XZ[N];
      for (I=0; I<=M; I++) {
         J = M - I;
         C[J] = XZ[J] - XU[J] * C[J+1];
         B[J] = (A[J+1] - A[J]) / H[J] - H[J] * (C[J+1] + 2.0 * C[J]) / 3.0;
         D[J] = (C[J+1] - C[J]) / (3.0 * H[J]);
      }
      OUTPUT(N, M, X, A, B, C, D);
   }
   return 0;
}
void INPUT(int *OK, double *X, double *A, int *N)
{
   int I;
   char NAME[30];
   FILE *INP;
   printf("Dar nombre de archivo de entrada en el formato");
   printf("nombre.txt\n");
   scanf("%s", NAME);
   INP = fopen(NAME, "r");
   *OK = false;
   while (!(*OK)) {
      printf("Dar valor de n\n");
      scanf("%d", N);
      if (*N > 0) {
```

```
         for (I=0; I<=*N; I++)
            fscanf(INP, "%lf %lf", &X[I], &A[I]);
         fclose(INP);
         *OK = true;
      }
      else printf("Debe ser un entero positivo\n");
   }
}
void OUTPUT(int N, int M, double *X, double *A, double *B, double *C, double *D)
{
   int I;
   char NAME[30];
   FILE *OUP;
   printf("Dar el nombre del archivo de salida en el formato:\n");
   printf("nombre.txt\n");
   scanf("%s", NAME);
   OUP = fopen(NAME, "w");
   for (I=0; I<=N; I++) fprintf(OUP, " %11.8f", X[I]);
   fprintf(OUP,"\n\nLos coeficientes son\n");
   fprintf(OUP, "para I = 0, ..., N-1\n");
   fprintf(OUP, "     A(I)          B(I)           C(I)         D(I)\n");
   for (I=0; I<=M; I++)
      fprintf(OUP, " %13.8f %13.8f %13.8f %13.8f \n", A[I], B[I], C[I], D[I]);
   fclose(OUP);
}
```

Algoritmo de splines sujetos

Obtiene los coeficientes de los polinomios interpolantes $S_i(x)$ de la función f definida en los puntos x_0, x_1, . . . ,x_n, con $S'(x_0) = f'(x_0)$ y $S'(x_n) = f'(x_n)$.

ENTRADA n, x_i, $f(x_i)$ identificados por a_i, $f'(x_0) = FP0$, $f'(x_n) = FPN$.

SALIDA a_j, b_j, c_j, d_j, coeficientes de los polinomios

$$S_j(x) = a_j + b_j(x - x_j) + c_j(x - x_j)^2 + d_j(x - x_j)^3$$

definidos en $x_j \leq x \leq x_{j+1}$.

Paso 1 Para $i = 0, ..., n-1$ hacer $h_i = x_{i+1} - x_i$

Paso 2 Hacer $\alpha_0 = \frac{3(a_1 - a_0)}{h_0} - 3FP0$, $\alpha_n = 3FPN - \frac{3(a_n - a_{n-1})}{h_{n-1}}$

Paso 3 Para $i = 1, ..., n-1$ hacer

Paso 4 Hacer $\alpha_i = \frac{3}{h_i}(a_{i+1} - a_i) - \frac{3}{h_{i-1}}(a_i - a_{i-1})$

Paso 5 Hacer $\ell_0 = 2h_0$, $\mu_0 = 0.5$, $z_0 = \alpha_0/\ell_0$

Paso 6 Para $i = 1, ..., n-1$ hacer pasos 7 al 9

Paso 7 Hacer $\ell_i = 2(x_{i+1} - x_{i-1}) - h_{i-1}\mu_{i-1}$

Paso 8 Hacer $\mu_i = \frac{h_i}{\ell_i}$

Paso 9 Hacer $z_i = \frac{\alpha_i - h_{i-1}z_{i-1}}{\ell_i}$

Paso 10 Hacer $\ell_n = h_{n-1}(2 - \mu_{n-1})$

Paso 11 Hacer $z_n = \frac{\alpha_n - h_{n-1}z_{n-1}}{\ell_n}$

Paso 12 Hacer $c_n = z_n$

Paso 13 Para $j = n-1, ..., 0$ hacer pasos 14 al 16

Paso 14 Hacer $c_j = z_j - \mu_j c_{j+1}$

Paso 15 $b_j = \frac{a_{j+1} - a_j}{h_j} - \frac{h_j(c_{j+1} + 2c_j)}{3}$

Paso 16 $d_j = \frac{c_{j+1} - c_j}{3h_j}$

Paso 17 SALIDA (Los coeficientes son a_j, b_j, c_j, d_j)

TERMINAR

Código de interpolación con splines sujetos

El siguiente ejemplo calcula los coeficientes de los *splines* cúbicos, usando como entrada un conjunto de n datos en un archivo de texto, que debe tener como formato en cada renglón: "x[tabulador]y"; además de los valores de la derivada en los extremos del intervalo de interés dados por teclado durante la corrida. Los resultados los escribe en un archivo de texto.

```
#include<stdio.h>
#include<math.h>
void INPUT(int *, double *, double *, int *, double *, double *);
void OUTPUT(int, int, double *, double *, double *, double *, double *);
main()
{
   double X[26],A[26],B[26],C[26],D[26],H[26],XA[26],XL[26],XU[26],XZ[26];
   double XX,S,Y, FP0, FPN;
   int I,J,N,M,OK;
   INPUT(&OK, X, A, &N, &FP0, &FPN);
   if (OK) {
      M = N - 1;
      for (I=0; I<=M; I++) H[I] = X[I+1] - X[I];
      XA[0] = 3.0 * (A[1] - A[0]) / H[0] - 3.0 * FP0;
      XA[N] = 3.0 * FPN - 3.0 * (A[N] - A[N-1]) / H[N-1];
      for (I=1; I<=M; I++)
         XA[I] = 3.0 * (A[I+1] * H[I-1] - A[I] *
                  (X[I+1] - X[I-1]) + A[I-1] * H[I])  /
                  (H[I] * H[I-1]);
      XL[0] = 2.0 * H[0]; XU[0] = 0.5; XZ[0] = XA[0] / XL[0];
      for (I=1; I<=M; I++) {
         XL[I] = 2.0 * (X[I+1] - X[I-1]) - H[I-1] * XU[I-1];
         XU[I] = H[I] / XL[I];
         XZ[I] = (XA[I] - H[I-1] * XZ[I-1]) / XL[I];
      }
      XL[N] = H[N-1] * (2.0 - XU[N-1]);
      XZ[N] = (XA[N] - H[N-1] * XZ[N-1]) / XL[N];
      C[N] = XZ[N];
      for (I=1; I<=N; I++) {
         J = N - I;
```

```
         C[J] = XZ[J] - XU[J] * C[J+1];
         B[J] = (A[J+1] - A[J]) / H[J] - H[J] * (C[J+1] + 2.0 * C[J]) / 3.0;
         D[J] = (C[J+1] - C[J]) / (3.0 * H[J]);
      }
      OUTPUT(N, M, X, A, B, C, D);
   }
   return 0;
}
void INPUT(int *OK, double *X, double *A, int *N, double *FP0, double *FPN)
{
   int I;
   char NAME[30];
   FILE *INP;
            printf("Dar el nombre de archivo en el formato:");
            printf("nombre.txt\n");
            scanf("%s", NAME);
            INP = fopen(NAME, "r");
            *OK = false;
            while (!(*OK)) {
               printf("Dar valor de n\n");
               scanf("%d", N);
               if (*N > 0) {
                  for (I=0; I<=*N; I++)
                     fscanf(INP, "%lf %lf", &X[I], &A[I]);
                  fclose(INP);
                  *OK = true;
               }
               else printf("Debe ser un entero positivo\n");
            }
   if ( *OK ) {
      printf ("Dar F'(X(0)) y F'(X(N)) separados por un espacio\n");
      scanf("%lf %lf", FP0, FPN);
   }
}
void OUTPUT(int N, int M, double *X, double *A, double *B, double *C, double *D)
{
   int I;
   char NAME[30];
   FILE *OUP;
   printf("Dar nombre del archivo de salida en el formato\n");
   printf("nombre.txt\n");
   scanf("%s", NAME);
   OUP = fopen(NAME, "w");
   for (I=0; I<=N; I++) fprintf(OUP, " %11.8f", X[I]);
   fprintf(OUP,"\n\nLos coeficientes son\n");
   fprintf(OUP, "para I = 0, ..., N-1\n");
   fprintf(OUP, "     A(I)          B(I)           C(I)         D(I)\n");
   for (I=0; I<=M; I++)
      fprintf(OUP, " %13.8f %13.8f %13.8f %13.8f\n", A[I], B[I], C[I], D[I]);
   fclose(OUP);
}
```

Ejercicios

Para los conjuntos de puntos siguientes, construir los polinomios interpolantes por cada uno de los métodos estudiados en este capítulo y calcular el valor de la función en los puntos indicados. Graficar y comparar en cada caso la precisión de la interpolación.

1. Función

   $f(0.4) = 0.97354585$, $f(0.8) = 0.87036324$, $f(1.4) = 0.70389271$, $f(1.9) = 0.49805269$, $f(2.4) = 0.28144297$, $f(2.9) = 0.08249972$, $f(3.4) = -0.07515916$, $f(3.9) = -0.17635031$, $f(4.4) = -0.21627320$, $f(4.9) = -0.20050052$, $f(5.4) = -0.14310452$

   Puntos a interpolar

   $f(0.5)$, $f(1.5)$, $f(2.6)$, $f(3.5)$, $f(4.8)$

2. Función

   $f(0.0) = 1.00000000$, $f(0.1) = 1.1668506$, $f(0.2) = 1.2312561$, $f(0.3) = 1.1326472$, $f(0.4) = 0.80644190$, $f(0.5) = 0.19228365$, $f(0.6) = -0.75433779$, $f(0.7) = -2.0472527$, $f(0.8) = -3.6523356$, $f(0.9) = -5.4693198$, $f(1.0) = -7.3151121$

   Puntos a interpolar

   $f(0.05)$, $f(0.25)$, $f(0.32)$, $f(0.64)$, $f(0.88)$

3. Función

   $f(0.8) = 1.2806249$, $f(1.6) = 1.8867962$, $f(2.4) = 2.6000001$, $f(3.2) = 3.3526111$, $f(4.0) = 4.1231055$, $f(4.8) = 4.9030604$, $f(5.6) = 5.6885853$, $f(6.4) = 6.4776545$, $f(7.2) = 7.2691135$, $f(8.0) = 8.0622587$

   Puntos a interpolar

   $f(1.0)$, $f(2.0)$, $f(3.0)$, $f(6.0)$, $f(7.0)$

4. Función

   $f(0.8) = 1.3104793$, $f(1.6) = 1.4140629$, $f(2.4) = 1.2943968$, $f(3.2) = 0.97037405$, $f(4.0) = 0.49315059$, $f(4.8) = 0.06193067$, $f(5.6) = 0.60723442$, $f(6.4) = 1.0566692$, $f(7.2) = 1.3392791$, $f(8.0) = 1.4104460$

   Puntos a interpolar

   $f(1.2)$, $f(2.8)$, $f(3.5)$, $f(5.0)$, $f(7.5)$

5. Función

   $f(0.6) = 1.8141849$, $f(1.4) = 1.0432273$, $f(2.0) = 1.3729484$, $f(2.8) = 2.1105540$, $f(3.5) = 1.4031190$, $f(4.1) = -0.390864$, $f(4.8) = -0.912879$, $f(5.5) = 0.5717366$, $f(6.2) = 2.0162477$, $f(6.9) = 1.7937250$

   Puntos a interpolar

   $f(1.5)$, $f(3.0)$, $f(4.5)$, $f(6.0)$, $f(6.5)$

6. Función

   $f(1.1) = 0.1048412$, $f(1.7) = 0.9020680$, $f(2.3) = 1.9156914$, $f(2.9) = 3.0876613$, $f(3.5) = 4.3846703$, $f(4.0) = 5.7850466$, $f(4.6) = 7.2735434$, $f(5.2) = 8.8388453$, $f(5.8) = 10.472218$, $f(6.4) = 12.166713$

   Puntos a interpolar

   $f(1.5)$, $f(2.0)$, $f(3.0)$, $f(4.5)$, $f(6.0)$

7. Función

   $f(0.5) = 0.11673335$, $f(1.2) = 0.96874416$, $f(1.6) = -0.06534098$, $f(2.1) = -0.11674739$, $f(2.5) = -0.98229235$, $f(3.0) = -0.69182622$, $f(3.4) = -0.94116682$, $f(3.9) = -0.27498683$, $f(4.3) = 0.08018288$, $f(4.8) = 0.12955102$

   Puntos a interpolar

   $f(1.1)$, $f(3.3)$, $f(3.8)$, $f(4.0)$, $f(4.5)$

8. Función

   $f(1.2) = 1.2509235$, $f(1.9) = 3.2814288$, $f(2.7) = 1.3315443$, $f(3.4) = 1.0966663$, $f(4.1) = 13.400121$, $f(4.9) = 23.144955$, $f(5.6) = 9.8527460$, $f(6.4) = 1.1469774$, $f(7.1) = 32.654453$, $f(7.9) = 62.621590$

   Puntos a interpolar

   $f(1.8)$, $f(3.1)$, $f(4.6)$, $f(6.6)$, $f(7.3)$

9. Función

   $f(1.0) = 1.00000000$, $f(2.1) = 1.1668506$, $f(3.2) = 1.2312561$, $f(4.3) = 1.1326472$, $f(5.4) = 0.80644190$, $f(6.5) = 0.19228365$, $f(7.6) = -0.75433779$, $f(8.7) = -2.0472527$, $f(9.8) = -3.6523356$, $f(10.9) = -5.4693198$, $f(12.0) = -7.3151121$

   Puntos a interpolar

   $f(1.8)$, $f(3.9)$, $f(5.7)$, $f(7.5)$, $f(9.4)$

10. Función

    $f(1.2) = 1.2509235$, $f(1.6) = 3.2814288$, $f(2.5) = 1.3315443$, $f(3.5) = 1.0966663$, $f(4.4) = 13.400121$, $f(4.8) = 23.144955$, $f(5.6) = 9.8527460$, $f(6.5) = 1.1469774$, $f(7.3) = 32.654453$, $f(7.9) = 62.621590$

    Puntos a interpolar

    $f(1.5)$, $f(2.6)$, $f(3.7)$, $f(5.9)$, $f(7.6)$

11. Resolver el siguiente ejercicio usando cada uno de los métodos estudiados. Comparar las respuestas obtenidas en cada uno y decidir cuál método de interpolación es el más conveniente para este caso.

    Se toman lecturas de velocidad y tiempos a un automóvil viajando por una carretera recta, en ciertas posiciones (distancias) previamente elegidas. Los resultados se dan en la siguiente tabla:

Tiempo (s)	Distancia (m)	Velocidad (m/s)
0	0	75
3	75	77
5	120	80
8	200	74
13	315	72

    a) Interpolar un polinomio y encontrar la posición del auto en $t = 10$ s.

    b) Usar la derivada del polinomio interpolado para ver si el auto excede el límite de velocidad de 90 km/h. Si es así, encontrar el intervalo de tiempo donde eso ocurre.

    c) ¿Cuál es la velocidad máxima del auto?

Capítulo 3

Integración y derivación

En este capítulo se trata el problema de encontrar la integral de una función cuya antiderivada no se puede calcular, así como el de aproximar la derivada de una función dada en forma tabular en lugar de en forma analítica. Existen varios métodos para hacer dicha aproximación. Aquí se examinan los métodos de rectángulos, trapecios y de Simpson para una variable, así como rectángulos para la integral doble. Asimismo, se trata el problema de aproximar las derivadas primera y segunda de una función dada por medio de una tabla.

3.1. Integración por medio de rectángulos

En el curso de cálculo integral se enseña que la integral definida de una función continua $f(x)$ entre a y b se encuentra por medio de la fórmula

$$\int_a^b f(x)\ dx = F(b) - F(a), \tag{3.1}$$

donde $F(x)$ es una función (llamada *primitiva*) cuya derivada es $f(x)$. Sin embargo, la mayoría de las funciones que se pueden formar combinando las llamadas *funciones elementales*, no son derivadas de alguna función. Es por eso que la fórmula anterior sólo sirve en un número limitado de casos. Para cualquier otra función es necesario hacer alguna aproximación numérica.

El área bajo una curva es aproximadamente

$$\int_a^b f(x)\ dx \approx \frac{b-a}{n}(y_0 + y_1 + y_2 + \cdots + y_{n-1}), \tag{3.2}$$

o bien

$$\int_a^b f(x)\ dx \approx \frac{b-a}{n}(y_1 + y_2 + \cdots + y_n), \tag{3.3}$$

así que, si no calculamos el límite cuando $n \to \infty$, pero usamos un número suficiente de rectángulos, podemos hallar una aproximación con la precisión necesaria.

Para aproximar integrales por medio de rectángulos podemos usar el algoritmo que se presenta enseguida.

Algoritmo de integración por rectángulos

Obtiene un valor aproximado para la integral

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

usando n rectángulos de bases equidistantes.

ENTRADA $f(x)$, a, b, n.

SALIDA Valor aproximado de la integral.

Paso 1 Hacer $h = \frac{b-a}{n}$

Paso 2 Hacer $int = 0$, $x = a$

Paso 3 Para i desde 1 hasta n hacer pasos 4 al 6

Paso 4 Hacer $y = f(x)$

Paso 5 Hacer $int = int + h \cdot y$

Paso 6 Hacer $x = x + h$

Paso 7 SALIDA (La integral vale: int)

TERMINAR

A continuación se da el código para este algoritmo.

Código de integración con rectángulos

En este ejemplo se calcula la integral

$$\int_a^b \frac{1}{x}\ dx.$$

```
#include<stdio.h>
#include<math.h>
   double F(double);
   void INPUT(int *, double *, double *, int *);
   void OUTPUT(double, double, double);
main() {
   double A,B,XI0,XI1,XI2,H,XI,X;
   int I,N,NN,OK;
   INPUT(&OK, &A, &B, &N);
   if (OK) {
      H = (B - A) / N;
      XI0 = F(B);
      XI1 = 0.0;
      XI2 = 0.0;
      NN = N - 1;
      for (I=1; I<=NN; I++) {
     X = A + I * H;
     if ((I%2) == 0) XI2 = XI2 + F(X);
     else XI1 = XI1 + F(X);
      }
      XI = (XI0 + XI2 + XI1) * H ;
      OUTPUT(A, B, XI);
   }
```

```
   return 0;
}
double F(double X)
{
   double f;
   f = 1/X;
   return f;
}
void INPUT(int *OK, double *A, double *B, int *N)
{
   char AA;
      *OK = false;
      while (!(*OK)) {
     printf("Dar el limite de integración inferior A\n");
     scanf("%lf", A);
     printf("Dar el limite de integración superior B\n");
     scanf("%lf", B);
     if (*A > *B) printf("A debe ser menor que B\n");
     else *OK = true;
      }
      *OK = false;
      while (!(*OK)) {
     printf("Dar el numero de divisiones para la particion\n");
     scanf("%d", N);
     if ((*N > 0) && (((*N)%2) == 0)) *OK = true;
     else printf("Debe ser un numero natural par\n");
      }
}
void OUTPUT(double A, double B, double XI)
{
   printf("La integral de F desde %12.8f hasta %12.8f es\n", A, B);
   printf("%12.8f\n", XI);
   getchar();
   getchar();
}
```

3.2. Integración por trapecios

Si en lugar de usar como figura de aproximación un rectángulo usamos un trapecio, tendremos una mejor aproximación a la integral definida buscada. En este caso las alturas serán los puntos medios de los subintervalos usados, esto es $y_i = \frac{x_{i-1}+x_i}{2}$, con lo cual la integral es aproximadamente

$$\int_a^b f(x)\,dx \approx \frac{b-a}{n}\left(\frac{y_0+y_n}{2} + y_1 + y_2 + \cdots + y_{n-1}\right). \tag{3.4}$$

Algoritmo de integración con trapecios

Obtiene un valor aproximado para la integral

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

usando n trapecios de bases equidistantes.

ENTRADA $f(x)$, a, b, n.

SALIDA Valor aproximado de la integral.

Paso 1 Hacer $h = \frac{b-a}{n}$

Paso 2 Hacer $int = h \cdot \frac{f(a)+f(b)}{2}$, $x = a + h$

Paso 3 Para i desde 1 hasta $n-1$ hacer pasos 4 al 6

Paso 4 Hacer $y = f(x)$

Paso 5 Hacer $int = int + h \cdot y$

Paso 6 Hacer $x = x + h$

Paso 7 SALIDA (La integral vale: int)

TERMINAR

Código de integración con trapecios

En este ejemplo se calcula la integral

$$\int_a^b \frac{\sqrt{x}}{x-1}\ dx.$$

```
#include<stdio.h>
#include<math.h>
   double F(double);
   void INPUT(int *, double *, double *, int *);
   void OUTPUT(double, double, double);
main() {
   double A,B,XI0,XI1,XI2,H,XI,X;
   int I,N,NN,OK;
   INPUT(&OK, &A, &B, &N);
   if (OK) {
      H = (B - A) / N;
      XI0 = F(A) + F(B);
      XI1 = 0.0;
      XI2 = 0.0;
      NN = N - 1;
      for (I=1; I<=NN; I++) {
     X = A + I * H;
     if ((I%2) == 0) XI2 = XI2 + F(X);
     else XI1 = XI1 + F(X);
      }
      XI = (XI0/2 + XI2 + XI1) * H ;
      OUTPUT(A, B, XI);
   }
   return 0;
```

```
}
double F(double X)
{
   double f;
   f = sqrt(X)/(X-1);
   return f;
}
void INPUT(int *OK, double *A, double *B, int *N)
{
      *OK = false;
      while (!(*OK)) {
     printf("Dar el limite de integracion inferior A\n");
     scanf("%lf", A);
     printf("Dar el limite de integracion superior B\n");
     scanf("%lf", B);
     if (*A > *B) printf("A debe ser menor que B\n");
     else *OK = true;
      }
      *OK = false;
      while (!(*OK)) {
     printf("Dar el numero de divisiones para la particion\n");
     scanf("%d", N);
     if ((*N > 0) && (((*N)%2) == 0)) *OK = true;
     else printf("Debe ser un numero natural par\n");
      }
}
void OUTPUT(double A, double B, double XI)
{
   printf("La integral de F desde %12.8f hasta %12.8f es\n", A, B);
   printf("%12.8f\n", XI);
   getchar();
   getchar();
}
```

3.3. Métodos de Simpson

El área bajo una parábola que pasa por los puntos (x_{i-1}, y_{i-1}), (x_i, y_i) y (x_{i+1}, y_{i+1}) es

$$A_i = \frac{\Delta x}{3}(y_{i-1} + 4y_i + y_{i+1}), \tag{3.5}$$

así que si sumamos sobre un número PAR (o sea, $n = 2m$) de subintervalos obtendremos

$$\int_a^b f(x)\ dx \approx \frac{b-a}{6m}[y_0 + y_{2m} + 2(y_2 + y_4 + ... + y_{2m}) + 4(y_1 + y_3 + ... + y_{2m-1})]. \tag{3.6}$$

Este método se debe a Thomas Simpson, por lo que la fórmula (3.6) también es conocida como *fórmula de 1/3 de Simpson.*

Algoritmo de integración de 1/3 de Simpson

Obtiene un valor aproximado de la integral

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

usando el método de 1/3 de Simpson en $2n$ puntos equidistantes.

ENTRADA $f(x)$, a, b, n.

SALIDA Valor aproximado de la integral.

Paso 1 Hacer $h = \frac{b-a}{2n}$

Paso 2 Hacer $int = (f(a) + f(b)) \cdot h$, $x = a$

Paso 3 Hacer $suma_1 = 0$, $suma_2 = 0$

Paso 4 Hacer $x = a + h$

Paso 5 Para i desde 1 hasta n hacer los pasos 6 al 8

- Paso 6 Hacer $y = f(x)$
- Paso 7 Hacer $suma_1 = suma_1 + y$
- Paso 8 Hacer $x = x + 2h$

Paso 9 Hacer $x = a + 2h$

Paso 10 Para i desde 1 hasta n hacer pasos 11 al 13

- Paso 11 Hacer $y = f(x)$
- Paso 12 Hacer $suma_2 = suma_2 + y$
- Paso 13 Hacer $x = x + 2h$

Paso 14 Hacer $int = \frac{1}{3}(int + 4h \cdot suma_1 + 2h \cdot suma_2)$

Paso 15 SALIDA (La integral vale: int)

TERMINAR

Código de integración de 1/3 de Simpson

En este ejemplo se calcula la integral

$$\int_a^b \sqrt{\operatorname{sen} x - x}\ dx.$$

```
/*Metodo de Homero Simpson*/
#include<stdio.h>
#include<math.h>
   double F(double);
   void INPUT(int *, double *, double *, int *);
   void OUTPUT(double, double, double);
main() {
```

```
   double A,B,XI0,XI1,XI2,H,XI,X;
   int I,N,NN,OK;
   INPUT(&OK, &A, &B, &N);
   if (OK) {
      H = (B - A) / N;
      XI0 = F(A) + F(B);
      XI1 = 0.0;
      XI2 = 0.0;
      NN = N - 1;
      for (I=1; I<=NN; I++) {
     X = A + I * H;
     if ((I%2) == 0) XI2 = XI2 + F(X);
     else XI1 = XI1 + F(X);
      }
      XI = (XI0 + 2.0 * XI2 + 4.0 * XI1) * H / 3.0;
      OUTPUT(A, B, XI);
   }
   return 0;
}
double F(double X)
{
   double f;
   f = sqrt(sin(X)-X);
   return f;
}
void INPUT(int *OK, double *A, double *B, int *N)
{
      *OK = false;
      while (!(*OK)) {
     printf("Dar el límite de integracion inferior A\n");
     scanf("%lf", A);
     printf("Dar el límite de integracion superior B\n");
     scanf("%lf", B);
     if (*A > *B) printf("A debe ser menor que B\n");
     else *OK = true;
      }
      *OK = false;
      while (!(*OK)) {
     printf("Dar el numero de divisiones para la particion\n");
     scanf("%d", N);
     if ((*N > 0) && (((*N)%2) == 0)) *OK = true;
     else printf("Debe ser un numero natural par\n");
      }
}
void OUTPUT(double A, double B, double XI)
{
   printf("La integral de F desde %12.8f hasta %12.8f es\n", A, B);
   printf("%12.8f\n", XI);
   getchar();
   getchar();
}
```

Si en lugar de usar segmentos de parábolas cada tres puntos usamos segmentos de polinomio cúbico cada cuatro puntos, obtendremos la llamada *fórmula de 3/8 de Simpson* dada abajo

$$\int_a^b f(x)\,dx \approx \frac{3}{8}h(f_0+f_n)+\frac{9}{8}h(f_1+f_4+...+f_{n-2})+ \tag{3.7}$$
$$+\frac{9}{8}h(f_2+f_5+...+f_{n-1})+\frac{3}{4}h(f_3+f_6+\cdots+f_{n-3}).$$

Algoritmo de integración de 3/8 de Simpson

Obtiene un valor aproximado de la integral

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

usando el método de 3/8 de Simpson en $3n$ puntos equidistantes.

ENTRADA $f(x)$, a, b, n.

SALIDA Valor aproximado de la integral.

Paso 1 Hacer $h = \frac{b-a}{3n}$

Paso 2 Hacer $int = (f(a)+f(b))\cdot h$, $x = a$

Paso 3 Hacer $suma_1 = 0$, $suma_2 = 0$, $suma_3 = 0$

Paso 4 Hacer $x = a+h$

Paso 5 Para i desde 1 hasta n hacer los pasos 6 al 8

- Paso 6 Hacer $y = f(x)$
- Paso 7 Hacer $suma_1 = suma_1 + y$
- Paso 8 Hacer $x = x+2h$

Paso 9 Hacer $x = a+2h$

Paso 10 Para i desde 1 hasta n hacer pasos 11 al 13

- Paso 11 Hacer $y = f(x)$
- Paso 12 Hacer $suma_2 = suma_2 + y$
- Paso 13 Hacer $x = x+2h$

Paso 14 Hacer $x = a+3h$

Paso 15 Para i desde 1 hasta n hacer pasos 16 al 18

- Paso 16 Hacer $y = f(x)$
- Paso 17 Hacer $suma_3 = suma_3 + y$
- Paso 18 Hacer $x = x+3h$

Paso 19 Hacer $int = \frac{3}{8}h(int + 3suma_1 + 3suma_2 + \frac{1}{2}suma_3)$

Paso 20 SALIDA (La integral vale: int)

TERMINAR

Código de integración de 3/8 de Simpson

En este ejemplo se calcula la integral

$$\int_a^b \sqrt{\operatorname{sen}^2 x + 1}\, dx.$$

```
/* Metodo de Homero Simpson */
#include<stdio.h>
#include<math.h>
   double F(double);
   void INPUT(int *, double *, double *, int *);
   void OUTPUT(double, double, double);
main() {
   double A,B,XI0,XI1,XI2,H,XI,X;
   int I,N,NN,OK;
   INPUT(&OK, &A, &B, &N);
   if (OK) {
      H = (B - A) / N;
      XI0 = F(A) + F(B);
      XI1 = 0.0;
      XI2 = 0.0;
      NN = N - 1;
      for (I=1; I<=NN; I++) {
     X = A + I * H;
     if ((I%2) == 0) XI2 = XI2 + F(X);
     else XI1 = XI1 + F(X);
      }
      XI = (XI0 + 2.0 * XI2 + 4.0 * XI1) * H / 3.0;
      OUTPUT(A, B, XI);
   }
   return 0;
}
double F(double X)
{
   double f;
   f = sqrt(sin(X)*sin(X)+1);
   return f;
}
void INPUT(int *OK, double *A, double *B, int *N)
{
      *OK = false;
      while (!(*OK)) {
     printf("Dar el limite de integracion inferior A\n");
     scanf("%lf", A);
     printf("Dar el limite de integracion superior B\n");
     scanf("%lf", B);
     if (*A > *B) printf("A debe ser menor que B\n");
     else *OK = true;
      }
      *OK = false;
      while (!(*OK)) {
     printf("Dar el numero de divisiones para la particion\n");
     scanf("%d", N);
```

```
      if ((*N > 0) && (((*N)%2) == 0)) *OK = true;
      else printf("Debe ser un numero natural par\n");
       }
}
void OUTPUT(double A, double B, double XI)
{
   printf("La integral de F desde %12.8f hasta %12.8f es\n", A, B);
   printf("%12.8f\n", XI);
   getchar();
   getchar();
}
}
```

Ejercicios

Aproximar numéricamente las siguientes integrales definidas usando los métodos estudiados, con un mismo valor de n para cada método. Comparar en cada caso la precisión de cada método.

1. $\int_1^5 \frac{1}{x}\, dx$ R: 1.609438
2. $\int_1^4 x^3\, dx$ R: 63.75000
3. $\int_0^1 \sqrt{1-x^3}\, dx$ R: 0.841299
4. $\int_1^3 \frac{1}{2x-1}\, dx$ R: 0.804719
5. $\int_1^{10} \log\left(\frac{1}{x}\right)\, dx$ R: -14.025844
6. $\int_0^1 \frac{4}{1+x^2}\, dx$ R: 3.141593
7. $\int_1^\pi \frac{\operatorname{sen} x}{x}\, dx$ R: 0.905854
8. $\int_1^\pi \frac{\cos x}{x}\, dx$ R: -0.263736
9. $\int_0^1 \sqrt{1+x^3}\, dx$ R: 1.111448
10. $\int_0^1 e^{-x^3}\, dx$ R: 0.807511
11. $\int_1^2 \ln x^3\, dx$ R: 1.158882
12. $\int_1^2 \ln^3 x\, dx$ R: 0.101097
13. $\int_{-1}^1 \left(x^3 e^x\right)^5\, dx$ R: 7.145648
14. $\int_0^2 \frac{1}{x^4+4}\, dx$ R: 0.354895
15. $\int_0^\pi (x^2 \cos x^2)^2\, dx$ R: 33.751135
16. $\int_0^2 e^{-x^2} \operatorname{sen} x^3\, dx$ R: 0.199815
17. $\int_1^3 \frac{x^{3/2}}{\sqrt{x^{2/3}+4}}\, dx$ R: 2.446318
18. $\int_3^5 \sqrt{x^{2/5}-4}\, dx$ R: 66.360315
19. $\int_0^{\pi/8} \operatorname{tg}^3 x^2\, dx$ R: 0.0343207
20. $\int_0^2 \frac{1}{x^{3/4}+4}\, dx$ R: 0.406686
21. $\int_3^4 \frac{x^{2/5}}{\sqrt{x^{2/3}+4}}\, dx$ R: 0.656753

22. $\int_0^{\pi/4}(x\cos x^2)^2\ dx$ R: 0.137204

23. $\int_1^2 \text{sen}\,\frac{1}{x}\ dx$ R: 0.632567

24. $\int_1^2 \cos\frac{1}{x}\ dx$ R: 0.761886

25. $\int_1^2 \left(\text{sen}\,\frac{1}{x^2}+\cos\frac{1}{x^2}\right)\ dx$ R: 1.328565

26. Se planea construir un arco de forma parabólica, dada por

$$y = 0.1x(30-x)$$

metros, con y la altura sobre el nivel del piso y x la posición horizontal, medida desde un extremo del arco. Calcular la longitud total del arco.

27. Una partícula de masa $m = 10$ kg se mueve a través de un fluido, sujeta a una resistencia R, que es función de su velocidad v. La relación entre la resistencia R, la velocidad v y el tiempo t está dada por la ecuación

$$t = \int_{v(t_0)}^{v(t)} \frac{m}{R(u)}\ du.$$

Suponiendo que $R(v) = -v^{3/2}$ para un fluido particular, donde R está en N y v en m/s, y suponiendo $v(0) = 10$ m/s, aproximar el tiempo requerido para que la partícula se frene hasta $v = 5$ m/s.

3.4. Integración doble

Para una integral doble

$$\int\int_R f(x,y)\ dA \tag{3.8}$$

en un rectángulo $R = \{(x,y)|\ a \le x \le b,\ c \le y \le d\}$, se puede calcular en forma iterada como

$$\int\int_R f(x,y)\ dA = \int_a^b \left(\int_c^d f(x,y)\ dy\right)\ dx \tag{3.9}$$

aplicando el método de Simpson (u otro) a cada integral obtenida.

Cuando la región de integración no es un rectángulo, sino que es de la forma

$$\int_a^b \int_{c(x)}^{d(x)} f(x,y)\ dy\ dx, \tag{3.10}$$

primero se transforma el intervalo $[c(x), d(x)]$ en cada x en el intervalo $[a,b]$, y luego se integra en una sola variable, haciendo después la suma correspondiente para la segunda integral.

Algoritmo de integración doble

Obtiene un aproximación para la integral

$$I = \int_a^b \int_{c(x)}^{d(x)} f(x,y)\ dy\ dx.$$

ENTRADA $f(x,y)$, $c(x)$, $d(x)$, a, b, enteros positivos pares m, n.

SALIDA Valor aproximado de I.

Paso 1 Hacer $h = \frac{b-a}{n}$, $J_1 = 0$, $J_2 = 0$, $J_3 = 0$

Paso 2 Para $i = 0, ..., n$ hacer pasos 3 al 8

Paso 3 Hacer $x = a + ih$

$HX = \frac{d(x)-c(x)}{m}$

$K_1 = f(x, c(x)) + f(x, d(x))$

$K_2 = 0$

$K_3 = 0$

Paso 4 Para $j = 1, ..., m-1$ hacer pasos 5 y 6

Paso 5 Hacer $y = c(x) + jHX$

$Q = f(x, y)$

Paso 6 Si j es par, entonces $K_2 = K_2 + Q$

Si no, entonces $K_3 = K_3 + Q$

Paso 7 Hacer $L = \frac{HX}{3}(K_1 + 2K_2 + 4K_3)$

Paso 8 Si $i = 0$ o $i = 1$, entonces hacer $J_1 = J_1 + L$

O bien, si i es par hacer $J_2 = J_2 + L$

si no, hacer $J_3 = J_3 + L$

Paso 9 Hacer $J = \frac{h}{3}(J_1 + 2J_2 + 4J_3)$

Paso 10 SALIDA (El valor de la integral es J)

TERMINAR

Código de integración doble en rectángulos

El siguiente ejemplo calcula la integral doble

$$I = \int_a^b \int_{x^3}^{x^2} \frac{y}{x}\, dy\, dx.$$

```
#include<stdio.h>
#include<math.h>
   double F(double, double);
   double C(double);
   double D(double);
   void INPUT(int *, double *, double *, int *, int *);
   void OUTPUT(double, double, double, int, int);
main() {
   double A,B,H,AN,AE,AO,X,HX,BN,YA,YB,BE,BO,Y,Z,A1,AC;
   int N,M,NN,MM,I,J,OK;
   INPUT(&OK, &A, &B, &N, &M);
```

```
   if (OK) {
      NN = 2 * N;
      MM = 2 * M - 1;
      H = (B - A) / NN;
      AN = 0.0;
      AE = 0.0;
      AO = 0.0;
      for (I=0; I<=NN; I++) {
     X = A + I * H;
     YA = C(X);
     YB = D(X);
     HX = (YB - YA) / (2.0 * M);
     BN = F(X, YA) + F(X, YB);
     BE = 0.0;
     BO = 0.0;
     for (J=1; J<=MM; J++) {
        Y = YA + J * HX;
        Z = F(X, Y);
        if ((J%2) == 0) BE = BE + Z;
        else BO = BO + Z;
     }
     A1 = (BN + 2.0 * BE + 4.0 * BO) * HX / 3.0;
     if ((I == 0) || (I == NN)) AN = AN + A1;
     else {
        if ((I%2) == 0) AE = AE + A1;
        else AO = AO + A1;
     }
      }
      AC = (AN + 2.0 * AE + 4.0 * AO) * H /3.0;
      OUTPUT(A, B, AC, N, M);
   }
   return 0;
}
double F(double X, double Y)
{
   double f;
   f = Y / X;
   return f;
}
double C(double X)
{
   double f;
   f = X * X * X;
   return f;
}
double D(double X)
{
   double f;
   f = X * X;
   return f;
}
void INPUT(int *OK, double *A, double *B, int *N, int *M)
{
   *OK = false;
```

```
   while (!(*OK)) {
   printf("Dar los limites de integracion");
   printf("separados por un espacio\n");
   scanf("%lf %lf", A, B);
   if (*A > *B) printf("El limite superior debe ser mayor que el inferior\n");
   else *OK = true;
   }
   *OK = false;
   while (!(*OK)) {
      printf("Dar el numero de intervalos en x\n");
      scanf("%d",N);
      printf("Dar el numero de intervalos en y\n");
      scanf("%d",M);
      if ((*M <= 0) || (*N <= 0)) printf("Deben ser enteros positivos\n");
      else *OK = true;
   }
}
void OUTPUT(double A, double B, double AC, int N, int M)
{
   printf("\nLa integral es:\n");
   printf("%12.8f", AC);
   getchar();
   getchar();
}
```

Ejercicios

1. $\int_0^3 \int_0^2 (4 - y^2)\ dy\, dx$ R: 16.000000
2. $\int_{-1}^0 \int_{-1}^1 (x + y + 1)\ dx\, dy$ R: 1.000000
3. $\int_0^\pi \int_0^x x \operatorname{sen} y\ dy\, dx$ R: 3.570796
4. $\int_1^{\ln 8} \int_0^{\ln y} e^{x+y}\ dx\, dy$ R: 16.635532
5. $\int_0^1 \int_0^{y^2} 3y^3 e^{xy}\ dx\, dy$ R: 0.718282
6. $\int_{-\pi/3}^{\pi/3} \int_0^{\sec y} 3 \cos y\ dx\, dy$ R: 6.283185
7. $\int_0^{1/16} \int_{y^{1/4}}^{1/2} \cos(16\pi x^5)\ dx\, dy$ R: 0.00397887
8. $\int_0^{2\sqrt{\ln 3}} \int_{y/2}^{\sqrt{\ln 3}} e^{x^2}\ dx\, dy$ R: 2.000000
9. $\int_0^\pi \int_x^\pi \frac{\operatorname{sen} y}{y}\ dy\, dx$ R: 2.000000
10. $\int_0^1 \int_y^1 x^2 e^{xy}\ dx\, dy$ R: 0.359141
11. $\int_0^1 \int_2^{4-2x}\ dy\, dx$ R: 1.000000
12. $\int_0^{\ln 2} \int_{e^x}^2\ dx\, dy$ R: 0.386294
13. $\int_0^{3/2} \int_0^{9-4x^2} 16x\ dy\, dx$ R: 81.000000
14. $\int_0^2 \int_0^{4-y^2} y\ dx\, dy$ R: 4.000000
15. $\int_0^1 \int_{-\sqrt{1-y^2}}^{\sqrt{1-y^2}} 3y\ dx\, dy$ R: 2.000000

16. $\int_0^2 \int_{-\sqrt{4-x^2}}^{\sqrt{4-x^2}} 6x \; dy\, dx$ R: 32.000000

17. $\int_0^2 \int_x^2 2y^2 \operatorname{sen} xy \; dy\, dx$ R: 4.756802

18. $\int_0^2 \int_0^{4-x^2} \frac{xe^{2y}}{4-y} \; dy\, dx$ R: 744.989497

19. $\int_0^3 \int_{\sqrt{x/3}}^1 e^{y^3} \; dy\, dx$ R: 1.718182

20. $\int_0^8 \int_{\sqrt[3]{x}}^2 \frac{1}{y^4+1} \; dy\, dx$ R: 0.708303

21. $\int\int_R (2x^2 - xy - y^2) \; dx \; dy$, con R la región del primer cuadrante encerrada por las rectas $y = -2x + 4$, $y = -2x + 7$, $y = x - 2$ y $y = x + 1$
    Transformación sugerida: $u = x - y$, $v = 2x + y$ R: 8.250000

22. $\int\int_R (3x^2 + 14xy + 8y^2) \; dx \; dy$, con R la región del primer cuadrante encerrada por las rectas $y = -(3/2)x + 1$, $y = -(3/2)x + 3$, $y = -(1/4)x$ y $y = -(1/4)x + 1$
    Transformación sugerida: $u = 3x + 2y$, $v = x + 4y$ R: 12.800000

23. $\int\int_R 2(x - y) \; dx \; dy$, con R la región cuyas fronteras son $x = -3$, $x = 0$, $y = x$ y $y = x + 1$
    Transformación sugerida: $u = 2x - 3y$, $v = -x + y$ R: -3.000000

24. $\int\int_R (x + y)^3 \; dx \; dy$, con R el paralelogramo con vértices (1,0), (3,1), (2,2), (0,1).
    Transformación sugerida: $u = x + y$, $v = x - 2y$ R: -63.750000

25. $\int\int_R (x - y)^2 \operatorname{sen}^2(x + y) \; dx \; dy$, con R el paralelogramo con vértices (π,0), ($2\pi, \pi$), ($\pi, 2\pi$), (0,π).
    Transformación sugerida: $u = x - y$, $v = x + y$ R: 32.469697

3.5. Derivación numérica

A diferencia del problema de la integración, la derivación siempre es factible para una función dada en forma analítica. El problema es que no siempre se tiene la función en forma analítica, sino que a veces sólo se tiene una tabla de valores. En tales casos es necesario derivar también en forma tabular.

Por definición, la derivada de una función $f(x)$ en el punto x_0 es

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h}, \tag{3.11}$$

por lo que, tomando h *suficientemente pequeña*, podremos aproximar la derivada. Nótese que si se tienen $n + 1$ puntos, al derivar se tendrá una tabla de sólo n puntos. El problema de aproximar la derivada con la razón de cambio finita es que a veces la h en la tabla no es *suficientemente pequeña*, lo cual puede hacer que el error de la aproximación sea mayor de lo deseable. Por ello se han desarrollado fórmulas para aproximar derivadas que usan más de dos puntos, a fin de reducir el error en la aproximación. Tales fórmulas usan desarrollos en series de potencias alrededor del punto x_0 y aproximan las derivadas en términos de los valores de la función en varios puntos alrededor del que nos interesa. Las fórmulas en cuestión son:

Para tres puntos:

$$f'(x_0) \approx \frac{1}{2h} \left[-3f(x_0) + 4f(x_0 + h) - f(x_0 + 2h) \right], \tag{3.12}$$

o bien

$$f'(x_0) \approx \frac{1}{2h}\left[f(x_0+h) - f(x_0-h)\right]. \tag{3.13}$$

Para cinco puntos:

$$f'(x_0) \approx \frac{1}{12h}\left[f(x_0-2h) - 8f(x_0-h) + 8f(xx_0+h) - f(x_0+2h)\right], \tag{3.14}$$

o bien

$$f'(x_0) \approx \frac{1}{12h}\left[-25f(x_0) + 48f(x_0+h) - 36f(x_0+2h) + 6f(x_0+3h) - 3f(x_0+4h)\right]. \tag{3.15}$$

Asimismo, cuando lo que se desea es aproximar la segunda derivada, es necesario usar la aproximación usando al menos tres puntos, en la forma

$$f''(x_0) \approx \frac{1}{h^2}\left[f(x_0-h) - 2f(x_0) + f(x_0+h)\right]. \tag{3.16}$$

Como podrá notar el lector, en este caso se pierden dos puntos en lugar de uno. Es decir, si se tienen $n+1$ puntos, se obtendrá la segunda derivada en $n-1$ puntos.

A continuación se da un algoritmo para calcular las derivadas primera y segunda usando dos y tres puntos, respectivamente.

Algoritmo de derivación numérica

Aproxima las derivadas de una función dada en forma tabular.

ENTRADA n, x_i, $f(x_i)$ en $n+1$ puntos.

SALIDA Derivadas $f'(x_i)$ y $f''(x_i)$ en $n-1$ puntos.

Paso 1 Para $i = 2, ..., n$ hacer pasos 2 al 5

Paso 2 Hacer $h = x_i - x_{i-1}$

Paso 3 Hacer $f'_i = \frac{f(x_i)-f(x_{i-1})}{h}$

Paso 4 Hacer $f''_i = \frac{f(x_{i+1})-2f(x_i+f(xi-1)}{h^2}$

Paso 5 Salida (f'_i,f''_i)

Paso 6 TERMINAR

Código de derivación tabular

El siguiente ejemplo calcula las derivadas primera y segunda de una función dada en forma tabular, usando como entrada un archivo de texto con formato: "x[tabulador]y". La salida se escribe en un archivo de texto con el formato: "x[tabulador]y[tabulador]y'[tabulador]y''".

```
#include<stdio.h>
   int i,n;
   float h;
   float x[100], fx[100], dx[100], ddx[100];
   char NAME1[30],NAME2[30];
```

```
   FILE *INP,*OUP;
main() {
   printf("Dar el nombre y la extension del archivo de entrada en el formato ");
   printf("nombre.txt\n");
   scanf("%s", NAME1);
   INP = fopen(NAME1, "r");
   printf("Dar valor de n\n");
   scanf("%d", &n);
      if (n > 0) {
          for (i=1; i<=n; i++)
            fscanf(INP, "%f\t%f\n", &x[i], &fx[i]);
            fclose(INP);
       }
        else printf("Debe ser un entero positivo\n");
   printf("Dar nombre de archivo de salida en el formato: archivo.txt\n");
   scanf("%s", NAME2);
   OUP = fopen(NAME2, "w");
      for (i=2; i<=n-1; i++) {
         h=x[i]-x[i-1];
         dx[i]=(fx[i]-fx[i-1])/h;
         ddx[i]=(fx[i+1]-2*fx[i]+fx[i+1])/(h*h);
         fprintf(OUP, "%f\t%f\t%f\t%f\n", x[i], fx[i], dx[i], ddx[i]);
   }
   fclose(OUP);
   return 0;
}
```

Ejercicios

Para los conjuntos de puntos siguientes, calcular las derivadas primera y segunda. Graficar los resultados obtenidos y comparar en cada caso con las gráficas dadas como soluciones.

1. $f(0.4) = 0.97354585$, $f(0.8) = 0.87036324$, $f(1.4) = 0.70389271$, $f(1.9) = 0.49805269$, $f(2.4) = 0.28144297$, $f(2.9) = 0.08249972$, $f(3.4) = -0.07515916$, $f(3.9) = -0.17635031$, $f(4.4) = -0.21627320$, $f(4.9) = -0.20050052$, $f(5.4) = -0.14310452$
2. $f(0.0) = 1.00000000$, $f(0.1) = 1.1668506$, $f(0.2) = 1.2312561$, $f(0.3) = 1.1326472$, $f(0.4) = 0.80644190$, $f(0.5) = 0.19228365$, $f(0.6) = -0.75433779$, $f(0.7) = -2.0472527$, $f(0.8) = -3.6523356$, $f(0.9) = -5.4693198$, $f(1.0) = -7.3151121$
3. $f(0.8) = 1.2806249$, $f(1.6) = 1.8867962$, $f(2.4) = 2.6000001$, $f(3.2) = 3.3526111$, $f(4.0) = 4.1231055$, $f(4.8) = 4.9030604$, $f(5.6) = 5.6885853$, $f(6.4) = 6.4776545$, $f(7.2) = 7.2691135$, $f(8.0) = 8.0622587$
4. $f(0.8) = 1.3104793$, $f(1.6) = 1.4140629$, $f(2.4) = 1.2943968$, $f(3.2) = 0.97037405$, $f(4.0) = 0.49315059$, $f(4.8) = 0.06193067$, $f(5.6) = 0.60723442$, $f(6.4) = 1.0566692$, $f(7.2) = 1.3392791$, $f(8.0) = 1.4104460$
5. $f(0.6) = 1.8141849$, $f(1.4) = 1.0432273$, $f(2.0) = 1.3729484$, $f(2.8) = 2.1105540$, $f(3.5) = 1.4031190$, $f(4.1) = -0.390864$, $f(4.8) = -0.912879$, $f(5.5) = 0.5717366$, $f(6.2) = 2.0162477$, $f(6.9) = 1.7937250$
6. $f(1.1) = 0.1048412$, $f(1.7) = 0.9020680$, $f(2.3) = 1.9156914$, $f(2.9) = 3.0876613$, $f(3.5) = 4.3846703$, $f(4.0) = 5.7850466$, $f(4.6) = 7.2735434$, $f(5.2) = 8.8388453$, $f(5.8) = 10.472218$, $f(6.4) = 12.166713$
7. $f(0.5) = 0.11673335$, $f(1.2) = 0.96874416$, $f(1.6) = -0.06534098$, $f(2.1) = -0.11674739$, $f(2.5) = -0.98229235$, $f(3.0) = -0.69182622$, $f(3.4) = -0.94116682$, $f(3.9) = -0.27498683$, $f(4.3) = 0.08018288$, $f(4.8) = 0.12955102$
8. $f(1.2) = 1.2509235$, $f(1.9) = 3.2814288$, $f(2.7) = 1.3315443$, $f(3.4) = 1.0966663$, $f(4.1) = 13.400121$, $f(4.9) = 23.144955$, $f(5.6) = 9.8527460$, $f(6.4) = 1.1469774$, $f(7.1) = 32.654453$, $f(7.9) = 62.621590$

9. $f(1.0) = 1.00000000$, $f(2.1) = 1.1668506$, $f(3.2) = 1.2312561$, $f(4.3) = 1.1326472$, $f(5.4) = 0.80644190$, $f(6.5) = 0.19228365$, $f(7.6) = -0.75433779$, $f(8.7) = -2.0472527$, $f(9.8) = -3.6523356$, $f(10.9) = -5.4693198$, $f(12.0) = -7.3151121$

10. $f(1.2) = 1.2509235$, $f(1.6) = 3.2814288$, $f(2.5) = 1.3315443$, $f(3.5) = 1.0966663$, $f(4.4) = 13.400121$, $f(4.8) = 23.144955$, $f(5.6) = 9.8527460$, $f(6.5) = 1.1469774$, $f(7.3) = 32.654453$, $f(7.9) = 62.621590$

11. Considerar una viga uniforme de 1 m de longitud, soportada en sus dos extremos, cuyo momento de torsión está dado por

$$y'' = \frac{M(x)}{EI},$$

con $y(x)$ la deflección, $M(x)$ el momento de torsión y EI la rigidez a la flexión. Calcular el momento de torsión en cada punto de la malla, incluyendo los extremos. Usar la distribución de deflección dada por la tabla siguiente:

i	x_i (m)	y_i (cm)
0	0.0	0.0
1	0.2	7.78
2	0.4	10.68
3	0.6	8.37
4	0.8	3.97
5	1.0	0.0

Suponer $EI = 1.2 \times 10^7$ Nm^2.

12. En un circuito con una fuerza electromotriz $E(t)$, resistencia $R = 0.142\ \Omega$ e inductancia L =0.98 H, la primera ley de Kirchoff da la relación

$$E(t) = L\frac{di}{dt} + Ri,$$

donde i es la corriente. Suponiendo que se mide la corriente para diferentes valores de t, obteniéndose la siguiente tabla, aproximar el valor de $E(t)$ cuando t =1.00, 1.01, 1.02, 1.03, 1.04 y 1.05 s.

t (s)	i (A)
0.0	0.0
0.2	7.78
0.4	10.68
0.6	8.37
0.8	3.97
1.0	0.0

Capítulo 4

Ecuaciones diferenciales ordinarias

Algunos métodos de integración de ecuaciones diferenciales ordinarias que se estudian en los cursos correspondientes sirven sólo para un espectro muy limitado de problemas. En general, la integración de una ecuación diferencial sólo puede hacerse en forma numérica en la mayoría de los casos. A ese problema dedicaremos este capítulo. Nos limitaremos al problema de ecuaciones diferenciales ordinarias con condiciones iniciales, es decir, a problemas de valor inicial.

4.1. Método de Euler

Si tenemos una ecuación diferencial de primer orden de la forma

$$\frac{dy}{dx} = f(x, y), \tag{4.1}$$

podemos escribirla en forma aproximada como

$$\frac{y_{i+1} - y_i}{x_{i+1} - x_i} \approx f(x_i, y_i). \tag{4.2}$$

De la anterior ecuación se desprende de inmediato la fórmula de recurrencia

$$y_{i+1} = y_i + hf(x_i, y_i), \tag{4.3}$$

con $h = x_{i+1} - x_i$.

Algoritmo de Euler

Obtiene una aproximación del problema de valor inicial

$$y' = f(x, y), \quad y(x_0) = y_0, \quad x \in [a, b].$$

ENTRADA $f(x, y)$, a, b, n, y_0.

SALIDA Aproximación a $y(x)$ en n valores de x equidistantes.

Paso 1 Hacer $h = (b - a)/n$, $x = a$, $y = y_0$

Paso 2 Para $i = 1, ..., n$ hacer pasos 3 al 5

Paso 3 Hacer $y = y + hf(x, y)$

Paso 4 Hacer $x = x + h$

Paso 5 SALIDA (La solución es (x, y))

Paso 6 TERMINAR

Código de Euler

Ejemplo de código de Euler para la ecuación

$$\frac{dy}{dt} = y - t^2 + 1.$$

```
#include<stdio.h>
#include<math.h>
   double F(double, double);
   void INPUT(int *, double *, double *, double *, int *);
   void OUTPUT(FILE **);
main() {
   double A,B,ALPHA,H,T,W;
   int I,N,OK;
   FILE *OUP;
   INPUT(&OK, &A, &B, &ALPHA, &N);
   if (OK) {
      OUTPUT(OUP);
      H = (B - A) / N;
      T = A;
      W = ALPHA;
      fprintf(*OUP, "%5.3f %11.7f\n", T, W);
      for (I=1; I<=N; I++) {
         W = W + H * F(T, W);
         T = A + I * H;
         fprintf(*OUP, "%5.3f %11.7f\n", T, W);
      }
      fclose(*OUP);
   }
   return 0;
}
double F(double T, double Y)
{
   double f;
   f = Y - T*T + 1.0;
   return f;
}
void INPUT(int *OK, double *A, double *B, double *ALPHA, int *N)
{
   double X;
      *OK = false;
      while (!(*OK)) {
         printf("Dar el valor del extremo A\n");
         scanf("%lf", A);
         printf("Dar el valor del extremo B\n");
         scanf("%lf", B);
         if (*A >= *B)
```

```
            printf("El valor de B debe ser mayor que el de A\n");
         else *OK = true;
      }
      printf("Dar la condicion inicial\n");
      scanf("%lf", ALPHA);
      *OK = false;
      while(!(*OK)) {
         printf("dar el numero de subintervalos\n");
         scanf("%d", N);
         if (*N <= 0.0) printf("Debe ser un numero natural\n");
         else *OK = true;
      }
}
void OUTPUT(FILE **OUP)
{
   char NAME[30];
   printf("Dar el nombre de un archivo para escribir la solucion\n");
   printf("Debe estar en el formato\n");
   printf("nombre.txt\n");
   scanf("%s", NAME);
   *OUP = fopen(NAME, "w");
   fprintf(*OUP, "Metodo de Euler\n\n");
   fprintf(*OUP, "    t           y\n\n");
   printf("Presionar enter para terminar el programa \n ");
   getchar();
   getchar();
}
```

4.2. Método de Euler mejorado

Si en la ecuación

$$\frac{dy}{dx} = f(x, y), \tag{4.4}$$

en lugar de aproximar la ecuación con

$$y_{i+1} - y_i = h\ f(x_i, y_i)$$

hacemos

$$y_{i+1} - y_i = \frac{h}{2}\left[f(x_i, y_i) + f(x_i + h, y_i)\right], \tag{4.5}$$

la aproximación se mejora de forma equivalente al aproximar la integral con trapecios en lugar de con rectángulos, según lo estudiado en el capítulo 3.

Algoritmo de Euler mejorado

Obtiene una aproximación del problema de valor inicial

$$y' = f(x, y), \quad y(x_0) = y_0, \quad x \in [a, b].$$

ENTRADA $f(x,y)$, a, b, n, y_0.

SALIDA Aproximación a $y(x)$ en n valores de x equidistantes.

Paso 1 Hacer $h = (b-a)/n$, $x = a$, $y = y_0$

Paso 2 Para $i = 1, ..., n$ hacer pasos 3 al 7

Paso 3 Hacer $k_1 = f(x,y)$

Paso 4 Hacer $k_2 = f(x+h, y+h\ k_1)$

Paso 5 Hacer $y = y + \frac{h}{2}(k_1 + k_2)$

Paso 6 Hacer $x = x + h$

Paso 7 SALIDA (La solución es (x,y))

Paso 8 TERMINAR

Código de Euler mejorado

Ejemplo de código de Euler mejorado para la ecuación

$$\frac{dy}{dt} = y - t^2 + 1.$$

```
#include<stdio.h>
#include<math.h>
   FILE *OUP;
   double F(double, double);
   void INPUT(int *, double *, double *, double *, int *);
   void OUTPUT(FILE **);
main() {
   double A,B,ALPHA,H,T,W;
   int I,N,OK;
   INPUT(&OK, &A, &B, &ALPHA, &N);
   if (OK) {
      OUTPUT(OUP);
      H = (B - A) / N;
      T = A;
      W = ALPHA;
      fprintf(*OUP, "%5.3f %11.7f\n", T, W);
      for (I=1; I<=N; I++) {
         W = W + H * F(T, W);
         T = A + I * H;
         fprintf(*OUP, "%5.3f %11.7f\n", T, W);
      }
      fclose(*OUP);
   }
   return 0;
}
double F(double T, double Y)
{
   double f;
   f = Y - T*T + 1.0;
   return f;
```

```
}
void INPUT(int *OK, double *A, double *B, double *ALPHA, int *N)
{
   double X;
      *OK = false;
      while (!(*OK)) {
         printf("Dar el valor del extremo A\n");
         scanf("%lf", A);
         printf("Dar el valor del extremo B\n");
         scanf("%lf", B);
         if (*A >= *B)
            printf("El valor de B debe ser mayor que el de A\n");
         else *OK = true;
      }
      printf("Dar la condicion inicial\n");
      scanf("%lf", ALPHA);
      *OK = false;
      while(!(*OK)) {
         printf("dar el numero de subintervalos\n");
         scanf("%d", N);
         if (*N <= 0.0) printf("Debe ser un numero natural\n");
         else *OK = true;
      }
}
void OUTPUT(FILE **OUP)
{
   char NAME[30];
   printf("Dar el nombre de un archivo para escribir la solucion\n");
   printf("Debe estar en el formato\n");

   printf("nombre.txt\n");
   scanf("%s", NAME);
   *OUP = fopen(NAME, "w");
   fprintf(*OUP, "Metodo de Euler mejorado\n\n");

   fprintf(*OUP, "    t           y\n\n");
   printf("Presionar enter para terminar el programa \n ");
   getchar();
   getchar();
}
```

4.3. Método de Runge-Kutta

En una forma equivalente a la del método de Simpson para integrales (capítulo 3), el método de Runge-Kutta aproxima la solución de la ecuación

$$\frac{dy}{dx} = f(x, y), \tag{4.6}$$

usando promedios pesados para los valores de $f(x, y)$ en diferentes puntos del intervalo $x_i \leq x \leq x_{i+1}$. La aproximación está dada por

$$y_{i+1} - y_i = \frac{h}{6}\left[k_1 + 2k_2 + 2k_3 + k_4\right], \tag{4.7}$$

donde

$$\begin{aligned} k_1 &= f(x_i, y_i) \\ k_2 &= f(x_i + h/2, y_i + (h/2)\ k_1) \\ k_3 &= f(x_i + h/2, y_i + (h/2)\ k_2) \\ k_4 &= f(x_i + h, y_i + hk_3). \end{aligned}$$

k_1 es la pendiente de la función en el extremo izquierdo del intervalo $[x_i, x_{i+1}]$, k_2 es la pendiente de x_i a $x_i + h/2$, k_3 es la pendiente de $x_i + h/2$ a x_{i+1} y k_4 es la pendiente en el extremo derecho del intervalo.

Algoritmo de Runge-Kutta de orden 4

Obtiene una aproximación del problema de valor inicial

$$y' = f(x, y), \quad y(x_0) = y_0, \quad x \in [a, b].$$

ENTRADA $f(x, y)$, a, b, n, y_0.

SALIDA Aproximación a $y(x)$ en n valores de x equidistantes.

Paso 1 Hacer $h = (b - a)/n$, $x = a$, $y = y_0$

Paso 2 Para $i = 1, ..., n$ hacer pasos 3 al 9

Paso 3 Hacer $k_1 = f(x, y)$

Paso 4 Hacer $k_2 = f(x + \frac{h}{2}, y + \frac{h}{2}k_1)$

Paso 5 Hacer $k_3 = f(x + \frac{h}{2}, y + \frac{h}{2}k_2)$

Paso 6 Hacer $k_4 = f(x + h, y + h\ k_3)$

Paso 7 Hacer $y = y + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

Paso 8 Hacer $x = x + h$

Paso 9 SALIDA $((x, y))$

Paso 10 TERMINAR

Código de Runge-Kutta de orden 4

Ejemplo de código de Runge-Kutta de orden 4 para la ecuación

$$\frac{dy}{dt} = y - t^2 + 1.$$

```
#include<stdio.h>
#include<math.h>
   FILE *OUP[1];
   double F(double, double);
   void INPUT(int *, double *, double *, double *, int *);
   void OUTPUT(FILE **);
main() {
   double A,B,ALPHA,H,T,W,K1,K2,K3,K4;
   int I,N,OK;
   INPUT(&OK, &A, &B, &ALPHA, &N);
   if (OK) {
      OUTPUT(OUP);
      H = (B - A) / N;
      T = A;
      W = ALPHA;
      fprintf(*OUP, "%5.3f %11.7f\n", T, W);
      for (I=1; I<=N; I++) {
         K1 = H * F(T, W);
         K2 = H * F(T + H / 2.0, W + K1 / 2.0);
         K3 = H * F(T + H / 2.0, W + K2 / 2.0);
         K4 = H * F(T + H, W + K3);
        W = W + (K1 + 2.0 * (K2 + K3) + K4) / 6.0;
         T = A + I * H;
         fprintf(*OUP, "%5.3f %11.7f\n", T, W);
      }
      fclose(*OUP);
   }
   return 0;
}
double F(double T, double Y)
{
   double f;
   f = Y - T*T + 1.0;
   return f;
}
void INPUT(int *OK, double *A, double *B, double *ALPHA, int *N)
{
   double X;
      *OK = false;
      while (!(*OK)) {
         printf("Dar el valor del extremo A\n");
         scanf("%lf", A);
         printf("Dar el valor del extremo B\n");
         scanf("%lf", B);
         if (*A >= *B)
            printf("El valor de B debe ser mayor que el de A\n");
         else *OK = true;
      }
      printf("Dar la condicion inicial\n");
      scanf("%lf", ALPHA);
      *OK = false;
      while(!(*OK)) {
         printf("dar el numero de subintervalos\n");
         scanf("%d", N);
```

```
         if (*N <= 0.0) printf("Debe ser un numero natural\n");
         else *OK = true;
      }
}
void OUTPUT(FILE **OUP)
{
   char NAME[30];
   printf("Dar el nombre de un archivo para escribir la solucion\n");
   printf("Debe estar en el formato\n");
   printf("nombre.txt\n");
   scanf("%s", NAME);
   *OUP = fopen(NAME, "w");
   fprintf(*OUP, "Metodo de Runge-Kutta de orden 4\n\n");
   fprintf(*OUP, "    t           w\n\n");
   printf("Presionar enter para terminar el programa \n ");
   getchar();
   getchar();
}
```

4.4. Método de Fehlberg

Los métodos de aproximación estudiados anteriormente usan un paso de integración h de longitud fija. Sin embargo, si se desea mantener acotado el error de la aproximación, a veces es mejor usar una longitud variable a cada paso en la integración. El *método de Fehlberg* usa el algoritmo de Runge-Kutta, pero agrega control del paso de integración en regiones donde pueda incrementarse el error.

Algoritmo de Fehlberg

Obtiene una solución aproximada para la ecuación $y' = f(t, y)$, con condición inicial $y(a) = \alpha$, en el intervalo $a \leq t \leq b$, con un error menor a la tolerancia especificada.

ENTRADA $f(t, y)$, a, b, α, TOL, cotas de error h_{max} y h_{min}.

SALIDA t, $y(t)$, h, o mensaje de error.

Paso 1 Hacer $t = a$, $y = \alpha$, $h = h_{max}$, $FLAG = 1$

Imprimir (t, y)

Paso 2 Mientras $FLAG = 1$ hacer pasos 3 al 11

Paso 3 Hacer $k_1 = h \cdot f(t, y)$

$k_2 = h \cdot f(t + \frac{h}{4}, y + \frac{k_1}{4})$

$k_3 = h \cdot f(t + 3\frac{h}{8}, y + 3\frac{k_1}{32}, +9\frac{k_2}{32})$

$k_4 = h \cdot f(t + \frac{12}{13}h, y + \frac{1932}{2197}k_1 - \frac{7200}{2197}k_2 + \frac{7296}{2197}k_3)$

$k_5 = h \cdot f(t + h, y + \frac{439}{216}k_1, -8k_2 + \frac{3680}{513}k_3 - \frac{845}{4104}k_4)$

$k_6 = h \cdot f(t + \frac{h}{2}, y - \frac{8}{27}k_1 + 2k_2 - \frac{3544}{2565}k_3 + \frac{1859}{4104}k_4 - \frac{11}{40}k_5)$

Paso 4 Hacer $R = \frac{1}{h}\left|\frac{1}{360}k_1 - \frac{128}{4275}k_3 - \frac{2197}{75240}k_4 + \frac{1}{50}k_5 + \frac{2}{55}k_6\right|$

Paso 5 Si $R \leq TOL$, hacer pasos 6 y 7

Paso 6 Hacer $t = t + h$

$y = y + \frac{25}{16}k_1 + \frac{1408}{2565}k_3 + \frac{2197}{4104}k_4 - \frac{1}{5}k_5$

Paso 7 Salida (t, y, h)

Paso 8 Hacer $\delta = 0.84(TOL/R)^{1/4}$

Paso 9 Si $\delta \leq 0.1$ entonces

Hacer $h = 0.1h$

O si $\delta \geq 4$ entonces

Hacer $h = 4h$

O bien

Hacer $h = \delta h$

Paso 10 Si $h > h_{max}$ entonces

Hacer $h = h_{max}$

Paso 11 Si $t \geq b$ entonces hacer $FLAG = 0$

O si $t + h > b$ entonces

Hacer $h = b - t$

O si $h < h_{min}$ entonces

Hacer $FLAG = 0$,

SALIDA (Se ha rebasado h_{min})

Paso 12 TERMINAR

Código de Fehlberg

Ejemplo de código de Fehlberg para la ecuación

$$\frac{dy}{dt} = y - t^2 + 1.$$

```
#include<stdio.h>
#include<math.h>
   FILE *OUP[1];
   double F(double, double);
   void INPUT(int *, double *, double *, double *, double *L, double *,
   double *, int *);
   void OUTPUT(FILE **);
main() {
   double A,B,TOL,ALPHA,HMAX,HMIN,H,T,W,K1,K2,K3,K4,K5,K6,R,DELTA;
   int I,N,OK;
   INPUT(&OK, &A, &B, &ALPHA, &TOL, &HMIN, &HMAX, &N);
   if (OK) {
      OUTPUT(OUP);
      H = HMAX;
```

```
      T = A;
      W = ALPHA;
      fprintf(*OUP, "%12.7f %11.7f           0           0\n", T, W);
      OK = true;
      while ((T < B) && OK) {
         K1 = H*F(T,W);
         K2 = H*F(T+H/4,W+K1/4);
         K3 = H*F(T+3*H/8,W+(3*K1+9*K2)/32);
         K4 = H*F(T+12*H/13,W+(1932*K1-7200*K2+7296*K3)/2197);
         K5 = H*F(T+H,W+439*K1/216-8*K2+3680*K3/513-845*K4/4104);
         K6 = H*F(T+H/2,W-8*K1/27+2*K2-3544*K3/2565
                   +1859*K4/4104-11*K5/40);
         R = absval(K1/360-128*K3/4275-2197*K4/75240.0
                    +K5/50+2*K6/55)/H;
         if (R <= TOL) {
            T = T + H;
            W = W+25*K1/216+1408*K3/2565+2197*K4/4104-K5/5;
            fprintf(*OUP, "%12.7f %11.7f %11.7f %11.7f\n", T, W, H, R);
         }
         if (R > 1.0E-20) DELTA = 0.84 * exp(0.25 * log(TOL / R));
         else DELTA = 10.0;
         if (DELTA <= 0.1) H = 0.1 * H;
         else {
            if (DELTA >= 4.0) H = 4.0 * H;
            else H = DELTA * H;
         }
         if (H > HMAX) H = HMAX;
         if (H < HMIN) OK = false;
         else {
            if (T+H > B)
               if (absval(B-T) < TOL) T = B;
               else H = B - T;
         }
      }
      if (!OK) fprintf(*OUP, "Se excedio el H minimo\n");
      fclose(*OUP);
   }
   return 0;
}
double F(double T, double Y)
{
   double f;
   f = Y - T*T + 1.0;
   return f;
}
void INPUT(int *OK, double *A, double *B, double *ALPHA, double *TOL, double
*HMIN, double *HMAX, int *N)
{
   double X;
   *OK = false;
   while (!(*OK)) {
      printf("Dar los limites de integracion separados por un espacio\n");
      scanf("%lf %lf", A, B);
      if (*A >= *B)
```

```
         printf("El limite inferior debe ser menor que el limite superior\n");
      else *OK = true;
   }
   printf("Dar la condicion inicial\n");
   scanf("%lf", ALPHA);
   *OK = false;
   while(!(*OK)) {
      printf("Dar la tolerancia\n");
      scanf("%lf", TOL);
      if (*TOL <= 0.0) printf("Debe ser positiva\n");
      else *OK = true;
   }
   *OK = false;
   while(!(*OK)) {
      printf("Dar el espaciado minimo y maximo separados por un espacio");
      scanf("%lf %lf", HMIN, HMAX);
      if ((*HMIN < *HMAX) && (*HMIN > 0.0)) *OK = true;
      else {
         printf("El espaciado minimo debe ser positivo y");
         printf("menor que el espaciado maximo\n");
      }
   }
}
void OUTPUT(FILE **OUP)
{
   char NAME[30];
   printf("Dar el nombre del archivo de salida en el formato:\n");
   printf("nombre.txt\n");
   scanf("%s", NAME);
   *OUP = fopen(NAME, "w");
   fprintf(*OUP, "        T(I)        W(I)           H           R\n\n");
}
```

4.5. Métodos multipasos

Para reducir aún más el error, también se pueden usar varios puntos al calcular el próximo paso. A veces la fórmula de recurrencia en estos métodos sólo involucra valores de y_i previos, por lo que se les llama *métodos explícitos*. Otras veces, la fórmula de recurrencia contiene el valor que se está tratando de calcular, por lo que se les llama *métodos implícitos*. Con frecuencia se combinan métodos explícitos e implícitos para dar los llamados *métodos predictor-corrector*, donde el método explícito predice la aproximación, mientras que el implícito la corrige. A continuación se da el algoritmo para el *método de Adams*, uno de los más usuales de los métodos predictor-corrector.

Algoritmo predictor-corrector de Adams

Aproxima una solución del problema

$$y' = f(t, y), \quad a \leq t \leq b, \quad y(a) = \alpha$$

en $n+1$ valores de t equiespaciados en $[a, b]$.

ENTRADA $f(t, y)$, a, b, n, α.

SALIDA Aproximación a y en $n+1$ puntos.

Paso 1 Hacer $h = \frac{b-a}{n}$, $t_0 = a$, $y_0 = \alpha$

Imprimir (t_0, y_0)

Paso 2 Para $i = 1, 2, 3$ hacer los pasos 3 al 5

Paso 3 Hacer $k_1 = h\ f(t_{i-1}, y_{i-1})$

$k_2 = h\ f(t_{i-1} + \frac{h}{2}, y_{i-1} + \frac{k_1}{2})$

$k_3 = h\ f(t_{i-1} + \frac{h}{2}, y_{i-1} + \frac{k_2}{2})$

$k_4 = h\ f(t_{i-1} + h, y_{i-1} + k_3)$

Paso 4 Hacer $y_i = y_{i-1} + \frac{1}{6}(k_1 + 2k - 2 + 2k_3 + k_4)$

$t_i = a + ih$

Paso 5 Salida (t_i, y_i)

Paso 6 Para $i = 4, ..., n$ hacer pasos 7 al 10

Paso 7 Hacer $t = a + ih$

$y = y_3 + \frac{h}{24}\left[55f(t_3, y_3) - 59f(t_2, y_2) + 37f(t_1, y_1) - 9f(t_0, y_0)\right]$

$y = y_3 + \frac{h}{24}\left[9f(t, y) + 19f(t_3, y_3) - 5f(t_2, y_2) + f(t_1, y_1)\right]$

Paso 8 Salida (t_i, y_i)

Paso 9 Para $j = 0, 1, 2$ hacer $t_j = t_{j+1}$, $y_j = y_{j+1}$

Paso 10 Hacer $t_3 = t$, $y_3 = y$

Paso 11 TERMINAR

Código predictor-corrector de Adams

Ejemplo de código de Adams para la ecuación

$$\frac{dy}{dt} = y - t^2 + 1.$$

```
#include<stdio.h>
#include<math.h>
   FILE *OUP[1];
   double F(double, double);
   void RK4(double , double , double , double *, double *);
   void INPUT(int *, double *, double *, double *, int *);
   void OUTPUT(FILE **);
main() {
   double T[4],W[4];
   double A,B,ALPHA,H,TO,WO;
   int I,N,J,OK;
   INPUT(&OK, &A, &B, &ALPHA, &N);
```

```
   if (OK) {
      OUTPUT(OUP);
      H = (B - A) / N;
      T[0] = A;
      W[0] = ALPHA;
      fprintf(*OUP, "%5.3f %11.7f\n", T[0], W[0]);
      for (I=1; I<=3; I++) {
         RK4(H, T[I-1], W[I-1], &T[I], &W[I]);
         fprintf(*OUP, "%5.3f %11.7f\n", T[I], W[I]);
      }
      for (I=4; I<=N; I++) {
         T0 = A + I * H;
         W0 = W[3]+H*(55.0*F(T[3],W[3])-59.0*F(T[2],W[2])
              +37.0*F(T[1],W[1])-9.0*F(T[0],W[0]))/24.0;
         W0 = W[3]+H*(9.0*F(T0,W0)+19.0*F(T[3],W[3])
              -5.0*F(T[2],W[2])+F(T[1],W[1]))/24.0;
         fprintf(*OUP, "%5.3f %11.7f\n", T0, W0);
         for (J=1; J<=3; J++) {
            T[J-1] = T[J];
            W[J-1] = W[J];
         }
         T[3] = T0;
         W[3] = W0;
      }
      fclose(*OUP);
   }
   return 0;
}
double F(double T, double Y)
{
   double f;
   f = Y - T*T + 1.0;
   return f;
}
void RK4(double H, double T0, double W0, double *T1, double *W1)
{
   double K1,K2,K3,K4;
   *T1 = T0 + H;
   K1 = H * F(T0, W0);
   K2 = H * F(T0 + 0.5 * H, W0 + 0.5 * K1);
   K3 = H * F(T0 + 0.5 * H, W0 + 0.5 * K2);
   K4 = H * F(*T1, W0 + K3);
   *W1 = W0 + (K1 + 2.0 * (K2 + K3) + K4) / 6.0;
}
void INPUT(int *OK, double *A, double *B, double *ALPHA, int *N)
{
   double X;
   *OK = false;
   while (!(*OK)) {
      printf("Dar los limites del intervalo separados con un espacio\n");
      scanf("%lf %lf", A, B);
      if (*A >= *B)
         printf("El limite derecho debe ser mayor que el izquierdo\n");
      else *OK = true;
```

```
   }
   printf("Dar la condicion inicial\n");
   scanf("%lf", ALPHA);
   *OK = false;
   while(!(*OK)) {
      printf("Dar un numero de subintervalos mayor a 3 ");
      scanf("%d", N);
      if (*N < 4) printf("Debe ser al menos 4\n");
      else *OK = true;
   }
}
void OUTPUT(FILE **OUP)
{
   char NAME[30];
   printf("Dar el nombre del archivo de salida en el formato:\n");
   printf("nombre.txt\n");
   scanf("%s", NAME);
   *OUP = fopen(NAME, "w");
   fprintf(*OUP, "    t           w\n");
}
```

Ejercicios

Resolver numéricamente los siguientes problemas de valor inicial en el intervalo dado, usando los métodos estudiados con anterioridad. Comparar la precisión de cada método usando igual número de puntos en cada caso y graficando las soluciones.

1. $y' = 3 + x - y$, $y(0) = 1$, $0 \leq x \leq 1$
2. $y' = 1 - x + 4y$, $y(0) = 1$, $0 \leq x \leq 2$
3. $y' = -3x + 2y$, $y(0) = 1$, $0 \leq x \leq 3$
4. $y' = 5x - 3\sqrt{y}$, $y(0) = 2$, $0 \leq x \leq 2$
5. $y' = 2x + e^{-xy}$, $y(0) = 1$, $0 \leq x \leq 1$
6. $y' = (x^2 - y^2)\operatorname{sen} y$, $y(0) = -1$, $0 \leq x \leq 2$
7. $y' = \frac{y^2+2xy}{3+x^2}$, $y(0) = 0.5$, $0 \leq x \leq 3$
8. $y' = 0.5 - x + 2y$, $y(0) = 1$, $0 \leq x \leq 2$
9. $y' = \frac{4-xy}{1+y^2}$, $y(0) = -2$, $0 \leq x \leq 1$
10. $y' = \sqrt{x+y}$, $y(0) = 3$, $0 \leq x \leq 2$
11. $y' = x^2 + y^2$, $y(0) = 1$, $0 \leq x \leq 3$
12. $y' = \cos 5\pi x$, $y(0) = 1$, $0 \leq x \leq 2$
13. $y' = \frac{3x^2}{3y^2-4}$, $y(0) = 0$, $0 \leq x \leq 1$
14. $y' = xe^{3x} - 2y$, $y(0) = 0$, $0 \leq x \leq 1$
15. $y' = 1 + (x - y)^2$, $y(2) = 1$, $2 \leq x \leq 3$
16. $y' = 1 + y/x$, $y(1) = 2$, $1 \leq x \leq 2$
17. $y' = \cos 2x + \operatorname{sen} 3x$, $y(0) = 1$, $0 \leq x \leq 1$
18. $y' = y/x - (y/x)^2$, $y(1) = 1$, $1 \leq x \leq 2$

19. $y' = 1 + y/x + (y/x)^2$, $y(1) = 0$, $1 \le x \le 3$
20. $y' = -(y+1)(y+3)$, $y(0) = -2$, $0 \le x \le 2$
21. $y' = -5y + 5x^2 + 2x$, $y(0) = \frac{1}{3}$, $0 \le x \le 1$
22. $y' = \frac{2}{x}y + x^2 e^x$, $y(1) = 0$, $1 \le x \le 2$
23. $y' = \operatorname{sen} x + e^{-x}$, $y(0) = 0$, $0 \le x \le 1$
24. $y' = \frac{1}{x(y^2+y)}$, $y(1) = -2$, $1 \le x \le 3$
25. $y' = (t + 2x^3)y^3 - xy$, $y(0) = \frac{1}{3}$, $0 \le x \le 2$
26. Una pieza de metal de masa $m = 0.1$ kg que estaba a una temperatura $T = 200$ °C se lleva a una habitación que está a $T = 25$ °C. Si la temperatura del metal sigue la ecuación

$$\frac{dT}{dt} = \frac{A}{\rho c v}\left[\varepsilon\sigma(297^4 - T^4) + h_c(297 - T)\right], \quad T(0) = 473$$

con t en segundos, y $\rho = 300$ kg/m^3, $v = 0.001$ m^3, $A = 0.25$ m^2, $c = 900$ J/kg K, $h_c = 30$ J/m^2 K, $\varepsilon = 0.8$ y $\sigma = 5.67 \times 10^{-8}$ w/m^2 K^4, calcular las temperaturas del metal cada 10 segundos, hasta llegar a 10 minutos.

27. Un circuito con una resistencia $R = 1.4\ \Omega$, inductancia $L = 1.7$ H, capacitancia $C = 0.3$ F y una fuerza electromotriz dada por

$$E(t) = e^{-0.06\pi t}\operatorname{sen}(2t - \pi),$$

satisface la ecuación

$$\frac{di}{dt} = C\frac{d^2E}{dt^2} + \frac{1}{R}\frac{dE}{dt} + \frac{1}{L}E.$$

Si $i(0) = 0$, encontrar el valor de i para $t = 0.1j$, con $j = 0, 1, 2, \ldots, 100$.

4.6. Ecuaciones de orden superior y sistemas

Aquí estudiaremos la solución de sistemas con valores iniciales, no considerando problemas con valores de frontera. Se sabe, del curso de ecuaciones diferenciales ordinarias, que toda ecuación diferencial de orden superior se puede transformar en un sistema de ecuaciones diferenciales de primer orden y viceversa. Por ejemplo, una ecuación de segundo orden puede sustituirse por un sistema de dos ecuaciones diferenciales de primer orden. Entonces será suficiente con tratar o analizar la solución de sistemas de ecuaciones diferenciales de primer orden.

Para resolver el sistema

$$\begin{aligned} \frac{dx}{dt} &= f(t, x, y) \\ \frac{dy}{dt} &= g(t, x, y) \end{aligned}$$

en forma numérica, es suficiente con calcular los valores de x y y con alguno de los métodos estudiados anteriormente (Euler, Adams, etc.) y sustituirlos en cada nueva iteración

$$\begin{aligned} x_{i+1} &= x_i + hf(t_i, x_i, y_i) \\ y_{i+1} &= y_i + hg(t_i, x_i, y_i). \end{aligned}$$

Por ejemplo, usando el método de Runge-Kutta, podemos hacer

$$\begin{aligned} x_{i+1} &= x_i + \frac{h}{6}\left[k_1 + 2k_2 + 2k_3 + k_4\right] \\ y_{i+1} &= y_i + \frac{h}{6}\left[\ell_1 + 2\ell_2 + 2\ell_3 + \ell_4\right] \end{aligned}$$

donde

$$\begin{aligned} k_1 &= f_1(t_i, x_i, y_i) \\ k_2 &= f_1(t_i + h/2, x_i + (h/2)\ k_1, y_i + (h/2)\ \ell_1) \\ k_3 &= f_1(x_i + h/2, x_i + (h/2)\ k_2, y_i + (h/2)\ \ell_2) \\ k_4 &= f_1(x_i + h, x_i + h\ k_3, y_i + h\ell_3) \\ \ell_1 &= f_2(t_i, x_i, y_i) \\ \ell_2 &= f_2(t_i + h/2, x_i + (h/2)\ k_1, y_i + (h/2)\ \ell_1) \\ \ell_3 &= f_2(x_i + h/2, x_i + (h/2)\ k_2, y_i + (h/2)\ \ell_2) \\ \ell_4 &= f_2(x_i + h, x_i + h\ k_3, y_i + h\ell_3). \end{aligned}$$

Por supuesto, este método se puede extender a tres o más ecuaciones, dependiendo del orden de la ecuación o sistema que se desee resolver numéricamente.

Algoritmo de Runge-Kutta para sistemas

Obtiene una solución aproximada del problema de valor inicial

$$x' = f_1(t, x, y), y' = f_2(t, x, y), \quad x(t_0) = x_0, \quad y(t_0) = y_0, \quad t \in [a, b].$$

ENTRADA $f_1(t, x, y)$, $f_2(t, x, y)$, a, b, n, x_0, y_0.

SALIDA Aproximación a $x(t)$, $y(t)$ en n valores de t equidistantes.

Paso 1 Hacer $h = (b - a)/n$, $t = a$, $x = x_0$, $y = y_0$

Paso 2 Para $i = 1, ..., n$ hacer pasos 3 y 4

Paso 3 Hacer $k_1 = f_1(x, y)$

$k_2 = f_1(x + \frac{h}{2}, y + \frac{h}{2}k_1)$

$k_3 = f_1(x + \frac{h}{2}, y + \frac{h}{2}k_2)$

$k_4 = f_1(x + h, y + h\ k_3)$

$x = x + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)$

Paso 4 Hacer $\ell_1 = f_2(x, y)$

$\ell_2 = f_2(x + \frac{h}{2}, y + \frac{h}{2}k_1)$

$\ell_3 = f_2(x + \frac{h}{2}, y + \frac{h}{2}k_2)$

$\ell_4 = f_2(x + h, y + h\ k_3)$

$y = y + \frac{h}{6}(\ell_1 + 2\ell_2 + 2\ell_3 + \ell_4)$

$t = t + h$

SALIDA $((t, x, y))$

Paso 5 TERMINAR

Código de Runge-Kutta para sistemas

Ejemplo de código de Runge-Kutta para el sistema

$$\frac{dx}{dt} = -4x - 3y + 6$$

$$\frac{dy}{dt} = -2.4x + 1.6y + 3.6.$$

```
#include<stdio.h>
#include<math.h>
   FILE *OUP[1];
   double F1(double, double, double);
   double F2(double, double, double);
   void INPUT(int *, double *, double *, double *, double *, int *);
   void OUTPUT(FILE **);
main() {
   double A,B,ALPHA1,ALPHA2,H,T,W1,W2,X11,X12,X21,X22,X31,X32,X41,X42;
   int I,N,OK;
   INPUT(&OK, &A, &B, &ALPHA1, &ALPHA2, &N);
   if (OK) {
      OUTPUT(OUP);
      H = (B - A) / N;
      T = A;
      W1 = ALPHA1;
      W2 = ALPHA2;
      fprintf(*OUP, "%5.3f %11.8f %11.8f\n", T, W1, W2);
      for (I=1; I<=N; I++) {
         X11 = H * F1(T, W1, W2);
         X12 = H * F2(T, W1, W2);
         X21 = H * F1(T + H / 2.0, W1 + X11 / 2.0, W2 + X12 / 2.0);
         X22 = H * F2(T + H / 2.0, W1 + X11 / 2.0, W2 + X12 / 2.0);
         X31 = H * F1(T + H / 2.0, W1 + X21 / 2.0, W2 + X22 / 2.0);
         X32 = H * F2(T + H / 2.0, W1 + X21 / 2.0, W2 + X22 / 2.0);
         X41 = H * F1(T + H, W1 + X31, W2 + X32);
         X42 = H * F2(T + H, W1 + X31, W2 + X32);
         W1 = W1 + (X11 + 2.0 * X21 + 2.0 * X31 + X41) / 6.0;
         W2 = W2 + (X12 + 2.0 * X22 + 2.0 * X32 + X42) / 6.0;
         T = A + I * H;
         fprintf(*OUP, "%5.3f %11.8f %11.8f\n", T, W1, W2);
      }
      fclose(*OUP);
   }
   return 0;
} double F1(double T, double X1, double X2) {
```

```
   double f;
   f = -4*X1+3*X2+6;
   return f;
} double F2(double T, double X1, double X2) {
   double f;
   f = -2.4*X1+1.6*X2+3.6;
   return f;
} void INPUT(int *OK, double *A, double *B, double *ALPHA1, double *ALPHA2, int *N) {
   double X;
      *OK = false;
      while (!(*OK)) {
         printf("Dar el valor del extremo A\n");
         scanf("%lf", A);
         printf("Dar el valor del extremo B\n");
         scanf("%lf", B);
         if (*A >= *B)
            printf("El valor de B debe ser mayor que el de A\n");
         else *OK = true;
      }
      printf("Dar la condicion inicial y1(A)\n");
      scanf("%lf", ALPHA1);
      printf("Dar la condicion inicial y2(A)\n");
      scanf("%lf", ALPHA2);
      *OK = false;
      while(!(*OK)) {
         printf("dar el numero de subintervalos\n");
         scanf("%d", N);
         if (*N <= 0.0) printf("Debe ser un numero natural\n");
         else *OK = true;
      }
} void OUTPUT(FILE **OUP) {
   char NAME[30];
   printf("Dar el nombre de un archivo para escribir la solucion\n");
   printf("Debe estar en el formato\n");
   printf("nombre.txt\n");
   scanf("%s", NAME);
   *OUP = fopen(NAME, "w");
   fprintf(*OUP, "Metodo de Runge-Kutta para un sistema de
   dos ecuaciones de primer orden\n");
   fprintf(*OUP, "    t          y1          y2\n\n");
   printf("Presionar enter para terminar el programa \n ");
   getchar();
   getchar();
}
```

Ejercicios

Resolver los sistemas de ecuaciones diferenciales dados o ecuaciones diferenciales de orden superior (previa transformación a un sistema).

1. $x' = x - 4y$, $y' = -x + y$, $x(0) = 1$, $y(0) = 0$, $0 \leq t \leq 1$
2. $x' = x + y + t$, $y' = 4x - 2y$, $x(0) = 1$, $y(0) = 0$, $0 \leq t \leq 2$
3. $x' = 2x + yt$, $y' = xy$, $x(0) = 1$, $y(0) = 1$, $0 \leq t \leq 3$

4. $x' = -tx - y - 1$, $y' = x$, $x(0) = 1$, $y(0) = 1$, $0 \le t \le 2$
5. $x' = x - y + xy$, $y' = 3x - 2y - xy$, $x(0) = 0$, $y(0) = 1$, $0 \le t \le 1$
6. $x' = x(1 - 0.5x - 0.5y)$, $y' = y(-0.25 + 0.25x)$, $x(0) = 4$, $y(0) = 1$, $0 \le t \le 2$
7. $x' = e^{-x+y} - \cos x$, $y' = \operatorname{sen}(x - 3y)$, $x(0) = 1$, $y(0) = 2$, $0 \le t \le 3$
8. $x' = x(4 - 0.0003x - 0.0004y)$, $y' = y(2 - 0.0002x - 0.0001y)$, $x(0) = 10000$, $y(0) = 10000$, $0 \le t \le 4$
9. $x' = 3x + 2y - (2t^2 + 1)e^{2t}$, $y' = 4x + y + (t^2 + 2t - 4)e^{2t}$, $x(0) = 1$, $y(0) = 1$, $0 \le t \le 2$
10. $x' = -4x - 2y + \cos t + 4 \operatorname{sen} t$, $y' = 3x + y - 3 \operatorname{sen} t$, $x(0) = 0$, $y(0) = -1$, $0 \le t \le 3$
11. $x' = (1 + x) \operatorname{sen} y$, $y' = 1 - x - \cos y$, $x(0) = 0$, $y(0) = 0$, $0 \le t \le 1$
12. $x' = x + y^2$, $y' = x + y$, $x(0) = 0$, $y(0) = 0$, $0 \le t \le 2$
13. $x' = y$, $y' = -x - 2e^t + 1$, $z' = -x - e^t + 1$, $x(0) = 1$, $y(0) = 0$, $z(0) = 1$, $1 \le t \le 2$
14. $x' = y - z + t$, $y' = 3t^2$, $z' = y + e^{-t}$, $x(0) = 1$, $y(0) = 1$, $z(0) = -1$, $0 \le t \le 1$
15. $x'' + t^2 x' + 3x = t$, $x(0) = 1$, $x'(0) = 2$, $0 \le t \le 2$
16. $x'' - x' + x = te^t - t$, $x(0) = 0$, $x'(0) = 0$, $1 \le t \le 2$
17. $x''' + 2x'' - x' - 2x = e^t$, $x(0) = 1$, $x'(0) = 2$, $x''(0) = 0$, $0 \le t \le 3$
18. $t^2 x'' - 2tx' + 2x = t^3 \ln t$, $x(1) = 1$, $x'(1) = 0$, $1 \le t \le 2$
19. $t^3 x''' - t^2 x'' + 3tx' - 4x = 5t^3 \ln t + 9t^3$, $x(1) = 0$, $x'(1) = 1$, $x''(1) = 3$, $1 \le t \le 2$
20. $\theta'' + 16 \operatorname{sen} \theta = 0$, $\theta(0) = \frac{\pi}{6}$, $\theta'(0) = 0$
21. En un lago viven dos tipos de peces, uno de los cuales se alimenta del otro. Si la población del pez que es la presa es $x_1(t)$, y la población del depredador es $x_2(t)$, las poblaciones satisfacen el sistema de ecuaciones

$$x_1'(t) = 3x_1(t) - 0.002x_1(t)x_2(t)$$
$$x_2'(t) = 0.0006x_1(t)x_2(t) - 0.5x_2(t).$$

Encontrar las poblaciones para $0 \le t \le 4$, suponiendo que $x_1(0) = 1000$ y $x_2(0) = 500$. Graficar ambas poblaciones como función del tiempo en la misma figura y comparar los máximos y mínimos de cada una de ellas.

22. El movimiento de una partícula sujeta a un resorte, que experimenta frenado por viscosidad, se puede modelar por medio de la ecuación

$$y'' + 2\zeta\omega y' + \omega^2 y = \frac{F(t)}{M},$$

donde $\omega = \sqrt{k/M}$, $\zeta = \frac{c}{2M\omega}$, $k = 3.2$ kg/s^2, $M = 5$ kg y

$$F(t) = \begin{cases} 9.8 & 0 \le t \le 1 \\ 0 & t > 1. \end{cases}$$

Determinar el movimiento de la partícula en $0 \le t \le 10$ s.

Capítulo 5

Sistemas de ecuaciones lineales y no lineales

En este capítulo se presentan dos algoritmos para obtener una solución de sistemas de ecuaciones lineales y no lineales. Aunque el método de eliminación gaussiana se estudia en cursos de álgebra lineal, es conveniente hacer una modificación para reducir la acumulación de error de redondeo. El método del punto fijo es también muy usual para aumentar la eficiencia de cálculo, al evitar resolver matrices demasiado grandes como las usuales en el método de eliminación gaussiana. Para sistemas de ecuaciones lineales se estudia el método de eliminación gaussiana con pivoteo y el método del punto fijo.

Los sistemas no lineales son especialmente difíciles, tanto por los cálculos involucrados (el jacobiano, con las derivadas parciales involucradas), como porque no hay un método fácil para encontrar los puntos iniciales para comenzar a aproximar. Se recomienda, por tanto, usar gráficas tridimensionales cuando sea posible. Para estos sistemas se estudia el método de Newton y el de Broyden.

5.1. Método de eliminación gaussiana con pivoteo

Aunque los sistemas de ecuaciones lineales pueden resolverse en forma analítica, en la práctica, la gran cantidad de cálculos involucrados hace necesario el uso de computadoras para llevarlos a cabo, lo cual tiene como consecuencia la acumulación de error de redondeo. Para reducirlo, al método de eliminación gaussiana convencional hay que añadir el *pivoteo*, que consiste en hacer intercambios de renglones aun cuando el elemento pivote no es cero: si el elemento pivote es pequeño comparado con los otros elementos de la misma columna, se hace un intercambio de renglón con el que contenga el elemento de mayor valor absoluto.

Algoritmo de eliminación gaussiana con pivoteo

Obtiene la solución del sistema de ecuaciones lineales

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n &= a_{1,n+1} \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{1n}x_n &= a_{2,n+1} \\ &\vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n &= a_{n,n+1}. \end{aligned}$$

ENTRADA Número de ecuaciones n, matriz aumentada $A = (a_{ij})$.

SALIDA Solución $x_1, \ldots, x_n$ o mensaje de que el sistema no está determinado.

Paso 1 Para $i = 1, ..., n$ hacer $NROW(i) = i$ $NCOL(i) = i$

Paso 2 Para $i = 1, ..., n-1$ hacer pasos 3 al 6

Paso 3 Tomar p y q, los enteros más pequeños que cumplan $i \leq p, q \leq n$ y $|a(ROW(p), NCOL(q)| = \text{máx}_{i \leq k, j \leq n} |a(NROW(k), NCOL(j))|$

{Se usa la notación: $a(NROW(i), NCOL(j)) = a_{NROW(i),NCOL(j)}$}

Paso 4 Si $a(NROW(p), NCOL(q)) = 0$ entonces

SALIDA ('El sistema está indeterminado')

TERMINAR

Paso 5 Si $NROW(i) \neq NROW(p)$ hacer

$NCOPY = NROW(i)$

$NROW(i) = NROW(p)$

$NROW(p) = NCOPY$

Si $NCOL(i) \neq NCOL(p)$ hacer

$NCOPY = NCOL(i)$

$NCOL(i) = NCOL(q)$

$NCOL(q) = NCOPY$

Paso 6 Para $j = i + 1, ..., n$ hacer pasos 7 y 8

Paso 7 Hacer $m(NROW(j), NCOL(i)) = \frac{a(NROW(j),NCOL(i))}{a(NROW(i),NCOL(i))}$

Paso 8 Hacer $(E_{NROW(j)} - m(NROW(j), NCOL(i)) \cdot E_{NROW(i)}) \rightarrow (E_{NROW(j)})$

Paso 9 Si $a(NROW(n), NCOL(n)) = 0$ entonces

SALIDA (No hay solución única)

TERMINAR

Paso 10 Hacer $x(NCOL(n)) = \frac{a(NROW(n),n+1)}{a(NROW(n),NCOL(n))}$

Paso 11 Hacer

$$x(NCOL(i)) = \frac{a(NROW(i), n+1) - \sum_{j=i+1}^{n} a(NROW(i), NCOL(j)) \cdot x(NCOL(j))}{a(NROW(i), NCOL(i))}$$

Paso 12 SALIDA ('La solución es')

('x(',$NCOL(i)$,')=',$x(NCOL(i))$)

TERMINAR

Código para el método de eliminación con pivoteo

```
#include<stdio.h>
#include<math.h>
void INPUT(int *, double [][11], int *);
void OUTPUT(int, double *);
main()
{
   double A[10][11], X[10];
   double AMAX,XM,SUM;
   int NROW[10];
   int N,M,ICHG,I,NN,IMAX,J,JJ,IP,JP,NCOPY,I1,J1,N1,K,N2,LL,KK,OK;
   INPUT(&OK, A, &N);
   if (OK) {
      M = N + 1;
      for (I=1; I<=N; I++) NROW[I-1] = I;
      NN = N - 1;
      ICHG = 0;
      I = 1;
      while ((OK) && (I <= NN)) {
         IMAX = NROW[I-1];
         AMAX = abs(A[IMAX-1][I-1]);
         IMAX = I;
         JJ = I + 1;
         for (IP=JJ; IP<=N; IP++) {
            JP = NROW[IP-1];
            if (abs(A[JP-1][I-1]) > AMAX) {
               AMAX = abs(A[JP-1][I-1]);
               IMAX = IP;
            }
         }
         if (AMAX <= 0) OK = false;
         else {
            if ( NROW[I-1] != NROW[IMAX-1]) {
               ICHG = ICHG + 1;
               NCOPY = NROW[I-1];
               NROW[I-1] = NROW[IMAX-1];
               NROW[IMAX-1] = NCOPY;
            }
            I1 = NROW[I-1];
            for (J=JJ; J<=N; J++) {
               J1 = NROW[J-1];
```

```
               XM = A[J1-1][I-1] / A[I1-1][I-1];
               for (K=JJ; K<=M; K++)
                  A[J1-1][K-1] = A[J1-1][K-1] - XM * A[I1-1][K-1];
               A[J1-1][I-1] = 0.0;
            }
         }
         I++;
      }
      if (OK) {
         N1 = NROW[N-1];
         if (abs(A[N1-1][N-1]) <= 0) OK = false;
         else {
            X[N-1] = A[N1-1][M-1] / A[N1-1][N-1];
            for (K=1; K<=NN; K++) {
               I = NN - K + 1;
               JJ = I + 1;
               N2 = NROW[I-1];
               SUM = 0.0;
               for (KK=JJ; KK<=N; KK++) {
                  SUM = SUM - A[N2-1][KK-1] * X[KK-1];
               }
               X[I-1] = (A[N2-1][N] + SUM) / A[N2-1][I-1];
            }
            OUTPUT(N, X);
         }
      }
      if (!OK) printf("El sistema tiene infinitas soluciones\n");
   }
   getchar();
   getchar();
}
void INPUT(int *OK, double A[][11], int *N)
{
   int I, J;
   printf("Solucion de un sistema de N ecuaciones lineales con N incognitas");
   printf(" usando eliminacion con pivoteo, dada la matriz AUMENTADA\n");
   printf("A(1,1), A(1,2), ..., A(1,N+1)\n");
   printf("A(2,1), A(2,2), ..., A(2,N+1)\n");
   printf(". . . ");
   printf("A(N,1), A(N,2), ..., A(N,N+1)\n\n");
      *OK = false;
      while (!(*OK)) {
         printf("Dar el numero de ecuaciones\n");
         scanf("%d", N);
         if (*N > 1) {
            for (I=1; I<=*N; I++) {
               for (J=1; J<=*N+1; J++) {
               printf("Dar el coeficiente A(%d,%d)  ", I, J);
               scanf("%lf", &A[I-1][J-1]);}
            }
            *OK = true;
         }
         else printf("Debe ser un entero mayor que 2\n");
      }
```

```
}
void OUTPUT(int N, double *X)
{
   int I, J;
   printf("\n\nEl sistema tiene el vector solucion:\n");
   for (I=1; I<=N; I++) {
      printf("  %12.8f", X[I-1]);
   }
   printf("\n");
}
```

5.2. Método del punto fijo

Otra técnica muy usual para resolver sistemas de ecuaciones lineales comienza por pasar el sistema de la forma $\mathbb{A}\vec{x} = \vec{b}$ a la forma $\vec{x} = \mathbb{B}\vec{x} + \vec{c}$, lo cual se logra despejando cada variable en cada una de las ecuaciones. A continuación, se toma alguna aproximación inicial $\vec{x}^0$ (a falta de algo mejor, se puede iniciar con $\vec{x}^{(0)} = \vec{0}$), y se itera sucesivamente, haciendo

$$\vec{x}^{(k)} = \mathbb{B}\vec{x}^{(k-1)} + \vec{c}. \tag{5.1}$$

Cuando esto se realiza, se le llama *método de Jacobi*. Para acelerar la convergencia, en cada iteración se pueden usar no sólo los valores obtenidos en la iteración anterior, sino también los generados en la presente iteración. Cuando se usa esta variación se llama *método de Gauss-Seidel*.

Algoritmo del punto fijo

Obtiene una solución aproximada para el sistema $\vec{x} = \mathbb{B}\vec{x}+\vec{c}$ dada una aproximación inicial $\vec{x}^{(0)}$.

ENTRADA Número de ecuaciones y de incógnitas n, coeficientes b_{ij}, términos independientes c_i, aproximación inicial $\vec{x}^{(0)}$, tolerancia TOL, número de iteraciones máximo N.

SALIDA Solución aproximada $\vec{x}$ o mensaje de error.

Paso 1 Hacer $k = 1$

Paso 2 Mientras $k \leq N$ hacer pasos 3 al 6

Paso 3 Para $i = 1, ..., n$ hacer

$$x_i = \frac{-\sum_{j=1}^{i-1} a_{ij}x_j - \sum_{j=i+1}^{n} a_{ij}x_j^{(0)} + b_i}{a_{ii}}$$

Paso 4 Si $||\vec{x} - \vec{x}_0|| < TOL$ entonces

SALIDA (La solucion es $\vec{x}$)

TERMINAR

Paso 5 Hacer $k = k + 1$

Paso 6 Para $i = 1, ..., n$ hacer $x^{(0)} = i = x_i$

Paso 7 SALIDA ('El método fracasó después de N iteraciones')

TERMINAR

Código para iteración del punto fijo

```
#include<stdio.h>
#include<math.h>
void INPUT(int *, double *, double [][11], double *, int *, int *);
void OUTPUT(int, double *);
main()
{
   double A[10][11],X1[10];
   double S,ERR,TOL;
   int N,I,J,NN,K,OK;
   INPUT(&OK, X1, A, &TOL, &N, &NN);
   if (OK) {
      K = 1;
      OK = false;
      while ((!OK) && (K <= NN )) {
         ERR = 0.0;
         for (I=1; I<=N; I++) {
            S = 0.0;
            for (J=1; J<=N; J++) if (J!=I) S = S - A[I-1][J-1] * X1[J-1];
            S = (S + A[I-1][N])/A[I-1][I-1];
            if (abs(S) > ERR) ERR  = abs(S);
            X1[I-1] = X1[I-1] + S;
         }
         if (ERR <= TOL) OK = true;
         else
            K++;
      }
      if (!OK) printf("Se excedio el maximo de iteraciones\n");
      else OUTPUT(N, X1);
   }
   getchar();
   getchar();
}
void INPUT(int *OK, double *X1, double A[][11], double *TOL, int *N, int *NN)
{
   int I, J;
   printf("Solucion de un sistema de N ecuaciones lineales con N incognitas");
   printf(" usando eliminacion con pivoteo, dada la matriz AUMENTADA\n");
   printf("A(1,1), A(1,2), ..., A(1,N+1)\n");
   printf("A(2,1), A(2,2), ..., A(2,N+1)\n");
   printf(". . . ");
   printf("A(N,1), A(N,2), ..., A(N,N+1)\n\n");
      *OK = false;
      while (!(*OK)) {
         printf("Dar el numero de ecuaciones\n");
         scanf("%d", N);
         if (*N > 1) {
```

```
            for (I=1; I<=*N; I++) {
               for (J=1; J<=*N+1; J++) {
               printf("Dar el coeficiente A(%d,%d)  ", I, J);
               scanf("%lf", &A[I-1][J-1]);}
            }
            *OK = true;
         }
         else printf("Debe ser un entero mayor que 2\n");
      }
      *OK = false;
      while(!(*OK)) {
         printf("Dar la tolerancia\n");
         scanf("%lf", TOL);
         if (*TOL > 0) *OK = true;
         else printf("Debe ser un numero positivo\n");
      }
      *OK = false;
      while(!(*OK)) {
         printf("Dar el numero de iteraciones maximo\n");
         scanf("%d", NN);
         if (*NN > 0) *OK = true;
         else printf("Debe ser un numero natural\n");
      }
}
void OUTPUT(int N, double *X1)
{
   int I;
   printf("\n\nEl sistema tiene el vector solucion: \n");
   for (I=1; I<=N; I++) {
      printf("  %12.8f", X1[I-1]);
   }
   printf("\n");
}
```

Cabe señalar que, con las modificaciones adecuadas, este método puede aplicarse igualmente a sistemas de ecuaciones no lineales. El lector avezado sabrá hacer las modificaciones necesarias, tanto al algoritmo como al código presentados, para poder resolver sistemas de ecuaciones no lineales.

Ejercicios

Encontrar soluciones aproximadas para los siguientes sistemas de ecuaciones lineales, usando los dos métodos estudiados. Comparar la precisión de ambos métodos por sustitución directa en el sistema.

1. $4x_1 - x_2 + x_3 = 8$ R: $x_1 = 1$, $x_2 = -1$, $x_3 = 3$
   $x_1 + 2x_2 + 4x_3 = 11$
   $2x_1 + 5x_2 + 2x_3 = 3$

2. $2x_1 - 1.5x_2 + 3x_3 = 1$ R: $x_1 = -1$, $x_2 = 0$, $x_3 = 1$
   $-x_1 + 2x_3 = 3$
   $4x_1 - 4.5x_2 + 5x_3 = 1$

3. $4x_1 + x_2 + 2x_3 = 9$
   $2x_1 + 4x_2 - x_3 = -5$
   $x_1 + x_2 - 3x_3 = -9$
   R: $x_1 = 1$, $x_2 = -1$, $x_3 = 3$

4. $x_1 - x_2 + 3x_3 = 2$
   $3x_1 - 3x_2 + x_3 = -1$
   $x_1 + x_2 = 3$
   R: $x_1 = 1.1875$, $x_2 = 1.8125$, $x_3 = 0.875$

5. $x_1 - 0.5x_2 + x_3 = 4$
   $2x_1 - x_2 - x_3 + x_4 = 5$
   $x_1 + x_2 = 2$
   $x_1 - 0.5x_2 + x_3 + x_4 = 5$
   R: $x_1 = 2.4444$, $x_2 = -0.4444$, $x_3 = 1.3333$, $x_4 = 1$

6. $-x_1 + 5x_2 - 2x_3 = 1$
   $7x_1 - 4x_2 + 2x_3 = -2$
   $8x_1 + 5x_2 - 3x_3 = 7$
   R: $x_1 = 0.122449$, $x_2 = 1.734694$, $x_3 = 4.897959$

7. $4x_1 + 2x_2 - 8x_3 = 6$
   $3x_1 - 5x_2 + 2x_3 = -7$
   $5x_1 + x_2 - 4x_3 = -2$
   R: $x_1 = -1.666667$, $x_2 = -0.259259$, $x_3 = -1.648148$

8. $9x_1 - 6x_2 - 2x_3 = 5$
   $-x_1 - 3x_2 + 5x_3 = -6$
   $-2x_1 + 4x_2 - 8x_3 = 7$
   R: $x_1 = 0.634146$, $x_2 = 0.396341$, $x_3 = -0.835366$

9. $x_1 + x_2 + x_4 = 2$
   $2x_1 + x_2 - x_3 + x_4 = 1$
   $-x_1 + 2x_2 + 3x_3 - x_4 = 4$
   $3x_1 - x_2 - x_3 + 2x_4 = -3$
   R: $x_1 = -1$, $x_2 = 2$, $x_3 = 0$, $x_4 = 1$

10. $x_1 + 2x_2 + 5x_3 = -9$
    $x_1 - x_2 + 3x_3 = 2$
    $3x_1 - 6x_2 - x_3 = 25$
    R: $x_1 = 2$, $x_2 = -3$, $x_3 = -1$

11. $4x_1 + x_2 + x_3 + x_4 = 7$
    $x_1 + 3x_2 - x_3 + x_4 = 3$
    $x_1 - x_2 + 2x_3 + 2x_4 = 2$
    $x_1 + x_2 + x_3 + x_4 = 5$
    R: $x_1 = 0.666667$, $x_2 = 2.444444$, $x_3 = 3.444444$, $x_4 = 1.555556$

12. $x_1 + 2x_2 + x_3 + 2x_4 = 5$
    $3x_1 + 2x_2 + 2x_3 - 3x_4 = 2$
    $x_1 + 2x_2 + 2x_3 - 2x_4 = -2$
    $x_1 + 2x_2 + 3x_3 - 4x_4 = 3$
    R: $x_1 = 2.857143$, $x_2 = -0.571428$, $x_3 = -0.142857$, $x_4 = 1.714286$

13. $2x_1 + 4x_2 - 8x_3 + 2x_4 = 5$
    $-5x_1 - 2x_2 + 24x_3 - 13x_4 = 3$
    $6x_1 + 3x_2 - 24x_3 + 16x_4 = 7$
    $4x_1 + x_2 - 8x_3 + 4x_4 = -1$
    R: $x_1 = -0.162791$, $x_2 = 3.395349$, $x_3 = 1.595930$, $x_4 = 2.255814$

14. $-8x_1 + 2x_2 - 9x_3 + x_4 = -11$
    $7x_1 - 3x_2 + 7x_3 + x_4 = 9$
    $2x_1 - 5x_2 - x_3 + 7x_4 = -7$
    $-2x_1 - 3x_2 - 5x_3 + 6x_4 = 4$
    R: $x_1 = 90$, $x_2 = 10$, $x_3 = -80$, $x_4 = -31$

15. $-2x_1 + x_2 - 5x_3 + 6x_4 = 6$
$-9x_1 - 10x_2 + x_3 + 2x_4 = 5$
$-x_1 - 4x_2 + 5x_3 - 5x_4 = -2$
$-6x_1 - 8x_2 + 3x_3 - x_4 = -8$
R: $x_1 = -28$, $x_2 = 5$, $x_3 = -67$, $x_4 = -65$

16. $4x_1 + x_2 + x_3 + x_4 = -2$
$x_1 + 3x_2 - x_3 + x_4 = 6$
$x_1 - x_2 + 2x_3 + 2x_4 = -3$
$x_1 + x_2 + x_3 + x_4 = 7$
R: $x_1 = -3$, $x_2 = 6.666667$, $x_3 = 7.166667$, $x_4 = -3.833333$

17. $x_1 + 2x_2 + x_3 + 2x_4 = -2$
$3x_1 + 2x_2 + 2x_3 - 3x_4 = 5$
$x_1 + 2x_2 + 2x_3 - 2x_4 = 12$
$x_1 + 2x_2 + 3x_3 - 4x_4 = -5$
R: $x_1 = -6.428571$, $x_2 = 10.642857$, $x_3 = 9.428571$, $x_4 = 5.857143$

18. $2x_1 + 4x_2 - 8x_3 + 2x_4 = 3$
$-5x_1 - 2x_2 + 24x_3 - 13x_4 = 2$
$6x_1 + 3x_2 - 24x_3 + 16x_4 = -5$
$4x_1 + x_2 - 8x_3 + 4x_4 = -6$
R: $x_1 = -2.5$, $x_2 = 1$, $x_3 = -0.625$, $x_4 = -0.5$

19. $-8x_1 + 2x_2 - 9x_3 + x_4 = 8$
$7x_1 - 3x_2 + 7x_3 + x_4 = -5$
$2x_1 - 5x_2 - x_3 + 7x_4 = -9$
$-2x_1 - 3x_2 - 5x_3 + 6x_4 = 13$
R: $x_1 = 102$, $x_2 = -5$, $x_3 = -98$, $x_4 = -48$

20. $-2x_1 + x_2 - 5x_3 + 6x_4 = 7$
$-9x_1 - 10x_2 + x_3 + 2x_4 = 11$
$-x_1 - 4x_2 + 5x_3 - 5x_4 = 4$
$-6x_1 - 8x_2 + 3x_3 - x_4 = -12$
R: $x_1 = -82$, $x_2 = 32$, $x_3 = -129$, $x_4 = -139$

21. $2x_1 - x_2 + 3x_3 - 2x_4 + 5x_5 = 6$
$3x_1 + 4x_2 - 2x_3 + 3x_4 - 9x_5 = 11$
$5x_1 - 5x_2 - 7x_3 + 6x_4 + 2x_5 = 15$
$5x_1 - 2x_2 + 5x_3 - x_4 + 4x_5 = -6$
$7x_1 + 3x_2 - 2x_3 + 2x_4 + x_5 = -3$
R: $x_1 = 12.594$, $x_2 = 14.259$, $x_3 = 14.681$, $x_4 = 33.276$, $x_5 = 11.191$

22. $x_1 - x_2 + 3x_3 - 4x_4 + 5x_5 = -2$
$x_1 + 4x_2 - 2x_3 + 3x_4 - 3x_5 = 1$
$x_1 - 5x_2 - 7x_3 + 4x_4 + 2x_5 = 5$
$x_1 - x_2 + 2x_3 - x_4 + 4x_5 = 5$
$x_1 + x_2 - 2x_3 + 2x_4 + 2x_5 = 3$
R: $x_1 = 1.221184$, $x_2 = -1.068536$, $x_3 = 2.155763$, $x_4 = 3.183801$, $x_5 = 0.395639$

23. $x_1 + 2x_2 - x_3 + 2x_4 + 3x_5 = -1$
$-x_1 - x_2 + 2x_3 - 2x_4 + 2x_5 = 1$
$2x_1 + 2x_2 - x_3 + 11x_4 + 5x_5 = -9$
$-3x_1 + x_2 + x_3 + 5x_4 + 4x_5 = 7$
$4x_1 - 5x_2 + 3x_3 - 3x_4 + 11x_5 = 3$
R: $x_1 = -4.269816$, $x_2 = -3.092074$, $x_3 = -5.759007$, $x_4 = 0.807846$, $x_5 = 1.770216$

24. $x_1 + 4x_2 - 2x_3 + 6x_4 - 7x_5 = 3$ R: $x_1 = 5.291826$, $x_2 = 1.586716$, $x_3 = 2.805121$, $x_4 = 4.196302$, $x_5 = 0.704125$
$2x_1 - x_2 + 4x_3 + 3x_4 - 5x_5 = 6$
$x_1 + 4x_2 + 2x_3 - x_4 + 6x_5 = -3$
$-3x_1 + x_2 + x_3 - 5x_4 - 3x_5 = -6$
$7x_1 + 2x_2 - x_3 + 9x_4 - 11x_5 = -13$

25. $2x_1 - 2x_2 + 3x_3 + x_4 + 5x_5 = 1$ R: $x_1 = -2.6625$, $x_2 = -1.5$, $x_3 = 7.9625$, $x_4 = 4.9125$, $x_5 = 6.425$
$3x_1 - x_2 + 2x_3 + 5x_4 + 7x_5 = -2$
$-2x_1 + 4x_2 - x_3 + 5x_4 + 3x_5 = 2$
$6x_1 - 8x_2 - 3x_3 + 11x_4 + 5x_5 = -2$
$2x_1 + 3x_2 + x_3 + x_4 + 4x_5 = 3$

26. Un circuito que sigue las leyes de Kirchoff, cumple con el siguiente sistema de ecuaciones para las corrientes que circulan a través de él:

$$\begin{aligned} 5i_1 + 5i_2 &= 6\cos(\pi t), \\ i_3 - i_4 - i_5 &= 0, \\ 2i_4 - 3i_5 &= 0, \\ i_1 - i_2 - i_3 &= 0, \\ 5i_2 - 7i_3 - 2i_4 &= 0. \end{aligned}$$

Encontrar las corrientes involucradas.

5.3. Método de Newton

Para resolver el sistema $\vec{F}(\vec{x}) = \vec{0}$, dada una aproximación inicial $\vec{x}_0$, donde

$$\vec{F}(\vec{x}) = (f_1(x_1, x_2, ..., x_n), f_2(x_1, x_2, ..., x_n), \cdots, f_n(x_1, x_2, ..., x_n)), \text{ y } \vec{x}_0 = (x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)}).$$

Primero explicamos el *método de Newton*. Éste consiste en obtener la matriz jacobiana

$$J(\vec{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \tag{5.2}$$

así como su inversa $J^{-1}(\vec{x})$, e iterar en la forma siguiente

$$\vec{x}^{(k)} = \vec{x}^{k-1} - J^{-1}(\vec{x}^{(k-1)})\vec{F}(\vec{x}^{(k-1)}). \tag{5.3}$$

El cálculo de $J^{-1}(\vec{x})$ puede ser demasiado laborioso, razón por la cual, en vez de eso se hace un truco, que consiste en hallar un vector $\vec{y}$ que satisfaga $J(\vec{x}^{(k-1)})\vec{y} = -\vec{F}(\vec{x}^{(k-1)})$. Con ello, la nueva aproximación de $\vec{x}^{(k-1)}$ se obtiene con

$$\vec{x}^{(k)} = \vec{x}^{(k-1)} + \vec{y}, \tag{5.4}$$

iterándose hasta obtener la precisión deseada.

Algoritmo de Newton

Aproxima a la solución del sistema $\vec{F}(\vec{x}) = \vec{0}$, dada una aproximación inicial $\vec{x}^{(0)}$.

ENTRADA Número de ecuaciones e incógnitas n, aproximación inicial $\vec{x} = 0$, tolerancia TOL, número de iteraciones máximo N.

SALIDA Solución aproximada $\vec{x}$ o mensaje de error.

Paso 1 Hacer $k = 1$

Paso 2 Mientras $k \leq N$, hacer pasos 3 al 7

Paso 3 Calcular $\vec{F}(\vec{x})$ y $J(\vec{x})$

Paso 4 Resolver el sistema lineal $J(\vec{x}) = -\vec{F}(\vec{x})$

Paso 5 Hacer $\vec{x} = \vec{x} + \vec{y}$

Paso 6 Si $||\vec{y}|| < TOL$

SALIDA (La solución es $\vec{x}$)

TERMINAR

Paso 7 Hacer $k = k + 1$

Paso 8 SALIDA (El método fracasó después de N_0 iteraciones)

TERMINAR

Código de Newton

El siguiente código se ejemplifica para el sistema

$$3x - \cos(yz) - 0.5 = 0$$

$$x^2 - 81(y + 0.1)(y + 0.1) + \text{sen}(z) + 1.06 = 0$$

$$e^{-xy} + 20z + \frac{10\pi - 3}{3} = 0.$$

```
#include<stdio.h>
#include<math.h>
#define pi 4*atan(1)
   double F(int, double *);
   double P(int, int, double *);
   void INPUT(int *, int *, double *, int *, double *);
   void LINSYS(int, int *, double [][11], double *);
main() {
   double A[10][11], X[10], Y[10];
   double TOL,R;
   int N,NN,I,J,K,OK;
   INPUT(&OK, &N, &TOL, &NN, X);
   if (OK) {
      K = 1;
```

```
      while (OK && (K <= NN)) {
         for (I=1; I<=N; I++) {
            for (J=1; J<=N; J++) A[I-1][J-1] = P(I, J, X);
            A[I-1][N] = -F(I, X);
         }
         LINSYS(N, &OK, A, Y);
         if (OK) {
            R = 0.0;
            for (I=1; I<=N; I++) {
               if (abs(Y[I-1]) > R) R = abs(Y[I-1]);
               X[I-1] = X[I-1] + Y[I-1];
            }
               printf(" %2d", K);
               for (I=1; I<=N; I++) printf(" %11.8f", X[I-1]);
               printf("\n%12.6e\n", R);
            if (R < TOL) {
               OK = false;
               printf("La iteracion %d da la solucion:\n\n", K);
               for (I=1; I<=N; I++) printf(" %11.8f", X[I-1]);
            }
            else K++;
         }
      }
      if (K > NN)
         printf("El proceso no converge despues de %d iteraciones\n", NN);
   }
   getchar();
   getchar();
}
double F(int I, double *X)
{
   double f;
   switch (I) {
      case 1:
         f = 3*X[0] - cos(X[1]*X[2]) -0.5;
         break;
      case 2:
         f = X[0]*X[0] - 81*(X[1]+0.1)*(X[1]+0.1) + sin(X[2]) + 1.06;
         break;
      case 3:
         f = exp(-X[0]*X[1]) + 20*X[2] + (10*pi-3)/3;
         break;
   }
   return f;
}
double P(int I, int J, double *X)
{
   double p;
   switch (I) {
      case 1:
         switch (J) {
            case 1:
               p = 3;
               break;
```

```
            case 2:
               p = X[2]*sin(X[1]*X[2]);
               break;
            case 3:
               p = X[1]*sin(X[1]*X[2]);
               break;
         }
         break;
      case 2:
         switch (J) {
            case 1:
               p = 2*X[0];
               break;
            case 2:
               p = -162*(X[1]+0.1);
               break;
            case 3:
               p = cos(X[2]);
               break;
         }
         break;
      case 3:
         switch (J) {
            case 1:
               p = -X[1]*exp(-X[0]*X[1]);
               break;
            case 2:
               p = -X[0]*exp(-X[0]*X[1]);
               break;
            case 3:
               p = 20;
               break;
         }
         break;
   }
   return p;
}
void INPUT(int *OK, int *N, double *TOL, int *NN, double *X)
{
   int I;
   char AA;
   printf("Metodo de Newton para un sistema no lineal\n");
   *OK = false;
      while (!(*OK)) {
         printf("Dar el numero de ecuaciones\n");
         scanf("%d", N);
         if (*N >= 2) *OK = true;
         else printf("Debe ser mayor que 2\n");
      }
      *OK = false;
      while(!(*OK)) {
         printf("Dar la tolerancia\n");
         scanf("%lf", TOL);
         if (*TOL > 0.0) *OK = true;
```

```
         else printf("La tolerancia debe ser positiva\n");
      }
      *OK = false;
      while (!(*OK)) {
         printf("Dar el numero de iteraciones maximo\n");
         scanf("%d", NN);
         if (*NN > 0) *OK = true;
         else printf("Debe ser un numero natural\n");
      }
      for (I=1; I<=*N; I++) {
         printf("Dar aproximacion inicial X(%d).\n", I);
         scanf("%lf", &X[I-1]);
      }
}
void LINSYS(int N, int *OK, double A[][11], double *Y)
{
   int L,I,K,IR,IA,J,JA;
   double Z,C;
   K = N - 1;
   *OK = true;
   I = 1;
   while (OK && (I <= K)) {
      Z = abs(A[I-1][I-1]);
      IR = I;
      IA = I + 1;
      for (J=IA; J<=N; J++)
         if (abs(A[J-1][I-1]) > Z) {
            IR = J;
            Z = abs(A[J-1][I-1]);
         }
      if (Z <= 0) *OK = false;
      else {
         if (IR != I)
            for (J=I; J<=N+1; J++) {
               C = A[I-1][J-1];
               A[I-1][J-1] = A[IR-1][J-1];
               A[IR-1][J-1] = C;
            }
         for (J=IA; J<=N; J++) {
            C = A[J-1][I-1] / A[I-1][I-1];
            if (abs(C) <= 0) C = 0.0;
            for (L=I; L<=N+1; L++)
               A[J-1][L-1] = A[J-1][L-1] - C * A[I-1][L-1];
         }
      }
      I++;
   }
   if (OK) {
      if (abs(A[N-1][N-1]) <= 0) *OK = false;
      else {
         Y[N-1] = A[N-1][N] / A[N-1][N-1];
         for (I=1; I<=K; I++) {
            J = N - I;
            JA = J + 1;
```

```
            C = A[J-1][N];
            for (L=JA; L<=N; L++) C = C - A[J-1][L-1] * Y[L-1];
            Y[J-1] = C / A[J-1][J-1];
         }
      }
   }
   if (!(*OK)) printf("El sistema lineal es singular\n");
}
double abs(double val) {
   if (val >= 0) return val;
   else return -val;
}
```

5.4. Método de Broyden

En el método de Newton es necesario calcular y evaluar derivadas parciales en cada iteración. A veces es preferible evitar esto, por lo que se prefiere el *método de Broyden*, que consiste en aproximar las derivadas por medio de secantes, análogamente al método de las secantes para ecuaciones de una variable (ver sección 1.4). El método consiste en lo siguiente:

Primero se busca una matriz que cumpla la propiedad

$$A_1(\vec{x}^{(1)} - \vec{x}^{(0)}) = \vec{F}(\vec{x}^{(1)}) - \vec{F}(\vec{x}^{(0)}), \tag{5.5}$$

a partir de la cual se generan aproximaciones sucesivas

$$A_i = A_{i-1} + \frac{\vec{F}(\vec{x}^{(i)}) - \vec{F}(\vec{x}^{(i-1)}) - A_{i-1}(\vec{x}^{(i)} - \vec{x}^{(i-1)})}{||\vec{x}^{(i)} - \vec{x}^{(i-1)}||}(\vec{x}^{(i)} - \vec{x}^{(i-1)}), \tag{5.6}$$

y con ella se genera cada nueva aproximación a la solución

$$\vec{x}^{(i+1)} = \vec{x}^{(i)} A_i^{-1} \vec{F}(\vec{x}^{(i)}). \tag{5.7}$$

El siguiente algoritmo establece los pasos para aplicar este método.

Algoritmo de Broyden

Obtiene una solución aproximada al sistema no lineal $\vec{F}(\vec{x}) = \vec{0}$ dada una aproximación inicial $\vec{x}^{(0)}$.

ENTRADA Número de ecuaciones e incógnitas n, aproximación inicial $\vec{x}^{(0)}$, tolerancia TOL, número de iteraciones máximo N.

SALIDA Solución aproximada o mensaje de error.

Paso 1 Hacer $A_0 = J(\vec{x})$, siendo $J(\vec{x})$ el jacobiano de $\vec{F}(\vec{x})$

Hacer $\vec{v} = \vec{F}(\vec{x})$

Paso 2 Hacer $A = A_0^{-1}$

Paso 3 Hacer $\vec{s} = -A\vec{v}$

Hacer $\vec{x} = \vec{x} + \vec{s}$

Hacer $k = 2$

Paso 4 Mientras $k \leq N$ hacer pasos 5 al 13

Paso 5 Hacer $\vec{w} = \vec{v}$

Hacer $\vec{v} = \vec{F}(\vec{x})$

Hacer $\vec{y} = \vec{v} - \vec{w}$

Paso 6 Hacer $\vec{z} = -A\vec{y}$

Paso 7 Hacer $p = -\vec{s}^t\vec{z}$

Paso 8 Hacer $\vec{u}^t = \vec{s}^t A$

Paso 9 Hacer $A = A + \frac{1}{p}(\vec{s} + \vec{z})\vec{u}^t$

Paso 10 Hacer $\vec{s} = -A\vec{v}$

Paso 11 Hacer $\vec{x} = \vec{x} + \vec{s}$

Paso 12 Si $||\vec{s}|| < TOL$ entonces

SALIDA (La solución es $\vec{x}$)

TERMINAR

Paso 13 Hacer $k = k + 1$

Paso 14 SALIDA (El método fracasó después de N_0 iteraciones)

TERMINAR

Código de Broyden

El siguiente código se ejemplifica para el sistema

$$3x - \cos(yz) - 0.5 = 0$$

$$x^2 - 81(y + 0.1)(y + 0.1) + \text{sen}(z) + 1.06 = 0$$

$$e^{-xy} + 20z + \frac{10\pi - 3}{3} = 0.$$

```
#include<stdio.h>
#include<math.h>
#define pi 4*atan(1)
   double F(int, double *);
   double PD(int, int, double *);
   void INPUT(int *, int *, double *, int *, double *);
   void MULT(int, double *, double *, double *, double [][10]);
   void INVERT(int, double [][10], int *, double [][10]);
main() {
   double A[10][10], B[10][10], U[10], X[10], V[10], S[10], Y[10], Z[10];
   double SN,TOL,VV,ZN,P;
   int N,NN,I,J,L,K,KK,OK;
```

```
INPUT(&OK, &N, &TOL, &NN, X);
if (OK) {
   for (I=1; I<=N; I++) {
      for (J=1; J<=N; J++) A[I-1][J-1] = PD(I, J, X);
      V[I-1] = F(I, X);
   }
   INVERT(N, B, &OK, A);
   if (OK) {
      K = 2;
      MULT(N, &SN, V, S, A);
      for (I=1; I<=N; I++) X[I-1] = X[I-1] + S[I-1];
         printf("%d",K-1);
         for (I=1;I<=N;I++) printf("%11.8f",X[I-1]);
         printf("\n %12.6e\n",SN);
      while ((K < NN) && OK) {
         for (I=1; I<=N; I++) {
            VV = F(I, X);
            Y[I-1] = VV - V[I-1];
            V[I-1] = VV;
         }
         MULT(N, &ZN, Y, Z, A);
         P = 0.0;
         for (I=1; I<=N; I++) {
            P = P - S[I-1] * Z[I-1];
         }
         for (I=1; I<=N; I++) {
            U[I-1] = 0.0;
            for (J=1;J<=N;J++)
               U[I-1] = U[I-1]+S[J-1]*A[J-1][I-1];
         }
         for (I=1;I<=N;I++) {
            for (J=1;J<=N;J++)
               A[I-1][J-1] = A[I-1][J-1]+(S[I-1]+Z[I-1])*U[J-1]/P;
         }
         MULT(N, &SN, V, S, A);
         for (I=1; I<=N; I++) X[I-1] = X[I-1] + S[I-1];
         KK = K + 1;
            printf(" %2d", K);
            for (I=1; I<=N; I++) printf(" %11.8f", X[I-1]);
            printf("\n%12.6e\n", SN);
         if (SN <= TOL) {
            OK = false;
            printf("La iteracion numero %d", K);
            printf(" da la solucion:\n\n");
            for (I=1; I<=N; I++) printf(" %11.8f", X[I-1]);
         }
         else
            K = KK;
      }
      if (K >= NN)
         printf("El procedimiento no converge\n");
   }
}
getchar();
```

```
   getchar();
}
double F(int I, double *X)
{
   double f;
   switch (I) {
      case 1:
         f = 3*X[0] - cos(X[1]*X[2]) - 0.5;
         break;
      case 2:
         f = X[0]*X[0] - 81*(X[1]+0.1)*(X[1]+0.1) + sin(X[2]) + 1.06;
         break;
      case 3:
         f = exp(-X[0]*X[1]) + 20*X[2] + (10*pi-3)/3;
         break;
   }
   return f;
}
double PD(int I, int J, double *X)
{
   double pd;
   switch (I) {
      case 1:
         switch (J) {
            case 1:
               pd = 3;
               break;
            case 2:
               pd = X[2]*sin(X[1]*X[2]);
               break;
            case 3:
               pd = X[1]*sin(X[1]*X[2]);
               break;
         }
         break;
      case 2:
         switch (J) {
            case 1:
               pd = 2*X[0];
               break;
            case 2:
               pd = -162*(X[1]+0.1);
               break;
            case 3:
               pd = cos(X[2]);
               break;
         }
         break;
      case 3:
         switch (J) {
            case 1:
               pd = -X[1]*exp(-X[0]*X[1]);
               break;
            case 2:
```

```
               pd = -X[0]*exp(-X[0]*X[1]);
               break;
            case 3:
               pd = 20;
               break;
         }
         break;
   }
   return pd;
}
void INPUT(int *OK, int *N, double *TOL, int *NN, double *X)
{
   int I;
   *OK = false;
   printf("Resuelve un sistema no lineal con el metodo de Broyden\n");
      while (!(*OK)) {
         printf("Dar el numero de ecuaciones N\n");
         scanf("%d", N);
         if (*N >= 2) *OK = true;
         else printf("N debe ser mayor que 2\n");
      }
      *OK = false;
      while(!(*OK)) {
         printf("Dar la tolerancia\n");
         scanf("%lf", TOL);
         if (*TOL > 0.0) *OK = true;
         else printf("La tolerancia debe ser positiva\n");
      }
      *OK = false;
      while (!(*OK)) {
         printf("Dar el numero de iteraciones maximo\n");
         scanf("%d", NN);
         if (*NN > 0) *OK = true;
         else printf("Debe ser un numero natural\n");
      }
      for (I=1; I<=*N; I++) {
         printf("Dar aproximacion inicial X(%d)\n", I);
         scanf("%lf", &X[I-1]);
      }
}
void MULT(int N, double *SN, double *V, double *S, double A[][10])
{
   int I, J;
   *SN = 0.0;
   for (I=1; I<=N; I++) {
      S[I-1] = 0.0;
      for (J=1; J<=N; J++) S[I-1] = S[I-1] - A[I-1][J-1] * V[J-1];
      *SN = *SN + S[I-1] * S[I-1];
   }
   *SN = sqrt(*SN);
}
void INVERT(int N, double B[][10], int *OK, double A[][10])
{
   int I,J,I1,I2,K;
```

```
   double C;
   for (I=1; I<=N; I++) {
      for (J=1; J<=N; J++) B[I-1][J-1] = 0.0;
      B[I-1][I-1] = 1.0;
   }
   I = 1;
   while ((I <= N) && OK) {
      I1 = I + 1;
      I2 = I;
      if (I != N) {
         C = abs(A[I-1][I-1]);
         for (J=I1; J<=N; J++)
            if (abs(A[J-1][I-1]) > C) {
               I2 = J;
               C = abs(A[J-1][I-1]);
            }
         if (C <= 0) *OK = false;
         else {
            if (I2 != I)
               for (J=1; J<=N; J++) {
                  C = A[I-1][J-1];
                  A[I-1][J-1] = A[I2-1][J-1];
                  A[I2-1][J-1] = C;
                  C = B[I-1][J-1];
                  B[I-1][J-1] = B[I2-1][J-1];
                  B[I2-1][J-1] = C;
               }
         }
      }
      else if (abs(A[N-1][N-1]) <= 0) *OK = false;
      if (*OK)
         for (J=1; J<=N; J++) {
            if (J != I) {
               C = A[J-1][I-1] / A[I-1][I-1];
               for (K=1; K<=N; K++) {
                  A[J-1][K-1] = A[J-1][K-1] - C * A[I-1][K-1];
                  B[J-1][K-1] = B[J-1][K-1] - C * B[I-1][K-1];
               }
            }
         }
      I++;
   }
   if (*OK)
      for (I=1; I<=N; I++) {
         C = A[I-1][I-1];
         for (J=1; J<=N; J++) A[I-1][J-1] = B[I-1][J-1] / C;
      }
   else printf("El Jacobiano no tiene inversa\n");
}
```

Ejercicios

Aproximar al menos una solución para cada uno de los sistemas de ecuaciones siguientes, usando los métodos de Newton y de Broyden. Comparar la precisión de cada método por sustitución directa.

1. $x_1^2 - 2.5x_1 - 2x_1x_2 + x_2^2 + 2 = 0$
   $x_1^4 - 2x_2 + 2 = 0$
   R: $x_1 = 0.8692$, $x_2 = 1.2855$; $x_1 = 0.8692$, $x_2 = 2.4085$

2. $x_1^2 - 10x_1 + x_2^2 + 8 = 0$
   $x_1x_2^2 + x_1 - 10x_2 + 8 = 0$
   R: $x_1 = 0.9999$, $x_2 = 0.9999$; $x_1 = 2.1934$, $x_2 = 3.0204$

3. $x_1(1 - x_1) + 4x_2 = 12$
   $(x_1 - 2)^2 + (2x_2 - 3)^2 = 25$
   R: $x_1 = -1.0000$, $x_2 = 3.4999$; $x_1 = 2.5469$, $x_2 = 3.9849$

4. $-x_1(x_1 + 1) + 2x_2 = 18$
   $(x_1 - 1)^2 + (x_2 - 6)^2 = 25$
   R: $x_1 = 1.5469$, $x_2 = 10.9699$; $x_1 = -1.9999$, $x_2 = 9.9999$

5. $4x_1^2 - 20x_1 + 0.25x_2^2 + 8 = 0$
   $0.5x_1x_2^2 + 2x_1 - 5x_2 + 8 = 0$
   R: $x_1 = 0.4999$, $x_2 = 1.9999$; $x_1 = 1.0967$, $x_2 = 6.0409$

6. $\text{sen}(4\pi x_1x_2) - 2x_2 - x_1 = 0$
   $0.9204225(e^{2x_1-1} - 1) + 4x_2^2 - 2x_1 = 0$
   R: $x_1 = -0.51316$, $x_2 = -0.018376$

7. $(x_1 + x_2)^2 - x_1 - x_2 - 10 = 0$
   $(x_1 - x_2)^2 - 0.01(1 - x_1)^2 = 0$
   R: $x_1 = -1.2388$, $x_2 = -1.4627$; $x_1 = 1.8102$, $x_2 = 1.8912$
   $x_1 = -1.4745$, $x_2 = -1.2270$; $x_1 = 1.8955$, $x_2 = 1.8060$

8. $3x_1^2 - x_2^2 = 0$
   $3x_1x_2^2 - x_1^3 - 1 = 0$
   R: $x_1 = -0.4218$, $x_2 = -0.7307$; $x_1 = 0.4594$, $x_2 = 0.7958$
   $x_1 = 5.1585$, $x_2 = 8.9349$; $x_1 = -5.2326$, $x_2 = 9.0632$

9. $\ln(x_1^2 + x_2^2) - \text{sen}(x_1 \cdot x_2) = \ln(2\pi)$
   $e^{x_1 - x_2} + \cos(x_1 \cdot x_2) = 0$
   R: $x_1 = 1.7724$, $x_2 = 1.7724$

10. $x_1^4 - 2x_1x_2 + x_2^2 - 15 = 0$
    $2x_1^2 - 3x_2^2 - 7 = 0$
    R: $x_1 = 2.0381$, $x_2 = 0.6602$; $x_1 = -2.0381$, $x_2 = -0.6602$
    $x_1 = -1.9185$, $x_2 = 0.3471$; $x_1 = 1.9185$, $x_2 = -0.3471$

11. $3x_1^2 - 2x_2 - 2x_3 - 5 = 0$
    $x_2^2 + 4x_3^2 - 9 = 0$
    $8x_2x_3 + 4 = 0$
    R: $x_1 = \pm 1.8821$, $x_2 = 2.9811$, $x_3 = -0.1677$
    $x_1 = \pm 1.5610$, $x_2 = -0.3354$,$x_3 = 1.4905$
    $x_1 = \pm 0.9468$, $x_2 = 0.3354$, $x_3 = -1.4905$

12. $x_1^3 - x_2^2 + 4x_3 = 0$
    $x_1^2 - x_3 - 11 = 0$
    $x_3^3 - 16 = 0$
    R: $x_1 = 3.6769$, $x_2 = -7.7324$, $x_3 = 2.5198$
    $x_1 = 3.6769$, $x_2 = 7.7324$, $x_3 = 2.5198$

13. $x_1^2 + x_2 - 37 = 0$
    $x_1 - x_2^2 - 5 = 0$
    $x_1 + x_2 + x_3 - 3 = 0$
    R: $x_1 = 6.0000$, $x_2 = 1.0000$, $x_3 = -3.9999$
    $x_1 = 6.1710$, $x_2 = -1.0821$, $x_3 = -2.0889$

14. $x_1^3 + x_1^2x_2 - x_1x_3 + 6 = 0$
    $e^{x_1} + e^{x_2} - x_3 = 0$
    $x_2^2 - 2x_1x_3 = 4$
    R: $x_1 = -1.4560$, $x_2 = -1.6642$, $x_3 = 0.42249$

15. $x_1^2 + 2x_2^2 - 2x_1x_2 - x_3 + 2 = 0$
    $x_1 + 2x_2^2 + 3x_3^2 + 1 = 0$
    $2x_1 + x_1x_2x_3 - 5 = 0$
    R: $x_1 = -5.9733$, $x_2 = -2.2212$, $x_3 = 1.2772$
    $x_1 = 1.7348$, $x_2 = 0.7712$, $x_3 = 1.1437$
    $x_1 = -1.4346$, $x_2 = 2.6129$, $x_3 = -2.0992$; $x_1 = 0.8787$, $x_2 = -2.0183$, $x_3 = -1.8281$

16. $x_1^2 + x_2^2 - 4x_3 = 3$
    $x_1^2 + 10x_2 - x_3 = 4$
    $x_2^3 - 25x_3 = 12$
    R: $x_1 = 1.0099$, $x_2 = 0.2500$, $x_3 = -0.4793$
    $x_1 = -1.0099$, $x_2 = 0.2500$, $x_3 = -0.4793$

17. $3x_1 - \cos(x_2 \cdot x_3) - 0.5 = 0$
$4x_1^2 - 625x_2^2 + 2x_2 - 1 = 0$
$e^{-x_1 \cdot x_2} + 20x_3 + 9.4719755 = 0$
R: $x_1 = 0.5001$, $x_2 = 0.2508$, $x_3 = -0.5173$

18. $3x_1 - \cos(x_2 \cdot x_3) - 0.5 = 0$
$x_1^2 - 625x_2^2 - 0.25 = 0$
$e^{-x_1 \cdot x_2} + 20x_3 + 9.4719755 = 0$
R: $x_1 = 0.5000$, $x_2 = 0.0000$, $x_3 = -0.5235$

19. $3x_1 - \cos(x_2 \cdot x_3) - 0.5 = 0$
$9x_2 + \sqrt{x_1^2 + \operatorname{sen} x_3 + 1.06} + 0.9 = 0$
$e^{-x_1 \cdot x_2} + 20x_3 + 9.4719755 = 0$
R: $x_1 = 0.4981$, $x_2 = -0.1996$, $x_3 = -0.5288$

20. $4x_1^2 - x_2^2 + x3 + x_4 = 0$
$-x_1 + 2x_2 - 2x_3 - x_2x_4 = 0$
$x_1 - 2x_2 - 3x_3 - x_3x_4 = 0$
$x_1^2 + x_2^2 + x_4^2 = 1$
R: $x_1 = 0.4640$, $x_2 = 0.1896$, $x_3 = 0.0396$, $x_4 = -0.8652$
$x_1 = -0.9829$, $x_2 = -0.1977$, $x_3 = -0.0406$, $x_4 = -0.8530$

Bibliografía

Amelkin, V. *Ecuaciones diferenciales aplicadas a la práctica*. Editorial Mir, Moscú, 1983.

Burden, R. y J. D. Faires. *Numerical Analysis*. (7a ed.). Editorial Thomson, San Francisco, 2001.

Burden, R. y J. D. Faires. *Study Guide for Numerical Analysis*. (7a ed.). Editorial Thomson, San Francisco, 2001.

Krasnov, M. et al. *Curso de matemáticas superiores*. (2a ed.). Editorial URSS, Moscú, 2003.

Nakamura, S. *Numerical Methods with Software*. Editorial Prentice, Nueva Jersey, 1991.

Piskunov, N. *Cálculo diferencial e integral*. Editorial Mir, Moscú, 1977.

Samarski, A. A. et al. *Métodos numéricos*. Editorial URSS, Moscú, 2003.

Volkov, E. A. *Métodos numéricos*. Editorial Mir, Moscú, 1987.

Algoritmos y códigos para cálculos numéricos
de Fausto Cervantes Ortiz,
se terminó de imprimir en diciembre de 2016,
en los talleres de impresión de la
Universidad Autónoma de la Ciudad de México,
San Lorenzo 290, Col. del Valle, Del. Benito Juárez, C.P. 03100,
con un tiraje de 1000 ejemplares.
Cuidado de la edición: Ángeles Godínez Guevara
Formación de interiores: Fausto Cervantes Ortiz
Diseño de la portada: Sergio Cortés Becerril
Difusión y distribución: Ana Beatriz Alonso Osorio

El análisis numérico es una de las ramas de la matemática que mayor auge cobró con la invención de la computadora, ya que permitió simular infinidad de procesos matemáticos complejos para ser aplicados a fenómenos del mundo real. La habilidad en el manejo de esta herramienta numérica es de clara utilidad para el estudiante, quien deberá plantear modelos que enfrenten la solución de problemas prácticos.

El presente manual de algoritmos tiene la intención de acompañar el trabajo de los estudiantes de ingeniería y licenciaturas afines a lo largo del curso Métodos Numéricos que ofrece la UACM, el cual consta de dos partes esenciales: exponer al estudiante los fundamentos matemáticos de los métodos y desarrollar códigos que le permitan aplicarlos para realizar los cálculos requeridos, aprovechando las ventajas de la computadora.

Algoritmos y códigos para cálculos numéricos trata los temas matemáticos de uso más frecuente: ecuaciones no lineales, interpolación, integración y derivación, ecuaciones diferenciales ordinarias y sistemas lineales y no lineales. En cada capítulo se han integrado secciones de ejercicios, así como las soluciones a muchos de ellos.

UACM
Universidad Autónoma de la Ciudad de México
Nada humano me es ajeno

Biblioteca del Estudiante

ISBN 978-6079465223
9 786079 465223