Métodos numéricos

para CBS

Fausto Cervantes Ortiz

Índice

Capítulo 1

Elementos de Python 3

El lenguaje de programación Python tiene muchas capacidades que lo hacen un instrumento muy útil para diversos propósitos. A diferencia de otros lenguajes más populares, Python usa una sintaxis muy simple, lo que permite pasar de los algoritmos (o pseudocódigos) a los programas de manera casi inmediata. No se pretende aquí dar un curso completo del mismo, sino simplemente examinar algunas de las funciones básicas que usaremos para los métodos estudiados. Como se usan arreglos, gráficas, etc., se presentan también algunos de los módulos usuales en el campo. Existen algunos otros módulos (como `sympy`) que no se usarán aquí, pero que pueden ser de gran utilidad para otros cursos. Se recomienda a los interesados explorar este lenguaje y su ambiente de programación más allá de lo que se usará en este libro.

1.1. Instalación de Python 3

El sitio oficial de Python es `www.python.org`, y ahí se puede descargar siempre la versión más reciente del intérprete. Sin embargo, como esta distribución es la más estándar, no contiene los módulos que requerimos para realizar cálculos numéricos, gráficas, etc. Aunque tales módulos se pueden descargar e instalar, siempre existen problemas de incompatibilidad si no se instalan las versiones adecuadas. Por ello, en lugar de eso, se recomienda usar otra distribución de python 3.

En el sitio `www.pyzo.org` hay una distribución de python 3 llamada *Pyzo*. Esta distribución contiene todos los módulos adicionales sin problemas de compatibilidad. Además tiene la ventaja adicional de no requerir privilegios de administrador para instalarse, lo que le permite usarse en cualquier computadora. Se recomienda usar esta distribución para familiarizarse con el lenguaje y ambiente de programación de python 3.

1.2. Entrada y salida de datos

Para hacer que el programa despliegue información en la pantalla, basta la instrucción simple:

`print(mensaje)`.

Aquí, `mensaje` puede ser un mensaje de texto explícito (por ejemplo, el clásico `Hola!`) o el nombre de alguna variable, en cuyo caso se desplegará el valor presente de esa variable. También puede ser ambos. Para distinguir el mensaje de texto explícito de el nombre de una

variable, en los mensajes de texto se deben usar comillas, simples o dobles, para abrir y cerrar el mensaje.

Ejemplos

Si tenemos una variable llamada `x`, y su valor presente es 1, al dar

```
print(x)
```

se tendrá como salida

```
1
```

Pero si escribimos

```
print('x')
```

la salida será

```
x
```

Escribir

```
print('x vale ',x)
```

dará como salida

```
x vale 1
```

Para introducir información desde el teclado se usa

```
input(mensaje)
```

donde `mensaje` puede ser un mensaje opcional, como `Dar el valor de x` . Sin embargo, esta instrucción sólo permite introducir la información como texto, aún cuando se escriban números, así que si queremos que el programa lea un número, se debe dar

```
eval(input(mensaje))
```

Ejemplos

Si queremos leer el valor de una variable llamada `x`, podemos dar

```
x=input()
```

y si escribimos por ejemplo, 1, y después damos

```
x
```

veremos como salida

```
'1'
```

Pero no nos dejemos engañar, no es un número, sino una cadena de texto. Si damos

```
'1'+x
```

veremos como salida

```
'11'
```

o sea que no sumó dos números, sino que concatenó dos cadenas de texto.

Si queremos que lo reconozca como número, daremos

```
x=eval(input())
```

y si damos 1, y luego

```
x
```

dará la salida

```
1
```

y si damos

```
x+1
```

obtendremos como salida

```
2
```

1.3. Operaciones

Las cuatro operaciones aritméticas básicas se realizan con:

- Suma: `+`
- Resta: `-`
- Multiplicación: `*`
- División: `/`

Además de estas operaciones, se agregan las siguientes:

- Potencia: `**`
- División entera: `//`
- Módulo: `%`

Ejemplos

Si tenemos

```
x=3
```

y

```
y=5
```

al dar

```
x**2+y**2
```

tendremos como resultado

```
34
```

Si damos

```
2*y//x
```

el resultado es

```
3
```

y si damos

```
2*y%x
```

obtendremos como salida

```
1
```

1.4. Archivos

Aunque dar información con el teclado y recibir información a través de la pantalla es muy útil, además de ser lo más común, para algunos cálculos numéricos (que es el propósito de este libro) es necesario poder leer información desde algún archivo, así como escribir en archivo algunos resultados importante.

Antes de leer o escribir archivos, es necesario abrirlos. Para ello se usa la instrucción `open`, en la forma siguiente

```
f=open('nombre.txt','r')
```

donde `nombre.txt` es el nombre del archivo que deseamos leer, y `r` indica que sólo vamos a leer. Si queremos crear un archivo nuevo para escribir en él, debemos dar `w`. Si queremos un archivo nuevo donde podamos leer y escribir damos , `r+` y si queremos agregar texto a un archivo existente que ya tiene texto, damos `a`.

Una vez abierto el archivo, para leer se usan las instrucciones `read()`, `readline()` y `readlines()`. Si después de leer un archivo queremos regresar al inicio del mismo, usaremos la instrucción `seek`.

Ejemplos

Para ilustrar el funcionamiento de estas instrucciones, supondremos que se tiene un archivo que contiene las siguientes líneas:

```
1234567890 abcdefghijklmnopqrstuvwxyz ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

La instrucción

`f.read(5)`

devuelve

`12345`

la instrucción

`f.read()`

devuelve

```
1234567890\nabcdefghijklmnopqrstuvwxyz\nABCDEFGHIJKLMNOPQRSTUVWXYZ
```

la instrucción

`f.readline()`

devuelve

`1234567890`

y la instrucción

`f.readlines()`

devuelve

```
['1234567890\n','abcdefghijklmnopqrstuvwxyz\n','ABCDEFGHIJKLMNOPQRSTUVWXYZ']
```

El último caso es una lista. Más adelante se describirá esto. Para regresar al inicio, la instrucción es:

```
f.seek(0)
```

Para escribir en un archivo, es necesario que lo que se va a escribir sea una cadena de texto. Entonces, si se tienen números, primero hay que convertirlos a texto con la instrucción `str()`. Si no se escribe la instrucción apropiada, no se darán saltos de línea ni espacios, por lo cual es necesario tener en cuenta el uso de estas instrucciones.

Ejemplos

Supongamos que abrimos un archivo para escribir, con la instrucción

```
g=open('datos.dat','w')
```

y queremos escribir los valores de `x` y `y`, que presentemente son 0 y 1, por ejemplo. Entonces debemos dar las instrucciones

```
k1=str(x)
```

```
k2=str(y)
```

```
g.write(k1+k2)
```

Ahora bien, aunque esto escribe en el archivo los valores deseados, si no añadimos otra instrucción, ambas cadenas de texto quedarán concatenadas, es decir, se escribirá

```
01
```

por lo cual hay que dar alguna otra instrucción que los separe. Algunas opciones son

```
g.write(k1+' '+k2)
```

que intercalará un espacio entre los dos valores, o sea

```
0 1
```

Otra posibilidad es

```
g.write(k1+'\t'+k2)
```

que escribirá

```
0	1
```

es decir, intercalará un carácter de tabulación. O bien

```
g.write(k1+'\n'+k2)
```

que escribirá

```
0
```

```
1
```

es decir, escribirá cada carácter en una línea nueva.

Es importante tener en cuenta que, si declaramos un archivo para escribir como nuevo, pero el mismo ya existe, se borrará y se creará otro nuevo con el mismo nombre, pero vacío; entonces es importante tener cuidado de no borrar otros archivos por descuido.

1.5. Control de flujo

Cuando se ejecuta un programa, frecuentemente será necesario indicar opciones a seguir, dependiendo de los resultados de algún cálculo, o algún otro tipo de información que se reciba. Para ello es necesario hacer alguna comparación, además de establecer claramente el camino a seguir en cada caso. Esto se hace con `if`, `else` y `elif`. Las comparaciones se hacen con `>`, `<`, `==`, `<=`, `>=` y `!=`.

Ejemplos

Supongamos que se tiene un número `a`, y queremos que si es diferente de cero se sume a `b`, y si es cero se multiplique por dos a `b`. Las instrucciones serían

```
if a!=0:
    c=a+b
else:
    c=2*b
```

Nótese que se deben dar 4 espacios (a esto se le llama indentar) para que se reconozca como el conjunto de pasos a seguir después del `if`.

Supongamos ahora que tenemos un número `d`, y queremos que si es cero se multiplique por dos a `a`, que si es positivo se sume a `b` y que si es negativo se sume a `c`. Las instrucciones serían

```
if d==0:
    num=2*a
elif d>0:
    num=b+d
else:
    num=c+d
```

1.6. Ciclos

En los métodos numéricos (así como en muchas otras aplicaciones) es bastante frecuente la necesidad de iterar un proceso. Los iteradores a usar son `for` y `while`. La diferencia entre ellos es que en `for` se tiene un número bien definido de iteraciones, mientras que en el `while` el número está indeterminado y sólo se detendrá al cumplirse alguna condición, pudiendo cometerse el error de entrar a un ciclo infinito si no se tiene cuidado.

Ejemplos

Para escribir 10 veces la palabra *Salud*, podemos hacer

```
for i in range(10):
    print('Salud')
```

Para escribir los cuadrados de los primeros 20 números naturales, se puede usar

```
for i in range(1,21):
    print(i**2)
```

Nótese que en este caso se dan dos números, el principio y el fin. También que el fin no es 20, sino 21. Esto es porque una característica de Python es contar desde cero, no desde 1; además de ello, al llegar a la última iteración, si el contador ya llegó al último valor, ya no entra al ciclo para evaluar. Esta es una característica inusual de Python, lo que obliga a tener que aumentar uno al número de iteraciones, que puede llevar a muchas confusiones. Sin embargo, una vez que se toma costumbre, esto no es un problema serio.

Por otro lado, para escribir cuadrados de números naturales menores que 5000 se puede usar

```
i=0
k=0
while k<=5000:
    k=i**2
    i=i+1
    print(k)
```

Ahora bien, si se saben usar, las instrucciones `for` también se puede usar para iteraciones que puedan terminar antes de lo estimado, por ejemplo, el programa anterior se puede escribir también como

```
i=0
k=0
for i in range (0,101):
    k=i**2
    print(k)
    if k>=5000:
        break
```

Si se corren ambos programas, se verá que arrojan los mismos resultados. El límite de 100 se eligió porque es sabido que 100^2=1000, que cumple sobradamente con la condición. Esto sirve también para ilustrar el uso de `break` para salir de un ciclo.

1.7. Funciones

En matemáticas, prácticamente todo gira alrededor de las funciones. Para evaluar una función (no necesariamente matemática, aunque en nuestro caso sí), es necesario definirla primero, con `def`. Es importante especificar bien todos los argumentos de entrada, así como la salida que se desea obtener, usando `return`.

Ejemplos

Sea la función que, dados x y y, calcula $x^2 + y^2$. Esto se hace con las instrucciones

```
def f(x,y):
    z=x**2+y**2
    return z
```

Así, por ejemplo, si queremos calcular $f(2,3)$, después de definir a la función f, simplemente necesitamos dar la instrucción

```
z=f(3,4)
```

que nos devolverá el valor

```
25
```

1.8. Listas

Si queremos, por ejemplo, definir un vector, necesitamos dar una lista de tres números. Pero también necesitamos que los tres números estén siempre ligados entre sí. Esto se hace definiendo una lista. Para ello se usan paréntesis cuadrados y comas como separadores. Para referirnos a los elementos de usamos índices entre paréntesis cuadrados. Esto se muestra en los ejemplos.

Ejemplos

Definimos el vector $\vec{v} = (1, 2, 3)$ con

`v=[1,2,3]`

para referirnos a la componente v_x, se usa

`v[0]`

lo que nos devuelve

`1`

Podemos definir la matriz $A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix}$ con

`A=[[1,2],[3,4]]`

y para referirnos al elemento a_{11} usamos

`A[0][0]`

lo que nos devuelve

`1`

Recuérdese que Python cuenta desde 0, no desde 1, por lo que el primer elemento de una lista no se identifica con el número 1, sino con el 0, según se observa.

1.9. Cálculos con matrices

Python, como cualquier lenguaje de programación de alto nivel, puede usarse para programar diversas tareas de cálculo vectorial y matricial usando listas, como las que se describieron en la sección anterior. Sin embargo, existen módulos especiales que tienen como función simplificar algunos cálculos. A continuación se mencionan varias de estas tareas. En cada caso, además de dar el comando adecuado, se da con anticipación el módulo que se necesita importar para realizar los cálculos necesarios. Esto no quiere decir que se deba escribir tal instrucción cada que se use el comando, sino sólo una vez, al principio de cada programa donde se haga uso del mismo.

Definición de una matriz de m renglones por n columnas

```
from numpy import *
A=array([[a11,a12...,a1n],[a21,...a2n],...,[am1,...,amn]])
```

siendo `aij` el elemento de la matriz A en el renglón i y la columna j.

La suma de matrices se hace con el operador `+`. La multiplicación por un escalar se hace con `c*A`, la multiplicación de dos matrices se hace con `dot(A,B)`.

Para trasponer una matriz se usa:

`A.transpose`.

La matriz unitaria de orden n se obtiene con: `identity(n)`.

Si se desea obtener una matriz que conste sólo de ceros, se usa (para una de 2×2):

```
zeros((2,2))
```

Para una matriz que conste sólo de unos, se usa (para una de 2×2):

```
ones((2,2))
```

Para invertir una matriz se usa:

```
from numpy.linalg import * inv(A)
```

y para calcular su determinante:

```
from numpy.linalg import * det(A)
```

Para hallar los eigenvalores:

```
from numpy.linalg import * eigvals(A)
```

Y para la descomposición LU se usa:

```
from scipy.linalg import * lu(A)
```

Solución de sistemas de ecuaciones lineales
Si se tiene un sistema de n ecuaciones lineales con n incógnitas,

$$\begin{array}{ccccc} a_{11}x_1 & \ldots & a_{1n}x_n & = & b_1 \\ \vdots & & & = & \vdots \\ a_{n1}x_1 & \ldots & a_{nn}x_n & = & b_n \end{array} \tag{1.1}$$

se puede escribir como una operación matricial en la forma

$$A\vec{x} = \vec{b} \tag{1.2}$$

donde A es la matriz de coeficientes, $\vec{x}$ es el vector columna que da los valores de x_i, y $\vec{b}$ es el vector columna con los términos independientes del sistema.

Python nos permite resolver el sistema anterior con:

```
from numpy.linalg import *
x=solve(A,b)
```

1.10. Gráficas

Para graficar se debe importar el módulo `pylab`, que tiene bastantes capacidades de graficación. A continuación algunos ejemplos fáciles de entender, que se pueden modificar de acuerdo a las necesidades de cada caso.

1.10.1. Gráfica cartesiana

Para una gráfica cartesiana se requiere dar el dominio `x` antes de definir la función. Después se puede dar la función dentro de la instrucción `plot`, como se ve en el ejemplo siguiente:

```
from numpy import *
from pylab import *
x=linspace(-pi,pi,100)
plot(x,sin(x),'*',color='black')
show()
```

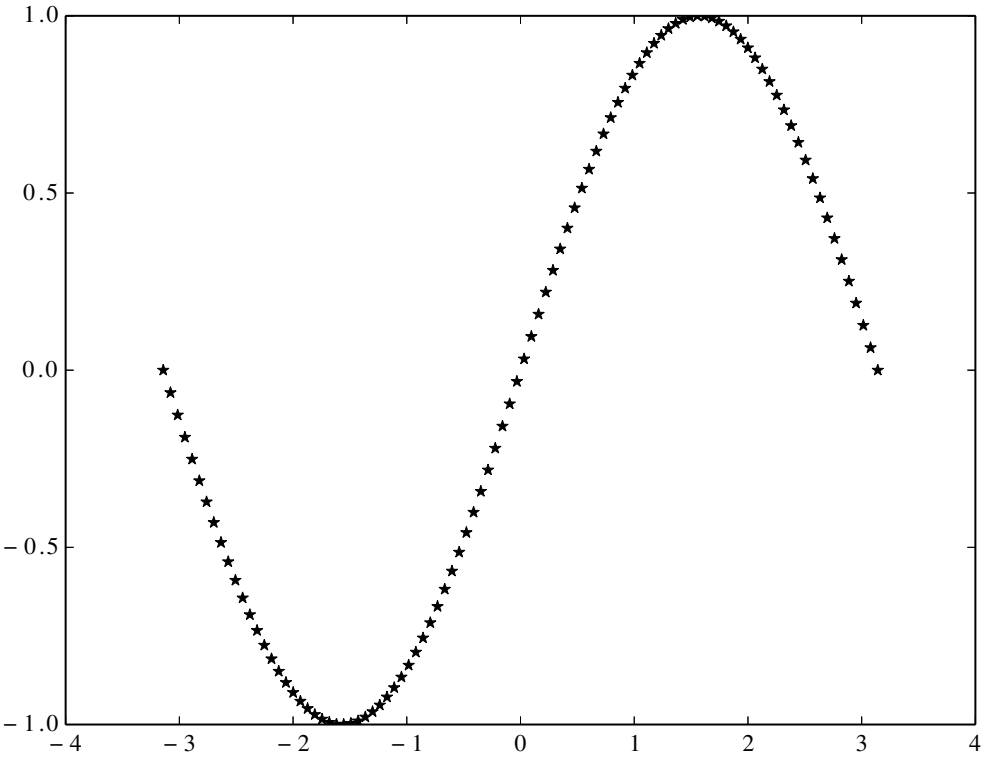

Figura 1.1: Gráfica cartesiana.

El símbolo entre apóstrofes * hará que la gráfica ponga marcas en forma de estrellas, como se ve en la figura 1.1. Si queremos que en la gráfica haya marcas de otro tipo, podemos escribir entre los apóstrofes `+` (cruces), `o` (círculos), `x` (taches). Si queremos que la gráfica esté unida por líneas se escribirá entre los apóstrofes `-` (continua), `--` (con guiones), : (punteada), `-.` (guiones y puntos). También se puede controlar el color de las líneas. Cabe señalar que se puede escribir solamente `plot(x,y);` sin dar el tipo de marca gráfica deseada ni el color, con

lo que se obtendrán líneas continuas de color azul. Estos modificadores también funcionan en los ejemplos de las siguientes gráficas.

1.10.2. Varias gráficas juntas

Para graficar varias funciones en una sola gráfica, se puede hacer, por ejemplo:

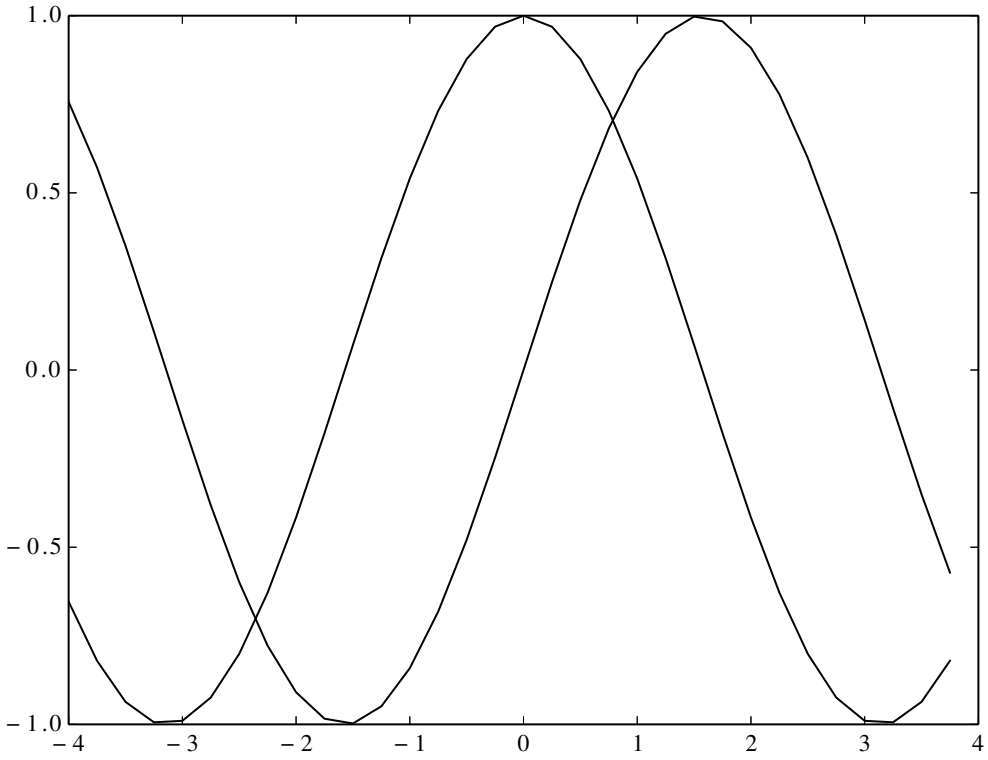

Figura 1.2: Varias funciones en una sola gráfica.

```
from numpy import *
from pylab import *
x=arange(-4,4,0.25)
y1=sin(x)
y2=cos(x)
plot(x,y1,x,y2)
show()
```

lo que desplegará ambas funciones en la misma gráfica. Si lo que se desea más bien, es tener una figura con varias gráficas por separado, se puede intentar algo como lo que sigue:

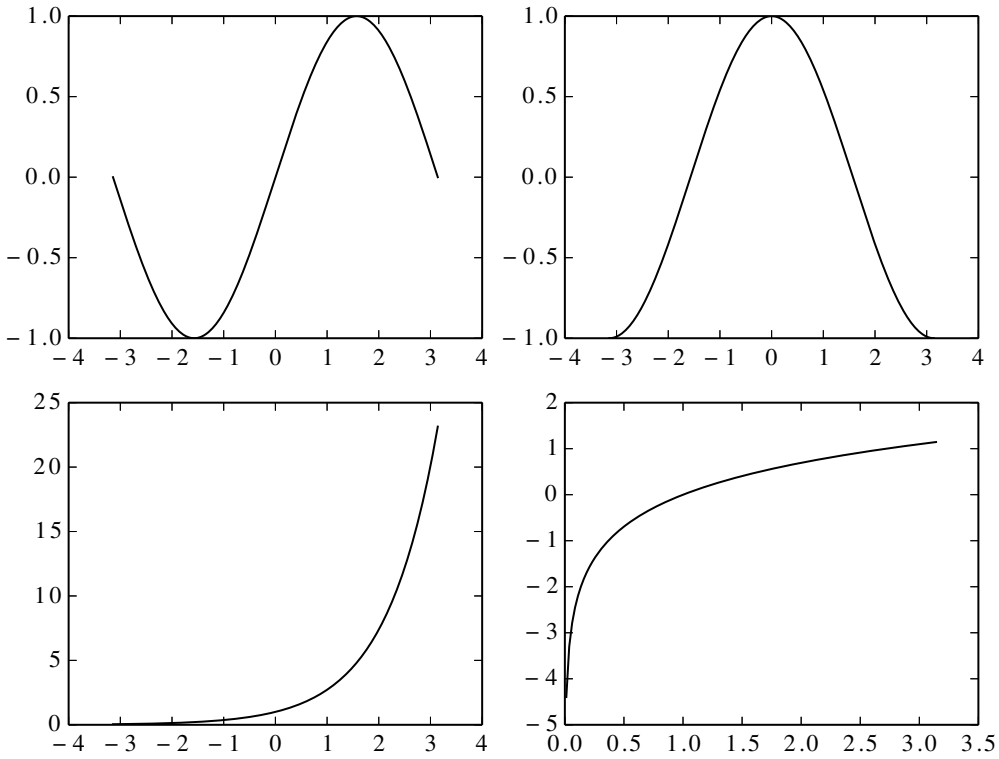

Figura 1.3: Varias gráficas en una sola figura.

```
from numpy import *
from pylab import *
x=linspace(-pi,pi,200)
subplot(2,2,1)
plot(x,sin(x))
subplot(2,2,2)
plot(x,cos(x))
subplot(2,2,3)
plot(x,exp(x))
subplot(2,2,4)
plot(x,log(x))
show()
```

Los números dados como parámetros para la instrucción `subplot` indican que habrá 2 renglones, 2 columnas, y que se está haciendo la gráfica 1, 2, 3 ó 4, según sea el caso.

1.10.3. Gráfica tridimensional de una superficie

Para graficar una función de dos variables, lo que implica una figura tridimensional, se requieren más instrucciones. Un ejemplo, con la función $f(x, y) = e^{-x^2/4+y^2/2}$, se da a continuación:

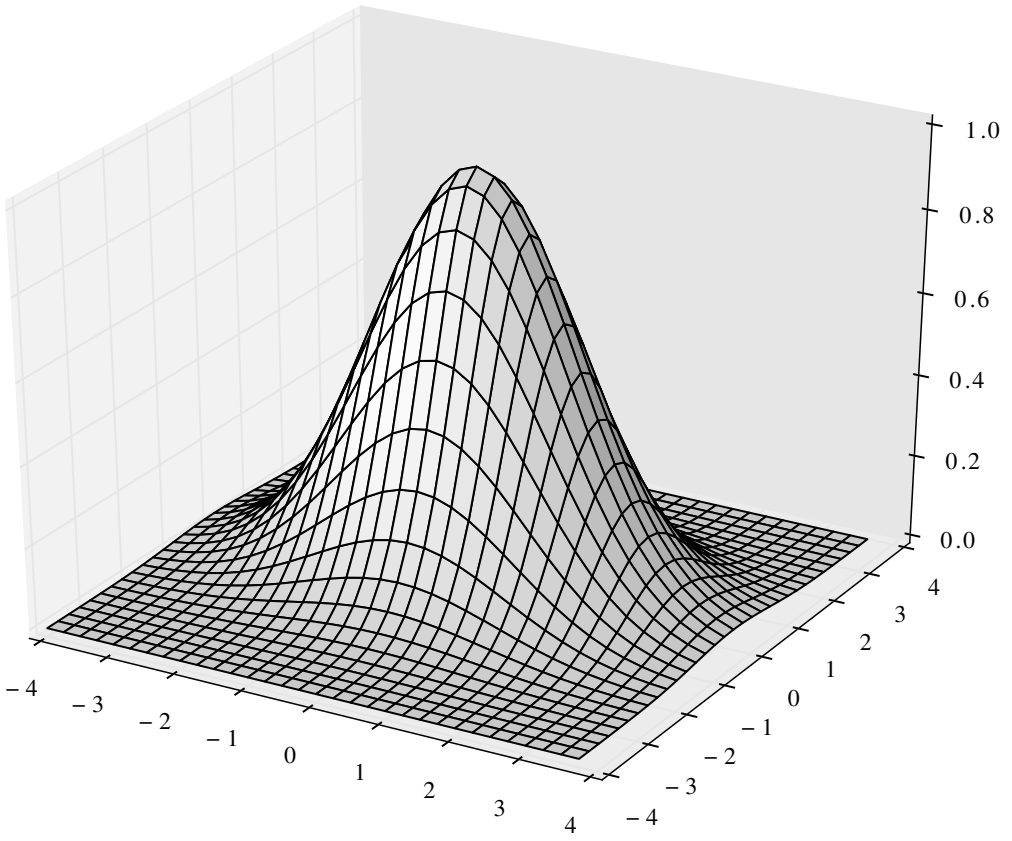

Figura 1.4: Gráfica de la función $f(x, y) = e^{-x^2/4+y^2/2}$.

```
from numpy import *
from pylab import *
from mpl_toolkits.mplot3d import axes3d
fig = figure()
ax=axes3d(fig)
x=arange(-4, 4, 0.25)
y=arange(-4, 4, 0.25)
x,y=meshgrid(x,y)
z=exp(-(x**2/4+y**2/2))
ax.plot_surface(x,y,z,rstride=1,cstride=1,color='white')
show()
```

1.10.4. Gráfica bidimensional de los contornos de una función de tres variables

Si en lugar de la superficie de la función tridimensional sólo queremos las curvas de nivel, es necesario definir en forma diferente las cosas. El siguiente ejemplo nos muestra cómo graficar las curvas de nivel de la función $f(x, y) = \ln(x^2 + y^2 + 0.1)$:

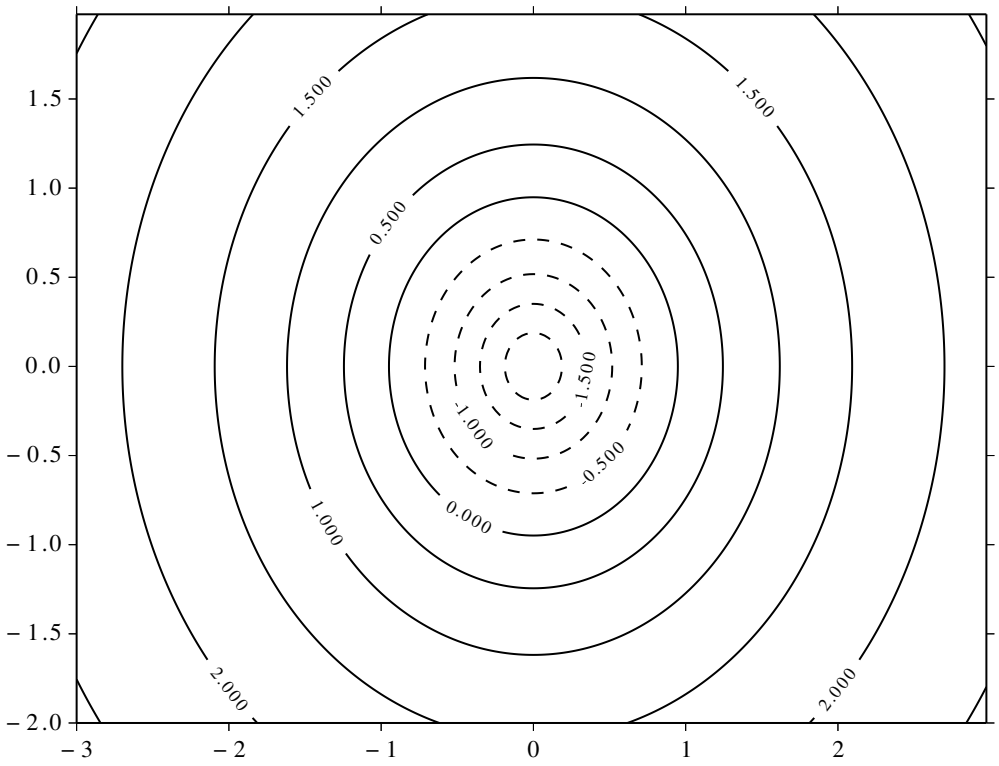

Figura 1.5: Gráfica de contornos.

```
from matplotlib import *
from numpy import *
from matplotlib.pyplot import *
rcParams['xtick.direction'] = 'out'
rcParams['ytick.direction'] = 'out'
delta = 0.025
x=arange(-3.0, 3.0, delta)
y= arange(-2.0, 2.0, delta)
X,Y=meshgrid(x, y)
```

```
Z=log(X**2+Y**2+0.1)
figure()
cosa=contour(X,Y,Z,10,colors='k')
clabel(cosa,fontsize=9, inline=1)
show()
```

1.10.5. Gráfica tridimensional de una curva paramétrica

Así como es posible graficar superficies tridimensionales, también se pueden tener curvas paramétricas. El siguiente ejemplo nos muestra una forma de graficar la curva paramétrica $\vec{r}(t) = e^{-0.1t}\,\text{sen}(t)\hat{i} + e^{-0.1t}\cos(t)\hat{j} + t\hat{k}$:

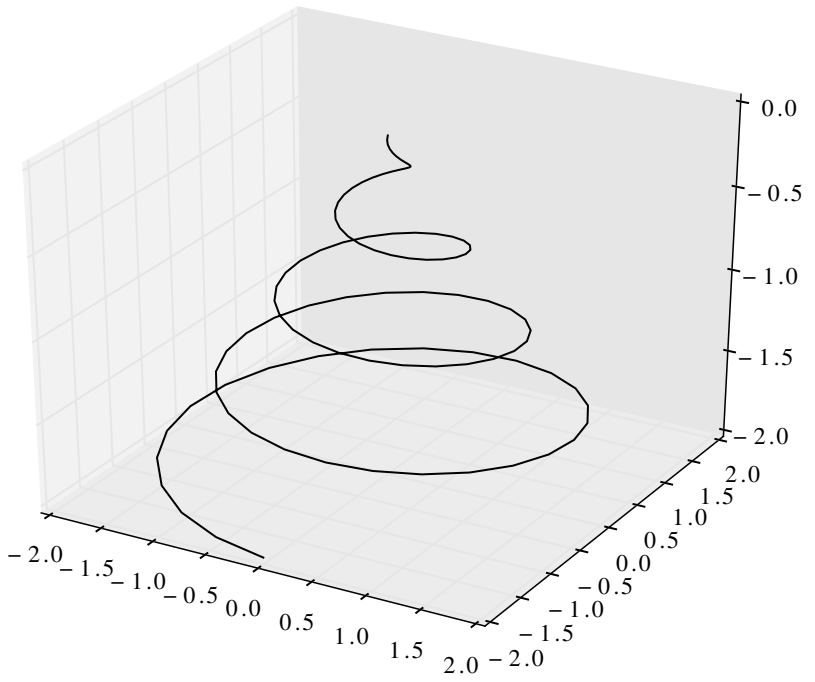

Figura 1.6: Gráfica de una curva paramétrica.

```
from matplotlib import *
from mpl_toolkits.mplot3d import Axes3D
from numpy import *
from matplotlib.pyplot import *
```

```
fig=figure()
ax=fig.gca(projection='3d')
t=linspace(0,6*pi,100)
x=exp(-0.1*t)*cos(t)
y=exp(-0.1*t)*sin(t)
ax.plot(x,y,t)
show()
```

1.10.6. Gráficas de conjuntos de datos

Si se tiene un conjunto de datos en parejas de la forma (x_i, y_i), para hacer una gráfica con ellos, simplemente se definen arreglos para x y para y independientemente, y se grafica como una función, según se ejemplifica.

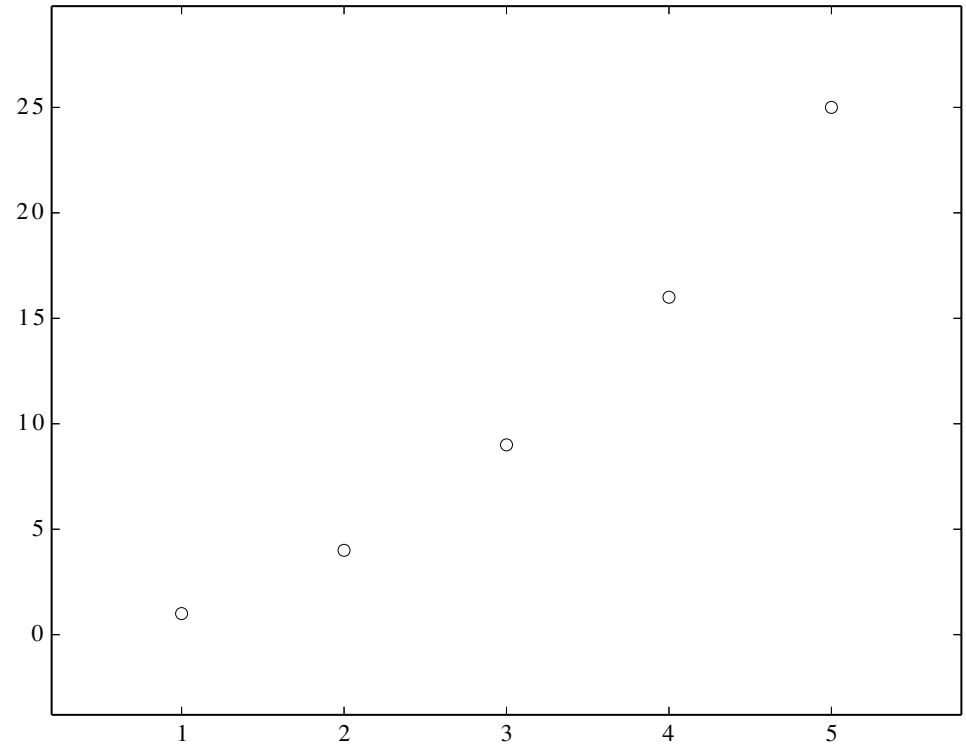

Figura 1.7: Gráfica de un conjunto de datos en el plano.

```
from numpy import *
from pylab import *
```

```
x=array([1,2,3,4,5])
y=array([0.1,0.4,0.9,1.6,2.5])
plot(x,y)
show()
```

1.10.7. Conjuntos de datos en tres dimensiones

Para ternas de datos podemos hacer una gráfica tridimensional. Para ello es necesario que cada arreglo contenga un renglón para cada valor de las abscisas y de las ordenadas. Esto se ve claramente en el siguiente ejemplo.

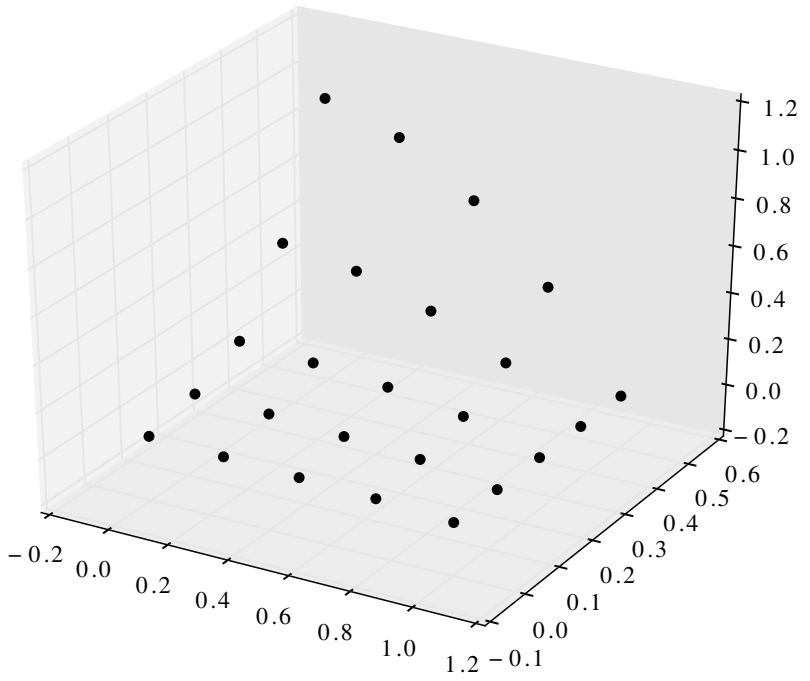

Figura 1.8: Gráfica de un conjunto de datos en el espacio.

```
from numpy import *
x=array([[0.,0.,0.,0.,0.],
        [0.25,0.25,0.25,0.25,0.25],
        [0.5,0.5,0.5,0.5,0.5],
        [0.75,0.75,0.75,0.75,0.75],
```

```
       [1.,1.,1.,1.,1.]])
y=array([[0.,0.125,0.25,0.375,0.5],
       [0.,0.125,0.25,0.375,0.5],
       [0.,0.125,0.25,0.375,0.5],
       [0.,0.125,0.25,0.375,0.5],
       [0.,0.125,0.25,0.375,0.5]])
z=array([[0.05,0.1,0.2,0.5,1.0],
       [0.04,0.09,0.18,0.45,0.9],
       [0.03,0.07,0.15,0.35,0.7],
       [0.02,0.05,0.1,0.2,0.4],
       [0.,0.,0.,0.,0.]])
from pylab import *
from mpl_toolkits.mplot3d import axes3d
fig = figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x,y,z)
show()
```

Si queremos que los datos estén unidos por líneas, en lugar de estar representados como puntos aislados, en lugar de

`ax.scatter(x,y,z)`

se debe usar

`ax.plot_wireframe(x,y,z)`

Ejercicios

1. Escribir un programa que pregunte al usuario su nombre, y que escriba `'Hola nombre'`, siendo `nombre` el nombre dado por el usuario.
2. Escribir un programa que pregunte el nombre del usuario y lo escriba en un archivo llamado `'nombre.txt'`.
3. Escribir un programa que pregunte al usuario su edad, y si es mayor o igual que 18, imprima `Ya puedes votar`, y que imprima `Todavía no puedes votar` en otro caso.
4. Escribir un programa que pregunte al usuario un número entero `n`, y que realize la suma

$$1 + 2 + 3 + \cdots + n = \sum_{i=1}^{n} i$$

   e imprima el resultado.
5. Escribir un programa que solicite al usuario un conjunto de calificaciones calcule el promedio de ellas.
6. Escribir un programa que defina una función para calcular el valor del polinomio $x^6 - 5x^5 + 4x^4 - 3x^3 + 2x^2 - x + 1$, dado un valor de x.
7. Escribir un programa que solicite al usuario las dimensiones y los valores de una matriz, y después imprima la matriz.
8. Escribir programas que hagan las gráficas de las siguientes funciones:

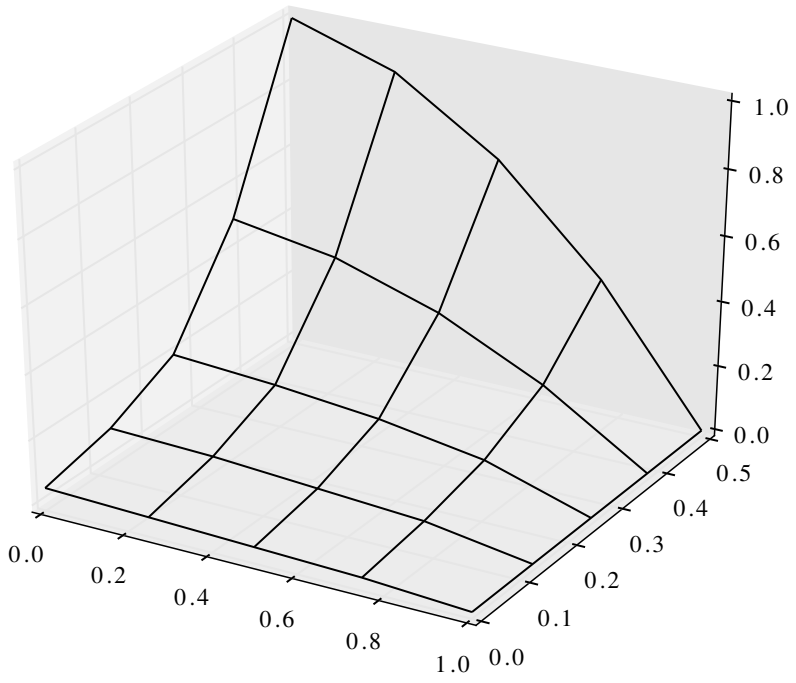

Figura 1.9: Gráfica de un conjunto de datos como superficie.

a) $f(x) = \text{sen}(2\pi x)$

b) $f(x) = \sqrt{x^2 + 1}$

c) $f(x) = \ln(1 + x)$

d) $f(x) = \frac{1}{x^2+1}$

e) $f(x) = e^{-x^2}$

9. Escribir un programa que solicite al usuario un conjunto de parejas de datos, y haga una gráfica de los mismos.

10. Escribir un programa que lea ternas de datos de un archivo llamado `'datos.txt'` y haga una gráfica tridimensional de los mismos.

Soluciones

1.
```
x=input('¿Cuál es tu nombre? ')
print('Hola ',x)
```

2.
```
x=input('¿Cuál es tu nombre? ')
```

```
f=open('nombre.txt','w')
f.write(x)
f.close()
```

3.
```
n=eval(input('Dar edad ')) if n>=18:
    print('Ya puedes votar ')
else:
    print('Aún no puedes votar ')
```

4.
```
n=eval(input('Dar un número entero positivo '))
suma=0
for i in range (n+1):
    suma=suma+i
print('El valor de la suma es ',suma)
```

5.
```
from numpy import *
n=eval(input('Dar el número de calificaciones '))
x=zeros(n)
for i in range (n):
    print('Dar la calificación número ',i+1)
    x[i]=eval(input())
suma=0
for i in range (n):
    suma=suma+x[i-1]
prom=suma/n
print('El promedio es ',prom)
```

6.
```
def f(x):
    y=x**6-5*x**5+4*x**4-3*x**3+2*x**2-x+1
    return y
x=eval(input('Dar x '))
y=f(x)
print('f(',x,')=',y)
```

7.
```
from numpy import *
n=eval(input('Dar el número de renglones '))
m=eval(input('Dar el número de columnas '))
x=zeros((n,m))
for i in range (n):
    for j in range (m):
        print('Dar el valor de x',i+1,j+1,' ')
        x[i][j]=eval(input())
print(x)
```

8. *a*)
```
from numpy import *
from pylab import *
x=linspace(0,1,100)
y=sin(2*pi*x)
plot(x,y)
show()
```

b)
```
from numpy import *
from pylab import *
x=linspace(-1,1,100)
y=sqrt(x**2+1)
plot(x,y)
show()
```

c)
```
from numpy import *
from pylab import *
x=linspace(0,2,100)
y=log(x+1)
plot(x,y)
show()
```

d)
```
from numpy import *
from pylab import *
x=linspace(-2,2,100)
y=1/(x**2+1)
plot(x,y)
show()
```

e)
```
from numpy import *
from pylab import *
x=linspace(-2,2,100)
y=exp(-x**2)
plot(x,y)
show()
```

9.
```
from numpy import *
n=eval(input('Dar el número de pares de datos '))
x=zeros(n)
y=zeros(n)
for i in range (n):
    print('Dar el dato x(',i+1,')')
    x[i]=eval(input())
    print('Dar el dato y(',i+1,')')
    y[i]=eval(input())
from pylab import *
plot(x,y)
show()
```

10.
```
from numpy import *
f=open('datos.txt','r')
n=len(f.readlines())
f.seek(0)
x=zeros(n)
y=zeros(n)
z=zeros(n)
for i in range (n):
    linea=f.readline()
    if not linea:
        break
    kk=linea.split('\t')
    x[i]=kk[0]
    y[i]=kk[1]
    z[i]=kk[2]
from pylab import *
from mpl_toolkits.mplot3d import axes3d
fig = figure()
ax = fig.add_subplot(111, projection='3d')
ax.scatter(x,y,z,color='black')
show()
```

Capítulo 2

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, posición falsa, secantes y Newton-Raphson. Cada uno de estos métodos puede ser más conveniente que los otros para diferentes casos, dependiendo eso tanto de la ecuación a resolver como de la facilidad para codificarlo.

2.1. 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

$$p_1 = a_1 + \frac{b_1 - a_1}{2} = \frac{a_1 + b_1}{2}. \tag{2.1}$$

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, lo que por inspección nos dará los puntos iniciales.

Enseguida se da el algoritmo general.

2.1.1. 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.

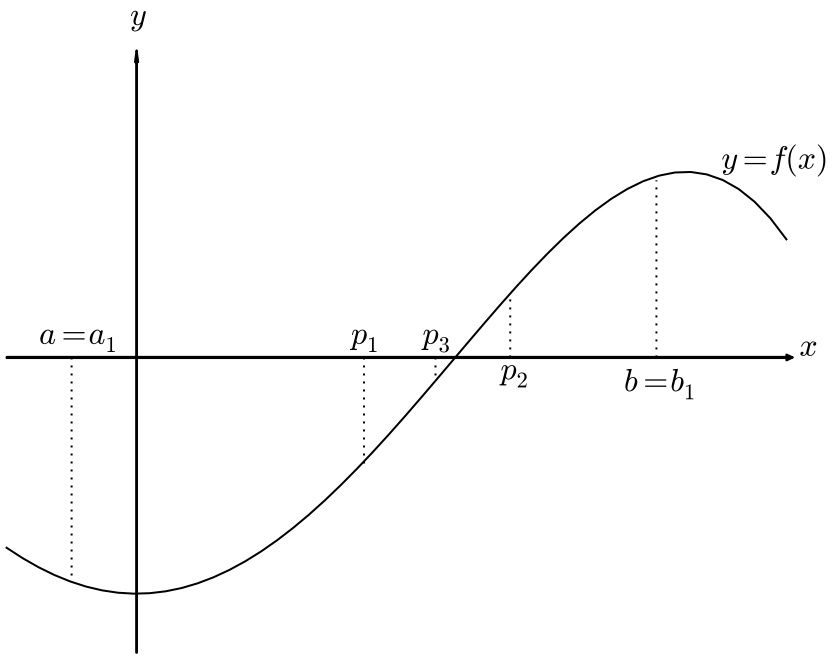

Figura 2.1: Método de bisección.

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

2.1.2. Código para el método de bisección

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

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

```
from math import *
def f(x):
    y=x-x**3
    return y
a=0.5
b=1.7
tol=0.001
n0=50
i=1
fa=f(a)
fb=f(b)
while i<=n0:
    p=a+(b-a)/2
    fp=f(p)
    if fp==0 or (b-a)/2<tol:
        print('La solucion es ',p)
        break
    i=i+1
    if fa*fp>0:
            a=p
            fa=fp
    else:
            b=p
            fb=fp
if i>=n0:
    print('El metodo no converge ')
```

2.2. 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

$$p_n = p_{n-1} - \frac{f(p_{n-1})}{f'(p_{n-1})}. \tag{2.2}$$

La ecuación (2.2) 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.

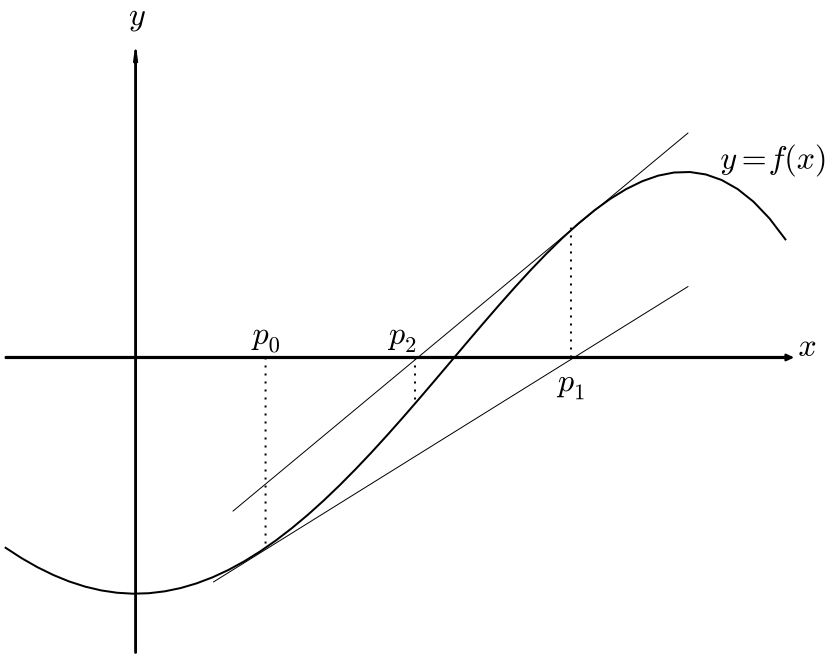

Figura 2.2: Método de Newton-Raphson

Como puede verse en la figura 1.4, 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.

2.2.1. 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

2.2.2. Código de Newton-Raphson

Este ejemplo utiliza

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

```
from math import *
def f(x):
    y=x**3-x
    return y
def df(x):
    y=3*x**2-1
    return y
p0=1.7
tol=0.001
n0=50
i=1
while i<=n0:
    p=p0-f(p0)/df(p0)
    if f(p)==0 or abs(p-p0)<tol:
        print('La solucion es ',p)
        break
    i=i+1
    p0=p
if i>=n0:
    print('El metodo no converge ')
```

2.3. 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

$$p_n = p_{n-1} - \frac{f(p_{n-1})(p_{n-1} - p_{n-2})}{f(p_{n-1}) - f(p_{n-2})}. \tag{2.3}$$

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 1.3 muestra gráficamente el método.

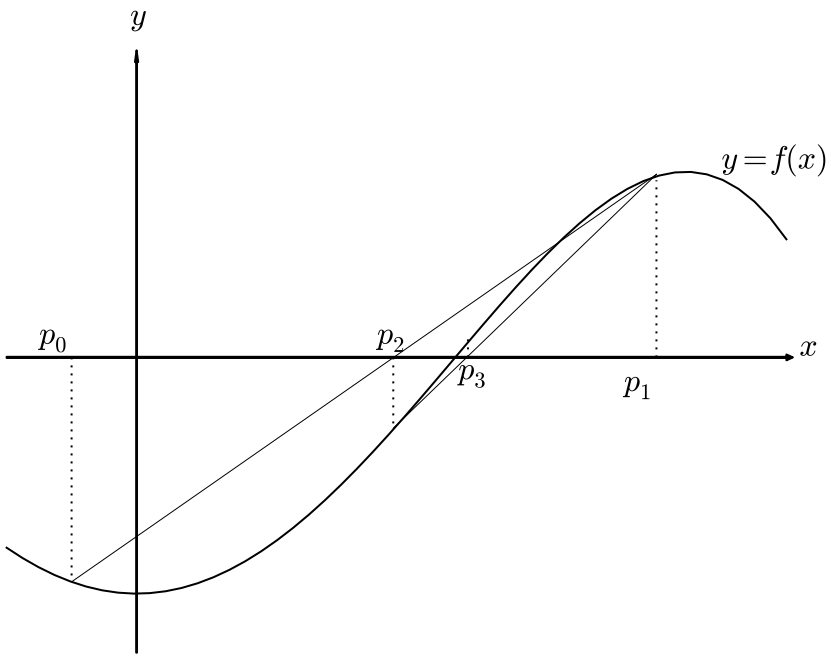

Figura 2.3: Método de las secantes

2.3.1. Algoritmo de las secantes

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

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

2.3.2. Código de las secantes

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

```
from math import *
def f(x):
    y=x**3-x
    return y
p0=0.5
p1=1.7
tol=0.001
n0=50
i=2
q0=f(p0)
q1=f(p1)
while i<=n0:
    p=p1-q1*(p1-p0)/(q1-q0)
    if f(p)==0 or abs(p-p1)<tol:
        print('La solucion es ',p)
        break
    i=i+1
    p0=p1
    q0=q1
    p1=p
    q1=f(p)
if i>=n0:
    print('El metodo no converge ')
```

2.4. 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 1.2 muestra gráficamente este método.

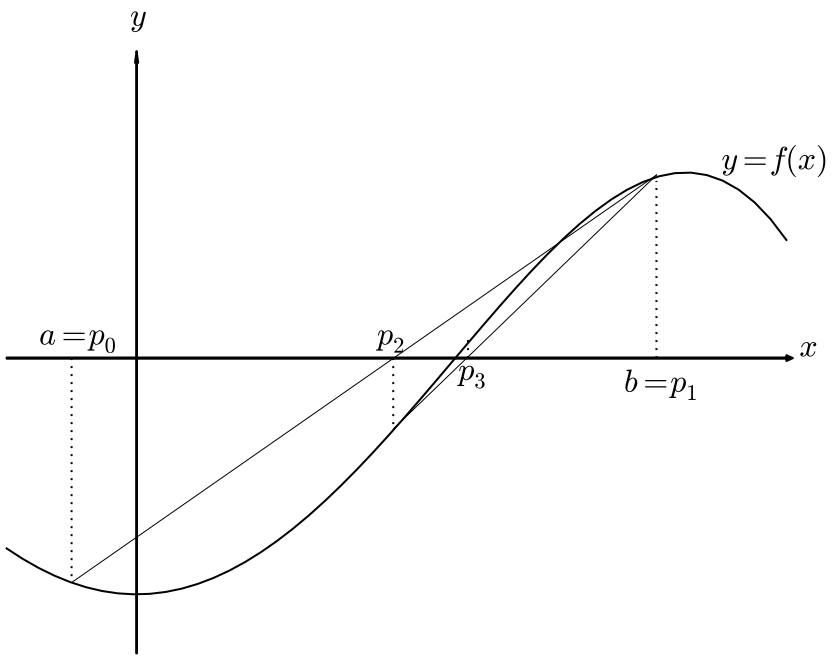

Figura 2.4: Método de la posición falsa

2.4.1. 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

2.4.2. Código de la posición falsa

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

```
from math import *
def f(x):
    y=x**3-x
    return y
p0=0.5
p1=1.7
tol=0.001
n0=50
i=2
q0=f(p0)
q1=f(p1)
while i<=n0:
    p=p1-q1*(p1-p0)/(q1-q0)
    if f(p)==0 or abs(p-p1)<tol:
        print('La solucion es ',p)
        break
    i=i+1
    q=f(p)
    if q*q1<0:
        p0=p1
        q0=q1
    p1=p
    q1=q
if i>=n0:
    print('El metodo no converge ')
```

Ejercicios

Usar cada uno de los métodos estudiados en este capítulo para encontrar las raíces de las siguientes ecuaciones, con una tolerancia de 0.001. Se recomienda graficar cada función 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 con la ecuación de estado de un gas ideal $pv = RT$.

Capítulo 3

Sistemas de ecuaciones no lineales

En este capítulo se presenta un algoritmo para obtener una solución de sistemas de ecuaciones no lineales, el método de Newton-Raphson generalizado a 2 y 3 variables. 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.

3.1. Método de Newton-Raphson

Para resolver el sistema $\vec{F}(\vec{x}) = \vec{0}$, dada una aproximación inicial $\vec{x}_0$, donde $\vec{F}(\vec{x}) = (f_1(x_1, x_2, ..., x_n), f_2(x_1, x_2, ..., x_n), ..., f_n(x_1, x_2, ..., x_n))$ y $\vec{x}_0 = (x_1^{(0)}, x_2^{(0)}, ..., x_n^{(0)})$, se puede usar el *método de Newton-Raphson* generalizado a varias variables. Éste consiste en obtener la matriz jacobiana

$$J(\vec{x}) = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & & & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \tag{3.1}$$

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

$$\vec{x}_{k+1} = \vec{x}_k - J^{-1}(\vec{x}_k)\vec{F}(\vec{x}_k). \tag{3.2}$$

3.1.1. Algoritmo de Newton-Raphson

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 6

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

Paso 4 Calcular $d\vec{x} = J^{-1}(\vec{x})\vec{F}(\vec{x})$

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

Paso 6 Si $||d\vec{x}|| < TOL$

SALIDA (La solución es $\vec{x}$)

TERMINAR

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

TERMINAR

3.1.2. Código de Newton-Raphson

El siguiente código se ejemplifica para el sistema

$$\text{sen}(x) + y^2 + \ln z - 7 = 0$$

$$3x + 2^y - z^3 + 1 = 0$$

$$x + y + z - 5 = 0.$$

```
from numpy import *
from numpy.linalg import *
x0=([2.0,1.0])
tol=1.0e-6
n=100
x=x0
def f(x):
    f=zeros(len(x))
    f[0]=x[0]**2+x[1]**2-1
    f[1]=(x[0]-1)**2+x[1]**2-1
    return f
def j(x):
    j=zeros((len(x),len(x)))
    j[0][0]=2*x[0]
    j[0][1]=2*x[1]
    j[1][0]=2*(x[0]-1)**2
    j[1][1]=2*x[1]
    return j
for i in range (n):
    jac=j(x)
    inversa=inv(jac)
    dx=dot(inversa,f(x))
    x=x-dx
    print(x)
    if sqrt(dot(dx,dx))<=tol:
        print('La solucion es: ',x)
```

```
        break
if i>=n:
    print('El metodo no converge')
```

3.2. El método de Broyden

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

$$\vec{x}_k = \vec{x}_{k-1} + \vec{y}, \tag{3.3}$$

iterándose hasta obtener la precisión deseada.

3.2.1. Algoritmo de Broyden

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

3.2.2. Código de Broyden

El siguiente código se ejemplifica para el sistema

$$\text{sen}(x) + y^2 + \ln z - 7 = 0$$

$$3x + 2^y - z^3 + 1 = 0$$

$$x + y + z - 5 = 0.$$

```
from numpy import *
from numpy.linalg import solve
def f(x):
    f=zeros(len(x))
    f[0]=sin(x[0])+x[1]**2+log(x[2])-7
    f[1]=3*x[0]+2**x[1]-x[2]**3+1
    f[2]=x[0]+x[1]+x[2]-5
    return f
x=array([1.0,1.0,1.0])
tol=1.0e-9
def jacobian(f,x):
    h = 1.0e-4
    n = len(x)
    jac = zeros((n,n))
    f0 = f(x)
    for i in range(n):
        temp = x[i]
        x[i] = temp + h
        f1 = f(x)
        x[i] = temp
        jac[:,i] = (f1 - f0)/h
    return jac,f0
for i in range(30):
    jac,f0 = jacobian(f,x)
    if sqrt(dot(f0,f0)/len(x)) < tol:
        print('La solucion es: ',x)
        break
    dx = solve(jac,-f0)
    x = x + dx
    if sqrt(dot(dx,dx)) < tol*max(max(abs(x)),1.0):
        print('La solucion es: ',x)
        break
if i>=30:
    print('Se excedio el maximo de iteraciones')
```

Ejercicios

Aproximar al menos una solución para cada uno de los sistemas de ecuaciones siguientes, usando el método de Newton-Raphson y el de Broyden, y comparar las soluciones obtenidas y el número de iteraciones requerido en cada caso.

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: $x_1=\pm1.8821$, $x_2=2.9811$, $x_3=-0.1677$
    $x_1=\pm1.5610$, $x_2=-0.3354$,$x_3=1.4905$
    $x_1=\pm0.9468$, $x_2=0.3354$, $x_3=-1.4905$

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: $x_1 = 0.4640$, $x_2 = 0.1896$, $x_3 = 0.0396$, $x_4 = -0.8652$
$x_1 = -0.9829$, $x_2 = -0.1977$, $x_3 = -0.0406$, $x_4 = -0.8530$

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

4.1.1. 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

4.1.2. Código de Euler

Ejemplo de código de Euler para el problema

$$\frac{dy}{dx} = e^{-0.5x}\cos x - 0.5y, \quad 0 \leq x \leq 10, \quad y(0) = 0$$

```
from numpy import *
def f(x,y):
    z=exp(-0.5*x)*cos(x)-0.5*y
    return z
a=0
b=10
y0=0
n=1000
x=zeros(n+1)
y=zeros(n+1)
h=(b-a)/n
x[0]=a
y[0]=y0
print(x[0],'\t',y[0],'\n')
for i in range (n):
    y[i+1]=y[i]+h*f(x[i],y[i])
    x[i+1]=x[i]+h
    print(x[i+1],'\t',y[i+1],'\n')
from pylab import *
plot(x,y,'-',color='black')
show()
```

La figura 4.1 muestra la gráfica de la solución obtenida.

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

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

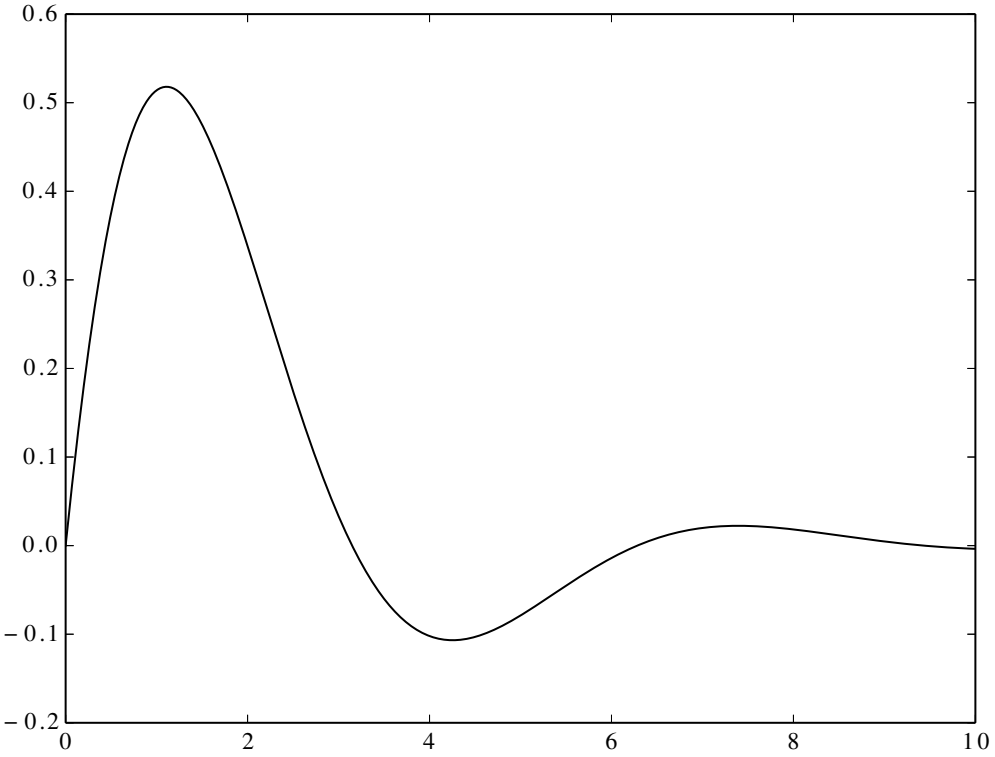

Figura 4.1: Solución por el método de Euler.

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 a aproximar la integral con trapecios en lugar de con rectángulos.

4.2.1. 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 5 TERMINAR

4.2.2. Código de Euler mejorado

Ejemplo de código de Euler mejorado para el problema

$$\frac{dy}{dx} = e^{-0.5x}\cos x - 0.5y, \quad 0 \leq x \leq 10, \quad y(0) = 0$$

```
from numpy import *
def f(x,y):
    z=exp(-0.5*x)*cos(x)-0.5*y
    return z
a=0
b=10
y0=0
n=1000
x=zeros(n+1)
y=zeros(n+1)
h=(b-a)/n
x[0]=a
y[0]=y0
print(x[0],'\t',y[0],'\n')
for i in range (n):
    y[i+1]=y[i]+h*f(x[i],y[i])
    x[i+1]=x[i]+h
    print(x[i+1],'\t',y[i+1],'\n')
from pylab import *
plot(x,y,'-',color='black')
show()
```

La figura 4.2 muestra la gráfica de la solución obtenida.

4.3. Método de Runge-Kutta

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

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

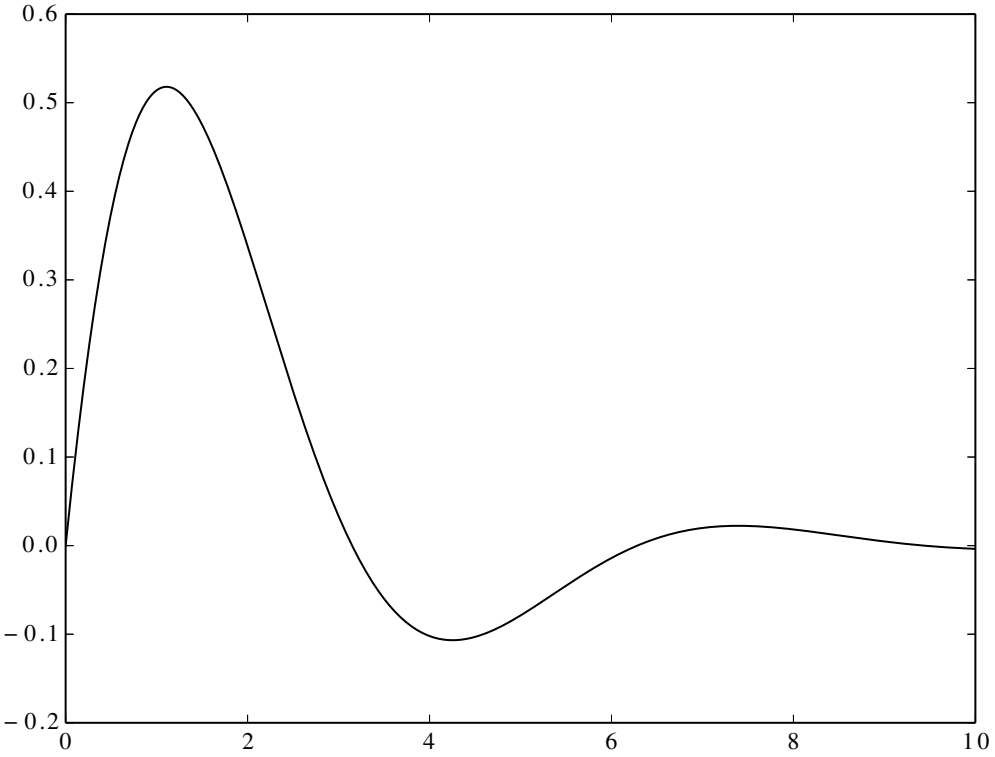

Figura 4.2: Solución por el método de Euler mejorado.

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.

4.3.1. 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

4.3.2. Código de Runge-Kutta de orden 4

Ejemplo de código de Runge-Kutta de orden 4 para el problema

$$\frac{dy}{dx} = e^{-0.5x}\cos x - 0.5y, \quad 0 \leq x \leq 10, \quad y(0) = 0$$

```
from numpy import *
def f(x,y):
    z=exp(-0.5*x)*cos(x)-0.5*y
    return z
a=0
b=10
y0=0
n=1000
x=zeros(n+1)
y=zeros(n+1)
h=(b-a)/n
x[0]=a
y[0]=y0
#print(x[0],'\t',y[0],'\n')
for i in range (n):
    k1=f(x[i],y[i])
    k2=f(x[i]+h/2,y[i]+h*k1/2)
    k3=f(x[i]+h/2,y[i]+h*k2/2)
```

```
    k4=f(x[i]+h,y[i]+h*k3)
    y[i+1]=y[i]+h*(k1+2*k2+2*k3+k4)/6
    x[i+1]=x[i]+h
#    print(x[i+1],'\t',y[i+1],'\n')
from pylab import *
plot(x,y,'-',color='black')
show()
```

La figura 4.3 muestra la gráfica de la solución obtenida.

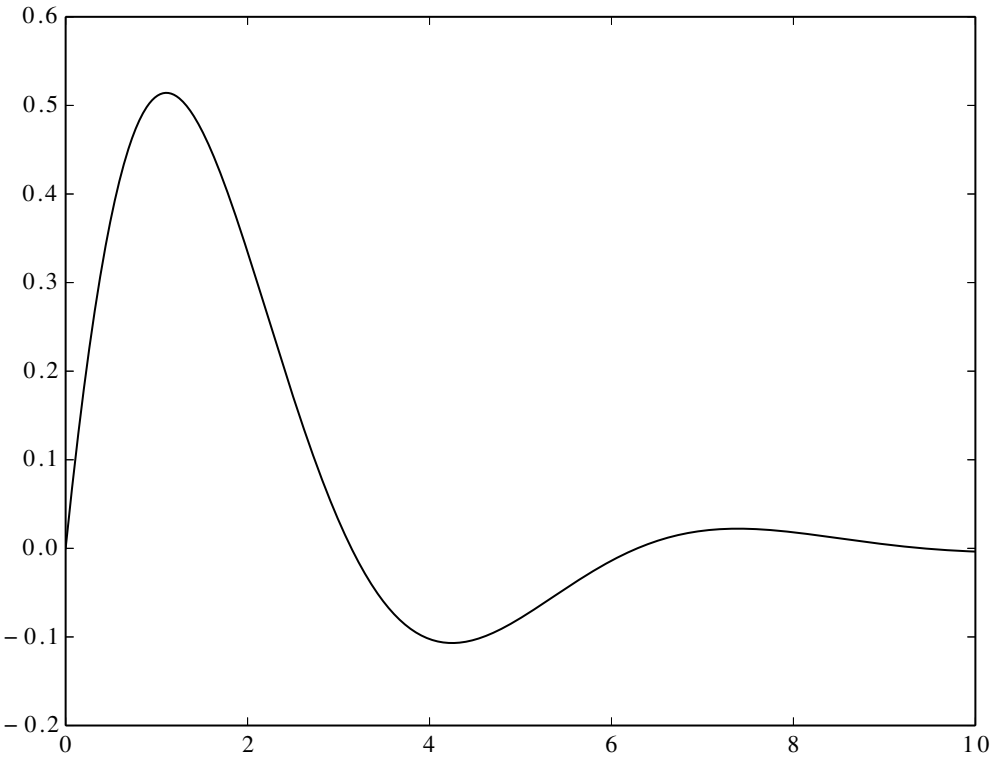

Figura 4.3: Solución por el método de Runge-Kutta.

Ejercicios

Resolver numéricamente los siguientes problemas de valor inicial en el intervalo dado, usando los métodos estudiados anteriormente. 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. $y' = -3x + 2y$, $y(0) = 1$, $0 \le x \le 3$
4. $y' = 5x - 3\sqrt{y}$, $y(0) = 2$, $0 \le x \le 2$
5. $y' = 2x + e^{-xy}$, $y(0) = 1$, $0 \le x \le 1$
6. $y' = (x^2 - y^2)\operatorname{sen} y$, $y(0) = -1$, $0 \le x \le 2$
7. $y' = \frac{y^2+2xy}{3+x^2}$, $y(0) = 0.5$, $0 \le x \le 3$
8. $y' = 0.5 - x + 2y$, $y(0) = 1$, $0 \le x \le 2$
9. $y' = \frac{4-xy}{1+y^2}$, $y(0) = -2$, $0 \le x \le 1$
10. $y' = \sqrt{x+y}$, $y(0) = 3$, $0 \le x \le 2$
11. $y' = x^2 + y^2$, $y(0) = 1$, $0 \le x \le 3$
12. $y' = \cos 5\pi x$, $y(0) = 1$, $0 \le x \le 2$
13. $y' = \frac{3x^2}{3y^2-4}$, $y(0) = 0$, $0 \le x \le 1$
14. $y' = xe^{3x} - 2y$, $y(0) = 0$, $0 \le x \le 1$
15. $y' = 1 + (x - y)^2$, $y(2) = 1$, $2 \le x \le 3$
16. $y' = 1 + y/x$, $y(1) = 2$, $1 \le x \le 2$
17. $y' = \cos 2x + \operatorname{sen} 3x$, $y(0) = 1$, $0 \le x \le 1$
18. $y' = y/x - (y/x)^2$, $y(1) = 1$, $1 \le x \le 2$
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. $y' = \frac{2}{x}y + x^2e^x$, $y(1) = 0$, $1 \le x \le 2$
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 $t = 0.1k$, con $k = 0, 1, 2, \cdots, 100$.

Capítulo 5

Sistemas de ecuaciones diferenciales ordinarias

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 en el capítulo anterior y sustituirlos en cada nueva iteración.

5.1. Método de Euler

Análogamente a lo que se hace para una sola ecuación, las derivadas se aproximan como

$$\begin{aligned} \frac{dx}{dt} &\approx \frac{x_{i+1}-x_i}{h} \\ \frac{dy}{dt} &\approx \frac{y_{i+1}-y_i}{h} \end{aligned}$$

con lo cual se tiene

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

Este método es el más fácil de los tres, igual que en una sola variable. Pero también como en una variable, es el que más error va acumulando.

5.1.1. Algoritmo de Euler 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 $i = 0, ..., n - 1$ hacer pasos 3 - 5

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

Paso 4 Hacer $y = y + hg(t, x, y)$

Paso 5 Hacer $t = t + h$

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

Paso 6 TERMINAR

5.1.2. Código de Euler para sistemas

Ejemplo de código para el sistema

$$\frac{dx}{dt} = x - 0.5xy$$

$$\frac{dy}{dt} = -0.75x + 0.25xy$$

```
from numpy import *
def f1(t,x,y):
    z=x-0.5*x*y
    return z
def f2(t,x,y):
    z=-0.75*y+0.25*x*y
    return z
a=0
b=20
x0=3.0
y0=5.0
n=1000
t=zeros(n+1)
x=zeros(n+1)
y=zeros(n+1)
h=(b-a)/n
t[0]=a
x[0]=x0
y[0]=y0
print(t[0],'\t',x[0],'\t',y[0],'\n')
```

```
for i in range (n):
    x[i+1]=x[i]+h*f1(t[i],x[i],y[i])
    y[i+1]=y[i]+h*f2(t[i],x[i],y[i])
    t[i+1]=t[i]+h
    print(t[i+1],'\t',x[i+1],'\t',y[i+1],'\n')
from pylab import *
plot(t,x,':',color='black')
plot(t,y,'-',color='black')
annotate(r'$x(t)$',xy=(7,9),fontsize=16)
annotate(r'$y(t)$',xy=(9.3,4),fontsize=16)
show()
```

La figura 5.1 muestra la gráfica de la solución obtenida.

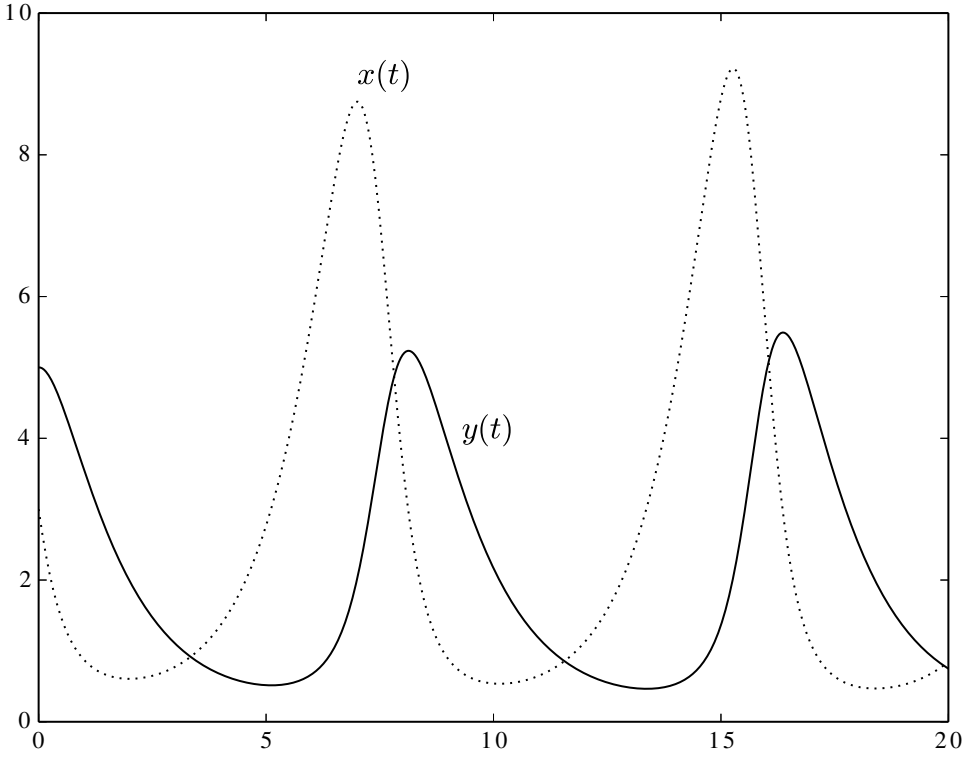

Figura 5.1: Solución por el método de Euler.

5.2. Método de Euler mejorado

En este caso, al igual que en una sola variable, se puede aproximar con trapecios en lugar de con rectángulos. En este caso las fórmulas de recurrencia tendrán el aspecto

$$\begin{aligned} x_{i+1} &= x_i + \frac{h}{2}\left[f(t_i, x_i, y_i) + f(t_i + h, x_i, y_i)\right] \\ y_{i+1} &= y_i + \frac{h}{2}\left[g(t_i, x_i, y_i) + g(t_i + h, x_i, y_i)\right] \end{aligned}$$

Este método nos da mayor precisión que el método de Euler básico.

5.2.1. Algoritmo de Euler mejorado 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 $i = 0, ..., n - 1$ hacer pasos 3 - 5

Paso 3 Hacer $x = x + \frac{h}{2}[f(t, x, y) + f(t + h, x, y)]$

Paso 4 Hacer $y = y + \frac{h}{2}[g(t, x, y) + g(t + h, x, y)]$

Paso 5 Hacer $t = t + h$

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

Paso 6 TERMINAR

5.2.2. Código de Euler mejorado para sistemas

Ejemplo de código para el sistema

$$\frac{dx}{dt} = x - 0.5xy$$

$$\frac{dy}{dt} = -0.75x + 0.25xy$$

```
from numpy import *
def f1(t,x,y):
    z=x-0.5*x*y
    return z
def f2(t,x,y):
    z=-0.75*y+0.25*x*y
    return z
a=0
b=20
x0=3.0
```

```
y0=5.0
n=1000
t=zeros(n+1)
x=zeros(n+1)
y=zeros(n+1)
h=(b-a)/n
t[0]=a
x[0]=x0
y[0]=y0
print(t[0],'\t',x[0],'\t',y[0],'\n')
for i in range (n):
    k1=f1(t[i],x[i],y[i])
    l1=f2(t[i],x[i],y[i])
    k2=f1(t[i]+h,x[i]+h*k1,y[i]+h*l1)
    l2=f2(t[i]+h,x[i]+h*k1,y[i]+h*l1)
    x[i+1]=x[i]+h*(k1+k2)/2
    y[i+1]=y[i]+h*(l1+l2)/2
    t[i+1]=t[i]+h
    print(t[i+1],'\t',x[i+1],'\t',y[i+1],'\n')
from pylab import *
plot(t,x,':',color='black')
plot(t,y,'-',color='black')
annotate(r'$x(t)$',xy=(7,8.5),fontsize=16)
annotate(r'$y(t)$',xy=(9.3,4),fontsize=16)
show()
```

La figura 5.2 muestra la gráfica de la solución obtenida.

5.3. Método de Runge-Kutta de orden 4

Ahora, para 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

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

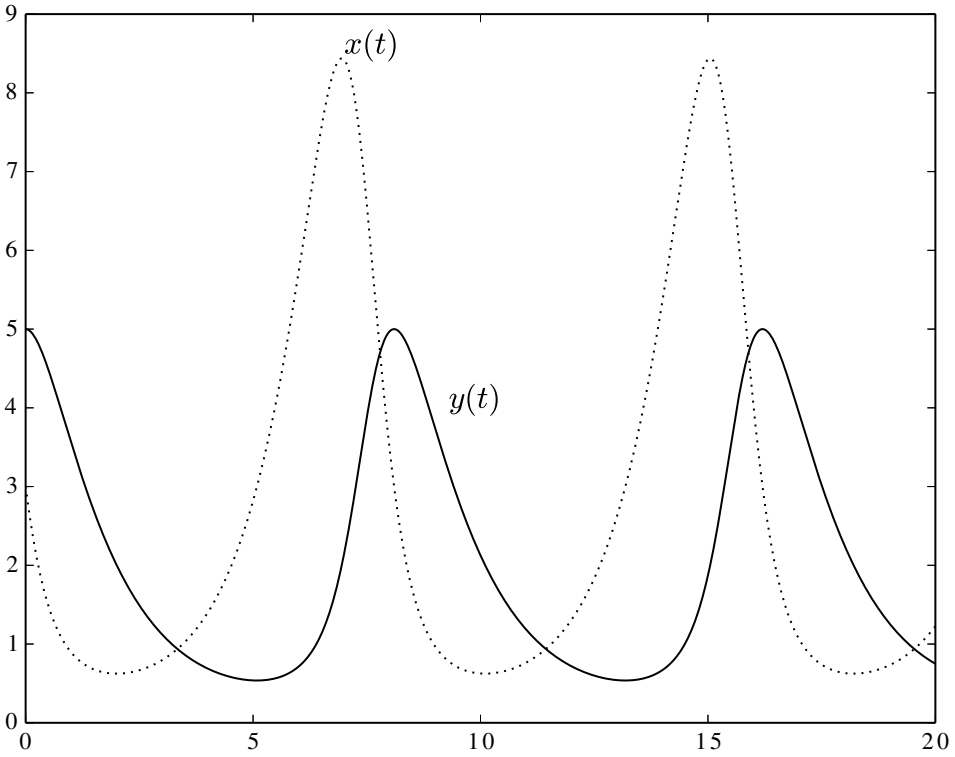

Figura 5.2: Solución por el método de Euler mejorado.

Por supuesto, este método tiene las mismas ventajas que en una sola variable.

5.3.1. 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

5.3.2. Código de Runge-Kutta para sistemas

Ejemplo de código de Runge-Kutta para el sistema

$$\frac{dx}{dt} = x - 0.5xy$$

$$\frac{dy}{dt} = -0.75x + 0.25xy$$

```
from numpy import *
def f1(t,x,y):
    z=x-0.5*x*y
    return z
def f2(t,x,y):
    z=-0.75*y+0.25*x*y
    return z
a=0
b=20
x0=3.0
y0=5.0
n=1000
t=zeros(n+1)
x=zeros(n+1)
y=zeros(n+1)
h=(b-a)/n
t[0]=a
x[0]=x0
y[0]=y0
print(t[0],'\t',x[0],'\t',y[0],'\n')
for i in range (n):
    k1=f1(t[i],x[i],y[i])
    l1=f2(t[i],x[i],y[i])
```

```
    k2=f1(t[i]+h/2,x[i]+h*k1/2,y[i]+h*l1/2)
    l2=f2(t[i]+h/2,x[i]+h*k1/2,y[i]+h*l1/2)
    k3=f1(t[i]+h/2,x[i]+h*k2/2,y[i]+h*l2/2)
    l3=f2(t[i]+h/2,x[i]+h*k2/2,y[i]+h*l2/2)
    k4=f1(t[i]+h,x[i]+h*k3,y[i]+h*l3)
    l4=f2(t[i]+h,x[i]+h*k3,y[i]+h*l3)
    x[i+1]=x[i]+h*(k1+2*k2+2*k3+k4)/6
    y[i+1]=y[i]+h*(l1+2*l2+2*l3+l4)/6
    t[i+1]=t[i]+h
    print(t[i+1],'\t',x[i+1],'\t',y[i+1],'\n')
from pylab import *
plot(t,x,':',color='black')
plot(t,y,'-',color='black')
annotate(r'$x(t)$',xy=(7,8.5),fontsize=16)
annotate(r'$y(t)$',xy=(9.3,4),fontsize=16)
show()
```

La figura 5.3 muestra la gráfica de la solución obtenida.

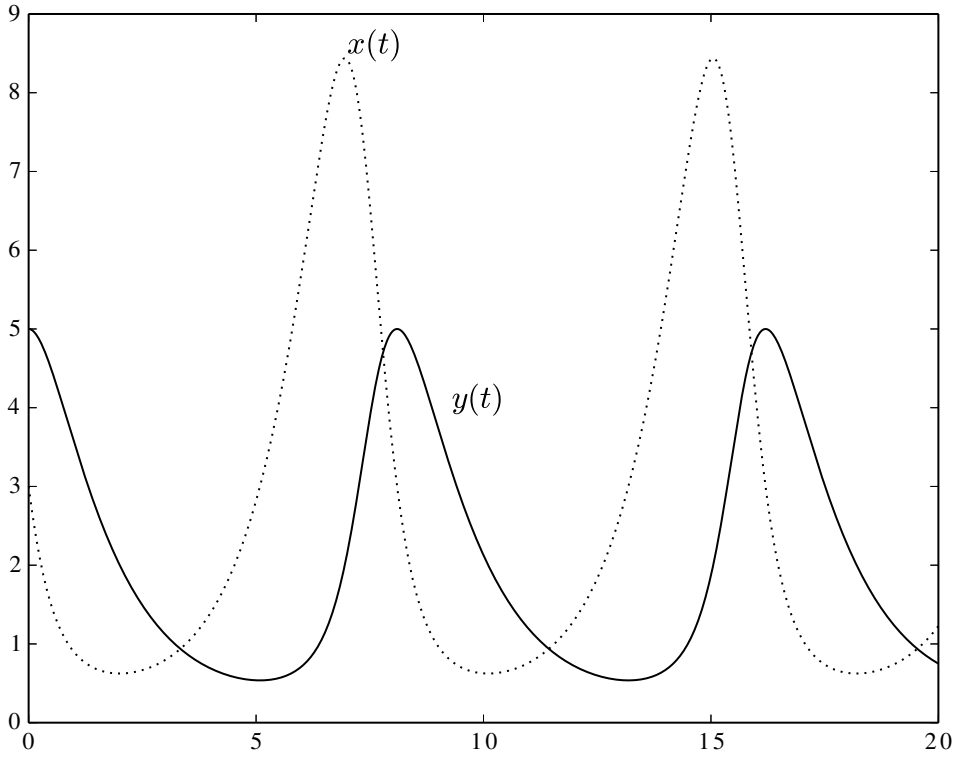

Figura 5.3: Solución por el método de Runge-Kutta.

Ejercicios

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

1. $x' = x - 4y$, $y' = -x + y$, $x(0) = 1$, $y(0) = 0$, $0 \le t \le 1$
2. $x' = x + y + t$, $y' = 4x - 2y$, $x(0) = 1$, $y(0) = 0$, $0 \le t \le 2$
3. $x' = 2x + yt$, $y' = xy$, $x(0) = 1$, $y(0) = 1$, $0 \le t \le 3$
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. $x'' + t^2x' + 3x = t$, $x(0) = 1$, $x'(0) = 2$, $0 \le t \le 2$
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. $t^2x'' - 2tx' + 2x = t^3 \ln t$, $x(1) = 1$, $x'(1) = 0$, $1 \le t \le 2$
19. $t^3x''' - t^2x'' + 3tx' - 4x = 5t^3 \ln t + 9t^3$, $x(1) = 0$, $x'(1) = 1$, $x''(1) = 3$, $1 \le t \le 2$
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 6

Ecuaciones diferenciales parciales

Al buscar la solución de ecuaciones diferenciales parciales, necesitamos tener condiciones iniciales y/o de frontera. Esto nos lleva a estudiar el problema de contorno en una incógnita, para después estudiar el la ecuación de difusión.

6.1. Método de diferencias finitas para problemas de contorno

Aquí estudiaremos la solución numérica de una ecuación diferencial ordinaria de segundo orden. La diferencia con lo estudiado anteriormente es que ahora se tienen condiciones de frontera en lugar de condiciones iniciales. Esto nos da el problema siguiente:

$$y'' = p(x)y' + q(x)y + r(x), \quad a \leq x \leq b, \quad y(a) = \alpha, \quad y(b) = \beta. \tag{6.1}$$

Para resolver este problema de contorno usaremos el método de diferencias finitas.

Primero dividimos el dominio de integración en $n+1$ subintervalos de longitudes h iguales, con lo cual se tiene que $h = (b-a)/(n+1)$. Los extremos de estos subintervalos están en $x_i = a + ih$, con $i = 0, 1, ..., n, n+1$.

Como aproximaciones de las derivadas se utilizan los cocientes:

$$y'' \approx \frac{y(x_{i+1}) - 2y(x_i) + y(x_{i-1})}{h^2}, \quad y' \approx \frac{y(x_{i+1}) - y(x_{i-1})}{2h}, \tag{6.2}$$

que al sustituir en la ecuación original nos da la ecuación:

$$\frac{w_{i+1} - 2w_i + w_{i-1}}{h^2} = p(x_i)\frac{w_{i+1} - w_{i-1}}{2h} + q(x_i)w_i + r(x_i), \tag{6.3}$$

donde w_i es la aproximación a $y(x_i)$ buscada.

De las condiciones de frontera tenemos:

$$w_0 = \alpha, \quad w_{n+1} = \beta, \tag{6.4}$$

y para los puntos internos, es decir, w_i, con $i = 1, ..., n$, se tiene la ecuación:

$$-\left(1 - \frac{h}{2}p(x_i)\right) w_{i+1} + (2 + h^2 q(x_i))w_i - \left(1 + \frac{h}{2}p(x_i)\right) = -h^2 r(x_i), \tag{6.5}$$

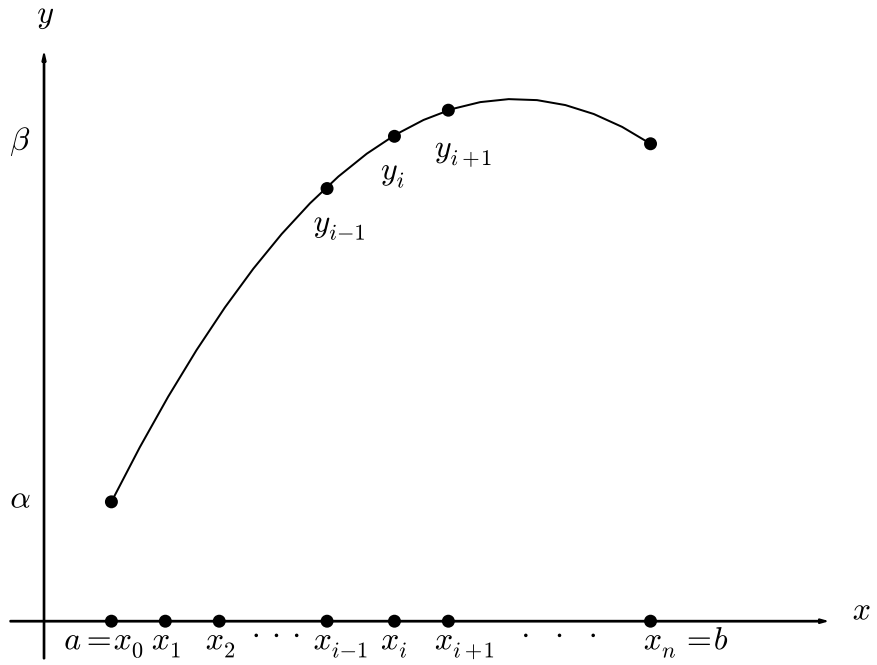

Figura 6.1: Método de diferencias finitas.

lo que nos da un sistema de ecuaciones lineales de la forma:

$$A\vec{w} = \vec{b}, \tag{6.6}$$

con A la matriz tridiagonal

$$A = \begin{bmatrix} 2+h^2q(x_1) & -1+\frac{h}{2}p(x_1) & 0 & 0 & \cdots & 0 \\ -1-\frac{h}{2}p(x_2) & 2+h^2q(x_2) & -1+\frac{h}{2}p(x_2) & 0 & \cdots & 0 \\ 0 & \cdot & \cdot & \cdot & \cdots & 0 \\ 0 & 0 & \cdot & \cdot & \cdots & 0 \\ 0 & 0 & 0 & \cdot & \cdots & 0 \\ \vdots & \vdots & 0 & -1-\frac{h}{2}p(x_{n-1}) & 2+h^2q(x_{n-1}) & -1+\frac{h}{2}p(x_{n-1}) \\ 0 & 0 & \cdots & 0 & -1-\frac{h}{2}p(x_n) & 2+h^2q(x_n) \end{bmatrix} \tag{6.7}$$

y

$$\vec{b} = \begin{bmatrix} -h^2 r(x_1) + \left(1 + \frac{h}{2} p(x_1)\right) w_0 \\ -h^2 r(x_2) \\ \cdot \\ \cdot \\ \cdot \\ -h^2 r(x_{n-1}) \\ -h^2 r(x_n) + \left(1 + \frac{h}{2} p(x_n)\right) w_{n+1} \end{bmatrix}. \tag{6.8}$$

Aunque el sistema tridiagonal se puede resolver con el método de eliminación de Gauss-Jordan, es mucho más simple aplicar una técnica de sustitución en reversa, según se explica en el siguiente algoritmo.

6.1.1. Algoritmo de diferencias finitas para el problema de contorno

Obtiene aproximaciones w_i a $y(x_i)$ del problema

$$y'' = p(x)y' + q(x)y + r(x), \quad a \le x \le b, \quad y(a) = \alpha, \quad y(b) = \beta$$

ENTRADA $p(x)$, $q(x)$, $r(x)$, a, b, α, β, $n \ge 2$.

SALIDA Aproximaciones a $y(x_i)$, en $n+2$ valores de x_i equidistantes, para $i = 0, 1, ..., n, n+1$.

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

$x = a + h$

$a_1 = 2 + h^2 q(x)$

$b_1 = -1 + \frac{h}{2} p(x)$

$d_1 = -h^2 r(x) + \left(1 + \frac{h}{2} p(x)\right) \alpha$

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

$x = a + ih$

$a_i = 2 + h^2 q(x)$

$b_i = -1 + \frac{h}{2} p(x)$

$c_i = -1 - \frac{h}{2} p(x)$

$d_i = -h^2 r(x)$

Paso 3 Hacer $x = b - h$

$a_n = 2 + h^2 q(x)$

$c_n = -1 - \frac{h}{2} p(x)$

$d_n = -h^2 r(x) + \left(1 - \frac{h}{2} p(x)\right) \beta$

Paso 4 Hacer $\ell_1 = a_1$

$u_1 = b_1/a_1$

$z_1 = d_1/\ell_1$

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

$\ell_i = a_i - c_i u_{i-1}$

$u_i = b_i/\ell_i$

$z_i = (d_i - c_i z_{i-1})/\ell_i$

Paso 6 Hacer $\ell_n = a_n - c_n u_{n-1}$

$z_n = (d_n - c_n z_{n-1})/\ell_n$

Paso 7 Hacer $w_0 = \alpha$

$w_{n+1} = \beta$

$w_n = z_n$

Paso 8 Para $i = n-1, ..., 1$, hacer $w_i = z_i - u_i w_{i+1}$

Paso 9 Para $i = 0, ..., n+1$ hacer $x = a + ih$

Solución(x,w_i)

Paso 10 TERMINAR

6.1.2. Código para el problema de contorno

Ejemplo de código para el problema

$$y'' = -\frac{4}{x}y' + \frac{2}{x^2}y - \frac{2\ln x}{x^2}, \quad 1 \le x \le 2, \quad y(1) = -0.5, \quad y(2) = \ln 2,$$

```
from numpy import *
def p(x):
    y=-4/x
    return y
def q(x):
    y=2/x**2
    return y
def r(x):
    y=-2*log(x)/x**2
    return y
n=20
a0=1.0
b0=2.0
alfa=-0.5
beta=log(2.)
h=(b0-a0)/(n+1)
a=zeros(n+2)
b=zeros(n+2)
c=zeros(n+2)
d=zeros(n+2)
```

```
l=zeros(n+2)
u=zeros(n+2)
w=zeros(n+2)
x=zeros(n+2)
z=zeros(n+2)
x[0]=a0
x[n+1]=b0
x[1]=a0+h
a[0]=a0
a[1]=2+h**2*q(x[1])
b[1]=-1+(h/2)*p(x[1])
d[1]=-h**2*r(x[1])+(1+(h/2)*p(x[1]))*alfa
for i in range(2,n):
    x[i]=a0+i*h
    a[i]=2+h**2*q(x[i])
    b[i]=-1+(h/2)*p(x[i])
    c[i]=-1-(h/2)*p(x[i])
    d[i]=-h**2*r(x[i])
x[n]=b0-h
a[n]=2+h**2*q(x[n])
c[n]=-1-(h/2)*p(x[n])
d[n]=-h**2*r(x[n])+(1-(h/2)*p(x[n]))*beta
l[1]=a[1]
u[1]=b[1]/a[1]
z[1]=d[1]/l[1]
for i in range(2,n):
    l[i]=a[i]-c[i]*u[i-1]
    u[i]=b[i]/l[i]
    z[i]=(d[i]-c[i]*z[i-1])/l[i]
l[n]=a[n]-c[n]*u[n-1]
z[n]=(d[n]-c[n]*z[n-1])/l[n]
w[0]=alfa
w[n+1]=beta
w[n]=z[n]
for i in range(n-1,0,-1):
    w[i]=z[i]-u[i]*w[i+1]
for i in range(0,n+2):
    print(x[i],'\t',w[i],'\n')
from pylab import *
plot(x,w,color='black')
show()
```

Ejercicios

Resolver numéricamente los siguientes problemas de contorno, usando el método de diferencias finitas y graficando las soluciones.

1. $y'' = 4(y - x)$, $0 \leq x \leq 1$, $y(0) = 0$, $y(1) = 2$
2. $y'' = y' + 2y + \cos x$, $0 \leq x \leq \frac{\pi}{2}$, $y(0) = -0.3$, $y\left(\frac{\pi}{2}\right) = -0.1$
3. $y'' = -3y' + 2y + 2x + 3$, $0 \leq x \leq 1$, $y(0) = 2$, $y(1) = 1$

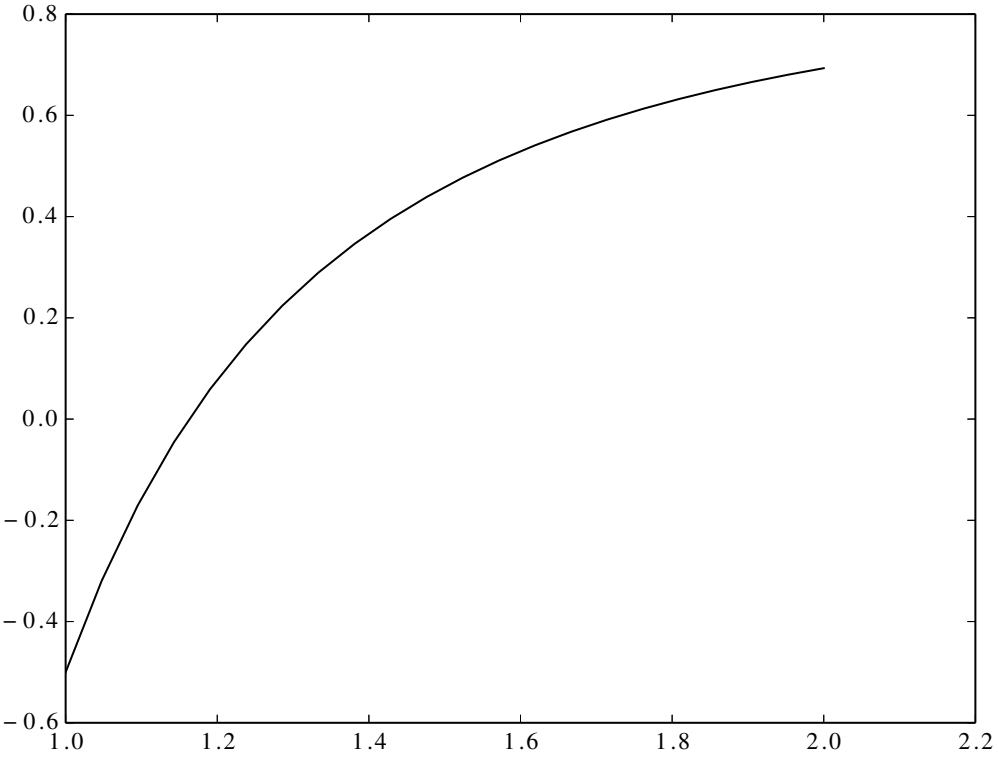

Figura 6.2: Solución numérica del problema de contorno.

4. $y'' = -\frac{4}{x}y' + \frac{2}{x^2}y - \frac{2}{x^2}\ln x$, $1 \le x \le 2$, $y(1) = -\frac{1}{2}$, $y(2) = \ln 2$
5. $y'' = -(x+1)y' + 2y + (1-x^2)e^{-x}$, $0 \le x \le 1$, $y(0) = -1$, $y(1) = 0$
6. $y'' = \frac{y'}{x} + \frac{3}{x^2}y + \frac{\ln x}{x} - 1$, $1 \le x \le 2$, $y(1) = 0$, $y(2) = 0$
7. $y'' + y = 0$, $0 \le x \le \frac{\pi}{4}$, $y(0) = 1$, $y\left(\frac{\pi}{4}\right) = 1$
8. $y'' + 4y = \cos x$, $0 \le x \le \frac{\pi}{4}$, $y(0) = 0$, $y\left(\frac{\pi}{4}\right) = 0$
9. $y'' = -\frac{4}{x}y' - \frac{2}{x^2}y + 2\frac{\ln x}{x^2}$, $1 \le x \le 2$, $y(1) = \frac{1}{2}$, $y(2) = \ln 2$
10. $y'' = 2y' - y + xe^x - x$, $0 \le x \le 2$, $y(0) = 0$, $y(2) = -4$
11. $y'' = 100y$, $0 \le x \le 1$, $y(0) = 1$, $y(1) = e^{-10}$

6.2. Ecuaciones parabólicas

Ahora estudiaremos la solución numérica de la ecuación diferencial parcial parabólica siguiente:

$$\frac{\partial u(x,t)}{\partial t} = \alpha^2 \frac{\partial^2 u(x,t)}{\partial x^2}. \tag{6.9}$$

Esta ecuación surge al resolver problemas de difusión, por ejemplo, de calor en una barra de metal de longitud ℓ. La constante α es diferente para cada metal. Como restricciones tomaremos:

$$u(x,0) = f(x), \quad u(0,t) = 0, \quad u(\ell,t) = 0. \tag{6.10}$$

Para resolver usaremos el método de diferencias finitas.

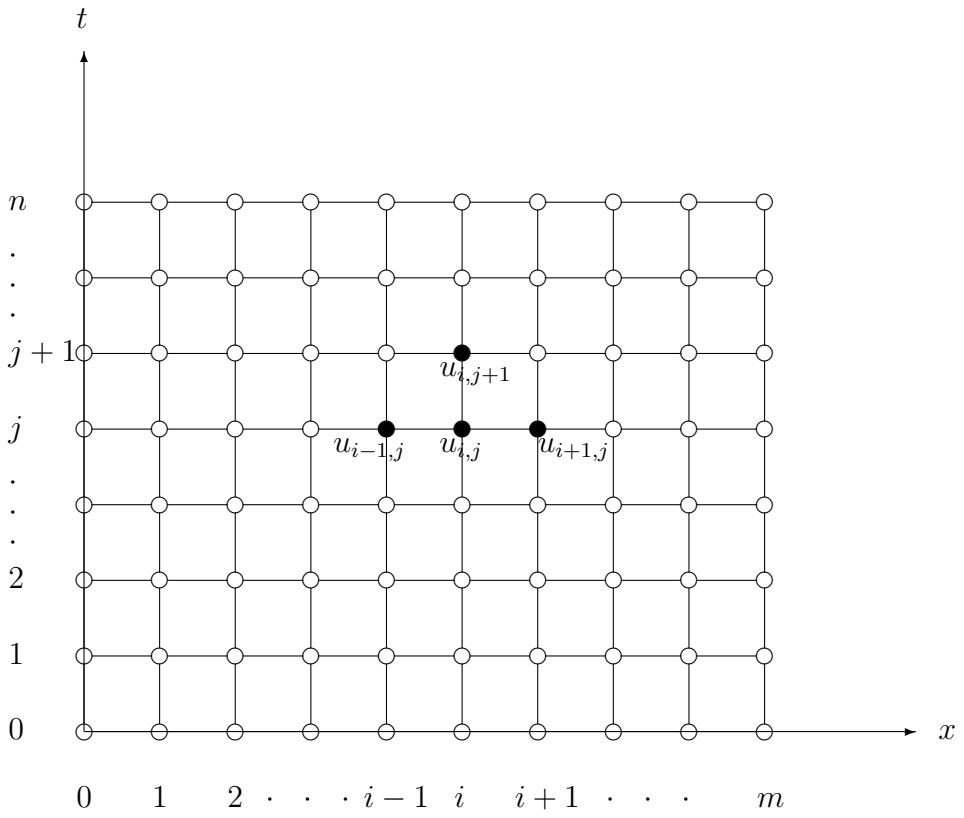

Figura 6.3: Malla para las diferencias finitas.

Primero dividimos el dominio de integración en m subintervalos de longitud h iguales, con lo cual se tiene que $h = \ell/m$. Después seleccionamos un intervalo de tiempo k como paso de integración.

Como aproximaciones de las derivadas se utilizan los cocientes:

$$\frac{\partial^2 u}{\partial x^2} \approx \frac{1}{h^2}[u(x+h,t) - 2u(x,t) + u(x-h,t)], \tag{6.11}$$

$$\frac{\partial u}{\partial t} \approx \frac{1}{k}[u(x,t+k) - u(x,t)], \tag{6.12}$$

con lo cual la ecuación se convierte en:

$$\frac{\alpha^2}{h^2}\frac{\partial^2 u}{\partial x^2} \approx \frac{1}{h^2}[u(x+h,t) - 2u(x,t) + u(x-h,t)] = \frac{1}{k}[u(x,t+k) - u(x,t)]. \tag{6.13}$$

Para acortar la escritura se usa la notación:

$$\lambda = \alpha^2 k/h^2, \quad u(x,t) = u_{ij}, \quad u(x+h,t) = u_{i+1,j}, \quad u(x-h,t) = u_{i-1,j}, \quad u(x,t+k) = u_{i,j+1},$$

así que la ecuación se vuelve

$$u_{i,j+1} = \lambda u_{i+1,j} + (1-2\lambda)u_{i,j} + \lambda u_{i-1,j}. \tag{6.14}$$

Aunque el tiempo puede extenderse hasta el infinito, aquí nos limitaremos a un tiempo máximo T. Entonces tendremos una malla rectangular con m franjas verticales de ancho h y largo T, o bien, n franjas horizontales de longitud ℓ y ancho k. Para especificar cada punto de esta malla, necesitamos dar las coordenadas:

$$x_i = ih, h = 0, 1, 2, ..., n \quad \text{y} \quad t_j = jk, j = 0, 1, 2, 3, ..., m.$$

A partir de aquí, se usa la fórmula XX para estimar los valores de $u(x,t)$ en los puntos de la $(j+1)$-ésima línea usando sólo valores de la línea j-ésima (en la primera línea se calcula en base a las condiciones para $t = 0$).

6.2.1. Algoritmo para la ecuación de difusión

Obtiene aproximaciones w_i a $u(x_i, t_j)$ del problema

$$\frac{\partial u}{\partial t} = \alpha^2 \frac{\partial^2 u}{\partial x^2} \quad u(x,0) = f(x), \quad u(0,t) = 0, \quad u(\ell,t) = 0$$

ENTRADA λ, $f(x)$, ℓ, m, n, k.

SALIDA Aproximaciones a $u(x_i, t_j)$, en n valores de x equidistantes para n instantes.

Paso 1 Hacer $h = \ell/m$

Paso 2 Para $i = 1, ..., m-1$ hacer $w_i = f(ih)$

Paso 3 Hacer $\ell_1 = 1 + 2\lambda$

$u_1 = -\lambda/\ell_1$

Paso 4 Para $i = 2, ..., m-2$ hacer

$\ell_i = 1 + 2\lambda + \lambda u_{i-1}$

$u_i = -\lambda/\ell_i$

Paso 5 Hacer $\ell_{m-1} = 1 + 2\lambda + \lambda u_{m-2}$

Paso 6 Para $j = 1, ..., n$, hacer los pasos 7 - 11

Paso 7 Hacer $t = jk$, $z_1 = w_1/\ell_1$

Paso 8 Para $i = 2, ..., m-1$, hacer $z_i = (w_i + \lambda z_{i-1})/\ell_i$

Paso 9 Hacer $w_{m-1} = z_{m-1}$

Paso 10 Para $i = m-2, ..., 1$, hacer $w_i = z_i - u_i w_{i+1}$

Paso 11 Para $i = 1, ..., m-1$, hacer $x = ih$

Solución(t_j,x,w_i)

Paso 12 TERMINAR

6.2.2. Código para la ecuación de difusión

Ejemplo de código para el problema

$$u_t = u_{xx}, \quad u(0,t) = u(\ell,t) = 0, \quad u(x,0) = \text{sen}(\pi x).$$

```
from numpy import *
l=1
T=0.5
alpha=1
m=20
n=20
h=l/m
k=T/n
lam=alpha**2*k/h**2
def f(x):
    return sin(pi*x)
t=zeros(n+1)
x=zeros(m+1)
w=zeros(m+1)
l=zeros(m+1)
u=zeros(m+1)
z=zeros(m+1)
X=zeros((m+1,n+1))
Y=zeros((m+1,n+1))
Z=zeros((m+1,n+1))
for i in range (m+1):
    x[i]=i*h
    for j in range (n+1):
        t[j]=j*k
        X[i][j]=x[i]
        Y[i][j]=t[j]
for i in range(1,m):
    w[i]=f(i*h)
    Z[i][0]=w[i]
```

```
l[1]=1+2*lam
u[1]=-lam/l[1]
for i in range(2,m-1):
    l[i]=1+2*lam+lam*u[i-1]
    u[i]=-lam/l[i]
l[m-1]=1+2*lam+lam*u[m-2]
for j in range(1,n+1):
    z[1]=w[1]/l[1]
    for i in range(2,m):
        z[i]=(w[i]+lam*z[i-1])/l[i]
    w[m-1]=z[m-1]
    Z[i][j]=w[m-1]
    for i in range(m-2,0,-1):
        w[i]=z[i]-u[i]*w[i+1]
        Z[i][j]=w[i]
for j in range (n+1):
    for i in range (m+1):
        print(Y[i][j],'\t',X[i][j],'\t',Z[i][j])
from pylab import *
from mpl_toolkits.mplot3d import axes3d
fig = figure()
ax = fig.add_subplot(111, projection='3d')
ax.plot_wireframe(X,Y,Z,color='black')
show()
```

La figura 6.4 muestra la solución numérica generada con este código.

Ejercicios

Resolver numéricamente los siguientes problemas de difusión, usando el método de diferencias finitas y graficando las soluciones en tres dimensiones.

1. $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$, $u(0,t) = u(2,t) = 0$, $u(x,0) = \text{sen}(\frac{\pi}{2}x)$
2. $\frac{\partial u}{\partial t} = \frac{1}{16}\frac{\partial^2 u}{\partial x^2}$, $u(0,t) = u(1,t) = 0$, $u(x,0) = 2\,\text{sen}(2\pi x)$
3. $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$, $u(0,t) = u(2,t) = 0$, $u(x,0) = \text{sen}(2\pi x)$
4. $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$, $u(0,t) = u(\pi,t) = 0$, $u(x,0) = \text{sen}\, x$
5. $\frac{\partial u}{\partial t} = \frac{4}{\pi^2}\frac{\partial^2 u}{\partial x^2}$, $u(0,t) = u(4,t) = 0$, $u(x,0) = \left(1 + 2\cos\left(\frac{\pi}{4}x\right)\right)\text{sen}\left(\frac{\pi}{4}x\right)$
6. $\frac{\partial u}{\partial t} = \frac{1}{\pi^2}\frac{\partial^2 u}{\partial x^2}$, $u(0,t) = u(1,t) = 0$, $u(x,0) = \cos\left[\pi\left(x - \frac{1}{2}\right)\right]$
7. $\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2}$, $u(0,t) = u(2,t) = 0$, $u(x,0) = \begin{cases} 1 & \text{para} \quad 0 \le x \le 1 \\ 0 & \text{para} \quad 1 < x \le 2 \end{cases}$
8. $\frac{\partial u}{\partial t} = \frac{K}{\gamma\rho}\frac{\partial^2 u}{\partial x^2}$, $u(0,t) = u(L,t) = 0$, $u(x,0) = f(x)$ $L = 50$, $K = 0.15$, $\rho = 8.0$, $\gamma = 0.11$, $f(x) = 30$
9. $\frac{\partial u}{\partial t} = \frac{K}{\gamma\rho}\frac{\partial^2 u}{\partial x^2}$, $u(0,t) = u(L,t) = 0$, $u(x,0) = f(x)$ $L = 20$, $K = 1.10$, $\rho = 2.7$, $\gamma = 0.22$, $f(x) = \frac{x(20-x)}{2}$
10. $\frac{\partial u}{\partial t} = \frac{K}{\gamma\rho}\frac{\partial^2 u}{\partial x^2}$, $u(0,t) = u(L,t) = 0$, $u(x,0) = f(x)$ $L = 100$, $K = 1.04$, $\rho = 10.6$, $\gamma = 0.06$, $f(x) = \begin{cases} 0.8x & \text{para} \quad 0 \le x \le 50 \\ 0.8(100 - x) & \text{para} \quad 50 < x \le 100 \end{cases}$

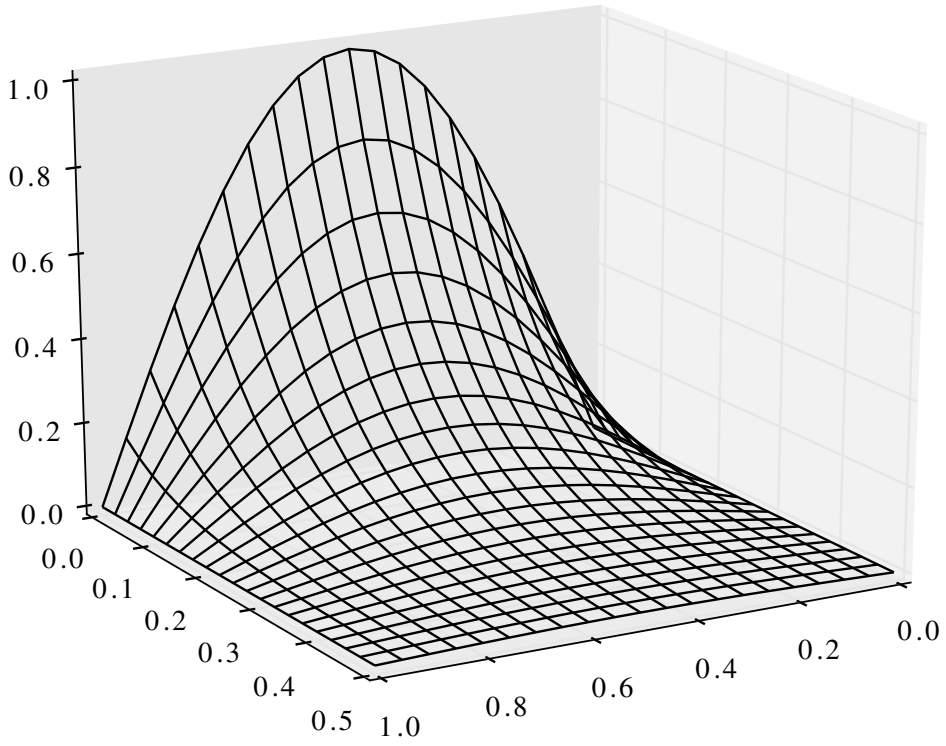

Figura 6.4: Solución numérica de la ecuación de calor.

Bibliografía

Burden, R. & J. D. Faires. *Numerical Analysis*. 7th ed. Thomson. San Francisco, 2001.

Burden, R. & J. D. Faires. *Study Guide for Numerical Analysis*. 7th ed. Thomson. San Francisco, 2001.

Kiusalaas, J. *Numerical Methods with Python 3*. Cambridge University Press. New York, 2013.

Lutz, M. *Programming Python*. 4th ed. O'Reilly. New York, 2011.

Lutz, M. *Python Pocket Reference*. 4th ed. O'Reilly. New York, 2011.

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

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