Métodos Numéricos

Dr. Félix Calderon Solorio

10 de septiembre de 2025

Índice general

Introducción a la Programación en Matlab

1.1. Introducción (Variables y Operadores)

1.1.1. Introducción

Matlab puede considerarse como un lenguaje de programación tal como C, Fortran, Java, etc. Algunas de las características de Matlab son:

- La programación es mucho más sencilla.
- Hay continuidad entre valores enteros, reales y complejos.
- La amplitud del intervalo y la exactitud de los números es mayor.
- Cuenta con una biblioteca matemática amplia.
- Abundantes herramientas gráficas, incluidas funciones de interfaz gráfica con el usuario.
- Capacidad de vincularse con los lenguajes de programación tradicionales.
- Transportabilidad de los programas.

Algunas de sus desventajas son:

- Necesita de muchos recursos de sistema como son Memoria, tarjeta de videos, etc. para funcionar correctamente.
- El tiempo de ejecución es lento.
- No genera código ejecutable.
- Es caro.

En una estación de trabajo UNIX, MATLAB puede abrirse tecleando

¿matlab

En el caso de sistemas LINUX, existe una versión similar al MATLAB , llamado OCTAVE, el cual tiene las mismas funciones y desempeño que el MATLAB. En este caso para iniciar la sesión se da:

¿octave

En Macintosh o Windows, haga clic en el icono de MATLAB.

1.1.2. Ayuda

Si no entiende el significado de un comando, teclee help y el nombre del comando que desea revisar. Este comando desplegara información concisa respecto que será de utilidad para el usuario. Así por ejemplo cuando se da el comando

≫ help help

Se despliega en la pantalla.

HELP On-line help, display text at command line. HELP, by itself, lists all primary help topics. Each primary topic corresponds to a directory name on the MATLABPATH.

"HELP TOPIC"gives help on the specified topic. The topic can be a command name, a directory name, or a MATLABPATH relative partial pathname (see HELP PARTIAL-PATH). If it is a command name, HELP displays information on that command. If it is a directory name, HELP displays the Table-Of-Contents for the specified directory. For example, "help general.ªnd "help matlab/general"both list the Table-Of-Contents for the directory toolbox/matlab/general.

HELP FUN displays the help for the function FUN.

T = HELP('topic') returns the help text in a \ n separated string.

LOOKFOR XYZ looks for the string XYZ in the first comment line of the HELP text in all M-files found on the MATLABPATH. For all files in which a match occurs, LOOKFOR displays the matching lines.

MORE ON causes HELP to pause between screenfuls if the help text runs to several screens.

In the online help, keywords are capitalized to make them stand out. Always type commands in lowercase since all command and function names are actually in lowercase.

For tips on creating help for your m-files 'type help.m'.

See also LOOKFOR, WHAT, WHICH, DIR, MORE.

El comando what (Qué) muestra una lista de archivos M, MAT y MEX presentes en el directorio de trabajo. Otra manera de realizar esta operación es utilizar el comando

dir.

El comando who (Quien) lista las variables utilizadas en el espacio de trabajo actual.

1.1.3. Tipos de Datos y Variables

No es necesario declarar los nombres de la variables ni sus tipos. Esto se debe a que los nombres de la variables en Matlab no son diferentes para los números enteros, reales y complejos. Sin embargo esto da lugar a conflictos cuando se utilizan los nombres de variables de palabras reservadas para:

1. nombres de variables especiales.

Nombre de variable	Significado
eps	Épsilon de la máquina (2.2204e-16)
pi	pi (3.14159...)
i y j	Unidad imaginaria
inf	infinito
NaN	no es número
date	fecha
flops	Contador de operaciones de punto flotante
nargin	Número de argumentos de entrada a una función
nargout	Número de argumentos de salida de una función

2. nombres de funciones.

Trigonométricas

Nombre de variable	Significado
sin	Seno
sinh	Seno Hiperbólico
asin	Seno Inverso
asinh	Seno hiperbólico inverso
cos	Coseno
cosh	Coseno Hiperbólico
acos	Coseno inverso
acosh	Coseno hiperbólico inverso
tan	Tangente
tanh	Tangente Hiperbólica
atan	Tangente inversa.
atan2	Tangente inversa en cuatro cuadrantes
atanh	Tangente hiperbólica inversa
sec	Secante
sech	Secante Hiperbólica
asec	Secante inversa
asech	Secante hiperbólica Inversa
csc	Cosecante
csch	cosecante hiperbólica
acsc	Cosecante Inversa
acsch	Inverse hyperbolic cosecant
cot	Cotangent
coth	Hyperbolic cotangent
acot	Inverse cotangent
acoth	Inverse hyperbolic cotangent

Exponencial.

Nombre de variable	Significado
exp	Exponencial
log	Logaritmo Natural
log10	logaritmo común (base 10)
log2	Logaritmo base 2
pow2	Potenciacion en Base 2
sqrt	Raíz Cuadrada
nextpow2	Próxima potencia de 2

Complejos.

Nombre de variable	Significado
abs	Valor Absoluto
angle	Ángulo de Fase
complex	Constructor de números complejos
conj	Conjugado de un complejo
imag	Parte imaginaria
real	Parte real
unwrap	Desenvolvimiento de fase
isreal	Verifica si un arreglo es real
cplxpair	Ordena números complejos

Redondeo y residuos.

Nombre de variable	Significado
fix	Parte entera de un número
floor	Parte fraccionaria
ceil	Parte entere de un número
round	Redondea al siguiente entero
mod	Residuo con signo de una división
rem	Residuo sin signo de una división
sign	Signo

3. nombres de comandos.

Comandos

Nombre de comando	Significado
what	Lista archivos en el directorio
dir	Lista archivos en el directorio
who	Variables utilizadas
clear	Borra variables utilizadas
etc	etc

1.1.4. Ejemplo

Calcular el volumen en una esfera.

```
clear;
r = 2
vol = (4/3)*pi*r^3
```

1.1.5. Operadores

Los operadores aritméticos como +, -, *, / son los mismos que en los lenguajes tradicionales como C, Java, Fortran, etc., así como la precedencia de estos.

Operador	Simbolo
suma	+
resta	-
multiplicación	*
división	/
reciproco	\

Un operador no convencional es el reciproco

```
clear;
c = 3 \ 1
```

Los operadores condicionales que existen en Matlab son:

Operador	Simbolo
Mayor que	$>$
Menor que	$<$
Mayor igual	$>=$
Menor igual	$<=$
Igual a	$==$
Diferente de	$\sim=$

Los cuales son utilizados para hacer condicionales con la sentencia if

```
clear;
r = -2
if  r > 0, (4/3)*pi*r\^3
end
```

Note que todas las sentencias if deben ir acompañadas por un end.

Los operadores lógicos and y or se denotan con & y ||

```
g = 5
if g>3 | g <0, a = 6
end
```

y se puede hacer agrupamiento con los operadores & y ||

```
clear;
a = 2
b = 3
c = 5
if((a==2 | b==3) & c < 5) g=1; end;
```

1.2. Instrucciones Secuenciales

1.3. Instrucciones Condicionales

Para hacer condicionales se utiliza la sentencia if, la cual, siempre debe ir terminada con la sentencia end

```
r = -2
if r > 0, (4/3)*pi*r^3
end
```

Los enunciados lógicos and y or se denotan con & y ||, pueden ser utilizados para implementar condicionales de la manera siguiente.

```
clear;
g = 5
if g>3 | g <0, a = 6
end
pause;
```

además los operadores & y || se puede agrupar como

```
clear;
a = 2
b = 3
c = 5

if((a==2 | b==3) & c < 5) g=1; end;

pause;
```

Las condicionales if se pueden utilizar con else o elseif

```
clear;

r = 2;

if r > 3       b = 1;
elseif r == 3  b = 2;
else           b = 0;
end;
```

Ver código ejem001.m

1.4. Instrucciones de Repetición

En Matlab existen dos maneras de implementar ciclos. La primera con los comandos for/end y la segundo con los comandos while/end, de manera muy similar a los lenguajes de alto nivel.

1.4.1. Ciclos for/end

La sintaxis de este comando es

```
for r=inicio: incremento: fin
   instrucciones_a_repetir
   instrucciones_a_repetir
   instrucciones_a_repetir
   instrucciones_a_repetir
   instrucciones_a_repetir
end;
```

imprimir los números del 1 a 100 se hace :

```
for x=1: 1:100

  x

end;
```

El siguiente conjunto de instrucciones realiza una cuenta de 100 a 80 con decrementos de 0.5.

```
for x=100:-0.5: 80

  x

end;
```

en el caso de decrementos o incrementos unitarios, se puede omitir el valor del incremento.

```
for x=1: 1:100, x, end
```

1.4.2. Ejemplo 1

Utilizando el comando for/end, calcular el volumen de cinco esferas de radio 1, 2, 3, 4 y 5 se hace:

```
for r=1:5
   vol = (4/3)*pi*r^3;
   disp([r, vol])
end;
```

Los ciclos pueden hacerse anidados de la siguiente manera.

```
for r=1:5
   for s=1:r
      vol = (4/3)*pi*(r^3-s^3);
      disp([r, s, vol])
   end
end
```

Podemos utilizar el comando break para detener la ejecución de un ciclo

```
for i=1:6
   for j=1:20
      if j>2*i, break, end
      disp([i, j])
   end
end
```

Ver código esferas.m

1.4.3. Ciclos while/end

La sintaxis de esta comando es

```
while condición
  instrucciones_a_repetir
end
```

Así por ejemplo podemos implementar al igual que con los ciclos for/end, un pequeño programa que imprima los números del 1 al 100.

```
x = 1;
while x <= 100
  x
  x = x + 1;
end
```

Ver ejemplo_while.m

El ejemplo para desplegar el volumen de una esfera con radios de 1, 2, 3, 4 y 5 queda.

```
r = 0;
while r<5
   r = r+1;
   vol = (4/3)*pi*r^3;
   disp([r, vol])
end;
```

otro ejemplo interesante es:

```
clear;
r = 0
while r<10
   r = input('Teclee el radio (o -1 para terminar): ');
   if  r< 0, break, end
   vol = (4/3)*pi*r*3;
   fprintf('volumen = \%7.3f\n', vol)
end
```

Ver esferas_while.m

1.4.4. Ejemplo 2

Hacer un programa que permita imprimir un triángulo rectángulo formado por asteriscos.

```
% Codigo para imprimir un triangulo

fprintf('\ntriangulo\n\n')

for k=1:7
   for l=1:k
      fprintf('*')
   end;
   fprintf('\n')
end
```

Ver triangulo.m

1.4.5. Ejemplo 3

Hacer un programa para desplegar un rectángulo de base 6 y altura 7.

```
% Codigo para imprimir un rectangulo
```

```
fprintf('\nrectangulo\n\n')

for k=1:7
   for l=1:6
      fprintf('*')
   end;
   fprintf('\n')
end
```

Ver rectangulo.m

1.4.6. Ejemplo 4

Hacer un programa para imprimir un pino utilizando un solo carácter.

```
a=10; %altura del follaje del pino
n=12; %Posición horizontal del vértice.
t=3;  %altura del tronco del pino
d=4;  %diámetro del tronco del pino

% Dibujar el follaje del pino, de altura 'a'

for i=1:a
   clear cad2 cad1
   num_ast=2*i-1;
   num_esp=n-i;
   cad1(1:num_esp)=' ';
   cad2(1:num_ast)='*';
   fprintf('\%s\%s\n',cad1,cad2)
end

% Dibujar el tronco del pino, de altura 't'

   clear cad2 cad1
   num_ast=d;
   num_esp=n-d/2;
   cad1(1:num_esp)=' ';
   cad2(1:num_ast)='I';

 for i=1:t
   fprintf('\%s\%s\n',cad1,cad2)
 end
```

Ver pino.m

1.4.7. Ejemplo 5

Hacer un programa para imprimir el triángulo de Pascal.

```
nr=8;   % Numero de renglones del triangulo de Pascal.
n=15;   % Numero de espacios en blanco antes del vértice.
x(1)=1;
cad1(1:n)=' ';
fprintf('%s%3.0f\n\n',cad1,x(1)); % vertice del triangulo
for k=2:nr-1;
    clear cad1 cad2
    num_esp=n-2*k+1;
    cad1(1:num_esp)=' ';
    clear x
    x(1)=1;
    for c=2:k;
    x(c)=x(c-1)*(k-c+1)/(c-1);
    end
    fprintf('\%s',cad1)
    for c=1:k
    fprintf(' \%3.0f',x(c))
    end
    fprintf('\n\n')
end
```

Ver pascal.m

1.5. Manejo de Matrices y Vectores

Un vector de datos puede definirse como

```
x = [0, 0.1, 0.2, 0.3, 0.4, 0.5]
```

si se desea imprimir un dato en particular se teclea

```
x(3)
```

el cual imprimirá el número en la tercer posición del arreglo, el primer elemento se numera con el uno.

Una forma equivalente de definir la misma x es

```
clear;
```

```
for i=1:6
   x(i) = (i-1)*0.1;
end

x
x(3)
```

Otra forma de escribir un arreglo es

```
clear;

x = 2: -0.4: -2

pause;
```

Podemos definir arreglos en fila o columna

```
clear;

z = [0; 0.1; 0.2; 0.3; 0.4; 0.5]

z = [0, 0.1, 0.2, 0.3, 0.4, 0.5]'
```

podemos realizar operaciones entre arreglos fila o columna con

```
clear;

x = [1, 2, 3, 4]
y = [4, 3, 2, 1]

suma  = x + y
resta = x - y
mult  = x .* y
div   = x ./ y
```

Las operaciones de suma y resta son iguales que para el álgebra lineal. En cambio .* y ./ son operadores nombrados para la división de arreglos. Si se omite el punto el significado es diferente lo cual es equivalente a

```
clear;

x = [1, 2, 3, 4]
y = [4, 3, 2, 1]
```

Para la suma hacemos

```
for i=1:4; suma(i)  = x(i) + y(i); end;
```

En la resta

```
for i=1:4; resta(i) = x(i) - y(i); end;
```

Multiplicación

```
for i=1:4; mult(i)  = x(i) * y(i); end;
```

y División

```
for i=1:4; div(i)   = x(i) / y(i); end;
```

Ver arreglos.m

Propiedades de los arreglos

Concatenación

```
clear;

x = [2, 3]
x = [x, 4]
```

en el caso de arreglos columna

```
clear;

y = [2, 3]'
y = [y; 4]
```

para extraer la una parte de un vector

```
clear;

y = [1,2,3,4,5,6,7,8,9]
w = y(3:6)
```

para obtener la longitud de un arreglo se utiliza

```
clear;

y = [1,2,3,4,5,6,7,8,9]
length(y)
```

La variables de cadena también puede tener caracteres

```
clear;

v = 'Hola Mundo'

w = ['H', 'o', 'l', 'a', ' ', 'M', 'u', 'n', 'd', 'o']
```

y podemos cambiar el orden de impresión haciendo

```
clear;

v = 'Hola Mundo'
v = v'

w = ['H', 'o', 'l', 'a', ' ', 'M', 'u', 'n', 'd', 'o']
w = w'
```

Ver arreglos_propiedades.m

1.5.1. Ejemplo 1

Dado un arreglo de datos calcular el promedio de este y el mayor de los valores.

```
x = [3 9 5 8 2]
n = length(x);
suma =0;
max = x(1);
for k=1:n
   suma = suma + x(k);
   if(max < x(k)); max = x(k); end;
end;
suma = suma/n;

fprintf('El promedio es = \%5.2f\n', suma);
fprintf('El mayor es    = \%5.2f\n', max);
```

Ver promedio_mayor.m

1.5.2. Ejemplo 2

Escribir un programa que verifique si una cadena de caracteres e un palíndromo.

```
x = '10011'
y = x(length(x):-1:1)
if(x == y)
```

```
   fprintf('Es palíndromo \n');
else
   fprintf('No es palíndromo \n');
end;
```

Ver palindroma.m

1.5.3. Arreglos Bidimensionales

Para definir arreglos bidimensionales o matrices hacemos

```
m = [0.1, 0.2, 0.3; 0.4, 0.5, 0.6; 0.7, 0.8, 0.9]
```

otra manera de definirlos es utilizar

```
clear;
m(1,1) = 0.1;
m(1,2) = 0.2;
m(1,3) = 0.3;
m(2,1) = 0.4;
m(2,2) = 0.5;
m(2,3) = 0.6;
m(3,1) = 0.7;
m(3,2) = 0.8;
m(3,3) = 0.9;
```

Podemos listar columnas de la matriz haciendo

```
m(:,1)
m(:,2)
m(:,3)
```

o también renglones

```
m(1,:)
m(2,:)
m(3,:)
```

podemos realizar operaciones de +. -, * y / con matrices

```
a = [0.1, 0.2, 0.3; 0.4, 0.5, 0.6; 0.7, 0.8, 0.9]
b = [0.3, 0.4, 1.3; 0.6, -0.7, 1.0; -2.0, 1.8, 9]

suma  = a + b
resta = a - b
mult  = a .* b
```

```
div   = a ./ b
```

lo cual es equivalente a

```
a = [0.1, 0.2, 0.3; 0.4, 0.5, 0.6; 0.7, 0.8, 0.9]
b = [0.3, 0.4, 1.3; 0.6, -0.7, 1.0; -2.0, 1.8, 9]

for i=1:3
   for j=1:3
      suma(i,j) = a(i,j) + b(i,j);
   end
end

%suma

for i=1:3
   for j=1:3
      resta(i,j) = a(i,j) - b(i,j);
   end
end

%resta

for i=1:3
   for j=1:3
      mult(i,j) = a(i,j) * b(i,j);
   end
end

%multiplicación

for i=1:3
   for j=1:3
      div(i,j) = a(i,j) / b(i,j);
   end
end

%división

pause;
```

también podemos utilizar el operador de potenciación en arreglos

```
clear;

a = [0.1, 0.2, 0.3; 0.4, 0.5, 0.6; 0.7, 0.8, 0.9]

g = a .^2

pause;
```

el cual es equivalente

```
a = [0.1, 0.2, 0.3; 0.4, 0.5, 0.6; 0.7, 0.8, 0.9]

for i=1:3
   for j=1:3
      g(i,j) = a(i,j)^2;
   end
end
```

Ver arreglos_2d.m

Algebra de Matrices

Cuando queremos realizar las operaciones del álgebra lineal de suma, resta, multiplicación y división hacemos

la suma y resta son iguales pero la multiplicación cambia

```
A = [0.1, 0.2, 0.3; 0.4, 0.5, 0.6; 0.7, 0.8, 0.9]

x = [1, 2, 3]'

b = A*x
```

para la división tendremos

```
A = [1, 4; 3, 5]

x = [2, 3]'

b = A\x
```

1.5.4. Ejemplos de arreglos sin utilizar ciclos

1. Dado un arreglo cualquiera, escribir el código que permita saber cual es el mayor múltiplo de 3 o 4.

a = randi([0, 1000], 1, 20) b = a(mod(a,3)==0 — mod(a,4) == 0)

max(b)

La ejecución queda:

a =

Columns 1 through 13:

253 615 183 578 840 211 974 213 173 579 541 535 793

Columns 14 through 20:

194 636 970 106 249 680 983

b =

615 183 840 213 579 636 249 680

ans = 840

2. Dado un arreglo cualquiera escribir código para generar los números impares múltiplos de 7 entre 0 y 100

a = [1:2:100] a(mod(a, 7)==0)

La ejecución es:

a =

Columns 1 through 15:

1 3 5 7 9 11 13 15 17 19 21 23 25 27 29

Columns 16 through 30:

31 33 35 37 39 41 43 45 47 49 51 53 55 57 59

Columns 31 through 45:

61 63 65 67 69 71 73 75 77 79 81 83 85 87 89

Columns 46 through 50:

91 93 95 97 99

ans =

7 21 35 49 63 77 91

3. Calcular la suma de la suma de los recíprocos potencia de dos

sum(2.^[1:1000])

y la respuesta es:

ans = 1

1.6. Estructuras de Programa y Funciones

Las funciones en MATLAB, que se guardan como archivos independientes, equivalen a las subrutinas y funciones de otros lenguajes.

1.6.1. Funciones que devuelven una sola variable

Consideremos el ejemplo de escribir la función

$$f(x) = 2x^3 + 4x^2 + 3$$

podemos escribir en MATLAB un archivo con el nombre fun00.m como

```
function y = fun01(x)
 y = 2*x^3+4*x^2+ 3;
```

Para ejecutar esta función, desde el ambiente de MATLAB podemos valuar f(2) como

fun01(2)

1.6.2. Funciones que devuelven mas de una variable

Una función puede devolver más de una variable y la sintaxis para escribir esta función es

```
function [Y1, Y2, ..., Yn ] = fun_regresa_varias(X)
...
...
...
Y1 = ....
Y2 = ....
...
...
```

Supongamos que dado un conjunto de datos queremos realizar una función que devuelva la media y la desviación estándar. Primero escribimos un archivo llamado fun02.m, que tenga las siguientes instrucciones.

```
function [media, des] = fun02(x)
   n = length(x);
```

```
   suma = 0;
   for k=1:n
      suma = suma + x(k);
   end;
   media = suma/n;
   suma = 0;
   for k=1:n
      suma = (x(k) - media)^2;
   end;
   des = sqrt(suma/n );
end
```

Guardamos el archivo y ejecutamos desde el MATLAB con:

[m d] = fun02(x)

y la ejecución regresa.

```
m =

2

d =

0.5774
```

Note que la función recibe dos argumentos a los que llamamos m y d. De hacer el llamado de la función sin poner estos dos, no se genera error alguno pero solo se imprime el primer parámetro que devuelve la función.

fun02(x)

ans =

2

1.6.3. Funciones anonimas

Las funciones anónimas en MATLAB, o funciones en línea, son funciones de una sola línea con una sola salida que se definen usando el símbolo @ y son ideales para tareas rápidas y de un solo uso. Permiten crear pequeñas funciones de manera concisa sin la necesidad de un archivo .m separado, asignándolas a una variable y pasándolas como argumentos a otras funciones.

Sintaxis y Creación

Para crear una función anónima, usa el símbolo @ seguido de los argumentos de entrada entre paréntesis, y luego la expresión que la función debe evaluar.

Ejemplo: Definir una función que eleve al cuadrado su entrada x y le sume 1:

Código

f = @(x) x.^2 + 1;

@: Indica que se está creando una función anónima.

(x): Los argumentos de entrada de la función. x.^2 + 1: La expresión que la función evalúa.

Uso y Características

Versatilidad: Se pueden definir directamente en la ventana de comandos o dentro de scripts.

Paso como argumento: Las funciones anónimas, al ser manipuladores de función, pueden ser argumentos para otras funciones (funciones de función).

Ejemplo: Pasar f (la función anónima definida arriba) a otra función que realiza una operación sobre ella.

Concisión: Son útiles para funciones pequeñas, de una sola línea, que simplifican el código al reducir la cantidad de archivos necesarios. Llamada: Se llama a la función anónima como cualquier otra función, usando el nombre de la variable a la que se ha asignado.

Ejemplo: Para usar la función definida anteriormente:

Código

resultado = f(5);

Arreglos de funciones anonimas

% Definir funciones anónimas (function handles)

suma_anonima = @(a, b) a + b;

resta_anonima = @(a, b) a - b;

multiplica_anonima = @(a, b) a * b;

% Crear un arreglo de celdas con las funciones anónimas

funciones = {suma_anonima, resta_anonima, multiplica_anonima};

% Acceder a una función del arreglo y ejecutarla

resultado_suma = funciones{1}(5, 3); % Llamar a la primera función (suma)

```
resultado_resta = funciones{2}(10, 2); % Llamar a la segunda función (resta)
resultado_multi = funciones{3}(4, 6); % Llamar a la tercera función (multiplicación)
disp(resultado_suma); % Mostrará 8
disp(resultado_resta); % Mostrará 8
disp(resultado_multi); % Mostrará 24
% También puedes iterar sobre las funciones en el arreglo
for i = 1:length(funciones)
  fprintf('Resultado de la función %d: %d \n', i, funcionesi(2, 3));
end
```

1.6.4. Ejemplo 1

Se tiene un circuito eléctrico formado por una fuente de voltaje variable en el tiempo $v(t) = 10cos(20t)$, una resistencia $R = 5$ y un diodo conectados en serie tal como se muestra en la figura 3.8.

Para este circuito hacer

- Escribir la función que modela el circuito,
- Escribir el código para resolver el problema y
- Graficar la corriente como función del tiempo

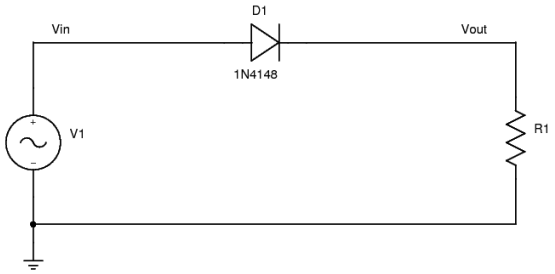


Figura 1.1: Circuito serie con diodo

La función para modelar el diodo en el circuito es:

```
function it = diodo(vt)
   R = 5;

   if vt > 0.7
       vd = 0.7;
   else
       vd = vt;
   end

   it = (vt - vd)/R;
end
```

Dado que tenemos un operador condicional, no es posible hacer la sectorización. Es este caso utilizamos la función arrayfun de matlab tal como se muestra en el siguiente código.

```
t = [0:0.01:2];             % tiempo
vt = 10*cos(20*t);          % Voltaje variante en el tiempo
it = arrayfun(@diodo, vt); % Calculo de la corriente

plot(t, it, t, vt);         % Corriente Calculada

xlabel('t (seg.)');
ylabel('v(t) y i(t)');
title ('Solución de un circuito con Diodo');
```

La Figura 1.2 muestra la solución encontrada en función del tiempo

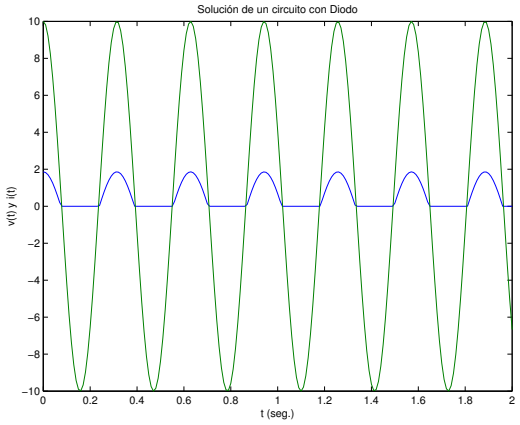


Figura 1.2: Solución gráfica del circuito con Diodo

1.6.5. Ejemplo 2

Determinar el cruce por cero de la función $f(x) = x–cos(x)$, utilizando el método de Newton Raphson. Este algoritmo iterativo se resuelve haciendo

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}$$

La implementación en MATLAB es:

```
function z = cero(x0)

 for k = 1: 100
   xnva = x0 - f(x0)/df(x0);
   x0 = xnva;
   fprintf('Iteracion \%d f(\%f) = \%f\n', k, x0, f(x0));
   if(abs(f(x0)) < 0.00001) break; end;
 end;

z =x0;

function y = f(x)
  y = x-cos(x);

function dy = df(x)
 dy = 1+sin(x);
```

1.6.6. Ejemplo 3

.

Determinar el cruce por cero de la función $f(x) = x–cos(x)$, utilizando el método de Bisecciones.

La implementación en MATLAB es:

```
function b = biseccion(x0,x1)
 n=x0:0.1:x1;
 plot(n,f(n))
 xlabel('eje x');
 ylabel('eje y');
 hold on;
 fmin = min(f(n));
```

```
 fmax = max(f(n));

 for  S = 1:100
     mau=((x1+x0)/2);
     a=f(x0);
     b=f(x1);
     fmau=f(mau);
     fprintf('Iteracion %d f(mau)(%f) = %f\n', S, mau, f(mau));

     plot(x0, fmin:0.01:fmax, x1, fmin:0.01:fmax);
     pause;

     if       a*fmau > 0   x0=mau;
     elseif   a*fmau==0    x0=mau;
     elseif   a*fmau==0    x1=mau;
     elseif   a*fmau < 0   x1=mau;

     if (abs(fmau) < 0.0001), break, end;
  end;
end;
  disp(['La solucion esta en:'])
  disp([mau]);

function y = f(x)
  y = x-cos(x);
```

Modelos, computadoras y errores

2.1. Modelos y Computadoras

2.2. Serie de Taylor y Errores de Truncamiento

En matemáticas, una serie de Taylor es una representación de una función como una infinita suma de términos. Estos términos se calculan a partir de las derivadas de la función para un determinado valor de la variable (respecto de la cual se deriva), lo que involucra un punto específico sobre la función. Si esta serie está centrada sobre el punto cero, se le denomina serie de McLaurin.

2.2.1. La serie de Taylor

Si la función f y sus primeras $n+1$ derivadas son continuas, en un intervalo que contiene a y x, entonces el valor de la función esta dado por:

$$f(x) = f(a) + f^{i}(a)(x-a) + \frac{1}{2!}f^{ii}(a)(x-a)^2 + \frac{1}{3!}f^{iii}(a)(x-a)^3 + \cdots + \frac{1}{n!}f^{n}(a)(x-a)^n \quad (2.1)$$

La cual puede representarse como una sumatoria para N términos mediante

$$f(x) = \sum_{n=0}^{N} \frac{f^n(a)}{n!}(x-a)^n$$

donde $f^0(a) = f(x)|_{x=a}$ y $f^n(a) = \left.\frac{d^n f(x)}{dx^n}\right|_{x=a}$

Con frecuencia es conveniente simplificar la serie de Taylor definiendo un paso h como $h = x^{(i+1)} - x^{(i)}$ con $a = x^{(i)}$. Con esto podemos expresar la serie de Taylor como:

$$f(x^{(i+1)}) = f(x^{(i)}) + f^{i}(x^{(i)})h + \frac{1}{2!}f^{ii}(x^{(i)})h^2 + \frac{1}{3!}f^{iii}(x^{(i)})h^3 + \cdots + \frac{1}{n!}f^{n}(x^{(i)})h^n \quad (2.2)$$

2.2.2. Ejemplo 1

Dada la función $f(x) = e^x$ hacer la implementación utilizando la serie de Taylor dada por (2.1) considerando que $a = 0$.

$$\begin{aligned} f(x) &= f(x) = e^x \\ f^{i}(x) &= \frac{df(x)}{dx} = e^x \\ f^{ii}(x) &= \frac{d^2f(x)}{dx^2} = e^x \\ f^{iii}(x) &= \frac{d^3f(x)}{dx^3} = e^x \\ f^{iv}(x) &= \frac{d^4f(x)}{dx^4} = e^x \end{aligned}$$

Hacemos la evaluación de la derivadas en $a = 0$

Orden n	$f^n(a)$	$f^n(0)$
0	e^x	1
1	e^x	1
2	e^x	1
3	e^x	1
$\vdots$	$\vdots$	$\vdots$
N	e^x	1

Sustituyendo en la ecuación (2.1) tenemos

$$e^x \approx f(a) + f^{i}(a)(x-a) + \frac{1}{2!}f^{ii}(a)(x-a)^2 + \frac{1}{3!}f^{iii}(a)(x-a)^3 + \cdots + \frac{1}{n!}f^{n}(a)$$

$$e^x \approx 1 + 1 \times (x-0) + \frac{1}{2!} \times 1 \times (x-0)^2 + \frac{1}{3!} \times 1 \times (x-0)^3 + \frac{1}{4!} \times 1 \times (x-0)^4 + \cdots$$

La aproximación polinomial final, para la función e^x es:

$$e^x \approx 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \frac{x^5}{5!} + \frac{x^6}{6!} + \frac{x^7}{7!} + \cdots$$

En forma compacta podemos representar la serie como:

$$e^x \approx \sum_{k=0}^{N} \frac{x^k}{k!}$$

La implementación en Matlab es:

```
function y = exponencial(x)
 y =0;
 for k= 0:40
     y = y + x.^k/factorial(k);
 end;
```

Ver exponencial.m

2.2.3. Ejemplo 2

Dada la función $f(x) = cos(x)$ hacer la implementación utilizando la serie de Taylor dada por (2.1) considerando que $a = 0$.

$$\begin{aligned} f^0(x) &= f(x) = cos(x) \\ f^i(x) &= \frac{df(x)}{dx} = -sin(x) \\ f^{ii}(x) &= \frac{d^2f(x)}{dx^2} = -cos(x) \\ f^{iii}(x) &= \frac{d^3f(x)}{dx^3} = -sin(x) \\ f^{iv}(x) &= \frac{d^4f(x)}{dx^4} = -cos(x) \end{aligned}$$

Sustituyendo para $x = a = 0$

Orden n	$f^n(a)$	$f^n(0)$
0	$cos(a)$	1
1	$-sen(a)$	0
2	$-cos(a)$	-1
3	$sen(a)$	0
4	$cos(a)$	1
5	$-sen(a)$	0
6	$-cos(a)$	-1
7	$sen(a)$	0
8	$cos(a)$	1
9	$-sen(a)$	0
10	$-cos(a)$	-1

La aproximación polinomial final, para la función coseno es:

$$cos(x) \approx 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \frac{x^8}{8!} - \frac{x^{10}}{10!}$$

En forma compacta podemos representar la serie como:

$$cos(x) \approx \sum_{k=0}^{N} \frac{(-1)^k x^{2k}}{(2k)!}$$

La implementación en Matlab es:

```
function y = coseno(x)
    y =0;
    N=80
    for k= 0:N
        y = y + (-1)^k*x.^(2*k)/factorial(2*k);
    end;
end
```

Ver coseno.m

En la Figura 2.3 podemos ver la gráfica generada por nuestra función coseno marcada con +, sobre impuesta a la gráfica de la función $cos(x)$ de Matlab. Note la cálida de la aproximación.

2.2.4. Ejemplo 3

Utilizar los términos de la serie de Taylor para $n = 0$ hasta 6, para aproximar la función $f(x) = cos(x)$ en $x^{(i+1)} = \pi/3$ y como condición inicial sus derivadas en $x^{(i)} = \pi/4$.

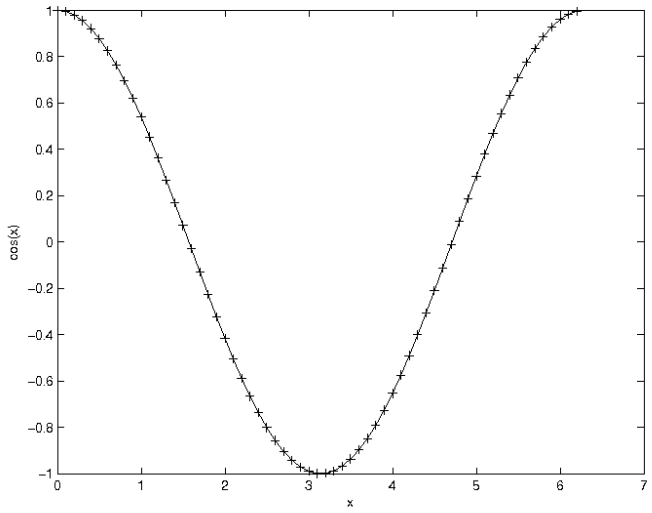


Figura 2.3: Implementación de la función coseno utilizando serie de Taylor

Para nuestra implementación tenemos que el incremento $h = \pi/3 - \pi/4 = \pi/12$. La aproximación en serie de Taylor utilizando (2.2) es:

$$f(x^{(i+1)}) = f(x^{(i)}) + f^i(x^{(i)})h + \frac{1}{2!}f^{ii}(x^{(i)})h^2 + \frac{1}{3!}f^{iii}(x^{(i)})h^3 + \cdots + \frac{1}{n!}f^n(x^{(i)})h^n$$

Sustituyendo valores tenemos

$$f(x^{(i+1)}) \approx f\left(\frac{\pi}{4}\right) + f^i\left(\frac{\pi}{4}\right)\frac{\pi}{12} + \frac{1}{2!}f^{ii}\left(\frac{\pi}{4}\right)\left(\frac{\pi}{12}\right)^2 + \frac{1}{3!}f^{iii}\left(\frac{\pi}{4}\right)\left(\frac{\pi}{12}\right)^3 + \cdots + \frac{1}{n!}f^n\left(\frac{\pi}{4}\right)\left(\frac{\pi}{12}\right)^n$$

La cual puede ser aproximada por la sumatoria

$$f(x^{(i+1)}) \approx \sum_{n=0}^{N} \frac{1}{n!}f^n\left(\frac{\pi}{4}\right)\left(\frac{\pi}{12}\right)^n$$

La evaluación para diferentes valores de n es

$$
\begin{aligned}
& N = 0 \\
f(x^{(i+1)}) &\approx 0.7071 \\
& N = 1 \\
f(x^{(i+1)}) &\approx 0.7071 + (-0.7071) \times \frac{\pi}{12} = 0.5219 \\
& N = 2 \\
f(x^{(i+1)}) &\approx 0.7071 + (-0.7071) \times \frac{\pi}{12} + \frac{1}{2} \times (-0.7071) \left(\frac{\pi}{12}\right)^2 = 0.4977
\end{aligned}
$$

Los valores de las derivadas y el error de aproximación se presenta en la siguiente tabla.

Orden n	$f^n(\frac{\pi}{4})$	$f(x^{(i+1)})$	error (%)
0	$\cos(\pi/4)$	0.707106781	-41.4
1	-sen$(\pi/4)$	0.521986659	-4.4
2	-cos$(\pi/4)$	0.497754491	0.449
3	sen$(\pi/4)$	0.499869147	2.62x10-2
4	cos$(\pi/4)$	0.500007551	-1.51x10-3
5	-sen$(\pi/4)$	0.500000304	-6.08x10-5
6	-cos$(\pi/4)$	0.499999988	2.40x10-6

Note, que a medida que se introducen más términos, la aproximación se vuelve más exacta y el porcentaje de error disminuye.

2.2.5. Ejemplo 4

Hacer la implementación en Serie de Taylor para la función $f(x) = sen(x)$.

De acuerdo con lo anterior la serie de Taylor para el seno esta dada como

$$sen(x) \approx x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \cdots$$

$$sen(x) \approx \sum_{k=0}^{N} \frac{(-1)^k x^{2k+1}}{(2k+1)!}$$

La implementación en Matlab es:

```
function y = seno(x)
    y = 0;
    N = 40;
```

```
    for k= 0:N
        y = y + (-1)^k*x.^(2*k+1)/factorial(2*k+1);
    end
end
```

ver seno.m

2.2.6. Uso de la serie de Taylor para estimar errores de Truncamiento

La serie de Taylor es muy útil para hacer la estimación de errores de truncamiento. Esta estimación ya la realizamos en los ejemplos anteriores. Recordemos que la serie de Taylor la podemos representar, con $a = x^{(i)}$, como:

$$v(x^{(i+1)}) = v(x^{(i)}) + v^{'}(x^{(i)})(x^{(i+1)} - x^{(i)}) + \frac{1}{2!}v^{''}(x^{(i)})(x^{(i+1)} - x^{(i)})^2 + \cdots + \frac{1}{n!}v^{n}(x_i)(x^{(i+1)} - x^{(i)})^n$$

Ahora, truncando la serie después del término con la primera derivada, se obtiene:

$$v(x^{(i+1)}) = v(x^{(i)}) + v^{'}(x^{(i)})(x^{(i+1)} - x^{(i)}) + R_1$$

Despejando el valor de v', tenemos:

$$v^{'}(x^{(i)}) = \frac{v(x^{(i+1)}) - v(x^{(i)})}{(x^{(i+1)} - x^{(i)})} - \frac{R_1}{(x^{(i+1)} - x^{(i)})}$$

El primer término de la ecuación represente la aproximación de la derivada y el segundo el error de truncamiento. Note que el error de truncamiento se hace más pequeño a medida que $x^{(i+1)} - x^{(i)}$ (incremento) se hace pequeño. Así que podemos hacer una buena aproximación de derivadas utilizando el primer término, siempre y cuando se utilicen incrementos pequeños.

El código en matlab para la implementación de la derivada queda como:

```
function [xnva, dy] = derivada(x, y)

        N  = length(x);
        dv = zeros(N-1);

        for k=1:N-1
                dy(k) = (y(k+1) - y(k))/(x(k+1) - x(k));
```

```
        end;

        xnva = x(1:N-1);
end
```

Ejemplo

Vamos a calcular la derivada de la función $y = cos(x)$y comparar con el valor exacto de la derivada $y' = -sen(x)$. Para esto consideramos que $0 \leq x \leq 2\pi$ y que el incremento es $h = 0.01$. El siguiente código muestra la manera de utilizar la función derivada y en la Figura 2.4 se muestra la gráfica de la derivada real y la aproximada con la serie de Taylor.

```
        h = 0.01;
        x = [0:h:2*pi];
        y = cos(x);
        dy = sin(x);
        [dx, dy] = derivada(x, y);
        plot(x, -sin(x), dx, dy)
        grid on
```

ver derivada.m

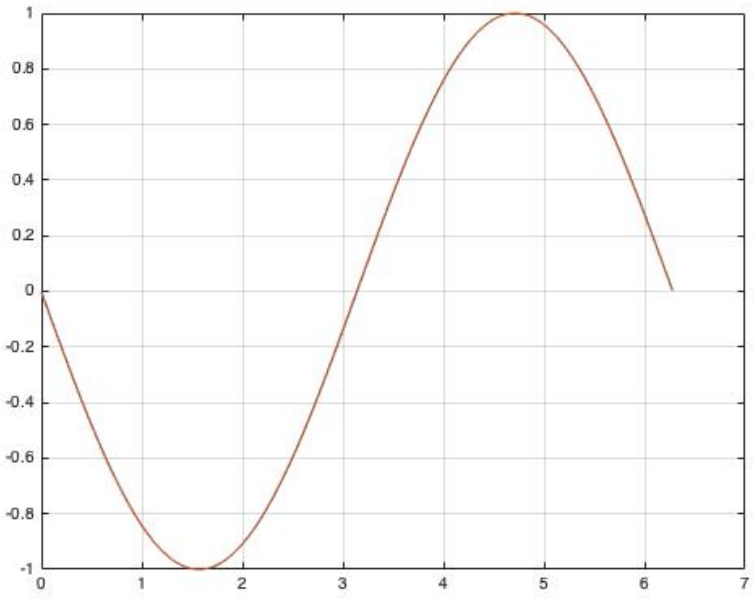


Figura 2.4: Gráfica mostrando la derivada real del coseno contra la estimación utilizando serie de Taylor

Solución de Ecuaciones no-lineales

3.1. Método Gráfico

Un método simple para obtener, visualmente la raíz de la ecuación $f(x) = 0$, consiste en graficar la función y observar en donde cruza el eje x. Este punto que representa el valor de x para el cual $f(x) = 0$, proporciona una aproximación inicial de la raíz.

3.1.1. Ejemplo

Utilice el método gráfico para observar algunas de las raíces de la función $f(x) = sen10x + cos3x$, en el intervalo $[0, 5]$

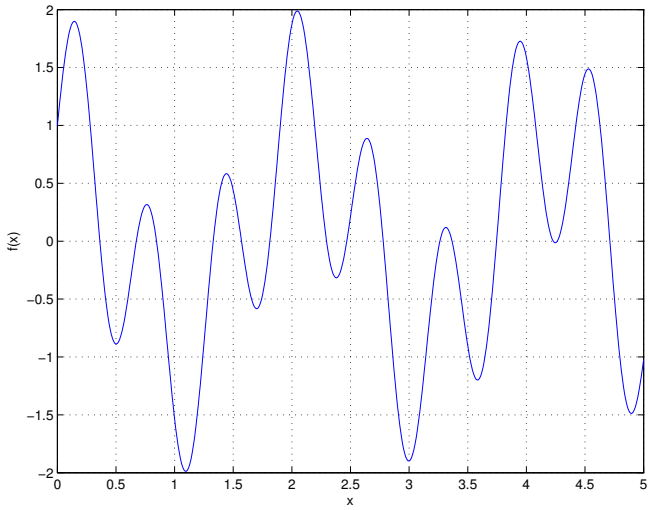


Figura 3.5: Método Gráfico

En la Figura 3.5 se muestran los cruces por cero de la función en el intervalo dado.

3.1.2. Implementación

La implementación en Matlab es

```
function Metodo_Grafico(funcion, x)
   f = funcion;
   y = f(x);
   plot(x, y);
   xlabel('x');
   ylabel('f(x)')
   grid on
```

Ver Metodo_Grafico.m. La implementación de la función

```
function y = f1(x)
   y  = sin (10*x) + cos(3*x);
```

La corrida del método es

```
Metodo_Grafico(@f1, 0:0.01:5)
```

3.2. Método de Bisección

Para este método debemos considerar una función continua dentro de un intervalo $[a, b]$ tal que $f(a)$ tenga diferente signo $f(a) * f(b) < 0$.

El proceso de decisión para encontrar la raíz consiste en dividir el intervalo $[inicio, fin]$ a la mitad $mitad = (inicio + fin)/2$ y luego analizar las tres posibilidades que se pueden dar.

1. Si $f(inicio)$ y $f(mitad)$ tienen signos opuestos, entonces hay un cero entre $[inicio, mitad]$.
2. Si $f(mitad)$ y $f(fin)$ tienen signos opuestos, entonces, hay un cero en $[mitad, fin]$.
3. Si $f(mitad)$ es igual a cero, entonces $mitad$ es un cero

La implementación en Matlab es:

```
function r = Biseccion(f, inicio, fin)
 mitad = 0;
 while abs((fin - inicio)/fin) > 0.0001
     mitad = (fin+inicio)/2.0;

   if(f(mitad) == 0)
       r = mitad;
       return;
```

```
  end;

  if f(inicio)*f(mitad) < 0
    fin = mitad;
  else
    inicio = mitad;
  end;
end;

r= mitad;
```

3.2.1. Ejemplo

Calcular los ceros de la función $f(x) = x - cos(x)$ utilizando el algoritmo de Bisección en el intervalo $[0, 1]$.

iter	a	c	b	f(a)	f(c)	f(b)
0	0.000000	0.500000	1.000000	-1.000000	-0.377583	0.459698
1	0.500000	0.750000	1.000000	-0.377583	0.018311	0.459698
2	0.500000	0.625000	0.750000	-0.377583	-0.185963	0.018311
3	0.625000	0.687500	0.750000	-0.185963	-0.085335	0.018311
4	0.687500	0.718750	0.750000	-0.085335	-0.033879	0.018311
5	0.718750	0.734375	0.750000	-0.033879	-0.007875	0.018311
6	0.734375	0.742188	0.750000	-0.007875	0.005196	0.018311
7	0.734375	0.738281	0.742188	-0.007875	-0.001345	0.005196
8	0.738281	0.740234	0.742188	-0.001345	0.001924	0.005196
9	0.738281	0.739258	0.740234	-0.001345	0.000289	0.001924
10	0.738281	0.738770	0.739258	-0.001345	-0.000528	0.000289
11	0.738770	0.739014	0.739258	-0.000528	-0.000120	0.000289
12	0.739014	0.739136	0.739258	-0.000120	0.000085	0.000289
13	0.739014	0.739075	0.739136	-0.000120	-0.000017	0.000085

Para ver los resultados correr

```
Biseccion(@f2, 0, 1)
```

3.3. Método de la falsa posición (Regula Falsi)

Una de las razones de la introducción de este método es que la velocidad de convergencia del método de Bisecciones es bastante baja. En el método de Bisección se usa el punto medio del intervalo $[a, b]$ para llevar a cabo el siguiente paso. Suele conseguirse una mejor

aproximación usando el punto $[c, 0]$ en el que la recta secante L, que pasa por los puntos $[a, f(a)]$ y $[b, f(b)]$.

En la Figura 3.6 se puede ver como funciona el método. En esta figura en azul esta la función de la cual queremos calcular el cruce por cero y en negro dos líneas rectas que aproximan la solución.

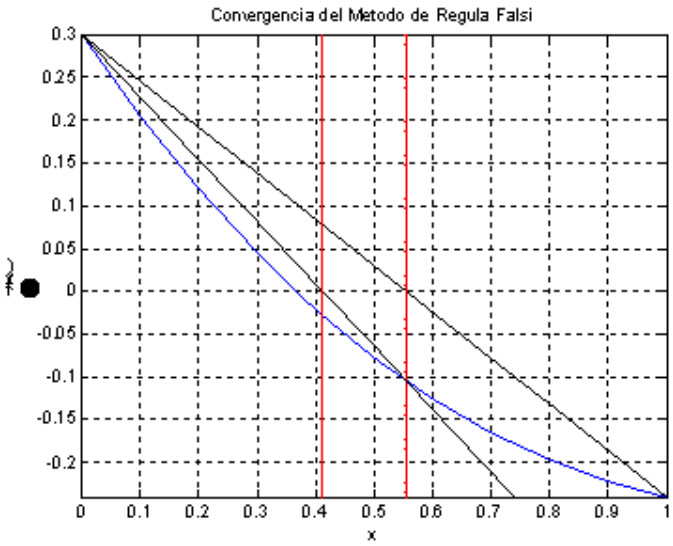


Figura 3.6: Método Regula Falsi

Para calcula la ecuación de la línea secante hacemos

$$\begin{aligned} p_1 &= [a, f(a)] \\ p_2 &= [b, f(b)] \end{aligned}$$

y sustituimos en la ecuación de la línea recta.

$$y–f(a) = (f(b)–f(a)) * (x–a)/(b–a)$$

El cruce por cero de esta ecuación esta en

$$c = a–f(a) * (b - a)/(f(b)–f(a))$$

Entonces el método de Bisecciones puede ser modificado, en lugar de calcular $c = (a+b)/2$ hacemos $c = a–f(a)*(b–a)/(f(b)–f(a))$ y aplicamos las mismas tres reglas de la Bisección. La implementación en Matlab es:

```
function r = Regula_Falsi(f, a, b)

 while abs((b - a)/a) > 0.0001
     c = a - f(a)*(b-a)/(f(b) - f(a));

   if(f(c) == 0)
       r = c;
       return;
   end;

   if f(a)*f(c) < 0
     b = c;
   else
     a = c;
   end;
 end;

 r= c;
```

Para ejecutar hacer

```
Regula_Falsi(@f2, 0, 1)
```

3.3.1. Ejemplo

Calcular los ceros de la función $f(x) = x - cos(x)$ utilizando el algoritmo de regula falsi en el intervalo $[0, 1]$.

iter.	a	c	b	f(a)	f(c)	f(b)
0	0.0000	0.6851	1.0000	-1.0000	-0.0893	0.4597
1	0.6851	0.7363	1.0000	-0.0893	-0.0047	0.4597
2	0.7363	0.7389	1.0000	-0.0047	-0.0002	0.4597
3	0.7389	0.7391	1.0000	-0.0002	0.0000	0.4597
4	0.7391	0.7391	1.0000	0.0000	0.0000	0.4597
5	0.7391	0.7391	1.0000	0.0000	0.0000	0.4597
6	0.7391	0.7391	1.0000	0.0000	0.0000	0.4597

3.4. Método de iteración de punto fijo

Los métodos de punto fijo son métodos que predicen la raíz de la ecuación a partir de una formula dada. La formula puede ser desarrollada por una sustitución sucesiva al rearreglar

la ecuación $f(x) = 0$ de tal modo que x quede del lado izquierdo de la ecuación

$$f(x) \equiv x - g(x) = 0$$

Lo que es igual a

$$x = g(x)$$

Esta transformación se puede llevar a cabo mediante operaciones algebraicas o simplemente agregando x a cada lado de la ecuación original. Por ejemplo:

$$\begin{aligned} x^2 - 2x + 3 &= 0 \\ x &= \frac{x^2 + 3}{2} \\ x &= g(x) \end{aligned}$$

mientras que $sen(x) = 0$ puede transformarse haciendo

$$x = sen(x) + x$$

Las iteraciones del método dado un punto inicial $x^{(0)}$, se pueden llevar a cabo calculando $x^{(k+1)}$

$$x^{(k+1)} = g(x^{(k)})$$

La implementación en Matlab es:

```
function r = Punto_Fijo(g, x1)

while 1
    x2 = g(x1)
    error = abs((x2-x1)/x2);
    if(error < 0.0001) break
    else x1 = x2;
    end;
end;
r = x2;
```

3.4.1. Ejemplo

Utilizando la iteración de punto fijo calcular las raíces $f(x) = e^{-x} - x$, para un valor inicial $x^{(0)} = 0$.

De acuerdo con lo revisado la función $g(x) = e^{-x}$ y las iteraciones hasta convergencia son:

k	$x^{(k)}$	$error(x^{(k)})$
1	1.0000	1.0000
2	0.3679	1.7183
3	0.6922	0.4685
4	0.5005	0.3831
5	0.6062	0.1745
6	0.5454	0.1116
7	0.5796	0.0590
8	0.5601	0.0348
9	0.5711	0.0193
⋮	⋮	⋮
19	0.5672	6.7008e-05

Para hacer la ejecución correr:

```
Punto_Fijo(@g1, 0)
```

con

```
function y = g4(x)

y = exp(-x);
```

3.4.2. Ejemplo

Calcular utilizando iteración de punto fijo, un cero de la función $f(x) = sen(10x) + cos(3x)$

Para llevar a cabo la iteración de punto fijo hacemos

$$x = sen(10x) + cos(3x) + x$$

La implementación en Matlab es

```
function y = g1(x)

y = sin(10*x)+cos(3*x) +x;
```

Para ejecutar hacer:

```
Punto_Fijo(@g1, 0)
```

3.5. Método de Newton-Raphson

Dada una función $f(x) : \mathbb{R} \to \mathbb{R}$ continua, con un valor inicial $x^{(0)}$ cerca de una raíz x^* y si la derivada $f'(x)$ existe, entonces puede utilizarse esta información para desarrollar algoritmos que produzcan sucesiones $x^{(k)}$ que converjan a x^* más rápidamente que los algoritmos como los vistos en la secciones anteriores.

El método considera que dada una función $f(x)$, podemos hacer una representación lineal utilizando la serie de Taylor. Así la representación lineal en Serie de Taylor es:

$$f(x^{(k+1)}) = f(x^{(k)}) + f'(x^{(k)})(x^{(k+1)} - x^{(k)}) + R_1$$

Si de la ecuación anterior despejamos el valor de $x^{(k+1)}$, suponemos que $f(x^{(k+1)}) = 0$ y despreciamos el residuo R_1, tenemos que la sucesión $x^{(0)}, x^{(1)}, x^{(2)}, \ldots, x^{(k)}, x^{(k+1)}$ puede ser calculada utilizando la formula:

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})}$$

La implementación del Método en Matlab es:

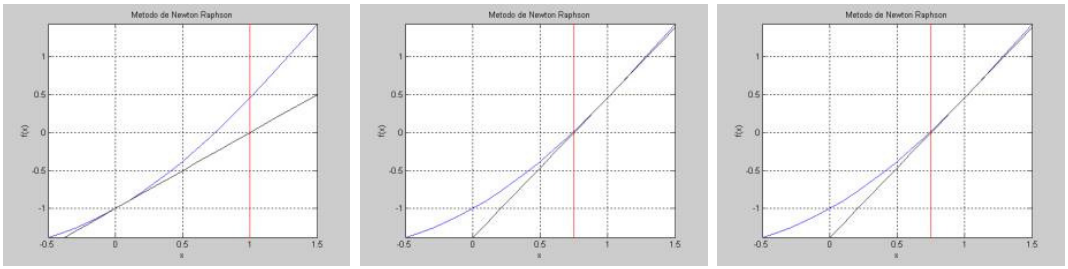
```
function r = Newton_Raphson(f, df, x1)

   while 1
       x2 = x1 - f(x1)/df(x1)
       if abs((x2-x1)/x2) < 0.0001 break;
       else x1 = x2;
       end;
   end;
   r = x2;
end
```

3.5.1. Ejemplo

Calcular los ceros de la función $f(x) = x - cos(x)$ utilizando el algoritmo de Newton Raphson con $x_0 = 0$.

k	$x^{(k)}$
0	0.0000
1	1.0000
2	0.7504
3	0.7391
4	0.7391
5	0.7391
6	0.7391
7	0.7391

En estas Figura 3.7 se muestra tres iteraciones del método. En la figura de la izquierda mostramos la línea recta que es tangente a la función $f(x)$ (en negro) en $x^{(0)} = 0$, note que la línea recta cruza el eje x en $x^{(1)} = 1$. En el centro la tenemos la segunda iteración tomando como valor inicial $x^{(1)} = 1$ y solución $x^{(2)} = 0.7504$. Finalmente a la izquierda tenemos la linea recta y la solución para $x^{(3)} = 0.7391$. Note como poco a poco se acerca a la solución.



(a) $x^{(0)} = 0$ **(b)** $x^{(1)} = 1$ **(c)** $x^{(2)} = 0.7391$

Figura 3.7: Iteraciones del Método de Newton-Raphson

Para correr este ejemplo dar:

```
Newton_Raphson(@f2, @df2, 0)
```

3.6. Aplicaciones

3.6.1. Cálculo de la Raíz Cuadrada

Hacer un algoritmo iterativo que permita hacer el cálculo de la raíz cuadrada de A.

Para este caso nuestra función a resolver es $f(x) = x^2 - A$. La solución cuando $f(x) = 0$ es $x = \sqrt{A}$. Para nuestro calculo hacemos

$$\begin{aligned} f(x) &= x^2 - A \\ f'(x) &= 2x \end{aligned}$$

Los cálculos numéricos suponiendo para calcular la raíz cuadrada de 5 son:

k	$x^{(k)}$
0	2.0000
1	2.2500
2	2.2361
3	2.2361
4	2.2361
5	2.2361
6	2.2361
7	2.2361
8	2.2361
9	2.2361
10	2.2361

Para ver la ejecución dar

```
Newton_Raphson(@f3, @df3, 2)
```

Si para este ejemplo ponemos sustituimos en la formulación original los valores de la derivada y de la función tenemos:

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k)})} = x^{(k)} - \frac{{x^{(k)}}^2 - A}{2x^{(k)}}$$

Reduciendo términos tenemos

$$x^{(k+1)} = \frac{{x^{(k)}}^2 + A}{2x^{(k)}}$$

La función en Matlab queda:

```
function y = Raiz_Cuadrada(A)
   r = A;
   while abs(r*r - A) > 0.00001
      r = (r*r+A)/(2*r);
   end;
end
```

3.6.2. Solución de un circuito con un diodo

Consideremos un circuito de corriente alterna formado por una fuente, una resistencia y un diodo tal como se muestra en la figura 3.8.

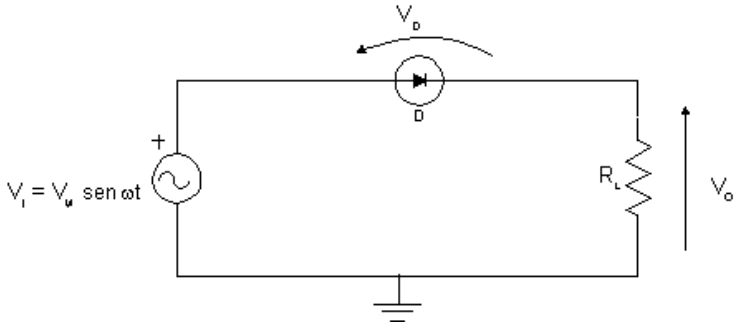


Figura 3.8: Circuito de corriente alterna con un diodo

La ecuación que modela al diodo esta dada por (3.3) y la curva se muestra en la figura 3.9:

$$i_d = I_s(e^{v_d/V_t} - 1) \tag{3.3}$$

donde $I_s = 1^{-12}$ y $V_t = 25.85^{-3}$

Para este ejemplo la ecuación que hay que resolver es:

$$V(t) - Ri(t) - V_d(t) = 0$$

$$V_d(t) = \begin{cases} log(I_d(t)/I_s + 1) * V_t & Si \quad I_d(t) \geq 0 \\ V & sino \quad Id(t) < 0 \end{cases}$$

La solución implementada se muestra en el siguiente código y la solución se muestra en la Fig. 3.10

```
function Simula_Circuito_Diodo()
    % Paramentros del Cicuito

    Vmax = 10;  % Volts;
    R    =  1.8; % ohms
```

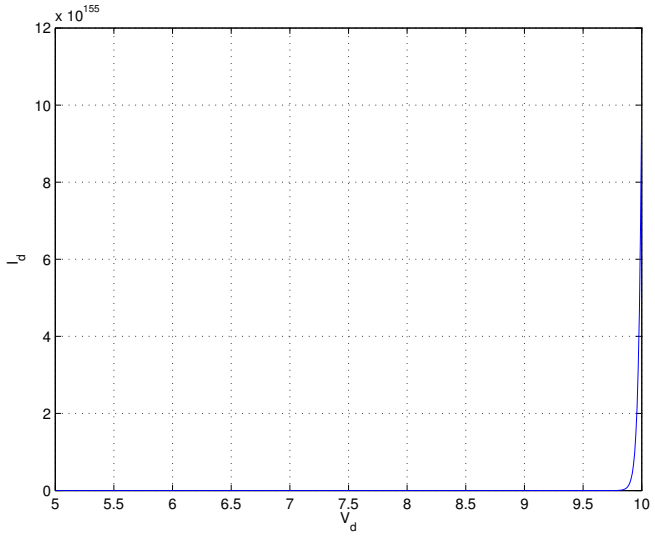


Figura 3.9: Comportamiento del diodo

```
    %tiempo de evaluación
    t = [0:0.001:0.1];
    %Voltaje en función del tiempo
    v = Vmax*sin(377*t);

    for k = 1:length(t)
        I(k) = ReglaFalsa(@Circuito_Diodo, -Vmax/R, Vmax/R, [v(k), R]);
    end

    plot(t, v, t, I);
    title('Solución de un circuito con Diodo');
    xlabel('tiempo s');
    legend('Voltaje', 'Corriente');
end


function di = Circuito_Diodo(I, p)
   V = p(1);
   R = p(2);
   di = V - R*I - Vd(V, I);
end
```

```
function r = ReglaFalsa(f, a, b, p)
%ReglaFalsa(f, a, b, p)
%   f : función a evaluar
%   a : inicio del intervalo
%   b : fin frl intervalo
%   p : parametros

 while 1
     c = a - f(a,p)*(b-a)/(f(b,p) - f(a,p));
     if(abs(f(c,p)) <= 1e-06)
         break;
     end;
     if f(a,p)*f(c,p) < 0
         b = c;
     else
         a = c;
   end;
 end;
 r = c;
end

function V = Vd(Vf, I)
% V = Vd(Vf, I)
%   Vf voltaje de la Fuente
%   I Corriente en el circuito

  Is = 1e-12;
  Vt = 25.85e-3;

  if I>=0
      V = log(I/Is+1)*Vt;
  else
      V = Vf;
  end
end
```

3.6.3. Solución del problema de flujos de Potencia

Se tiene un circuito formado por tres elementos: una fuente de voltaje V, una resistencia R y una carga P_L en una combinación en serie. Dados los valores $V = 10$ volts, $R = 2$ ohms y $P_L = 12$ Wats, calcular la corriente que circula por el circuito.

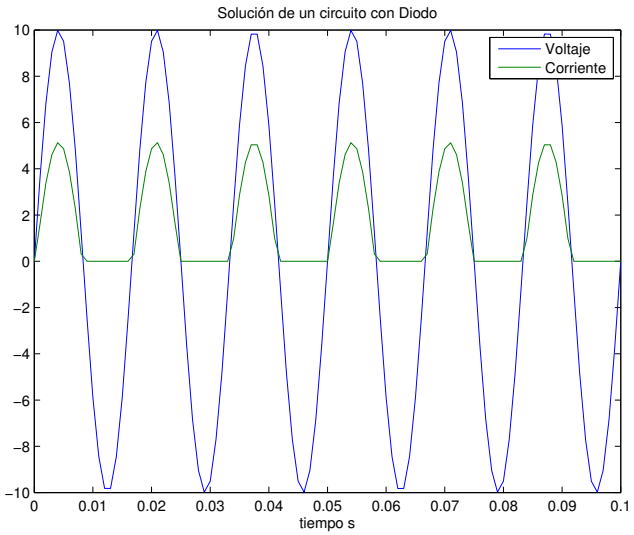


Figura 3.10: Solución al circuito con diodo

La ecuación de voltaje del circuito es:

$$V - Ri - V_L = 0$$

multiplicado por la corriente i en el circuito tenemos

$$Vi - Ri^2 - P_L = 0$$

La solución implementada es:

```
function Simula_Flujos_Potencia()
    % Parametros del Circuito

    V  =  20; % Volts
    R  =  2; % ohms
    P  = 12; % Watts

    fprintf('La solución es %10.6f\n', NR(@fp, @der_fp, V/R, [V, R, P]));
end

function d = fp(I, p)
```

```
   V = p(1);
   R = p(2);
   PL = p(3);
   d = V*I - R*I*I - PL;
end

function der = der_fp(I, p)
   V = p(1);
   R = p(2);
   PL = p(3);
   der = V-2*R*I;
end

function r = NR(f, df, x1, p)
k=1;
while 1
    x2 = x1 - f(x1, p)/df(x1, p);
    if abs(f(x2, p)) < 1e-6 break;
    else x1 = x2;
    end;
    k = k+1;
end;
r = x2;
end
```

La solución obtenida de la corrida es:

```
>> Simula_Flujos_Potencia
La solución es   3.000000
```

3.7. Evaluación de Polinomios

Consideremos como ejemplo el siguiente polinomio.

$$f_3(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3$$

En este caso los coeficientes del polinomio los podemos poner en un vector de datos dado como $a = [a_0, a_1, a_2, a_3]$, y realizar la evaluación polinomial utilizando ciclos. Adicionalmente necesitamos de una función que nos permita calcular la potencias de x.

Manipulando el polinomio, podemos dar una forma eficiente de realizar la evaluación. Así el mismo polinomio, lo podemos representar como:

$$f_3(x) = a_0 + x(a_1 + x(a_2 + x(0x + a_3)))$$

De manera general podemos expresar como

$$f_N(x) = a_0 + x(a_1 + x(a_2 + \cdots + x(a_{N-1} + x(0x + a_N))))$$

Note que ahora ya no tenemos potencia de x y su implementación resulta mas eficiente. La implementación en Matlab es:

```
function y = Evalua_Polinomio(x, a)

n = length(a);
p = 0.0;

for i=n:-1: 1
    p = p.*x + a(i);
end
y = p;
```

3.7.1. Ejemplo

Consideremos el siguiente polinomio $f_2(x) = 1 + 2x + 3x^2$, el cual deseamos evaluar en $x = 20$. Para implementarlo hacemos

i	$p = p * x + a_i$	$p = 0$
2	$p = 0 * 20 + 3$	$p = 3$
1	$p = 3 * 20 + 2$	$p = 62$
0	$p = 62 * 20 + 1$	$p = 1241$

3.7.2. Ejemplo

Hacer la evaluación del polinomio $f_5(x) = x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25$ en $x = 2.5$

i	$p = p * x + a_i$	$p = 0$
5	$p = 0.000000 * 2.500000 + 1.000000$	1.000000
4	$p = 1.000000 * 2.500000 - 3.500000$	-1.000000
3	$p = -1.000000 * 2.500000 + 2.750000$	0.250000
2	$p = 0.250000 * 2.500000 + 2.125000$	2.750000
1	$p = 2.750000 * 2.500000 - 3.875000$	3.000000
0	$p = 3.000000 * 2.500000 + 1.250000$	8.750000

3.7.3. Cálculo de derivadas

Cuando se buscan los ceros de una función, como es el caso del método de Newton, es necesario no solo hacer la evaluación del polinomio sino calcular también su derivada. En este caso la derivada de cualquier polinomio la podemos calcular como:

$$f_n(x) = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_{n-1}x^{n-1} + a_nx^n$$

$$f'_n(x) = a_1 + 2a_2x + 3a_3x^2 + \cdots + (n-1)a_{n-1}x^{n-2} + na_nx^{n-1}$$

Note que el polinomio $f'n(x)$ es un polinomio de menor grado.

Los coeficientes a del polinomio original pueden tomarse para calcular los coeficiente del nuevo polinomio, para esto tomamos en cuenta lo siguiente

$$a = [a_0, a_1, a_2, a_3, a_4, \cdots, a_{n-1}, a_n]$$

$$a' = [a_1, 2a_2, 3a_3, 4a_4, \cdots, (n-1)a_{n-1}, na_n]$$

Con esto tenemos que dado a los coeficientes de del polinomio de derivadas pueden ser calculados y la evaluación de la derivada se reduce a una simple evaluación polinomial.

Así por ejemplo considerando el polinomio $f_3(x) = 1 + 2x + 3x^2 + 4x^3$, tenemos que $a = [1, 2, 3, 4]$ y los coeficiente de $f'_3(x)$ son $a' = [2, 6, 12]$. La evaluación del polinomio resultante lo podemos hacer para $x = 20$ utilizando:

i	$p = p * x + a'_i$	p
2	0*20 +12	12
1	12*20 + 6	246
0	246*20 + 2	4922

3.7.4. Ejemplo

Dado el polinomio de la serie de Taylor correspondiente a la función $f(x) = seno(x)$, hacer la evaluación de la derivada en $x = 0.5$.

El polinomio de la serie de Taylor para seno es:

$$f(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \cdots$$

Los coeficientes de este polinomio son $a = [0, 1, 0, -1/3!, 0, 1/5!, 0, -1/7!, 0, 1/9!]$, por lo tanto los coeficientes para la derivada son $a^* = [1, 0, -3/3!, 0, 5/5!, 0, -7/7!, 0, 9/9!]$

i	$p = p * x + a_i^*$	p
8	$p = 0.000000 * 0.500000 + 0.000025$	0.000025
7	$p = 0.000025 * 0.500000 + 0.000000$	0.000012
6	$p = 0.000012 * 0.500000 - 0.001389$	-0.001383
5	$p = -0.001383 * 0.500000 + 0.000000$	-0.000691
4	$p = -0.000691 * 0.500000 + 0.041667$	0.041321
3	$p = 0.041321 * 0.500000 + 0.000000$	0.020660
2	$p = 0.020660 * 0.500000 - 0.500000$	-0.489670
1	$p = -0.489670 * 0.500000 + 0.000000$	-0.244835
0	$p = -0.244835 * 0.500000 + 1.000000$	0.877583

La implementación en Matlab es

```
function y = Evalua_Derivada(x, a)

    N = length(a);
    b = a(2:N).*[1:N-1];

    y = Evalua_Polinomio(x, b);

end
```

3.7.5. Calculo de la integral

Dado un polinomio de grado $f_n(x)$

$$f_n(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \cdots + a_{n-1} x^{n-1} + a_n x_n$$

La integral de $f_n(x)$ se calcula de manera analítica como

$$\int f(x)dx = CTE + a_0x + \frac{a_1}{2}x^2 + \frac{a_2}{3}x^3 + \frac{a_3}{4}x^4 \cdots + \frac{a_{n-1}}{n}x^n + \frac{a_n}{n+1}x^{n+1}$$

Note que los coeficiente del nuevo polinomio a^* pueden calcularse a partir de los coeficientes a de polinomio original :

$$a = [a_0, a_1, a_2, a_3, a_4, \cdots, a_{n-1}, a_n]$$

$$a^* = [CTE, \tfrac{a_0}{1}, \tfrac{a_1}{2}, \tfrac{a_2}{3}, \tfrac{a_3}{4}, \tfrac{a_4}{5}, \cdots, \tfrac{a_{n-1}}{n}, \tfrac{a_n}{n+1}]$$

Así la evaluación de la integral del polinomio se reduce a calcular los nuevos coeficientes del nuevo polinomio y hacer la evaluación del nuevo polinomio con coeficientes a^*.

En el caso de tener una integral definida debemos notar:

$$\int_{x_0}^{x_1} f(x)dx = \left(CTE + a_0x + \frac{a_1}{2}x^2 + \frac{a_2}{3}x^3 + \frac{a_3}{4}x^4 \cdots + \frac{a_{n-1}}{n}x^n + \frac{a_n}{n+1}x^{n+1}\right)\Bigg|_{x_0}^{x_1}$$

Lo que significa que debemos evaluar el polinomio con coeficientes a^* en x_1 y restar la evaluación en x_2.

La implementación en Matlab para este queda

```
function y = Evalua_Integral(x1, x2, a)
    N = length(a);
    b = [0, a./[1:N]];
    y = Evalua_Polinomio(x2, b) - Evalua_Polinomio(x1, b);
end
```

3.7.6. Ejemplo

Para la serie de Taylor de la función seno hacer la evaluación de la integral:

$$\int_0^{\pi/2} sin(x)dx$$

El polinomio de la serie de Taylor para seno es:

$$f(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!} + \cdots$$

Los coeficientes de este polinomio son $a = [0, 1, 0, -1/3!, 0, 1/5!, 0, -1/7!, 0, 1/9!]$, por lo tanto los coeficientes para la integral serán :

$$a^* = [0, 0, \tfrac{1}{2}, 0, \tfrac{-1}{4\times 3!}, 0, \tfrac{1}{6\times 5!}, 0, \tfrac{-1}{8\times 7!}, 0, \tfrac{1}{10\times 9!}]$$

$$a^* = [0, 0, 0.5, 0, -0.041667, 0, 0.001389, 0, -0.000025, 0]$$

La evaluación del polinomio es $x = \pi/2$ con los coeficientes a^* se muestra a continuación y la evaluación con $x = 0$.

i	$p = p * x + a_i^*$	p
10	p = 0.000000 * 1.570796 + 0.000000	0.000000
9	p = 0.000000 * 1.570796 + 0.000000	0.000000
8	p = 0.000000 * 1.570796 - 0.000025	-0.000024
7	p = -0.000024 * 1.570796 + 0.000000	-0.000038
6	p = -0.000038 * 1.570796 + 0.001389	0.001329
5	p = 0.001329 * 1.570796 + 0.000000	0.002088
4	p = 0.002088 * 1.570796 - 0.041667	-0.038387
3	p = -0.038387 * 1.570796 + 0.000000	-0.060297
2	p = -0.060297 * 1.570796 + 0.500000	0.405285
1	p = 0.405285 * 1.570796 + 0.000000	0.636620
0	p = 0.636620 * 1.570796 + 0.000000	**1.000000**

i	$p = p * x + a_i^*$	p
10	p = 0.000000 * 0.000000 + 0.000000	0.000000
9	p = 0.000000 * 0.000000 + 0.000000	0.000000
8	p = 0.000000 * 0.000000 - 0.000025	-0.000025
7	p = -0.000025 * 0.000000 + 0.000000	0.000000
6	p = 0.000000 * 0.000000 + 0.001389	0.001389
5	p = 0.001389 * 0.000000 + 0.000000	0.000000
4	p = 0.000000 * 0.000000 - 0.041667	-0.041667
3	p = -0.041667 * 0.000000 + 0.000000	0.000000
2	p = 0.000000 * 0.000000 + 0.500000	0.500000
1	p = 0.500000 * 0.000000 + 0.000000	0.000000
0	p = 0.000000 * 0.000000 + 0.000000	**0.000000**

Con esto tenemos que el valor de la integral es igual a $1 - 0 = 1$.

La solución en Matlab queda

```
>> a = [0, 1, 0, -1/factorial(3), 0, 1/factorial(5),
0, -1/factorial(7), 0, 1/factorial(9)]
```

```
>> Evalua_Integral(0, pi/2, a)

ans =

    1.0000
```

3.7.7. Ejemplo cálculo de raíces de un polinomio

Dado un polinomio de grado n

$$f_n(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n.$$

es posible realizar una factorización de la siguiente forma

$$f_n(x) = (x + s) * (a'_0 + a'_1x + a'_2x^2 + ... + a'_{n-1}x^{n-1}).$$

y podemos comprobar que tenemos un monomio de la forma $x - s$, si evaluando la función $f_n(x)$ en $x = -s$, el resultado es $f_n(-s) = 0$. Una manera de resolverlo es utilizar un método de prueba que sería la manera mas tediosa.

El problema lo podemos solucionar utilizando el método de Newton-Raphson, sin embargo hay que hacer un pequeño cambio en la formulación dado que el argumento de la función es negativo. A partir de la serie de Taylor tenemos:

$$f(x^{(k+1)}) \approx f(x^{(k)}) + f'(x^{(k)})(x^{(k+1)} - x^{(k)}) = 0$$

haciendo $x^{(k)} = -s^{(k)}$ tenemos

$$f(-s^{(k+1)}) \approx f(-s^{(k)}) + f'(-s^{(k)})(-s^{(k+1)} + s^{(k)}) = 0$$

Despejando para $s^{(k+1)}$ tenemos la iteración de Newton-Raphson para este problema.

$$s^{(k+1)} = s^{(k)} + \frac{f(-s^{(k)})}{f'(-s^{(k)})}$$

Para hacer la solución más eficiente evaluaremos la función utilizando evaluación polinomial tanto para la función como para la derivada.

```
function sn = factor_NR(s0, a)
    % factor_NR(s0, a) Calcula el factor (x+s) de un polinomio con
    % coeficiente a dado un valor inicial s0 mediante el
    % metodo de Newton-Raphson


    for iter = 1: 100000
        sn = s0 + Evalua_Polinomio(-s0, a)/Evalua_Derivada(-s0, a);

        if abs(Evalua_Polinomio(sn, a)) < 1e-6
        break;
        else
            s0 = sn
        end
    end;
end
```

Como ejemplo numérico calculamos la raíz del polinomio $f_6(x) = x^6 - 1$ con un valor inicial $s^{(0)} = 10$

$$s_1 = 10.000000 + \frac{999999.000000}{-600000.000000} = 8.333335$$

$$s_2 = 8.333335 + \frac{334897.378558}{-241126.784337} = 6.944450$$

$$s_3 = 6.944450 + \frac{112156.191251}{-96903.735989} = 5.787052$$

$$s_4 = 5.787052 + \frac{37560.618299}{-38943.785362} = 4.822569$$

$$s_5 = 4.822569 + \frac{12578.711852}{-15651.050576} = 4.018871$$

$$s_6 = 4.018871 + \frac{4212.321940}{-6290.306217} = 3.349218$$

$$s_7 = 3.349218 + \frac{1410.434918}{-2528.533032} = 2.791411$$

$$s_8 = 2.791411 + \frac{472.088718}{-1016.880869} = 2.327159$$

$$s_9 = 2.327159 + \frac{157.838757}{-409.526161} = 1.941741$$

$$s_{10} = 1.941741 + \frac{52.597923}{-165.618133} = 1.624156$$

$$s_{11} = 1.624156 + \frac{17.355481}{-67.809321} = 1.368210$$

$$s_{12} = 1.368210 + \frac{5.560199}{-28.768380} = 1.174936$$

$$s_{13} = 1.174936 + \frac{1.630779}{-13.434500} = 1.053548$$

$$s_{14} = 1.053548 + \frac{0.367497}{-7.787952} = 1.006360$$

$$s_{15} = 1.006360 + \frac{0.038774}{-6.193251} = 1.000100$$

$$s_{16} = 1.000100 + \frac{0.000598}{-6.002990} = 1.000000$$

Como resultado tenemos que $x^6 - 1 = (x+1)(a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0)$.

3.8. Deflación de polinomios y División de Polinomios

La deflación polomial se define como el cálculo de un polinomio de grado $f_{n_1}(x)$ tal que $f_n(x) = (x-s) * f_{n-1}(x)$. Para esto suponga que conoce, una de las raíces de un polinomio y que podemos realizar la división de este polinomio para obtener un polinomio de grado

menor. Así por ejemplo si tenemos

$$f_n(x) = a_0 + a_1x + a_2x^2 + ... + a_nx^n.$$

Y una de sus raíces es s, entonces podemos escribir

$$f_n(x) = (x + s) * (a'_0 + a'_1x + a'_2x^2 + ... + a'_{n-1}x^{n-1}).$$

En este caso el residuo de la división es cero y podemos calcular un polinomio de grado $n - 1$. Para un polinomio de orden 3 tenemos que

	a_3x^2	$(a_2 - a_3s)x$	$a_1 - (a_2 - a_3s)s$	
$x + s$	a_3x^3	$+a_2x^2$	$+a_1x$	$+a_0$
	$-a_3x^3$	$-a_3sx^2$		
	0	$(a_2 - a_3s)x^2$	$+a_1x$	$+a_0$
		$-(a_2 - a_3s)x^2$	$-(a_2 - a_3s)sx$	
		0	$(a_1 - (a_2 - a_3s)s)x$	$+a_0$
			$-(a_1 - (a_2 - a_3s)s)x$	$-(a_1 - (a_2 - a_3s)s)s$
			0	$a_0 - (a_1 - (a_2 - a_3s)s)s$

La división sintética, es una manera de hacer lo mismo, pero de forma compacta. Así tenemos para el ejemplo anterior que podemos hacer:

$x + s$	a_3	a_2	a_1	a_0
		$-a_3s$	$-(a_2 - a_3s)s$	$-(a_1 - (a_2 - sa_3)s)s$
	a_3	$a_2 - a_3s$	$a_1 - (a_2 - a_3s)s$	$a_0 - (a_1 - (a_2 - a_3s)s)s$

y de manera general suponiendo que el grado del polinomio es n:

$x + s$	a_n	a_{n-1}	a_{n-2}	a_{n-3}
		$-a_ns$	$-(a_{n-1} - a_ns)s$	$-(a_{n-2} - (a_{n-1} - sa_n)s)s$
	a_n	$a_{n-1} - a_ns$	$a_{n-2} - (a_{n-1} - a_ns)s$	$a_{n-3} - (a_{n-2} - (a_{n-1} - a_ns)s)s$

Renombrado términos tenemos

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} - a_n s = a_{n-1} - b_n s \\ b_{n-2} &= a_{n-2} - (a_{n-1} - a_n s)s = a_{n-2} - b_{n-1}s \\ \vdots &= \vdots \\ b_2 &= a_2 - b_3 s \\ b_1 &= a_1 - b_2 s \\ b_0 &= a_0 - b_1 s \end{aligned}$$

En general

$$\begin{aligned} b_n &= a_n \\ b_k &= a_k - b_{k+1}s \\ \\ \forall k &\in [n-1, n-2, \cdots, 2, 1, 0] \end{aligned}$$

La implementación en Matlab es:

```
function y = Division_Sintetica (a, s)

n = length(a);
b = zeros(1,n);

b(n) = a(n);


for k= n-1: -1: 1
    b(k) = a(k) - b(k+1) * s;
end;

disp('residuo ');
disp(b(1));

y = b(2:n);
```

Y para realizar la ejecución hacemos

```
Division_Sintetica([-120, -46, 79, -3, -7, 1], -4)
```

3.8.1. Ejemplo

Dado el polinomio $f_5(x) = x^5 - 7x^4$–$3x^3 + 79x^2$–$46x$–120 encontrar la división sintética con el monomio $(x - 4)$.

$x-4$	1	-7	-3	79	-46	-120
		4	-12	-60	76	120
	1	-3	-15	19	30	0

Para este ejemplo dar en Matlab

```
Division_Sintetica([-120, -46, 79, -3, -7, 1], -4)

El resultado es

30    19   -15    -3     1
```

El polinomio resultante, en este caso, es $f_4(x) = x^4 - 3x^3$–$15x^2 + 19x + 30$. Note que el residuo es cero y en este caso tenemos una deflación polinomial.

En algunos casos en deseable hacer la división sintética de la función $f_n(x)$ entre un polinomio de segundo orden de la forma $f_2(x) = x^2 + rx + s$. Aplicando la técnica anterior podemos realizar la división $f_n(x)/f_2(x)$ como

	a_5x^3	$+(a_4 - a_5r)x^2$	$+((a_3 - sa_5) - (a_4 - a_5r)r)x$		
$x^2 + rx + s$	a_5x^5	$+a_4x^4$	$+a_3x^3$	$+a_2x^2$	$\ldots$
	$-a_5x^5$	$-a_5rx^4$	$-a_5sx^3$		
		$(a_4 - a_5r)x^4$	$+(a_3 - sa_5)x^3$	$+a_2x^2$	$\ldots$
		$-(a_4 - a_5r)x^4$	$-(a_4 - a_5r)rx^3$	$-(a_4 - a_5r)sx^2$	$\ldots$
			$((a_3 - sa_5) - (a_4 - a_5r)r)x^3$	$+(a_2 - (a_4 - a_5r)s)x^2$	
			$\vdots$	$\vdots$	$\ldots$

Si en la ecuación anterior sustituimos

$$
\begin{aligned}
b_5 &= a_5 \\
b_4 = a_4 - a_5r &= a_4 - rb_5 \\
b_3 = a_3 - sa_5 - (a_4 - a_5r)r &= a_3 - sb_5 - rb_4 \\
b_2 &= a_2 - sb_4 - rb_3 \\
b_1 &= a_1 - sb_3 - rb_2 \\
b_0 &= a_0 - sb_2 - rb_1
\end{aligned}
$$

En general para un polinomio de grado $f_n(x)$ tenemos que podemos hacer la división sintética aplicando la siguiente recurrencia

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} - rb_n \\ b_k &= a_k - rb_{k+1} - sb_{k+2} \\ \forall k &\in \{n-2, n-3, \cdots 1\} \end{aligned}$$

Finalmente el residuo de la división se calcula como

$$R = b_1(x + r) + b_0$$

La implementación en Matlab es:

```
function [c, r] = Division_Sintetica2(a, r, s)

n = length(a);

b=zeros(n,1);

b(n) = a(n);
b(n-1) = a(n-1) - r*b(n);

for k=n-2: -1:1
      b(k) = a(k) - r*b(k+1) - s*b(k+2);
end;

fprintf(1,'Residuo %6.4f x + %6.4f\n', b(2), b(1)+r*b(2));

c = b(3:n);
r = [b(2); b(1)+r*b(2)];
end
```

3.8.2. Ejemplo

Hacer la división sintética del polinomio $f_3(x) = x^3 - 1$, entre el polinomio $f_2(x) = x^2 + 3x + 4$

De acuerdo con la sucesión

$$\begin{aligned} b_3 &= a_3 = 1.0 \\ b_2 &= a_2 - rb_3 = 0 - (1)(3) = -3 \\ b_1 &= a_1 - rb_2 - sb_3 = 0 - (3)(-3) - (4)(1) = 5 \\ b_0 &= a_0 - rb_1 - sb_2 = -1 - (3)(5) - (4)(-3) = -4 \end{aligned}$$

El residuo de acuerdo con lo expuesto es

$$R = b_1(x + r) + b_0 = 5(x + 3) - 4 = 5x + 11$$

La corrida correspondiente en Matlab queda

```
>> [b, r] = Division_Sintetica2([-1, 0, 0, 1], 3, 4)
Residuo 5.0000 x + 11.0000

b =

    -3
     1


r =

    11
     5
```

3.8.3. Ejemplo

Hacer la división sintética del polinomio $f_5(x) = x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25$, entre el polinomio $f_2(x) = x^2 + x - 2$

De acuerdo con la sucesión

$$\begin{aligned} b_5 &= a_5 = 1.0 \\ b_4 &= a_4 - rb_5 = -3.5 - (1)(1) = -4.5 \\ b_3 &= a_3 - rb_4 - sb_5 = 2.75 - (1)(-4.5) - (-2)(1) = 2.75 + 6.5 = 9.25 \\ b_2 &= a_2 - rb_3 - sb_4 = 2.125 - (1)(9.25) - (-2)(-4.5) = -16.125 \\ b_1 &= a_1 - rb_2 - sb_3 = -3.875 - (1)(-16.125) - (-2)(9.25) = 30.75 \\ b_0 &= a_0 - rb_1 - sb_2 = 1.25 - (1)(30.75) - (-2)(-16.125) = -61.75 \end{aligned}$$

Nuestro residuo queda

$$R = b_1(x + r) + b_0 = 30.75 \times (x + 1) + (-61.75) = 30.75x - 31.00$$

La siguiente tabla muestra el resumen de los coeficientes calculados. Note que el residuo no es cero.

	1	-3.5	2.75	2.125	-3.875	1.25
$x^2 + x - 2$		-1	6.5	-18.25	34.625	-63.00
	1	-4.5	9.25	-16.125	30.75	-61.75

La ejecución en Matlab sería

```
[b, r] = Division_Sintetica2([1.25, -3.875, 2.125, 2.75, -3.5, 1], 1, -2)
Residuo 30.7500 x - 31.0000

b =

  -16.1250
    9.2500
   -4.5000
    1.0000


r =

  -31.0000
   30.7500
```

Lo cual significa que

$$\frac{x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25}{x^2 + x - 2} = x^3 - 4.5x^2 + 9.25x - 16.125$$

3.8.4. Ejemplo

Repetir el problema anterior pero suponiendo que $f_2(x) = x^2 - x - 2$

Pare este ejemplo tenemos la sucesión

$$
\begin{aligned}
b_5 &= a_5 = 1.0 \\
b_4 &= a_4 - rb_5 = -3.5 - (-1)(1) = -2.5 \\
b_3 &= a_3 - rb_4 - sb_5 = 2.75 - (-1)(-2.5) - (-2)(1) = 2.25 \\
b_2 &= a_2 - rb_3 - sb_4 = 2.125 - (-1)(2.25) - (-2)(-2.5) = -0.625 \\
b_1 &= a_1 - rb_2 - sb_3 = -3.875 - (-1)(-0.625) - (-2)(2.25) = 0 \\
b_0 &= a_0 - rb_1 - sb_2 = 1.25 - (-1)(0) - (-2)(-0.625) = 0
\end{aligned}
$$

	1	-3.5	2.75	2.125	-3.875	1.25
$x^2 - x - 2$		1	-0.5	-2.75	3.875	-1.25
	1	-2.5	2.25	-0.625	0	0

con un residuo

$$R = 0 \times (x - 1) + 0 = 0$$

La corrida en Matlab es

```
[b, r] = Division_Sintetica2([1.25, -3.875, 2.125, 2.75, -3.5, 1], -1, -2)
Residuo 0.0000 x + 0.0000

b =

   -0.6250
    2.2500
   -2.5000
    1.0000


r =

     0
     0
```

Note que los residuos son iguales a cero, por lo tanto

$$\frac{x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25}{x^2 - x - 2} = x^3 - 2.5x^2 + 2.25x - 0.625$$

3.9. Solución de un polinomio de orden 2 en la forma general

Un polinomio de segundo orden lo podemos expresar como

$$f_2(n) = a_0 + a_1 x + a_2 x^2$$

La solución utilizando la formula general tenemos que

$$x = \frac{-a_1 \pm \sqrt{a_1^2 - 4a_2 a_0}}{2a_2}$$

Para su implementación en Matlab hacemos

```
function [x1, x2] = Formula_General(a)
x1 = (-a(2)+ sqrt(a(2)^2 - 4*a(3)*a(1)))/(2*a(3));
x2 = (-a(2)- sqrt(a(2)^2 - 4*a(3)*a(1)))/(2*a(3));
```

3.10. Método de Bairstow para la solución de polinomios

El método de Bairstow es un método iterativo, basado en el método de Newton Raphson. Dado un polinonio $f_n(x)$ se encuentran dos factores, un polinomio cuadrático $f_2(x) = x^2 + rx + s$ y un polinomio $f_{n-2}(x)$. El procedimiento general para el método de Bairstow es:

1. Dado $f_n(x)$ y r_0 y s_0
2. Utilizando el método de NR calculamos $f_2(x) = x^2 + r_0 x + s_0$ y $f_{n-2}(x)$, tal que, el residuo de $fn(x)/f_2(x)$ sea igual a cero.
3. Se determinan la raíces $f_2(x)$, utilizando la formula general.
4. Se calcula $f_{n-2}(x) = f_n(x)/f_2(x)$
5. Hacemos $f_n(x) = f_{n-2}(x)$
6. Si el grado del polinomio es mayor que tres regresamos al paso 2
7. Si no terminamos

La principal diferencia de este método, respecto a otros, es que permite calcular todas las raíces de un polinomio (reales e imaginarias).

Para calcular la división de polinomios, hacemos uso de la división sintética. Así dado

$$f_n(x) = a_n x^n + a_{n-1}x^{n-1} + \ldots + a_2 x^2 + a_1 x + a_0$$

Al dividir entre $f_2(x) = x^2 – rx – s$, tenemos como resultado el siguiente polinomio

$$f_{n-2}(x) = b_n x^{n-2} + b_{n-1}x^{n-3} + \ldots + b_3 x + b_2$$

con un residuo $R = b_1(x + r) + b_0$, el residuo será cero solo si b_1 y b_0 lo son.

Los términos b, los calculamos utilizamos división sintética, la cual puede resolverse utilizando la siguiente relación de recurrencia

$$\begin{aligned} b_n &= a_n \\ b_{n-1} &= a_{n-1} - rb_n \\ b_i &= a_i - rb_{i+1} - sb_{i+2} \end{aligned}$$

Una manera de determinar los valores de r y s que hacen cero el residuo es utilizar el Método de Newton-Raphson. Para ello necesitamos una aproximación lineal de b_1 y b_0 respecto a r y s la cual calculamos utilizando la serie de Taylor

$$\begin{aligned} b_1(r_{k+1}, s_{k+1}) &= b_1(r_k, s_k) + \frac{\partial b_1(r_k, s_k)}{\partial r}(r_k - r_{k+1}) + \frac{\partial b_1(r_k, s_k)}{\partial s}(s_k - s_{k+1}) \\ b_0(r_{k+1}, s_{k+1}) &= b_0(r_k, s_k) + \frac{\partial b_0(r_k, s_k)}{\partial r}(r_k - r_{k+1}) + \frac{\partial b_0(r_k, s_k)}{\partial s}(s_k - s_{k+1}) \end{aligned}$$

donde los valores de r_k y s_k están dados y calculamos los nuevos valores r_{k+1} y s_{k+1} que hacen a $b_1(r_{k+1}, s_{k+1})$ y $b_0(r_{k+1}, s_{k+1})$ igual a cero. El sistema de ecuaciones que tenemos que resolver es:

$$\begin{aligned} 0 &= b_1(r_k, s_k) + \frac{\partial b_1(r_k, s_k)}{\partial r}(r_k - r_{k+1}) + \frac{\partial b_1(r_k, s_k)}{\partial s}(s_k - s_{k+1}) \\ 0 &= b_0(r_k, s_k) + \frac{\partial b_0(r_k, s_k)}{\partial r}(r_k - r_{k+1}) + \frac{\partial b_0(r_k, s_k)}{\partial s}(s_k - s_{k+1}) \end{aligned}$$

En forma matricial tenemos

$$\begin{bmatrix} \frac{\partial b_1(r_k,s_k)}{\partial r} & \frac{\partial b_1(r_k,s_k)}{\partial s} \\ \frac{\partial b_0(r_k,s_k)}{\partial r} & \frac{\partial b_0(r_k,s_k)}{\partial s} \end{bmatrix} \begin{bmatrix} dr_k \\ ds_k \end{bmatrix} = \begin{bmatrix} -b_1(r_k,s_k) \\ -b_0(r_k,s_k) \end{bmatrix}$$

donde $dr_k = r_k - r_{k+1}$ y $ds_k = s_k - s_{k+1}$

Bairtow muestra que los coeficientes c del polinomio de derivadas parciales $f'_n(x) = c_0 + c_1x + c_2x^2 + \cdots$ pueden obtenerse haciendo un procedimiento similar a la división sintética, así

$$\begin{aligned} c_n &= b_n \\ c_{n-1} &= b_{n-1} - rc_n \\ c_i &= b_i - rc_{i+1} - sc_{i+2} \end{aligned}$$

donde

$$\begin{aligned} \frac{\partial b_0(r_k,s_k)}{\partial r} &= c_1(r_k,s_k) \\ \frac{\partial b_1(r_k,s_k)}{\partial r} &= c_2(r_k,s_k) \\ \frac{\partial b_0(r_k,s_k)}{\partial s} &= c_2(r_k,s_k) \\ \frac{\partial b_1(r_k,s_k)}{\partial s} &= c_3(r_k,s_k) \end{aligned}$$

El código en Matlab para implementar la división sintética y calculo de los coeficientes del polinomio de derivadas es

```
function [b, c] = Division_Sintetica_Derivadas(a, r, s)

n = length(a);

b=zeros(n,1);
c=zeros(n,1);

b(n) = a(n);
b(n-1) = a(n-1) - r*b(n);
c(n) = b(n);
```

```
c(n-1) = b(n-1) - r*c(n);
for k=n-2: -1:1
      b(k) = a(k) - r*b(k+1) - s*b(k+2);
      c(k) = b(k) - r*c(k+1) - s*c(k+2);
end;
```

Finalmente el sistema de ecuaciones que debe resolverse es:

$$\begin{bmatrix} c_2(r_k, s_k) & c_3(r_k, s_k) \\ c_1(r_k, s_k) & c_2(r_k, s_k) \end{bmatrix} \begin{bmatrix} dr_k \\ ds_k \end{bmatrix} = \begin{bmatrix} -b_1(r_k, s_k) \\ -b_0(r_k, s_k) \end{bmatrix}$$

y cada una de las iteraciones de Newton Raphson

$$\begin{bmatrix} r_{k+1} \\ s_{k+1} \end{bmatrix} = \begin{bmatrix} r_k \\ s_k \end{bmatrix} - \begin{bmatrix} c_2(r_k, s_k) & c_3(r_k, s_k) \\ c_1(r_k, s_k) & c_2(r_k, s_k) \end{bmatrix}^{-1} \begin{bmatrix} -b_1(r_k, s_k) \\ -b_0(r_k, s_k) \end{bmatrix}$$

La codificación en Matlab para llevar a cabo este procedimiento es:

```
function [c, r, s] = Itera_Bairstow(a, r0, s0)

N = length(a);
res =[r0, s0]';

for k=1:100
[b, c] = Division_Sintetica_Derivadas(a, res(1), res(2));

A = [c(3) c(4); c(2) c(3)];
d = [-b(2); -b(1)];

res = res - inv(A)*d;
if sqrt(norm(d)) < 0.0001 break; end
end;

r = res(1);
s = res(2);
c = b(3:N);
```

3.10.1. Ejemplo 1

Dado el polinomio $f_5(x) = x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25$, determinar los valores de r y s que hacen el resido igual a cero. Considere $r_0 = 1$ y $s_0 = -2$.

Solución.

Iteración 1

La división sintética con el polinomio $f_2(x) = x^2 + x - 2.0$ da como resultado

$$f_3(x) = x^3 - 4.5x^2 + 9.25x - 16.125$$

$$Residuo = [30.75, -61.75]$$

El polinomio de derivadas es

$$c = [-257.6250, 108.1250, -43.8750, 16.7500, -5.5000, 1.0000]$$

Aplicando el método de Newton tenemos

$$\begin{bmatrix} r_1 \\ s_1 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \end{bmatrix} - \begin{bmatrix} -43.875 & 16.750 \\ 108.125 & -43.875 \end{bmatrix}^{-1} \begin{bmatrix} -30.7500 \\ 61.7500 \end{bmatrix} = \begin{bmatrix} -1.7637 \\ -7.4034 \end{bmatrix}$$

Iteración 2

La división sintética con el polinomio $f_2(x) = x^2 - 1.7637x - 7.4034$ da como resultado

$$f_3(x) = x^3 - 1.7363x^2 + 7.0911x - 1.7768$$

$$Residuo = [51.7564, 105.6858]$$

Los coeficiente del polinomio de derivadas es

$$c_2 = [677.3337, 208.1484, 27.6280, 14.5427, 0.0274, 1.0000]$$

Aplicando el método de Newton tenemos

$$\begin{bmatrix} r_2 \\ s_2 \end{bmatrix} = \begin{bmatrix} -1.7637 \\ -7.4034 \end{bmatrix} - \begin{bmatrix} 27.62800 & 14.5427 \\ 208.1484 & 27.6280 \end{bmatrix}^{-1} \begin{bmatrix} -51.75640 \\ -105.68578 \end{bmatrix} = \begin{bmatrix} -1.7164 \\ -3.9343 \end{bmatrix}$$

Iteración 3

La división sintética con el polinomio $f_2(x) = x^2 - 1.7164x - 3.9343$ da como resultado

$$f_3(x) = x^3 - 1.7836x^2 + 3.6229x + 1.3262$$

$$Residuo = [12.6547, 28.1881]$$

Los coeficientes del polinomio de derivadas son

$$c_3 = [195.3505, 65.6792, 13.8350, 7.4418, -0.0672, 1.0000]$$

Aplicando el método de Newton tenemos

$$\begin{bmatrix} r_3 \\ s_3 \end{bmatrix} = \begin{bmatrix} -1.7164 \\ -3.9343 \end{bmatrix} - \begin{bmatrix} 13.8350 & 7.4418 \\ 65.6792 & 13.8350 \end{bmatrix}^{-1} \begin{bmatrix} -12.6547 \\ -28.1881 \end{bmatrix} = \begin{bmatrix} -1.5997 \\ -2.4507 \end{bmatrix}$$

En resumen

k	r	s	b_1	b_0
0	1.0000	- 2.0000	30.7500	-61.7500
1	-1.7637	-7.4034	51.7564	105.6858
2	-1.7164	-3.9343	12.6547	28.1881
3	-1.5997	-2.4507	2.8996	8.1547
4	-1.3335	-2.1867	0.7601	2.5222
5	-1.1183	-2.1130	0.2719	0.6077
6	-1.0271	-2.0232	0.0431	0.1119
7	-1.0017	-2.0015	0.0028	0.0063
8	-1.0000	-2.0000	0.0000	0.0000
9	-1.0000	-2.0000	0.0000	0.0000
10	-1.0000	-2.0000	0.0000	0.0000

La solución es:

$$\begin{aligned} f_3(x) &= x^3 - 2.5x^2 + 2.25x - 0.625 \\ f_2(x) &= x^2 - x - 2 \end{aligned}$$

Las raíces de $f_2(x) = x^2 - x - 2$, son $x_1 = 2$ y $x_2 = -1$

Para replicar el ejemplo damos

```
[c r s] =Itera_Bairstow([1.25, -3.875, 2.125, 2.75, -3.5, 1], 1, -2)

c =

   -0.6250
    2.2500
   -2.5000
    1.0000


r =

   -1.0000


s =

   -2.0000
```

Si repetimos el ejemplo pero ahora considerando el polinomio $f_3(x) = x^3 - 2.5x^2 + 2.25x - 0.625$, podemos calcular el total de las raíces del polinomio original.

La implementación del Método de Bairstow es:

```
function raices = Bairstow (b, r0, s0)

raices = [];

N = length(b);

while(N > 3)
    [b r s] = Itera_Bairstow(b, r0, s0);
    [x1, x2] = Formula_General(1, r, s);
    raices = [raices, x1, x2];

    N = N-2;
end;

if length(b) == 3
    [x1, x2] = Formula_General(b(3), b(2), b(1));
    raices = [raices, x1, x2];
end;
```

```
if length(b) == 2
    raices = [raices, -b(1)/b(2)];
end;
```

3.10.2. Ejemplo 2

Dado el polinomio $f_5(x) = x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25$, calcular la raíces utilizando el Método de Bairstow con valores iniciales $r_0 = 1$ y $s_0 = -2$

Iteración 1

Calculamos la deflación Polinomial

$$f_5(x) = (-0.625 + 2.25x - 2.5x^2 + x^3)(x^2 - x - 2)$$

Las raíces calculadas son

$$\begin{aligned} x_1 &= 2.0000 \\ x_2 &= -1.0000 \end{aligned}$$

Iteración 2

$$f_3(x) = (-0.625 + 2.25x - 2.5x^2 + x^3) = (x - 0.5)(x^2 - 2x + 1.25)$$

Las raíces son

$$\begin{aligned} x_3 &= 1 + j0.5 \\ x_4 &= 1 - j0.5 \end{aligned}$$

Iteración 3

Finalmente

$$f_1(x) = x - 0.5$$

tiene una raiz

$$x_5 = 0.5$$

Para correr en Matlab hacer

```
Bairstow([1.25, -3.875, 2.125, 2.75, -3.5, 1], 1, -2)

ans =

   2.0000            -1.0000             1.0000 + 0.5000i   1.0000 - 0.5000i   0.5000
```

3.10.3. Ejemplo 3

Dado el polinomio $f_5(x) = x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25$, determinar las raíces de este polinomio. Considere $r_0 = -1$ y $s_0 = -1$.

Iteración 1

$$\begin{aligned} f_5(x) &= x^5 - 3.5x^4 + 2.75x^3 + 2.125x^2 - 3.875x + 1.25 \\ f_5(x) &= (x^3 - 4x^2 + 5.25x - 2.5) * (x^2 + 0.5x - 0.5) \end{aligned}$$

Las raíces de $x^2 + 0.5x - 0.5 = 0$ son

$$\begin{aligned} x_1 &= 0.5 \\ x_2 &= -1.0 \end{aligned}$$

Iteración 2

$$\begin{aligned} f_3(x) &= x^3 - 4x^2 + 5.25x - 2.5 \\ f_3(x) &= (x - 2) * (x^2 - 2x + 1.25) \end{aligned}$$

Las raíces de $x^2 - 2x + 1.25 = 0$ son

$$\begin{aligned} x_3 &= 1.0 + j0.5 \\ x_4 &= -1.0 - j0.5 \end{aligned}$$

Iteración 3

$$f_1(x) = (x - 2)$$

La raíz de este polinomio es

$$x_5 = 2;$$

Todas la raíces de $f_5(x)$ son $x = [0.5, -1.0, (1.0 + j0.5), (1 - j0.5), 2]$. Para correr en Matlab dar

```
>> Bairstow([1.25, -3.875, 2.125, 2.75, -3.5, 1], -1, -1)

ans =

   2.0000     -1.0000        1.0000 + 0.5000i   1.0000 - 0.5000i   0.5000
```

Solución de sistemas de ecuaciones lineales y no-lineales

4.1. Solución de Sistemas lineales

Se llama sistema de ecuaciones lineales a un conjunto de ecuaciones de la forma:

donde x_1, ..., x_n son las incógnitas, b_1, ..., b_n se denominan términos independientes y los números $a_{i,j}$ se llaman coeficientes de las incógnitas, formando una matriz que denominaremos A, matriz de coeficientes.

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

En forma compacta

$$Ax = b$$

En las siguientes subsecciones se analizarán algunos de los métodos para resolver el sistema de ecuaciones.

4.1.1. Método iterativo de Jacobi

Consideremos el siguiente sistema de ecuaciones.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Vamos a representar cada una de las variables en términos de ellas mismas.

$$\begin{aligned} x_1 &= \frac{b_1 - a_{1,2}x_2 - a_{1,3}x_3}{a_{1,1}} \\ x_2 &= \frac{b_2 - a_{2,1}x_1 - a_{2,3}x_3}{a_{2,2}} \\ x_3 &= \frac{b_3 - a_{3,1}x_1 - a_{3,2}x_2}{a_{3,3}} \end{aligned}$$

Lo cual nos sugiere el siguiente esquema iterativo de solución.

$$\begin{aligned} x_1^{(t+1)} &= \frac{b_1 - a_{1,2}x_2^{(t)} - a_{1,3}x_3^{(t)}}{a_{1,1}} \\ x_2^{(t+1)} &= \frac{b_2 - a_{2,1}x_1^{(t)} - a_{2,3}x_3^{(t)}}{a_{2,2}} \\ x_3^{(t+1)} &= \frac{b_3 - a_{3,1}x_1^{(t)} - a_{3,2}x_2^{(t)}}{a_{3,3}} \end{aligned}$$

En general podemos escribir como

$$x_k^{(t+1)} = \frac{b_k - \sum_{l=1, l\neq k}^{N} a_{k,l}x_l^{(t)}}{a_{k,k}}$$

La implementación en Matlab es

```
function y = Jacobi(A, x, b)

N = length(x);
y = zeros(N,1);

for iter=1:100000
    for k = 1:N
        suma =0;
        for l= 1:N
            if k ~= l
```

```
                suma = suma + A(k,l)*x(l);
            end;
        end;
        y(k) = (b(k) - suma)/A(k,k);
    end;
    if sqrt(norm(x-y)) < 1e-6
        break;
    else
       x = y;
    end;
end;
```

4.1.2. Algoritmo iterativo de Gauss-Seidel

El cambio que debemos hacer respecto al de Jacobi, es que las variables nuevas son utilizadas una vez que se realiza el cálculo de ellas así, para un sistema de tres ecuaciones tendremos:

$$\begin{aligned} x_1^{(t+1)} &= \frac{b_1 - a_{1,2}x_2^{(t)} - a_{1,3}x_3^{(t)}}{a_{1,1}} \\ x_2^{(t+1)} &= \frac{b_2 - a_{2,1}x_1^{(t+1)} - a_{2,3}x_3^{(t)}}{a_{2,2}} \\ x_3^{(t+1)} &= \frac{b_3 - a_{3,1}x_1^{(t+1)} - a_{3,2}x_2^{(t+1)}}{a_{3,3}} \end{aligned}$$

La implementación en Matlab es

```
function y = Gauss_Seidel(A, x, b)

N = length(x);
y = zeros(N,1);

for iter=1:100000
    for k = 1:N
        suma =0;
        for l= 1:N
            if k ~= l
                suma = suma + A(k,l)*x(l);
            end;
```

```
        end;
        x(k) = (b(k) - suma)/A(k,k);
    end;
    if sqrt(norm(x-y)) < 1e-6
        break;
    else
        y=x;
    end;
end;
```

Ejemplo

Resolver el siguiente sistema de ecuaciones utilizando el método de Jacobi y comparar con el método de Gauss-Seidel.

$$\begin{bmatrix} 4 & -1 & 1 \\ 4 & -8 & 1 \\ -2 & 1 & 5 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 7 \\ -21 \\ 15 \end{bmatrix}$$

Las primeras 20 iteraciones del algoritmo de Jacobi son :

k	x_1	x_2	x_3
1	1.0000	2.0000	2.0000
2	1.7500	3.3750	3.0000
3	1.8438	3.8750	3.0250
4	1.9625	3.9250	2.9625
5	1.9906	3.9766	3.0000
6	1.9941	3.9953	3.0009
7	1.9986	3.9972	2.9986
8	1.9996	3.9991	3.0000
9	1.9998	3.9998	3.0000
10	1.9999	3.9999	2.9999
11	2.0000	4.0000	3.0000
12	2.0000	4.0000	3.0000

Para correr hacer

```
>> Jacobi([4 -1 1; 4 -8 1; -2 1 5], [1,2,2]', [7,-21, 15]')

ans =

    2.0000
```

```
    4.0000
    3.0000
```

La solución utilizando Gauss-Seidel es :

k	x_1	x_2	x_3
1	1.0000	2.0000	2.0000
2	1.7500	3.7500	2.9500
3	1.9500	3.9688	2.9863
4	1.9956	3.9961	2.9990
5	1.9993	3.9995	2.9998
6	1.9999	3.9999	3.0000
7	2.0000	4.0000	3.0000
8	2.0000	4.0000	3.0000

Note que Gauss-Seidel requiere de 7 iteraciones mientras Jacobi de 11, para convergir. Para correr hacer

```
>> Gauss_Seidel([4 -1 1; 4 -8 1; -2 1 5], [1,2,2]', [7,-21, 15]')

ans =

    2.0000
    4.0000
    3.0000
```

4.1.3. Ejemplo matrices dispersas

Resolver el sistema de ecuaciones utilizando matrices dispersas y los métodos de Jacobi y Gauss-Seidel

$$A = \begin{bmatrix} 10 & -1 & -1 & -1 & 1 \\ 1 & 4 & 0 & 0 & 0 \\ 1 & 0 & 4 & 0 & 0 \\ 1 & 0 & 0 & 4 & 0 \\ 1 & 0 & 0 & 0 & 3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \\ x_5 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{bmatrix}$$

```
A = sparse(5,5);
A(1,1) = 10;
A(1,2) = -1;
A(1,3) = -1;
A(1,4) = -1;
A(1,5) = 1;
```

```
A(2,1) = 1;
A(3,1) = 1;
A(4,1) = 1;
A(5,1) = 1;
A(2,2) = 4;
A(3,3) = 4;
A(4,4) = 4;
A(5,5) = 3;
disp(A);

size(A)

b = [1,2,3,4,5]';

disp(b);
Jacobi(A,[0,0,0,0,0]', b)
Gauss_Seidel(A,[0,0,0,0,0]', b)
```

La solución para ambos métodos es:

```
ans =

    0.1520
    0.4620
    0.7120
    0.9620
    1.6160


ans =

    0.1520
    0.4620
    0.7120
    0.9620
    1.6160
```

4.1.4. Solución de sistemas lineales de ecuaciones por el método de Eliminación Gaussiana

Consideremos que tenemos un sistema lineal Ax=b, donde la matriz A no tiene las condiciones de ser triangular superior.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Comenzaremos por despejar x_1 de la ecuación (I)

$$x_1 = \frac{b_1 - a_{1,2}x_2 - a_{1,3}x_3}{a_{1,1}}$$

y sustituimos en las ecuaciones (II)

$$a_{2,1}\left(\frac{b_1 - a_{1,2}x_2 - a_{1,3}x_3}{a_{1,1}}\right) + a_{2,2}x_2 + a_{2,3}x_3 = b_2$$

agrupando términos semejantes tenemos:

$$\left(a_{2,2} - \frac{a_{2,1} * a_{1,2}}{a_{1,1}}\right)x_2 + \left(a_{2,3} - a_{2,1}\frac{a_{1,3}}{a_{1,1}}\right)x_3 = \left(b_2 - a_{2,1}\frac{b_1}{a_{1,1}}\right)$$

Ahora sustituimos en la ecuación (III)

$$\left(a_{3,1}\frac{b_1 - a_{1,2}x_2 - a_{1,3}x_3}{a_{1,1}}\right) + a_{3,2}x_2 + a_{3,3}x_3 = b_3$$

$$\left(a_{3,2} - \frac{a_{3,1}a_{1,2}}{a_{1,1}}\right)x_2 + \left(a_{3,3} - \frac{a_{3,1}a_{1,3}}{a_{1,1}}\right)x_3 = \left(b_3 - a_{3,1}\frac{b_1}{a_{1,1}}\right)$$

Lo cual nos da un nuevo sistema simplificado dada por

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ 0 & a'_{2,2} & a'_{2,3} \\ 0 & a'_{3,2} & a'_{3,3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b'_2 \\ b'_3 \end{bmatrix}$$

donde los valores de $a'_{i,j}$ los calculamos como:

$$\begin{aligned} a'_{i,j} &= a_{i,j} - a_{i,k}\frac{a_{k,j}}{a_{k,k}} \\ b'_i &= b_i - a_{i,k}\frac{b_k}{a_{k,k}} \end{aligned}$$

Si repetimos el procedimiento, ahora para x_1, tendremos un sistema dado como:

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ 0 & a'_{2,2} & a'_{2,3} \\ 0 & 0 & a''_{3,3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} b_1 \\ b'_2 \\ b''_3 \end{bmatrix}$$

Solución de un sistema Triangular superior

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ 0 & a_{2,2} & a_{2,3} & a_{2,4} \\ 0 & 0 & a_{3,3} & a_{3,4} \\ 0 & 0 & 0 & a_{4,4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

Podemos resolverlo empezando con la ecuación 4 cuya solución es la más simple, para luego solucionar 3 y así sucesivamente. Este esquema de solución queda

$$\begin{aligned} x_4 &= \frac{b_4}{a_{4,4}} \\ x_3 &= \frac{b_3 - a_{3,4}x_4}{a_{3,3}} \\ x_2 &= \frac{b_2 - a_{2,3}x_3 - a_{2,4}x_4}{a_{2,2}} \\ x_1 &= \frac{b_1 - a_{1,2}x_2 - a_{1,3}x_3 - a_{1,4}x_4}{a_{1,1}} \end{aligned}$$

En general podemos calcular la solución haciendo

$$x_k = \frac{b_k - \sum_{i=k+1}^{N} a_{k,i}x_i}{a_{k,k}}$$

La implementación en Matlab queda

```
function x = Eliminacion_Gaussiana(A, b)

N = length(b);

x = zeros(N,1);

for k =1:1:N-1
    for n = k+1:1:N
        b(n) = b(n) - A(n,k)*b(k)/A(k,k);
        for m=N:-1:k
            A(n,m) = A(n,m) - A(n,k)*A(k,m)/A(k,k);
        end;
    end;
end;

for k=N:-1:1
    suma = 0;
    for m=k+1:1:N
        suma = suma +A(k,m)*x(m);
    end;
    x(k) = (b(k)-suma)/A(k,k);
end;
```

que corresponde a un sistema triangular superior que podemos solucionar utilizando sustitución hacia atrás.

Ejemplo 1

Dado el sistema lineal de ecuaciones, calcular el sistema Triangular Superior utilizando el algoritmo de Eliminación Gaussiana

$$\begin{bmatrix} 10 & -1 & 2 & 1 \\ -1 & 15 & -3 & 1 \\ 2 & -3 & 6 & -3 \\ 1 & 1 & -3 & 7 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 2 \\ 1 \end{bmatrix}$$

Primer paso $k = 1$

$$\begin{pmatrix} 10 & -1 & 2 & 1 \\ 0 & \frac{149}{10} & -\frac{14}{5} & \frac{11}{10} \\ 0 & -\frac{14}{5} & \frac{28}{5} & -\frac{16}{5} \\ 0 & \frac{11}{10} & -\frac{16}{5} & \frac{69}{10} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{21}{10} \\ \frac{9}{5} \\ \frac{9}{10} \end{pmatrix}$$

$$\begin{bmatrix} 10.0000 & -1.0000 & 2.0000 & 1.0000 \\ 0 & 14.9000 & -2.8000 & 1.1000 \\ 0 & -2.8000 & 5.6000 & -3.2000 \\ 0 & 1.1000 & -3.2000 & 6.9000 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1.0000 \\ 2.1000 \\ 1.8000 \\ 0.9000 \end{bmatrix}$$

Segundo paso $k = 2$

$$\begin{pmatrix} 10 & -1 & 2 & 1 \\ 0 & \frac{149}{10} & -\frac{14}{5} & \frac{11}{10} \\ 0 & 0 & \frac{756}{149} & -\frac{446}{149} \\ 0 & 0 & -\frac{446}{149} & \frac{1016}{149} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{21}{10} \\ \frac{327}{149} \\ \frac{111}{149} \end{pmatrix}$$

$$\begin{bmatrix} 10.0000 & -1.0000 & 2.0000 & 1.0000 \\ 0 & 14.9000 & -2.8000 & 1.1000 \\ 0 & 0 & 5.0738 & -2.9933 \\ 0 & 0 & -2.9933 & 6.8188 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1.0000 \\ 2.1000 \\ 2.1946 \\ 0.7450 \end{bmatrix}$$

Tercer paso $k = 3$

$$\begin{pmatrix} 10 & -1 & 2 & 1 \\ 0 & \frac{149}{10} & -\frac{14}{5} & \frac{11}{10} \\ 0 & 0 & \frac{756}{149} & -\frac{446}{149} \\ 0 & 0 & 0 & \frac{955}{189} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 1 \\ \frac{21}{10} \\ \frac{327}{149} \\ \frac{257}{126} \end{pmatrix}$$

$$\begin{bmatrix} 10.0000 & -1.0000 & 2.0000 & 1.0000 \\ 0 & 14.9000 & -2.8000 & 1.1000 \\ 0 & 0 & 5.0738 & -2.9933 \\ 0 & 0 & 0 & 5.0529 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 1.0000 \\ 2.1000 \\ 2.1946 \\ 2.0397 \end{bmatrix}$$

Sustitución hacia atras

Paso 1 $k = 4$

$$\begin{aligned} x_4 &= \frac{b_4}{a_{4,4}} \\ x_4 &= \frac{2.0397}{5.0529} = 0.4037 \end{aligned}$$

Paso 2 $k = 3$

$$\begin{aligned} x_3 &= \frac{b_3 - a_{3,4}x_4}{a_{3,3}} \\ x_3 &= \frac{2.1946 - (-2.9933)(0.4037)}{5.0738} = 0.6707 \end{aligned}$$

Paso 3 $k = 2$

$$\begin{aligned} x_2 &= \frac{b_2 - a_{2,3}x_3 - a_{2,4}x_4}{a_{2,2}} \\ x_2 &= \frac{2.1000 - (-2.8000)(0.6707) - (1.1000)(0.4037)}{14.9000} = 0.2372 \end{aligned}$$

Paso 4 $k = 1$

$$\begin{aligned} x_1 &= \frac{b_1 - a_{1,2}x_2 - a_{1,3}x_3 - a_{1,4}x_4}{a_{1,1}} \\ x_1 &= \frac{1.0000 - (-1.0000)(0.2372) - (2.0000)(0.6707) - (1.0000)(0.4037)}{10.0000} = -0.0508 \end{aligned}$$

La solución en Matlab es

```
>> Eliminacion_Gaussiana([10 -1 2 1; -1 15 -3 1; 2 -3 6 -3; 1 1 -3 7], [1,2,2,1]')

   10.0000   -1.0000    2.0000    1.0000
```

```
         0   14.9000   -2.8000    1.1000
         0         0    5.0738   -2.9933
         0         0         0    5.0529

    1.0000    2.1000    2.1946    2.0397

ans =

   -0.0508
    0.2372
    0.6707
    0.4037
```

Ejemplo 2

Calcular el sistema triangular superior utilizando eliminación Gaussiana.

$$5x + 2y + 1z = 3$$

$$2x + 3y - 3z = -10$$

$$1x - 3y + 2z = 4$$

Primer paso

$$5x + 2y + 1z = 3$$

$$0 + (11/5)y - (17/5)z = -(56/5)$$

$$0 - (17/5)y + (9/5)z = (17/5)$$

Segundo paso

$$5x + 2y + 1z = 3$$

$$0x + (11/5)y - (17/5)z = -(56/5)$$

$$0x - 0y - (38/11)z = -(153/11)$$

La solución en Matlab es

```
>> Eliminacion_Gaussiana([5 2 1; 2 3 -3; 1 -3 2], [3, -10, 4])
    5.0000    2.0000    1.0000
         0    2.2000   -3.4000
         0         0   -3.4545

    3.0000
  -11.2000
  -13.9091


ans =

   -0.6579
    1.1316
    4.0263
```

Ejemplo 3

Resolver el sistema de ecuaciones

$$\begin{bmatrix} 3 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 1 \end{bmatrix}$$

Aplicando Eliminación Gaussiana nos queda.

$$\begin{bmatrix} 3 & -1 & -1 \\ 0 & 2/3 & -1/3 \\ 0 & 0 & 1/2 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 3/2 \end{bmatrix}$$

La solución en Matlab es

```
>> Eliminacion_Gaussiana([3 -1 -1; -1 1 0; -1 0 1], [0 1 1])
    3.0000   -1.0000   -1.0000
         0    0.6667   -0.3333
         0         0    0.5000

         0
    1.0000
    1.5000
```

```
ans =

    2.0000
    3.0000
    3.0000
```

Problema con la Eliminación Gaussiana

Un ejemplo de sistema donde es necesario hacer un cambio de renglón por renglón para que tenga solución es el siguiente sistema.

$$\begin{bmatrix} 1 & 2 & 6 \\ 4 & 8 & -1 \\ -2 & 3 & 5 \end{bmatrix}$$

Aplicando el primer paso de la eliminación gaussiana tenemos:

$$\begin{bmatrix} 1 & 2 & 6 \\ 0 & 0 & -25 \\ 0 & 7 & 17 \end{bmatrix}$$

note, que aparece un cero en el elemento 22, lo cual nos da un sistema sin solución. Permutando los renglones 2 y 3 el sistema tiene solución.

$$\begin{bmatrix} 1 & 2 & 6 \\ -2 & 3 & 5 \\ 4 & 8 & -1 \end{bmatrix}$$

La solución en Matlab tenemos

```
>> Eliminacion_Gaussiana([1 2 6; 4 8 -1; -2 3 5], [1 1 1])
     1     2     6
     0     0   -25
     0   NaN   Inf

     1
    -3
   Inf
```

```
ans =

   NaN
   NaN
   NaN
```

Haciendo el cambio de renglones tenemos

```
>> Eliminacion_Gaussiana([1 2 6;  -2 3 5; 4 8 -1], [1 1 1])
     1     2     6
     0     7    17
     0     0   -25

     1
     3
    -3


ans =

    0.0057
    0.1371
    0.1200
```

4.1.5. Gauss-Jordan

El método de Gauss-Jordan es una variación de la eliminación de Gauss. La principal diferencia consiste em que cuando una incógnita se elimina en el método de Gauss-Jordan, esta es eliminada de todas las ecuaciones en lugar de hacerlo en la subsecuentes. Además todos los renglones se normalizan al dividirlos entre su elemento privote. De esta forma, el paso de eliminación genera una matriz identidad en vez de una matriz triangular. En consecuencia no es necesario usar la sustitución hacia atrás para obtener la solución.

Comenzaremos planteando un sistema de 3 ecuaciones con 3 incógnitas dado por

$$\left[\begin{array}{c|c|c} a_{1,1} & a_{1,2} & a_{1,3} \\ \hline a_{2,1} & a_{2,2} & a_{2,3} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right] = \left[\begin{array}{c} b_1 \\ b_2 \\ b_3 \end{array}\right]$$

Este sistema de ecuaciones lo podemos escribir como:

$$Ax = Ib$$

donde I es la matriz identidad

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Lo cual da lugar a la matriz aumentada que hemos utilizado en el procedimiento de Gauss-Jordan

$$\left[\begin{array}{ccc|ccc|c} a_{1,1} & a_{1,2} & a_{1,3} & 1 & 0 & 0 & b_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 & 1 & 0 & b_2 \\ a_{3,1} & a_{3,2} & a_{3,3} & 0 & 0 & 1 & b_3 \end{array}\right] \tag{4.4}$$

Primer paso

Vamos a despejar la variable x_1 del sistema y la sustituimos en las otras dos ecuaciones

$$\frac{b_1 - a_{1,2}x_2 - a_{1,3}x_3}{a_{1,1}} = x_1 \tag{4.5}$$

Sustituyendo en la ecuación 2

$$a_{2,1}\left(\frac{b_1 - a_{1,2}x_2 - a_{1,3}x_3}{a_{1,1}}\right) + a_{2,2}x_2 + a_{2,3}x_3 = b_2$$

$$0x_1 + \left(\frac{a_{2,1}}{a_{1,1}}\right)b_1 + \left(a_{2,2} - \frac{a_{2,1}a_{1,2}}{a_{1,1}}\right)x_2 + \left(a_{2,3} - \frac{a_{2,1}a_{1,3}}{a_{1,1}}\right)x_3 = b_2 \tag{4.6}$$

Sustituyendo en la ecuación 3

$$a_{3,1}\left(\frac{b_1 - a_{1,2}x_2 - a_{1,3}x_3}{a_{1,1}}\right) + a_{3,2}x_2 + a_{3,3}x_3 = b_3$$

$$0x_1 + \left(\frac{a_{3,1}}{a_{1,1}}\right)b_1 + \left(a_{3,2} - \frac{a_{3,1}a_{1,2}}{a_{1,1}}\right)x_2 + \left(a_{3,3} - \frac{a_{3,1}a_{1,3}}{a_{1,1}}\right)x_3 = b_3 \tag{4.7}$$

En forma matricial podemos escribir las ecuaciones 4.5, 4.6 y 4.7

$$\left[\begin{array}{c|c|c} 1 & \frac{a_{1,2}}{a_{1,1}} & \frac{a_{1,3}}{a_{1,1}} \\ \hline 0 & \left(a_{2,2}-\frac{a_{2,1}a_{1,2}}{a_{1,1}}\right) & \left(a_{2,3}-\frac{a_{2,1}a_{1,3}}{a_{1,1}}\right) \\ \hline 0 & \left(a_{3,2}-\frac{a_{3,1}a_{1,2}}{a_{1,1}}\right) & \left(a_{3,3}-\frac{a_{3,1}a_{1,3}}{a_{1,1}}\right) \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right] = \left[\begin{array}{c|c|c} \frac{1}{a_{1,1}} & 0 & 0 \\ \hline -\frac{a_{2,1}}{a_{1,1}} & 1 & 0 \\ \hline -\frac{a_{3,1}}{a_{1,1}} & 0 & 1 \end{array}\right] \left[\begin{array}{c} b_1 \\ b_2 \\ b_3 \end{array}\right]$$

La matriz aumentada dada por (4.4) tenemos:

$$\left[\begin{array}{c|c|c||c|c|c} 1 & \frac{a_{1,2}}{a_{1,1}} & \frac{a_{1,3}}{a_{1,1}} & \frac{1}{a_{1,1}} & 0 & 0 & \frac{1}{a_{1,1}}b_1 \\ \hline 0 & \left(a_{2,2}-\frac{a_{2,1}a_{1,2}}{a_{1,1}}\right) & \left(a_{2,3}-\frac{a_{2,1}a_{1,3}}{a_{1,1}}\right) & -\frac{a_{2,1}}{a_{1,1}} & 1 & 0 & b_2-\frac{a_{2,1}}{a_{1,1}}b_1 \\ \hline 0 & \left(a_{3,2}-\frac{a_{3,1}a_{1,2}}{a_{1,1}}\right) & \left(a_{3,3}-\frac{a_{3,1}a_{1,3}}{a_{1,1}}\right) & -\frac{a_{3,1}}{a_{1,1}} & 0 & 1 & b_3-\frac{a_{3,1}}{a_{1,1}}b_1 \end{array}\right]$$

Segundo Paso

Para un sistema equivalente

$$\left[\begin{array}{c|c|c} 1 & a'_{1,2} & a'_{1,3} \\ \hline 0 & a'_{2,2} & a'_{2,3} \\ \hline 0 & a'_{3,2} & a'_{3,3} \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right] = \left[\begin{array}{c|c|c} c'_{1,1} & 0 & 0 \\ \hline c'_{2,1} & 1 & 0 \\ \hline c'_{3,1} & 0 & 1 \end{array}\right] \left[\begin{array}{c} b'_1 \\ b'_2 \\ b'_3 \end{array}\right] \tag{4.8}$$

Despejamos x_2 de la ecuación 2

$$x_2 = \frac{c'_{2,1}b'_1 + b'_2 - a'_{2,3}x_3}{a'_{2,2}}$$

Sustituyendo en la ecuación 1

$$x1 + a'_{1,2}\left(\frac{c'_{2,1}b'_1 + b'_2 - a'_{2,3}x_3}{a'_{2,2}}\right) + a'_{1,3} = c'_{1,1}b'_1$$

$$x1 + 0x_2 + \left(a'_{1,3} - \frac{a'_{1,2}a'_{2,3}}{a'_{2,2}}x_3\right) = \left(c'_{1,1}b'_1 - \frac{a'_{1,2}c'_{2,1}}{a'_{2,2}}\right) - \frac{a'_{2,1}}{a'_{2,2}}b'_2$$

Sustituyendo en la ecuación 3

$$0x_1 + 0x_2 + \left(a'_{3,3} - \frac{a'_{3,2}a'_{2,3}}{a'_{2,2}}\right)x_3 = \left(c'_{3,1} - \frac{a'_{3,2}c'_{2,1}}{a'_{2,2}}\right)b'_1 - \frac{a'_{3,2}}{a'_{2,2}}b'2 + b'3$$

En forma matricial

$$\left[\begin{array}{c|c|c} 1 & 0 & a'_{1,3} - \frac{a'_{1,2}a'_{2,3}}{a'_{2,2}} \\ \hline 0 & 1 & \frac{a'_{2,3}}{a'_{2,2}} \\ \hline 0 & 0 & a'_{3,3} - \frac{a'_{3,2}a'_{2,3}}{a'_{2,2}} \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right] = \left[\begin{array}{c|c|c} c'_{1,1} - \frac{a'_{1,2}c'_{2,1}}{a'_{2,2}} & \frac{a'_{1,2}}{a'_{2,2}} & 0 \\ \hline \frac{c'_{2,1}}{a'_{2,2}} & \frac{1}{a'_{2,2}} & 0 \\ \hline c'_{3,1} - \frac{a'_{3,2}c'_{2,1}}{a'_{2,2}} & -\frac{a'_{3,2}}{a'_{2,2}} & 1 \end{array}\right] \left[\begin{array}{c} b'_1 \\ b'_2 \\ b'_3 \end{array}\right]$$

Tercer Paso

Dado el sistema equivalente

$$\left[\begin{array}{c|c|c} 1 & 0 & a''_{1,3} \\ \hline 0 & 1 & a''_{2,3} \\ \hline 0 & 0 & a''_{3,3} \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right] = \left[\begin{array}{c|c|c} c''_{1,1} & c''_{1,2} & 0 \\ \hline c''_{2,1} & c''_{2,2} & 0 \\ \hline c''_{3,1} & c''_{3,2} & 1 \end{array}\right] \left[\begin{array}{c} b''_1 \\ b''_2 \\ b''_3 \end{array}\right]$$

Despejamos x_3 de la ecuación 3 y sustituimos en la ecuaciones 1 y 2

$$\left[\begin{array}{c|c|c} 1 & 0 & 0 \\ \hline 0 & 1 & 0 \\ \hline 0 & 0 & 1 \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right] = \left[\begin{array}{c|c|c} c''_{1,1} & c''_{1,2} - \frac{a''_{1,3}c''_{3,2}}{a''_{3,3}} & -\frac{a''_{1,3}}{a''_{3,3}} \\ \hline c''_{2,1} & c''_{2,2} - \frac{a''_{2,3}}{a''_{3,3}} & -\frac{a''_{2,3}}{a''_{3,3}} \\ \hline \frac{c''_{3,1}}{a''_{3,3}} & \frac{c''_{3,2}}{a''_{3,3}} & \frac{1}{a''_{3,3}} \end{array}\right] \left[\begin{array}{c} b''_1 \\ b''_2 \\ b''_3 \end{array}\right]$$

Como resultado tenemos que la matrix c'' es la inversa de nuestro sistema y b'' la solución del sistema de ecuaciones

Resumen

Dado el sistema

$$\left[\begin{array}{ccc|c|c} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,M} \\ a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,M} \\ a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,M} \end{array}\right]$$

Paso 1

Dividimos la primer ecuación entre $a_{1,1}$

$$\frac{1}{a_{1,1}} \left[\begin{array}{c|c|c||c||c} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,M} \\ \hline a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,M} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,M} \end{array}\right]$$

Multiplicamos la ecuación 1 por $a_{2,1}$ y la restamos a la ecuación 2

$$a_{2,1}\left[\begin{array}{c|c|c||c||c} 1 & \frac{a_{1,2}}{a_{1,1}} & \frac{a_{1,3}}{a_{1,1}} & \cdots & \frac{a_{1,M}}{a_{1,1}} \\ \hline a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,M} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,M} \end{array}\right]$$

$$\left[\begin{array}{c|c|c||c||c} 1 & \frac{a_{1,2}}{a_{1,1}} & \frac{a_{1,3}}{a_{1,1}} & \cdots & \frac{a_{1,M}}{a_{1,1}} \\ \hline 0 & a_{2,2}-a_{2,1}\left(\frac{a_{1,2}}{a_{1,1}}\right) & a_{2,3}-a_{2,1}\left(\frac{a_{1,3}}{a_{1,1}}\right) & \cdots & a_{2,M}-a_{2,1}\left(\frac{a_{1,M}}{a_{1,1}}\right) \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,M} \end{array}\right]$$

Multiplicamos la ecuación 1 por $a_{3,1}$ y la restamos a la ecuación 3

$$a_{3,1}\left[\begin{array}{c|c|c||c||c} 1 & \frac{a_{1,2}}{a_{1,1}} & \frac{a_{1,3}}{a_{1,1}} & \cdots & \frac{a_{1,M}}{a_{1,1}} \\ \hline 0 & a_{2,2}-a_{2,1}\left(\frac{a_{1,2}}{a_{1,1}}\right) & a_{2,3}-a_{2,1}\left(\frac{a_{1,3}}{a_{1,1}}\right) & \cdots & a_{2,M}-a_{2,1}\left(\frac{a_{1,M}}{a_{1,1}}\right) \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,M} \end{array}\right]$$

$$\left[\begin{array}{c|c|c||c||c} 1 & \frac{a_{1,2}}{a_{1,1}} & \frac{a_{1,3}}{a_{1,1}} & \cdots & \frac{a_{1,M}}{a_{1,1}} \\ \hline 0 & a_{2,2}-a_{2,1}\left(\frac{a_{1,2}}{a_{1,1}}\right) & a_{2,3}-a_{2,1}\left(\frac{a_{1,3}}{a_{1,1}}\right) & \cdots & a_{2,M}-a_{2,1}\left(\frac{a_{1,M}}{a_{1,1}}\right) \\ \hline 0 & a_{3,2}-a_{3,1}\left(\frac{a_{1,2}}{a_{1,1}}\right) & a_{3,3}-a_{3,1}\left(\frac{a_{1,3}}{a_{1,1}}\right) & \cdots & a_{3,M}-a_{3,1}\left(\frac{a_{1,M}}{a_{1,1}}\right) \end{array}\right]$$

Paso 2

Dividimos la segunda ecuación entre $a_{2,2}$

$$\frac{1}{a_{2,2}}\left[\begin{array}{c|c|c||c||c} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,M} \\ \hline a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,M} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,M} \end{array}\right]$$

$$\left[\begin{array}{c|c|c||c||c} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,M} \\ \hline \frac{a_{2,1}}{a_{2,2}} & 1 & \frac{a_{2,3}}{a_{2,2}} & \cdots & \frac{a_{2,M}}{a_{2,2}} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,M} \end{array}\right]$$

Multiplicamos la ecuación 2 por $a_{1,2}$ y la restamos a la primer ecuación

$$a_{1,2}\left[\begin{array}{c|c|c||c||c} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,M} \\ \hline \frac{a_{2,1}}{a_{2,2}} & 1 & \frac{a_{2,3}}{a_{2,2}} & \cdots & \frac{a_{2,M}}{a_{2,2}} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,M} \end{array}\right]$$

$$\left[\begin{array}{c|c|c||c||c} a_{1,1}-a_{1,2}\left(\frac{a_{2,1}}{a_{2,2}}\right) & 0 & a_{1,3}-a_{1,2}\left(\frac{a_{2,3}}{a_{2,2}}\right) & \cdots & a_{1,M}-a_{1,2}\left(\frac{a_{2,M}}{a_{2,2}}\right) \\ \hline \frac{a_{2,1}}{a_{2,2}} & 1 & \frac{a_{2,3}}{a_{2,2}} & \cdots & \frac{a_{2,M}}{a_{2,2}} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,M} \end{array}\right]$$

Multiplicamos la ecuación 2 por $a_{3,2}$ y se la restamos a la ecuación 3

$$a_{3,2}\left[\begin{array}{c|c|c||c||c} a_{1,1}-a_{1,2}\left(\frac{a_{2,1}}{a_{2,2}}\right) & 0 & a_{1,3}-a_{1,2}\left(\frac{a_{2,3}}{a_{2,2}}\right) & \cdots & a_{1,M}-a_{1,2}\left(\frac{a_{2,M}}{a_{2,2}}\right) \\ \hline \frac{a_{2,1}}{a_{2,2}} & 1 & \frac{a_{2,3}}{a_{2,2}} & \cdots & \frac{a_{2,M}}{a_{2,2}} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,M} \end{array}\right]$$

$$\left[\begin{array}{c|c|c||c||c} a_{1,1}-a_{1,2}\left(\frac{a_{2,1}}{a_{2,2}}\right) & 0 & a_{1,3}-a_{1,2}\left(\frac{a_{2,3}}{a_{2,2}}\right) & \cdots & a_{1,M}-a_{1,2}\left(\frac{a_{2,M}}{a_{2,2}}\right) \\ \hline \frac{a_{2,1}}{a_{2,2}} & 1 & \frac{a_{2,3}}{a_{2,2}} & \cdots & \frac{a_{2,M}}{a_{2,2}} \\ \hline a_{3,1}-a_{3,2}\left(\frac{a_{2,1}}{a_{2,2}}\right) & 0 & a_{3,3}-a_{3,2}\left(\frac{a_{2,3}}{a_{2,2}}\right) & \cdots & a_{3,1}-a_{3,2}\left(\frac{a_{2,M}}{a_{2,2}}\right) \end{array}\right]$$

Paso 3

Dividimos la tercera ecuación entre $a_{3,3}$

$$\frac{1}{a_{3,3}}\left[\begin{array}{c|c|c||c||c} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,M} \\ \hline a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,M} \\ \hline a_{3,1} & a_{3,2} & a_{3,3} & \cdots & a_{3,M} \end{array}\right]$$

$$\left[\begin{array}{c|c|c||c||c} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,M} \\ \hline a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,M} \\ \hline \frac{a_{3,1}}{a_{3,3}} & \frac{a_{3,2}}{a_{3,3}} & 1 & \cdots & \frac{a_{3,M}}{a_{3,3}} \end{array}\right]$$

Multiplicamos la ecuación 3 por $a_{1,3}$ y la restamos a la primer ecuación

$$a_{1,3}\left[\begin{array}{c|c|c||c||c} a_{1,1} & a_{1,2} & a_{1,3} & \cdots & a_{1,M} \\ \hline a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,M} \\ \hline \frac{a_{3,1}}{a_{3,3}} & \frac{a_{3,2}}{a_{3,3}} & 1 & \cdots & \frac{a_{3,M}}{a_{3,3}} \end{array}\right]$$

$$\left[\begin{array}{c|c|c||c||c} a_{1,1}-a_{1,3}\left(\frac{a_{3,1}}{a_{3,3}}\right) & a_{1,2}-a_{1,3}\left(\frac{a_{3,2}}{a_{3,3}}\right) & 0 & \cdots & a_{1,M}-a_{1,3}\left(\frac{a_{3,M}}{a_{3,3}}\right) \\ \hline a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,M} \\ \hline \frac{a_{3,1}}{a_{3,3}} & \frac{a_{3,2}}{a_{3,3}} & 1 & \cdots & \frac{a_{3,M}}{a_{3,3}} \end{array}\right]$$

Multiplicamos la ecuación 3 por $a_{2,3}$ y la restamos a la segunda ecuación

$$a_{2,3}\left[\begin{array}{c|c|c||c||c} a_{1,1}-a_{1,3}\left(\frac{a_{3,1}}{a_{3,3}}\right) & a_{1,2}-a_{1,3}\left(\frac{a_{3,2}}{a_{3,3}}\right) & 0 & \cdots & a_{1,M}-a_{1,3}\left(\frac{a_{3,M}}{a_{3,3}}\right) \\ \hline a_{2,1} & a_{2,2} & a_{2,3} & \cdots & a_{2,M} \\ \hline \frac{a_{3,1}}{a_{3,3}} & \frac{a_{3,2}}{a_{3,3}} & 1 & \cdots & \frac{a_{3,M}}{a_{3,3}} \end{array}\right]$$

$$\left[\begin{array}{c|c|c||c||c} a_{1,1}-a_{1,3}\left(\frac{a_{3,1}}{a_{3,3}}\right) & a_{1,2}-a_{1,3}\left(\frac{a_{3,2}}{a_{3,3}}\right) & 0 & \cdots & a_{1,M}-a_{1,3}\left(\frac{a_{3,M}}{a_{3,3}}\right) \\ \hline a_{2,1}-a_{2,3}\left(\frac{a_{3,1}}{a_{3,3}}\right) & a_{2,2}-a_{2,3}\left(\frac{a_{3,2}}{a_{3,3}}\right) & 0 & \cdots & a_{2,M}-a_{2,3}\left(\frac{a_{3,M}}{a_{3,3}}\right) \\ \hline \frac{a_{3,1}}{a_{3,3}} & \frac{a_{3,2}}{a_{3,3}} & 1 & \cdots & \frac{a_{3,M}}{a_{3,3}} \end{array}\right]$$

Note la similitud del método con el método de sumas y restas. No olvidar que este método es equivalente al calculo parcial de la matriz inversa. El método se ilustra mejor con un ejemplo.

Ejemplo 1

Utilice la técnica de Gauss-Jordan para resolver el siguiente sistema de ecuaciones.

$$\begin{aligned} 30x_1 - x_2 - 2x_3 &= 78 \\ x_1 + 70x_2 - 3x_3 &= -193 \\ 3x_1 - 2x_2 + 100x_3 &= 714 \end{aligned}$$

Primero exprese los coeficientes y el lado derecho como una matriz aumentada

$$\left(\begin{array}{ccc|ccc|c} 30 & -1 & -2 & 1 & 0 & 0 & 78 \\ 1 & 70 & -3 & 0 & 1 & 0 & -193 \\ 3 & -2 & 100 & 0 & 0 & 1 & 714 \end{array}\right)$$

Primer iteración

Luego se normaliza el primer renglón, al dividirlo entre el elemento pivote, 30, se obtiene

$$\left[\begin{array}{ccc|ccc|c} 1 & -\frac{1}{30} & -\frac{1}{15} & \frac{1}{30} & 0 & 0 & \frac{13}{5} \\ 1 & 70 & -3 & 0 & 1 & 0 & -193 \\ 3 & -2 & 100 & 0 & 0 & 1 & 714 \end{array}\right]$$

El término x_1 se puede eliminar del segundo renglón restando el primer renglón multiplicado por 1 del segundo renglón. En forma similar restando el primer renglón multiplicando por 3 eliminará el término x_1 del tercer renglón

$$\left[\begin{array}{ccc|ccc|c} 1 & -\frac{1}{30} & -\frac{1}{15} & \frac{1}{30} & 0 & 0 & \frac{13}{5} \\ 0 & \frac{2101}{30} & -\frac{44}{15} & -\frac{1}{30} & 1 & 0 & -\frac{978}{5} \\ 0 & -\frac{19}{10} & \frac{501}{5} & -\frac{1}{10} & 0 & 1 & \frac{3531}{5} \end{array}\right]$$

$$\left[\begin{array}{ccc|ccc|c} 1. & -0.0333 & -0.0667 & 0.0333 & 0. & 0. & 2.6 \\ 0. & 70.0333 & -2.9333 & -0.0333 & 1. & 0. & -195.6 \\ 0. & -1.9 & 100.2 & -0.1 & 0. & 1. & 706.2 \end{array}\right]$$

Segunda Iteración

En seguida, se normaliza el segundo renglón dividiéndolo entre $\frac{2101}{30}$

$$\left[\begin{array}{ccc|ccc|c} 1 & -\frac{1}{30} & -\frac{1}{15} & \frac{1}{30} & 0 & 0 & \frac{13}{5} \\ 0 & 1 & -\frac{8}{191} & -\frac{1}{2101} & \frac{30}{2101} & 0 & -\frac{5868}{2101} \\ 0 & -\frac{19}{10} & \frac{501}{5} & -\frac{1}{10} & 0 & 1 & \frac{3531}{5} \end{array}\right]$$

$$\left[\begin{array}{ccc|ccc|c} 1. & -0.0333 & -0.0667 & 0.0333 & 0. & 0. & 2.6 \\ 0. & 1. & -0.0419 & -0.0005 & 0.0143 & 0. & -2.793 \\ 0. & -1.9 & 100.2 & -0.1 & 0. & 1. & 706.2 \end{array}\right]$$

Al reducir los términos x_2 de las ecuaciones 1 y 2 tenemos

$$\left[\begin{array}{ccc|ccc|c} 1 & 0 & -\frac{13}{191} & \frac{70}{2101} & \frac{1}{2101} & 0 & \frac{5267}{2101} \\ 0 & 1 & -\frac{8}{191} & -\frac{1}{2101} & \frac{30}{2101} & 0 & -\frac{5868}{2101} \\ 0 & 0 & \frac{19123}{191} & -\frac{212}{2101} & \frac{57}{2101} & 1 & \frac{1472577}{2101} \end{array}\right]$$

$$\left[\begin{array}{ccc|ccc|c} 1. & 0. & -0.0681 & 0.0333 & 0.0005 & 0. & 2.5069 \\ 0. & 1. & -0.0419 & -0.0005 & 0.0143 & 0. & -2.793 \\ 0. & 0. & 100.12 & -0.1009 & 0.0271 & 1. & 700.893 \end{array}\right]$$

Tercer iteración

El tercer renglón se normaliza entonces al dividirlo entre $\frac{19123}{191}$

$$\left[\begin{array}{ccc|ccc|c} 1 & 0 & -\frac{13}{191} & \frac{70}{2101} & \frac{1}{2101} & 0 & \frac{5267}{2101} \\ 0 & 1 & -\frac{8}{191} & -\frac{1}{2101} & \frac{30}{2101} & 0 & -\frac{5868}{2101} \\ 0 & 0 & 1 & -\frac{212}{210353} & \frac{57}{210353} & \frac{191}{19123} & \frac{1472577}{210353} \end{array}\right]$$

$$\left[\begin{array}{ccccccc} 1. & 0. & -0.0681 & 0.0333 & 0.0005 & 0. & 2.5069 \\ 0. & 1. & -0.0419 & -0.0005 & 0.0143 & 0. & -2.793 \\ 0. & 0. & 1. & -0.001 & 0.0003 & 0.01 & 7.0005 \end{array}\right]$$

Por último, los términos x_3 se pueden reducir de la primera y segunda ecuación para obtener

$$\left[\begin{array}{ccc|ccc|c} 1 & 0 & 0 & \frac{538}{16181} & \frac{8}{16181} & \frac{1}{1471} & \frac{48274}{16181} \\ 0 & 1 & 0 & -\frac{109}{210353} & \frac{3006}{210353} & \frac{8}{19123} & -\frac{525828}{210353} \\ 0 & 0 & 1 & -\frac{212}{210353} & \frac{57}{210353} & \frac{191}{19123} & \frac{1472577}{210353} \end{array}\right]$$

$$\left[\begin{array}{ccccccc} 1. & 0. & 0. & 0.0332 & 0.0005 & 0.0007 & 2.9834 \\ 0. & 1. & 0. & -0.0005 & 0.0143 & 0.0004 & -2.4997 \\ 0. & 0. & 1. & -0.001 & 0.0003 & 0.01 & 7.0005 \end{array}\right]$$

El código en Matlab es

```
function [x, Ainv] = Gauss_Jordan(A, b)
N = length(b);

B  = [A,  diag(ones(N,1)), b];

for n = 1:N;
    B(n,:) = B(n, :)./B(n,n);
    for m = 1: N
        if n ~= m
            B(m,:) = B(m,:) - B(n,:)*B(m,n);
        end;
    end;
end;
x = B(:,2*N+1);
```

```
Ainv = B(:, (N+1):(N+N));
```

Como resultado tenemos en primer termino la matriz identidad resultado de las eliminaciones, en segundo lugar la matriz inversa y finalmente la solución del sistema de ecuaciones.

Para el ejemplo hacemos

```
>> [b, Ainv] = Gauss_Jordan([3, -0.1, -0.2; 0.1 7 -0.3; 0.3 -0.2 10],
 [7.85; -19.3; 71.4])

x =

    3.0000
   -2.5000
    7.0000


Ainv =

    0.3325    0.0049    0.0068
   -0.0052    0.1429    0.0042
   -0.0101    0.0027    0.0999
```

Ejemplo 2

Dado el sistema lineal de ecuaciones calcular la solución utilizando el método de Gauss-Jordan

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

La matriz aumentada para este sistema es:

$$\left[\begin{array}{ccc|c} 1 & 1 & 2 & -1 \\ 1 & 3 & -6 & 7 \\ 2 & -1 & 2 & 0 \end{array}\right]$$

Primer iteración

$$\left[\begin{array}{ccc|c} 1 & 1 & 2 & -1 \\ 0 & 2 & -8 & 8 \\ 0 & -3 & -2 & 2 \end{array}\right]$$

Segunda iteración

$$\left[\begin{array}{ccc|c} 1 & 0 & 6 & -5 \\ 0 & 1 & -4 & 4 \\ 0 & 0 & -14 & 14 \end{array}\right]$$

Tercer iteración

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

Ejemplo 3

Dado el sistema lineal de ecuaciones, calcular la solución y la matriz inversa utilizando el método de Gauss-Jordan

$$\left[\begin{array}{cccc} 10 & -1 & 2 & 1 \\ -1 & 15 & -3 & 1 \\ 2 & -3 & 6 & -3 \\ 1 & 1 & -3 & 7 \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \\ x_4 \end{array}\right] = \left[\begin{array}{c} 1 \\ 2 \\ 2 \\ 1 \end{array}\right]$$

El sistema inicial aumentado es

$$\left[\begin{array}{cccc|cccc|c} 10 & -1 & 2 & 1 & 1 & 0 & 0 & 0 & 1 \\ -1 & 15 & -3 & 1 & 0 & 1 & 0 & 0 & 2 \\ 2 & -3 & 6 & -3 & 0 & 0 & 1 & 0 & 2 \\ 1 & 1 & -3 & 7 & 0 & 0 & 0 & 1 & 1 \end{array}\right]$$

Primer iteración

$$\left(\begin{array}{cccc|cccc|c} 1 & -\frac{1}{10} & \frac{1}{5} & \frac{1}{10} & \frac{1}{10} & 0 & 0 & 0 & \frac{1}{10} \\ 0 & \frac{149}{10} & -\frac{14}{5} & \frac{11}{10} & \frac{1}{10} & 1 & 0 & 0 & \frac{21}{10} \\ 0 & -\frac{14}{5} & \frac{28}{5} & -\frac{16}{5} & -\frac{1}{5} & 0 & 1 & 0 & \frac{9}{5} \\ 0 & \frac{11}{10} & -\frac{16}{5} & \frac{69}{10} & -\frac{1}{10} & 0 & 0 & 1 & \frac{9}{10} \end{array}\right)$$

$$\left[\begin{array}{cccc|cccc|c} 1.0000 & -0.1000 & 0.2000 & 0.1000 & 0.1000 & 0 & 0 & 0 & 0.1000 \\ 0 & 14.9000 & -2.8000 & 1.1000 & 0.1000 & 1.0000 & 0 & 0 & 2.1000 \\ 0 & -2.8000 & 5.6000 & -3.2000 & -0.2000 & 0 & 1.0000 & 0 & 1.8000 \\ 0 & 1.1000 & -3.2000 & 6.9000 & -0.1000 & 0 & 0 & 1.0000 & 0.9000 \end{array}\right]$$

Segunda iteración

$$\left[\begin{array}{cccc|cccc|c} 1 & 0 & \frac{27}{149} & \frac{16}{149} & \frac{15}{149} & \frac{1}{149} & 0 & 0 & \frac{17}{149} \\ 0 & 1 & -\frac{28}{149} & \frac{11}{149} & \frac{1}{149} & \frac{10}{149} & 0 & 0 & \frac{21}{149} \\ 0 & 0 & \frac{756}{149} & -\frac{446}{149} & -\frac{27}{149} & \frac{28}{149} & 1 & 0 & \frac{327}{149} \\ 0 & 0 & -\frac{446}{149} & \frac{1016}{149} & -\frac{16}{149} & -\frac{11}{149} & 0 & 1 & \frac{111}{149} \end{array}\right]$$

$$\left[\begin{array}{cccc|cccc|c} 1.0000 & 0 & 0.1812 & 0.1074 & 0.1007 & 0.0067 & 0 & 0 & 0.1141 \\ 0 & 1.0000 & -0.1879 & 0.0738 & 0.0067 & 0.0671 & 0 & 0 & 0.1409 \\ 0 & 0 & 5.0738 & -2.9933 & -0.1812 & 0.1879 & 1.0000 & 0 & 2.1946 \\ 0 & 0 & -2.9933 & 6.8188 & -0.1074 & -0.0738 & 0 & 1.0000 & 0.7450 \end{array}\right]$$

Tercer iteración

$$\left(\begin{array}{cccc|cccc|c} 1 & 0 & 0 & \frac{3}{14} & \frac{3}{28} & 0 & -\frac{1}{28} & 0 & \frac{1}{28} \\ 0 & 1 & 0 & -\frac{1}{27} & 0 & \frac{2}{27} & \frac{1}{27} & 0 & \frac{2}{9} \\ 0 & 0 & 1 & -\frac{223}{378} & -\frac{1}{28} & \frac{1}{27} & \frac{149}{756} & 0 & \frac{109}{252} \\ 0 & 0 & 0 & \frac{955}{189} & -\frac{3}{14} & \frac{1}{27} & \frac{223}{378} & 1 & \frac{257}{126} \end{array}\right)$$

$$\left[\begin{array}{cccc|cccc|c} 1.0000 & 0 & 0 & 0.2143 & 0.1071 & 0.0000 & -0.0357 & 0 & 0.0357 \\ 0 & 1.0000 & 0 & -0.0370 & 0 & 0.0741 & 0.0370 & 0 & 0.2222 \\ 0 & 0 & 1.0000 & -0.5899 & -0.0357 & 0.0370 & 0.1971 & 0 & 0.4325 \\ 0 & 0 & 0 & 5.0529 & -0.2143 & 0.0370 & 0.5899 & 1.0000 & 2.0397 \end{array}\right]$$

Cuarta iteración

$$\left[\begin{array}{cccc|cccc|c} 1 & 0 & 0 & 0 & \frac{111}{955} & -\frac{3}{1910} & -\frac{58}{955} & -\frac{81}{1910} & -\frac{97}{1910} \\ 0 & 1 & 0 & 0 & -\frac{3}{1910} & \frac{71}{955} & \frac{79}{1910} & \frac{7}{955} & \frac{453}{1910} \\ 0 & 0 & 1 & 0 & -\frac{58}{955} & \frac{79}{1910} & \frac{254}{955} & \frac{223}{1910} & \frac{1281}{1910} \\ 0 & 0 & 0 & 1 & -\frac{81}{1910} & \frac{7}{955} & \frac{223}{1910} & \frac{189}{955} & \frac{771}{1910} \end{array}\right]$$

$$\left[\begin{array}{cccc|cccc|c} 1.0000 & 0 & 0 & 0 & 0.1162 & -0.0016 & -0.0607 & -0.0424 & -0.0508 \\ 0 & 1.0000 & 0 & 0 & -0.0016 & 0.0743 & 0.0414 & 0.0073 & 0.2372 \\ 0 & 0 & 1.0000 & 0 & -0.0607 & 0.0414 & 0.2660 & 0.1168 & 0.6707 \\ 0 & 0 & 0 & 1.0000 & -0.0424 & 0.0073 & 0.1168 & 0.1979 & 0.4037 \end{array}\right]$$

```
>> [x, Ainv] = Gauss_Jordan([10 -1 2 1; -1 15 -3 1; 2 -3 6 -3; 1 1 -3 7],
[1,2,2,1]')

b =

   -0.0508
    0.2372
    0.6707
    0.4037


Ainv =

    0.1162   -0.0016   -0.0607   -0.0424
   -0.0016    0.0743    0.0414    0.0073
   -0.0607    0.0414    0.2660    0.1168
   -0.0424    0.0073    0.1168    0.1979
```

4.2. Métodos para sistemas no lineales

Una ecuación no lineal es aquella que tiene una forma diferente a $f(x) = a_0 + a_1 x$ en cuyo caso calcular la solución consiste en resolver despejando de $x = -a_0/a_1$. Pero el caso es que queremos resolver un sistema de ecuaciones no lineales de la forma

$$\begin{aligned} f_1(x_1, x_2, \cdots, x_n) &= 0 \\ f_2(x_1, x_2, \cdots, x_n) &= 0 \\ \vdots &= \vdots \\ f_n(x_1, x_2, \cdots, x_n) &= 0 \end{aligned}$$

donde n es el número de ecuaciones y se tiene el mismo números de funciones f_i que variables x_i. En esta sección veremos dos de estos métodos que son la iteración de punto fijo y el método de Newton-Raphson, los cuales se pueden utilizar para resolver sistemas de ecuaciones no lineales

4.2.1. Método Gráfico para sistemas no lineales en dos dimensiones

En el caso de tener un sistema de ecuaciones de dos variables, es posible hacer la gráfica de las curvas de nivel. Por ejemplo en el caso de tener dos funciones como el siguiente

$$\begin{aligned} f_1(x, y) &= x^2 + xy - 10 = 0 \\ f_2(x, y) &= y + 3xy^2 - 57 = 0 \end{aligned}$$

podemos graficar la curva de nivel con la función contour de matlab. El código correspondiente es el siguiente y en la Figura 4.11 se muestra las curvas de nivel obtenidas.

```
xa = [-10:0.01:10];
ya = [-10:0.01:10];
[x,y] = meshgrid(xa, ya);

f1 = x.^2 + x.*y - 10;
f2 = y + 3*x.*y.^2-57;

contour(x,y, f1,[0,0],'k')
hold on
contour(x,y, f2,[0,0],'r')
```

4.2.2. Método de iteración de punto fijo para sistemas

Considerando un sistema no lineal de ecuaciones

$$\begin{aligned} f_1(x_1, x_2, \cdots, x_n) &= 0 \\ f_2(x_1, x_2, \cdots, x_n) &= 0 \\ \vdots &= \vdots \\ f_n(x_1, x_2, \cdots, x_n) &= 0 \end{aligned}$$

El método de iteración de punto fijo intentara despejar de cada una de la las ecuaciones $f_i(x_1, x_2, \cdots, x_n)$ la i-esima variable tal que

$$\begin{aligned} x_1 - g_1(x_1, x_2, \cdots, x_n) &= f_1(x_1, x_2, \cdots, x_n) \\ x_2 - g_2(x_1, x_2, \cdots, x_n) &= f_2(x_1, x_2, \cdots, x_n) \\ \vdots &= \vdots \\ x_n - g_n(x_1, x_2, \cdots, x_n) &= f_n(x_1, x_2, \cdots, x_n) \end{aligned}$$

Para resolver de manera iterativa

$$\begin{aligned} x_1^{(t+1)} &= g_1(x_1^{(t)}, x_2^{(t)}, \cdots, x_n^{(t)}) \\ x_2^{(t+1)} &= g_2(x_1^{(t)}, x_2^{(t)}, \cdots, x_n^{(t)}) \\ \vdots &= \vdots \\ x_n^{(t+1)} &= g_n(x_1^{(t)}, x_2^{(t)}, \cdots, x_n^{(t)}) \end{aligned}$$

La implementación en Matlab es

```
function r = Punto_Fijo(g, x1)

while 1
    x2 = g(x1)
    error = abs(norm((x2-x1)/x2));
    if(error < 0.0001) break
    else x1 = x2;
    end;
end;
r = x2;
```

Ejemplo 1

Utilice el método de iteración de punto fijo para determinar las raíces del sistema de ecuaciones dado. Considere como valores iniciales $x_1^{(0)} = 1.5$ y $x_2^{(0)} = 3.5$.

$$\begin{aligned} x_1^2 + x_1 x_2 - 10 &= 0 \\ x_2 + 3x_1 x_2^2 - 57 &= 0 \end{aligned}$$

De acuerdo con lo descrito despejamos x_1 y x_2

$$\begin{aligned} f_1(x_1, x_2) &= x_1 - \sqrt{10 - x_1 x_2} \\ f_2(x_1, x_2) &= x_2 - \sqrt{\frac{57 - x_2}{3x_1}} \end{aligned}$$

Despejando tenemos el siguiente sistema iterativo de ecuaciones

$$\begin{aligned} x_1^{(t+1)} &= \sqrt{10 - x_1^{(t)} x_2^{(t)}} \\ x_2^{(t+1)} &= \sqrt{\frac{57 - x_2^{(t)}}{3x_1^{(t+1)}}} \end{aligned}$$

La implementación en Matlab es

```
function x = g1(x)

x(1) = sqrt(10 - x(1)*x(2));
x(2) = sqrt((57 - x(2))/(3 * x(1)));
```

El proceso iterativo será:

Iteración 1

Con $x^{(0)} = [1.5, 3.5]^T$ tenemos

$$\begin{aligned} x_1^{(1)} &= \sqrt{10 - x_1^{(0)} x_2^{(0)}} = \sqrt{10 - (1.5)(3.5)} = 2.1794 \\ x_2^{(1)} &= \sqrt{\frac{57 - x_2^{(0)}}{3x_1^{(1)}}} = \sqrt{\frac{57 - (3.5)}{3(2.1794)}} = 2.8605 \end{aligned}$$

Iteración 2

Con $x^{(1)} = [2.1794, 2.8605]^T$ tenemos

$$\begin{aligned} x_1^{(2)} &= \sqrt{10 - x_1^{(1)} x_2^{(1)}} = \sqrt{10 - (2.1794)(2.8605)} = 1.9405 \\ x_2^{(2)} &= \sqrt{\frac{57 - x_2^{(0)}}{3x_1^{(1)}}} = \sqrt{\frac{57 - (2.8605)}{3(1.9405)}} = 3.0496 \end{aligned}$$

El resumen el proceso iterativo se muestra en la siguiente tabla, donde podemos ver que la solución es $x^* = [2, 3]^T$ en 9 iteraciones.

k	$x_1^{(k)}$	$x_2^{(k)}$
0	1.5000	3.5000
1	2.1794	2.8605
2	1.9405	3.0496
3	2.0205	2.9834
4	1.9930	3.0057
5	2.0024	2.9981
6	1.9992	3.0007
7	2.0003	2.9998
8	1.9999	3.0001
9	2.0000	3.0000

Para realizar la ejecución dar en Matlab

```
Punto_Fijo(@g1, [1.5; 3.5])
```

Ejemplo 2

Utilice el método de iteración de punto fijo para determinar las raíces del sistema de ecuaciones dado. Considere como valores iniciales $x_1^{(0)} = 0.0$ y $x_2^{(0)} = 1.0$.

$$\begin{aligned} x_1^2 - 2x_1 - x_2 + 0.5 &= 0 \\ x_1^2 + 4x_2^2 - 4 &= 0 \end{aligned}$$

De acuerdo con lo descrito despejamos x_1 y x_2

$$\begin{aligned} f_1(x_1, x_2) &= x_1 - \sqrt{2x_1 + x_2 - 0.5} \\ f_2(x_1, x_2) &= x_2 - \sqrt{\frac{4 - x_1^2}{4}} \end{aligned}$$

Despejando tenemos el siguiente sistema iterativo de ecuaciones

$$\begin{aligned} x_1^{(t+1)} &= \sqrt{2x_1^{(t)} + x_2^{(t)} - 0.5} \\ x_2^{(t+1)} &= \sqrt{\frac{4 - \left(x_1^{(t+1)}\right)^2}{4}} \end{aligned}$$

La implementación en Matlab es

```
function x = g2(x)

x(1) = sqrt(2*x(1) + x(2) - 0.5);
x(2) = sqrt(4 -x(1)^2)/2;
```

El proceso iterativo será:

Iteración 1

Con $x^{(0)} = [0, 1]^T$ tenemos

$$\begin{aligned} x_1^{(1)} &= \sqrt{2x_1^{(0)} + x_2^{(0)} - 0.5} = \sqrt{2(0) + (1) - 0.5} = 0.7071 \\ x_2^{(1)} &= \sqrt{\frac{4 - \left(x_1^{(1)}\right)^2}{4}} = \sqrt{\frac{4 - (0.7071)^2}{4}} = 0.9354 \end{aligned}$$

Iteración 2

Con $x^{(1)} = [0.7071, 0.9354]^T$ tenemos

$$\begin{aligned} x_1^{(2)} &= \sqrt{2x_1^{(1)} + x_2^{(1)} - 0.5} = \sqrt{2(0.7071) + (0.9354) - 0.5} = 1.3600 \\ x_2^{(2)} &= \sqrt{\frac{4 - \left(x_1^{(2)}\right)^2}{4}} = \sqrt{\frac{4 - (1.3600)^2}{4}} = 0.7332 \end{aligned}$$

El resumen el proceso iterativo se muestra en la siguiente tabla, donde podemos ver que la solución es $x^* = [1.9007, 0.3112]^T$ en 8 iteraciones.

k	$x_1^{(k)}$	$x_2^{(k)}$
0	0.0000	1.0000
1	0.7071	0.9354
2	1.3600	0.7332
3	1.7185	0.5116
4	1.8570	0.3713
5	1.8935	0.3220
6	1.8997	0.3127
7	1.9006	0.3114
8	1.9007	0.3112

Para realizar la ejecución dar en Matlab

```
Punto_Fijo(@g2, [0; 1])
```

4.2.3. Método de Newton-Raphson para sistemas

Consideremos el sistema de ecuaciones no lineales

$$\begin{aligned} f_1(x_1, x_2, \cdots, x_n) &= 0 \\ f_2(x_1, x_2, \cdots, x_n) &= 0 \\ \vdots &= \vdots \\ f_n(x_1, x_2, \cdots, x_n) &= 0 \end{aligned}$$

Utilizando la serie de Taylor podemos hacer una aproximación lineal para una función $f_i(x_1^{(t)}+\delta x_1^{(t)}, x_2^{(t)}+\delta x_2^{(t)}, \cdots, x_n^{(t)}+\delta x_n^{(t)})$ en un incremento $\delta x_i^{(t)} = x_i^{(t+1)} - x_i^{(t)}$ como:

$$
\begin{aligned}
& f_1(x_1^{(t)}+\delta x_1^{(t)}, x_2^{(t)}+\delta x_2^{(t)}, \cdots, x_n^{(t)}+\delta x_n^{(t)}) \\
&= f_1(x_1^{(t)}, x_2^{(t)}+\cdots, +x_n^{(t)}) + \frac{\partial f_1}{\partial x_1}\delta x_1^{(t)} + \frac{\partial f_1}{\partial x_2}\delta x_2^{(t)} + \cdots + \frac{\partial f_1}{\partial x_n}\delta x_n^{(t)} \\
& f_2(x_1^{(t)}+\delta x_1^{(t)}, x_2^{(t)}+\delta x_2^{(t)}, \cdots, x_n^{(t)}+\delta x_n^{(t)}) \\
&= f_2(x_1^{(t)}, x_2^{(t)}+\cdots, +x_n^{(t)}) + \frac{\partial f_2}{\partial x_1}\delta x_1^{(t)} + \frac{\partial f_2}{\partial x_2}\delta x_2^{(t)} + \cdots + \frac{\partial f_2}{\partial x_n}\delta x_n^{(t)} \\
& \vdots \\
& f_n(x_1^{(t)}+\delta x_1^{(t)}, x_2^{(t)}+\delta x_2^{(t)}, \cdots, x_n^{(t)}+\delta x_n^{(t)}) \\
&= f_n(x_1^{(t)}, x_2^{(t)}+\cdots, +x_n^{(t)}) + \frac{\partial f_n}{\partial x_1}\delta x_1^{(t)} + \frac{\partial f_n}{\partial x_2}\delta x_2^{(t)} + \cdots + \frac{\partial f_n}{\partial x_n}\delta x_n^{(t)}
\end{aligned}
$$

Si escribimos el sistema en forma matricial tenemos

$$
\begin{bmatrix} f_1(x_1^{(t)}+\delta x_1^{(t)}, x_2^{(t)}+\delta x_2^{(t)}, \cdots, x_n^{(t)}+\delta x_n^{(t)}) \\ f_2(x_1^{(t)}+\delta x_1^{(t)}, x_2^{(t)}+\delta x_2^{(t)}, \cdots, x_n^{(t)}+\delta x_n^{(t)}) \\ \vdots \\ f_n(x_1^{(t)}+\delta x_1^{(t)}, x_2^{(t)}+\delta x_2^{(t)}, \cdots, x_n^{(t)}+\delta x_n^{(t)}) \end{bmatrix} =
$$

$$
\begin{bmatrix} f_1(x_1^{(t)}, x_2^{(t)}, \cdots, x_n^{(t)}) \\ f_2(x_1^{(t)}, x_2^{(t)}, \cdots, x_n^{(t)}) \\ \vdots \\ f_n(x_1^{(t)}, x_2^{(t)}, \cdots, x_n^{(t)}) \end{bmatrix} + \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_n} \\ \vdots & \vdots & \cdots & \vdots \\ \frac{\partial f_n(x)}{\partial x_1} & \frac{\partial f_n(x)}{\partial x_2} & \cdots & \frac{\partial f_n(x)}{\partial x_n} \end{bmatrix} \begin{bmatrix} \delta x_1^{(t)} \\ \delta x_2^{(t)} \\ \vdots \\ \delta x_n^{(t)} \end{bmatrix}
$$

En forma compacta

$$
\begin{aligned}
f(x^{(t)}+\delta x^{(t)}) &= f(x^{(t)}) + J(x^{(t)})\delta x^{(t)} \\
f(x^{(t)}+\delta x^{(t)}) &= f(x^{(t)}) + J(x^{(t)})(x^{(t+1)} - x^{(t)})
\end{aligned}
$$

donde $J(x)$ es la matriz de primeras derivada o Jacobiano.

Como queremos encontrar el cero de la función, la aproximación lineal que debemos resolver es:

$$0 = f(x^{(t)}) + J(x^{(t)})(x^{(t+1)} - x^{(t)})$$

donde las actualizaciones las hacemos como

$$x^{(t+1)} = x^{(t)} - \left[J(x^{(t)})\right]^{-1} f(x^{(t)})$$

La implementación del algoritmo en Matlab es:

```
function r = Newton_Raphson(f, J, x1)

while 1
    x2 = x1 - inv(J(x1))* f(x1);
    if norm((x1-x2)./x2) < 0.0001 break;
    else x1 = x2;
    end;
end;
r = x2;
```

Ejemplo 1

Resolver el siguiente sistema de ecuaciones dado por y cuya gráfica se muestra en la Figura 4.11

$$\begin{aligned} f_1(x_1, x_2) &= x_1^2 + x_1 x_2 - 10 \\ f_2(x_1, x_2) &= x_2 + 3x_1 x_2^2 - 57 \end{aligned}$$

El Jacobiano es

$$J(x) = \left[\begin{array}{c|c} 2x_1 + x_2 & x_1 \\ \hline 3x_2^2 & 1 + 6x_1 x_2 \end{array}\right]$$

y el arreglo de funciones

$$f(x) = \begin{bmatrix} x_1^2 + x_1 x_2 - 10 \\ x_2 + 3x_1 x_2^2 - 57 \end{bmatrix}$$

Considerando como valores iniciales $x^{(0)} = [1.5, 3.5]^T$ tenemos:

Primer iteración

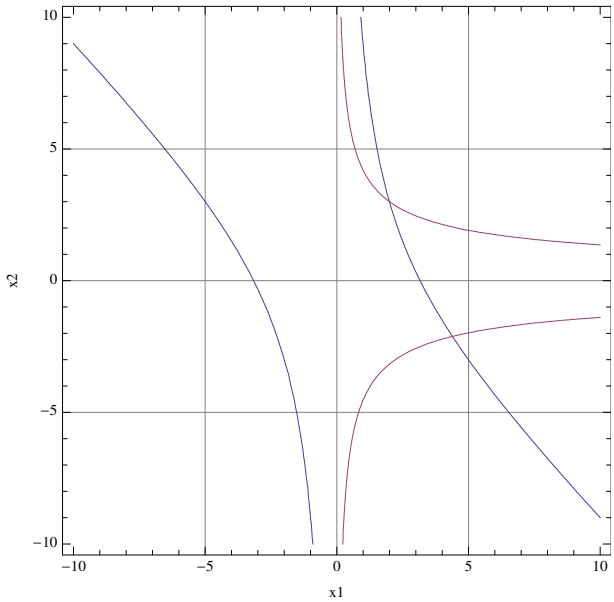


Figura 4.11: Curvas con la solución del sistema del ejemplo 1

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} 1.5000 \\ 3.5000 \end{bmatrix} - \begin{bmatrix} 6.5000 & 1.5000 \\ 36.7500 & 32.5000 \end{bmatrix}^{-1} \begin{bmatrix} -2.5000 \\ 1.6250 \end{bmatrix} = \begin{bmatrix} 2.0360 \\ 2.8439 \end{bmatrix}$$

Segunda iteración

$$\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} = \begin{bmatrix} 2.0360 \\ 2.8439 \end{bmatrix} - \begin{bmatrix} 6.9159 & 2.0360 \\ 24.2629 & 35.7413 \end{bmatrix}^{-1} \begin{bmatrix} -0.0644 \\ -4.7562 \end{bmatrix} = \begin{bmatrix} 1.9987 \\ 3.0023 \end{bmatrix}$$

Tercer iteración

$$\begin{bmatrix} x_1^{(3)} \\ x_2^{(3)} \end{bmatrix} = \begin{bmatrix} 1.9987 \\ 3.0023 \end{bmatrix} - \begin{bmatrix} 6.9997 & 1.9987 \\ 27.0412 & 37.0041 \end{bmatrix}^{-1} \begin{bmatrix} -0.0045 \\ 0.0496 \end{bmatrix} = \begin{bmatrix} 2.0000 \\ 3.0000 \end{bmatrix}$$

Cuarta iteración

$$\begin{bmatrix} x_1^{(4)} \\ x_2^{(4)} \end{bmatrix} = \begin{bmatrix} 2.0000 \\ 3.0000 \end{bmatrix} - \begin{bmatrix} 7.0000 & 2.0000 \\ 27.0000 & 37.0000 \end{bmatrix}^{-1} \begin{bmatrix} -0.00000129 \\ -0.00002214 \end{bmatrix} = \begin{bmatrix} 2.0000 \\ 3.0000 \end{bmatrix}$$

Note que solamente 4 iteraciones son necesarias para llegar a la solución $x = [2, 3]^T$

La implementación en Matlab para este ejemplo son:

Para la función tenemos

```
function f = f1(x)

n = length(x);
f = zeros(n,1);

f(1) = x(1)^2 + x(1)*x(2) - 10;
f(2) = x(2) + 3*x(1)*x(2)^2 - 57;
end
```

El Jacobiano

```
function J = J1(x)
n = length(x);
J = zeros(n, n);

J(1,1) = 2*x(1) + x(2);
J(1,2) = x(1);
J(2,1) = 3*x(2)^2;
J(2,2) = 1 + 6*x(1)*x(2);
end
```

y la ejecución

```
>> Newton_Raphson(@f1, @J1, [1.5,3.5]')

ans =

    2.0000
    3.0000
```

Ejemplo 2

Resolver el siguiente sistema de ecuaciones dado por las ecuaciones y cuya solución gráfica se muestra en la figura 4.12

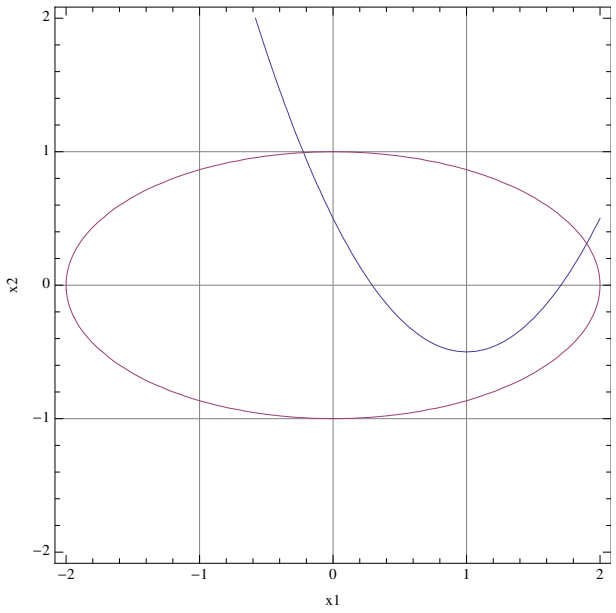


Figura 4.12: Curvas con la solución del sistema del ejemplo 2

$$\begin{aligned} f_1(x_1, x_2) &= x_1^2 – 2x_1 – x_2 + 0.5 \\ f_2(x_1, x_2) &= x_1^2 + 4x_2^2 – 4 \end{aligned}$$

El Jacobiano es

$$J(x) = \left[\begin{array}{c|c} 2x_1 - 2 & -1 \\ \hline 2x_1 & 8x_2 \end{array}\right]$$

y el arreglo de funciones

$$f(x) = \begin{bmatrix} x_1^2 – 2x_1 – x_2 + 0.5 \\ x_1^2 + 4x_2^2 – 4 \end{bmatrix}$$

Considerando como valores iniciales $x^{(0)} = [0, 1]^T$ tenemos:

Primer iteración

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \end{bmatrix} - \begin{bmatrix} -2 & -1 \\ 0 & 8 \end{bmatrix}^{-1} \begin{bmatrix} -0.5 \\ 0.0 \end{bmatrix} = \begin{bmatrix} -0.25 \\ 1.00 \end{bmatrix}$$

Segunda iteración

$$\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} = \begin{bmatrix} -0.2500 \\ 1.0000 \end{bmatrix} - \begin{bmatrix} 2.5000 & -1.0000 \\ -0.5000 & 8.0000 \end{bmatrix}^{-1} \begin{bmatrix} 0.0625 \\ 0.0625 \end{bmatrix} = \begin{bmatrix} -0.2226 \\ 0.9939 \end{bmatrix}$$

Tercer iteración

$$\begin{bmatrix} x_1^{(3)} \\ x_2^{(3)} \end{bmatrix} = \begin{bmatrix} -0.2226 \\ 0.9939 \end{bmatrix} - \begin{bmatrix} -2.4451 & -1.0000 \\ -0.4451 & 7.9512 \end{bmatrix}^{-1} \begin{bmatrix} -0.0008 \\ 0.0009 \end{bmatrix} = \begin{bmatrix} -0.2222 \\ 0.9938 \end{bmatrix}$$

Cuarta iteración

$$\begin{bmatrix} x_1^{(4)} \\ x_2^{(4)} \end{bmatrix} = \begin{bmatrix} -0.2222 \\ 0.9938 \end{bmatrix} - \begin{bmatrix} -2.4444 & -1.0000 \\ -0.4444 & 7.9505 \end{bmatrix}^{-1} \begin{bmatrix} -0.00002300 \\ 0.00000038 \end{bmatrix} = \begin{bmatrix} -0.2222 \\ 0.9938 \end{bmatrix}$$

La implementación en Matlab para este ejemplo son:

Para la función tenemos

```
function f = f2(x)

n = length(x);
f = zeros(n,1);

f(1) = x(1)*x(1) - 2*x(1) - x(2) + 0.5;
f(2) = x(1)*x(1) + 4*x(2)*x(2) - 4;
end
```

El Jacobiano

```
function y = J2(x)
n = length(x);
y = zeros(n, n);
y(1,1) = 2*x(1) -2;
y(1,2) = -2;
y(2,1) = 2*x(1);
y(2,2) = 8*x(2);
end
```

y la ejecución

```
>> Newton_Raphson(@f2, @J2, x)

ans =

   -0.2222
    0.9938
```

4.2.4. Ejemplo

Para el circuito que se muestra en la figura 4.13, esta constituido por dos mallas y dos cargas. Se sea calcular la corriente que circula por cada uno de los elementos y la potencia que debe suministrar la fuente de voltaje.

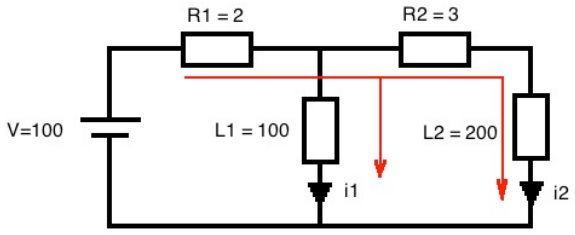


Figura 4.13: Circuito eléctrico de dos mallas

La ecuaciones de Voltaje para cada una de las mallas es:

$$V - R_1(i_1 + i_2) - L_1/i_1 = 0$$
$$V - R_1(i_1 + i_2) - R_2 i_2 - L_2/i_2 = 0$$

Reorganizando las funciones tenemos

$$F = \begin{bmatrix} Vi_1 - R_1(i_1 + i_2)i_1 - L_1 \\ Vi_2 - R_1(i_1 + i_2)i_2 - R_2 i_2^2 - L_2 \end{bmatrix} = 0$$

```
function f = Fun(i)
N = length(i);
f = zeros(N,1);

V = 100;
```

```
R1 = 2;
R2 = 3;
L1 = 100;
L2 = 200;

f(1) = V*i(1) - R1*(i(1)+i(2))*i(1) - L1;
f(2) = V*i(2) - R1*(i(1)+i(2))*i(2) - R2*i(2)^2 - L2;
end
```

El Jacobiano para este sistema es:

$$J = \left[\begin{array}{c|c} V - R_1(2i_1 + i_2) & -i_1 R_1 \\ \hline -i_2 R1 & V_1 - R_1(i_1 + 2i_2) - 2R_2 i_2 \end{array}\right]$$

```
function J = JacSistema(i)
V = 100;
R1 = 2;
R2 = 3;
%L1 = 100;
%L2 = 200;

J(1,1) = V - R1*(2*i(1) + i(2)) ;
J(1,2) = -i(1)*R1;
J(2,1) = -i(2)*R1;
J(2,2) = V - (i(1) + 2*i(2))*R1 - 2*i(2)*R2 ;
end
```

La solución considerando un valor inicial de corrientes $I^{(0)} = [0, 0]^T$:

```
>> I = NewtonRaphson(@FunSistema, @JacSistema, [0,0]')
1.-      1     2

2.-     1.0720    2.3114

3.-     1.0728    2.3185

4.-     1.0728    2.3185


I =
```

```
1.0728
2.3185
```

Potencia de la fuente $P_V = V(i_1 + i_2) = 100(1.0728 + 2.3185) = +339.1282$
Potencia consumida por R_1 es $P_{R_1} = -R_1(i_1 + i_2)^2 = -2(1.0728 + 2.3185)^2 = -23.0016$
Potencia consumida por R_2 es $P_{R_1} = -R_2 i_2^2 = -3(2.3185)^2 = -16.1266$
La potencia consumida por las cargas es $L1 + L2 = -300$.
La suma de la potencia es cero.

Desventajas del Método de Newton

Aunque el método de Newton-Raphson en general es muy eficiente, hay situaciones en que se comporta en forma deficiente. Un caso especial, raíces múltiples.

Ejemplo

Determinar la raíz de la función $f(x) = x^{10}–1$.

La solución utilizando el método de Newton-Raphson queda:

$$x^{(k+1)} = x^{(k)} - \frac{f(x^{(k)})}{f'(x^{(k+1)})}$$

Sustituyendo valores tenemos

$$x^{(k+1)} = x^{(k)} - \frac{\left(x^{(k)}\right)^{10} - 1}{10\left(x^{(k)}\right)^{9}}$$

Y la solución numérica es:

$x^{(k)}$	$f(x^{(k)})$	$f'(x^{(k)})$
0.5000	-0.9990	0.0195
51.6500	135114904483914000.0000	26159710451871000.0000
46.4850	47111654129711500.0000	10134807815362300.0000
41.8365	16426818072478500.0000	3926432199748670.0000
37.6529	5727677301318310.0000	1521180282851980.0000
33.8876	1997117586819850.0000	589336409039672.0000
30.4988	696351844868619.0000	228320999775654.0000
27.4489	242802875029547.0000	88456233382052.8000
24.7040	84660127717097.5000	34269757191973.2000
22.2336	29519161271064.1000	13276806089225.7000
20.0103	10292695105054.7000	5143706707446.1600
18.0092	3588840873655.1100	1992777367871.5700
16.2083	1251351437592.9200	772042782329.1500
14.5875	436319267276.5290	299105192259.1190
13.1287	152135121499.2910	115879479847.7330
11.8159	53046236848.5329	44894084747.9692
10.6343	18496079117.2577	17392888266.5936
9.5708	6449184014.3077	6738361277.7304
8.6138	2248691421.7628	2610579221.6818
7.7524	784070216.9426	1011391879.0870

Optimización

5.1. Optimización no-restringida. Método de búsqueda de la sección dorada

La búsqueda de la sección dorada es una técnica simple de búsqueda de una sola variable de propósito general. La clave para hacer eficiente este procedimiento es la mejor elección de los puntos intermedios. Esta meta se puede alcanzar al especificar que las siguientes dos condiciones se cumplan (ver [**?**]).

$$l_0 = l_1 + l_2 \tag{5.9}$$

$$\frac{l_1}{l_0} = \frac{l_2}{l_1} \tag{5.10}$$

Sustituyendo la ecuación 5.9 en la ecuación 5.10 obtenemos:

$$\frac{l_1}{l_1 + l_2} = \frac{l_2}{l_1} \tag{5.11}$$

Definimos $R = \frac{l_2}{l_1}$ y sustituimos en la ecuación 5.11 para obtener $R^2 + R - 1$, cuya solución positiva es $R = \frac{\sqrt{5}-1}{2}$. R es definido como la razón dorada y fue un número ampliamente utilizado por los Griegos en su arquitectura.

El algoritmo de la razón dorada es:

La implementación en Matlab es:

```
function y = Razon_Dorada(f, xl, xu)

R =  (sqrt(5) -1)/2
t = 0;
```

Algoritmo 1 Algoritmo de Búsqueda con Razón Dorada

Entrada: $x_u, x_l, f(x)$
Salida: x
  mientras $x_u - x_l > Tol$ **hacer**
    $d = R(x_u - x_l)$
    $x_1 = x_l + d$
    $x_2 = x_u - d$
    $k = k + 1$
    si $f(x_1) < f(x_2)$ **entonces**
      $x_l = x_2$
    si no
      $x_u = x_1$
    fin si
  fin mientras
  si $f(x_1) < f(x_2)$ **entonces**
    devolver x_1
  si no
    devolver x_2
  fin si

```
while (xu-xl > 0.0001)
    d = R*(xu - xl);
    x1 = xl + d;
    x2 = xu - d;
    t = t+1;
    if f(x1) < f(x2)
        xl = x2;
    else
        xu = x1;
    end;

    if f(x1) < f(x2)
        y = x1;
    else
        y=x2;
   end;
end;
```

5.1.1. Ejemplo 1

Use la búsqueda de la sección dorada para encontrar el mínimo de la función $f(x) = -2seno(x) + \frac{x^2}{10}$ en el intervalo [0,4].

Los resultados de este ejemplo se muestran en la siguiente tabla

k	x_l	f_l	x_2	f_2	x_1	f_1	x_u	f_u	d
0	0.0000	0.0000	1.5279	-1.7647	2.4721	-0.6300	4.0000	-3.1136	2.4721
1	0.0000	0.0000	0.9443	-1.5310	1.5279	-1.7647	2.4721	-0.6300	1.5279
2	0.9443	-1.5310	1.5279	-1.7647	1.8885	-1.5432	2.4721	-0.6300	0.9443
3	0.9443	-1.5310	1.3050	-1.7595	1.5279	-1.7647	1.8885	-1.5432	0.5836
4	1.3050	-1.7595	1.5279	-1.7647	1.6656	-1.7136	1.8885	-1.5432	0.3607
...	...	...	...	...	...	...	...	...	...
17	1.4267	-1.7757	1.4271	-1.7757	1.4274	-1.7757	1.4278	-1.7757	0.0007

Según los resultados de la tabla la solución es $x = 1.4267$ con $f(x) = -1.7757$.

Para realizar la ejecución dar

```
>> Razon_Dorada(@fun_1, 0, 4)

ans =

    1.4276
```

5.1.2. Ejemplo 2

Use la búsqueda de la sección dorada para encontrar el mínimo de la función $f(x) = -xe^{-x^2}$ en el intervalo [0,4].

Los resultados de este ejemplo se muestran en la siguiente tabla

k	x_l	f_l	x_2	f_2	x_1	f_1	x_u	f_u
0	0.0000	0.0000	1.5279	-0.1480	2.4721	-0.0055	4.0000	-0.0000
1	0.0000	0.0000	0.9443	-0.3871	1.5279	-0.1480	2.4721	-0.0055
2	0.0000	0.0000	0.5836	-0.4151	0.9443	-0.3871	1.5279	-0.1480
3	0.0000	0.0000	0.3607	-0.3167	0.5836	-0.4151	0.9443	-0.3871
4	0.3607	-0.3167	0.5836	-0.4151	0.7214	-0.4287	0.9443	-0.3871
...	...	...	...	...	...	...	...	...
22	0.7071	-0.4289	0.7071	-0.4289	0.7071	-0.4289	0.7072	-0.4289

Según los resultados de la tabla la solución es $x = 0.7071$ con $f(x) = -0.4289$.

Para realizar la ejecución dar

```
>> Razon_Dorada(@fun_2, 0, 4)

ans =

    0.7071
```

5.2. Optimización no-restringida. Método de Newton

5.2.1. Método de Newton en una dimensión

Consideremos que la función $f : \mathbb{R} \to \mathbb{R}$ es de clase dos, es decir que la segunda derivada puede ser calculada. La idea consiste en reemplazar en la vecindad del punto x^k de la función f por una aproximación cuadrática $q(x)$ dada por

$$q(x) = f(x^{(k)}) + f'(x^{(k)})(x - x^{(k)}) + \frac{1}{2}f''(x^{(k)})(x - x^{(k)})^2$$

llamaremos a $x^{(k+1)}$ el mínimo de $q(x)$. Para calcularlo, es necesario que la segunda derivada $f''(x^{(k)})$ sea positiva. La función $q(x)$ es entonces estrictamente convexa y tiene un mínimo único en x_{k+1} dado por

$$q'(x^{(k+1)}) = 0$$

Esto da lugar sistema lineal de ecuaciones:

$$q'(x^{(k+1)}) = f'(x^{(k)}) + f''(x^{(k)})(x^{(k+1)} - x^{(k)}) = 0$$

La solución para $x^{(k+1)}$ es el mínimo, lo cual da lugar a la recurrencia

$$x^{(k+1)} = x^{(k)} - \frac{f'(x^{(k)})}{f''(x^{(k)})} \tag{5.12}$$

Algoritmo 2 Algoritmo de Newton Unidimensional

Entrada: $f(x), x_0$
Salida: $x^{(k)}$
  mientras $\|f'(x^{(k)})\| \geq \varepsilon$ **hacer**
    $x^{(k+1)} = x^{(k)} - \frac{f'(x^{(k)})}{f''(x^{(k)})}$
    $k = k + 1$
  fin mientras
  devolver $x^{(k)}$

La implementación en Matlab es

```
function y = Newton_1d(derf, der2f, x)
x1 = x;
x2 = x;
while 1
    delta = derf(x1)/der2f(x1);
    x2 = x1 - delta;
    disp([x1, derf(x1), der2f(x1), x2]);
    if abs(delta) < 1e-6 break;
    else
        x1 =x2;
    end;
end;
y = x2;
```

5.2.2. Ejemplo 1

Calcular el máximo de la función $f(x) = -2seno(x) + \frac{x^2}{10}$ utilizando el método de Newton con un valor inicial $x_0 = 0.5$.

Para resolver tenemos que calcular $f'(x)$ y $f''(x)$

$$f'(x) = \frac{x}{5} - 2cos(x)$$

$$f''(x) = \frac{1}{5} + 2sin(x)$$

Para resolver hacemos

$$
\begin{aligned}
k &= 0 \\
x^{(1)} &= 0.5000 - \frac{-1.6551}{1.1588} = 1.9282 \\
k &= 1 \\
x^{(2)} &= 1.9282 - \frac{1.0854}{2.0735} = 1.4047 \\
k &= 2 \\
x^{(3)} &= 1.4047 - \frac{-0.0495}{2.1725} = 1.4275 \\
k &= 3 \\
x^{(4)} &= 1.4275 - \frac{8.20126 \times 10^{-5}}{2.1795} = 1.4275 \\
k &= 4 \\
x^{(5)} &= 1.4275 - \frac{2.02094 \times 10^{-10}}{2.1795} = 1.4275
\end{aligned}
$$

Para ejecutar hacer

```
>> Newton_1d(@der_fun_1, @der2_fun_1, 0.5)

ans =

    1.4276

function y = der_fun_1(x)
     y = x/5 - 2*cos(x);

function y = d2f1(x)
     y = 1/5 + 2*sin(x);
```

5.2.3. Ejemplo 2

Dada la función $f(x) = -xe^{-x^2}$, calcular el mínimo utilizando el método de Newton para un valor inicial $x_0 = -1.0$.

Para resolver tenemos que calcular $f'(x)$ y $f''(x)$

$$f'(x) = (2x^2 - 1)e^{-x^2}$$

$$f''(x) = (6x - 4x^3)e^{-x^2}$$

Para soluciona hacemos

$$\begin{aligned} k &= 0 \\ x^{(1)} &= -1.0000 - \frac{0.3679}{-0.7358} = -0.5000 \\ k &= 1 \\ x^{(2)} &= -0.5000 - \frac{-0.3894}{-1.9470} = -0.7000 \\ k &= 2 \\ x^{(3)} &= -0.7000 - \frac{-0.0123}{-1.7325} = -0.7071 \\ k &= 3 \\ x^{(4)} &= -0.7071 - \frac{-0.0001}{-1.7156} = -0.7071 \\ k &= 4 \\ x^{(5)} &= -0.7071 - \frac{-0.0000}{-1.7155} = -0.7071 \end{aligned}$$

Para ejecutar en Matlab

```
>> Newton_1d(@der_fun_2, @der2_fun_2, -1.0)

ans =

    -0.7071

function y = der_fun_2(x)
    y = (2*x^2 -1)*exp(-x^2);

function y = der2_fun_2(x)
    y = (6*x-4*x^3)*exp(-x^2);
```

5.2.4. Método de Newton en N dimensiones

Consideremos que la función f es de clase dos, es decir que la segunda derivada puede ser calculada. La idea consiste en reemplazar en la vecindad del punto x^k de la función f por una aproximación cuadrática $q(x)$ dada por

$$q(x) = f(x^{(k)}) + \nabla f^T(x^{(k)})(x - x^{(k)}) + \frac{1}{2}(x - x^{(k)})^T \nabla^2 f(x^{(k)})(x - x^{(k)})$$

donde $\nabla f(x^{(k)})$ es el gradiente de la función o vector de primeras derivadas y $\nabla f^2(x^{(k)})$ es la matriz de segundas derivadas o matriz Hessiana, ambos valuados en el muno $x^{(k)}$. Llamaremos a $x^{(k+1)}$ el mínimo de $q(x)$ y para calcularlo, es necesario que la matriz $\nabla^2 f(x^{(k)})$ sea una matriz definida positiva. La función $q(x)$ es entonces estrictamente convexa y tiene un mínimo único en $x^{(k+1)}$ dado por

$$\nabla_q(x^{(k+1)}) = 0$$

Esto da lugar sistema lineal de ecuaciones:

$$\nabla f(x^{(k)}) = -\nabla^2 f(x^{(k)})(x^{(k+1)} - x^{(k)})$$

La solución para x_{k+1} es el mínimo, lo cual da lugar a la recurrencia

$$x^{(k+1)} = x^{(k)} - \left[\nabla^2 f(x^{(k)})\right]^{-1} \nabla f(x^{(k)}) \tag{5.13}$$

En el algoritmo 3 se presenta el proceso para encontrar en mínimo de una función utilizando el algoritmo de Newton y abajo la implementación en Matlab

```
function r = Newton(g, H, x1)

while 1
    d= inv(H(x1))* g(x1);
    x2 = x1 - d;
    if norm(d) < 0.0001 break;
    else x1 = x2;
    end;
end;
r = x2;
```

Algoritmo 3 Algoritmo de Newton

Entrada: Dado $f(x)$ y $x^{(0)}$
Salida: $x^{(k)}$
  mientras $(\|\nabla f(x^{(k)})\| \leq \varepsilon)$ **hacer**
    $x^{(k+1)} = x^{(k)} - [\nabla^2 f(x^{(k)})]^{-1} * \nabla f(x^{(k)})$
    $k \leftarrow k + 1$
  fin mientras
  devolver $x^{(k)}$

5.2.5. Ejemplo 1

Minimizar es $f(x) = x_1 e^{-x_1^2 - x_2^2}$, con un punto inicial $x_1^{(0)} = -0.5$, $x_2^{(0)} = -0.1$ aplicando el Método de Newton.

El vector gradiente para esta función es:

$$\nabla f(x_1, x_2) = \begin{bmatrix} (1 - 2x_1^2)e^{(-x_1^2 - x_2^2)} \\ -2x_1 x_2 e^{(-x_1^2 - x_2^2)} \end{bmatrix}$$

```
function g = Gradiente_f1(x)
n = length(x);
g = zeros(n, 1);

val_e = exp(-x(1)^2-x(2)^2);

g(1) = val_e*(1-2*x(1)^2);
g(2) = val_e*(-2*x(1)*x(2));
```

El Hessiano es:

$$\nabla^2 f(x_1, x_2) = \begin{bmatrix} (-6x_1 + 4x_1^3)e^{(-x_1^2 - x_2^2)} & (-2x_2 + 4x_1^2 x_2)e^{(-x_1^2 - x_2^2)} \\ (-2x_2 + 4x_1^2 x_2)e^{(-x_1^2 - x_2^2)} & (-2x_1 + 4x_1 x_2^2)e^{(-x_1^2 - x_2^2)} \end{bmatrix}$$

```
function H = Hessiano_f1(x)
n = length(x);
H = zeros(n, n);

val_e = exp(-x(1)^2-x(2)^2);
H(1,1) = val_e*(-6*x(1)+4*x(1)^3);
H(1,2) = val_e*(-2*x(2)+4*x(1)^2*x(2));
H(2,1) = H(1,2);
H(2,2) = val_e*(-2*x(1)+4*x(1)*x(2)^2);
```

Primer iteración

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} -0.5000 \\ -0.1000 \end{bmatrix} - \begin{bmatrix} 1.9276 & 0.0771 \\ 0.0771 & 0.7556 \end{bmatrix}^{-1} \begin{bmatrix} 0.3855 \\ -0.0771 \end{bmatrix} = \begin{bmatrix} -0.7049 \\ 0.0230 \end{bmatrix}$$

Segunda iteración

$$\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} = \begin{bmatrix} -0.7049 \\ 0.0230 \end{bmatrix} - \begin{bmatrix} 1.7199 & -0.0002 \\ -0.0002 & 0.8564 \end{bmatrix}^{-1} \begin{bmatrix} 0.0038 \\ 0.0197 \end{bmatrix} = \begin{bmatrix} -0.7071 \\ -0.0000 \end{bmatrix}$$

Tercer iteración

$$\begin{bmatrix} x_1^{(3)} \\ x_2^{(3)} \end{bmatrix} = \begin{bmatrix} -0.7071 \\ -0.0000 \end{bmatrix} - \begin{bmatrix} 1.7155 & 0.0000 \\ 0.0000 & 0.8578 \end{bmatrix}^{-1} \begin{bmatrix} 0.0180 \\ -0.2114 \end{bmatrix} \times 10^{-4} = \begin{bmatrix} -0.7071 \\ -0.0000 \end{bmatrix}$$

En la siguiente tabla se muestra como la función $f(x)$ disminuye su valor en el proceso de convergencia

k	$f(x^{(k)})$
0	-0.3855
1	-0.4287
2	-0.4289
3	-0.4289

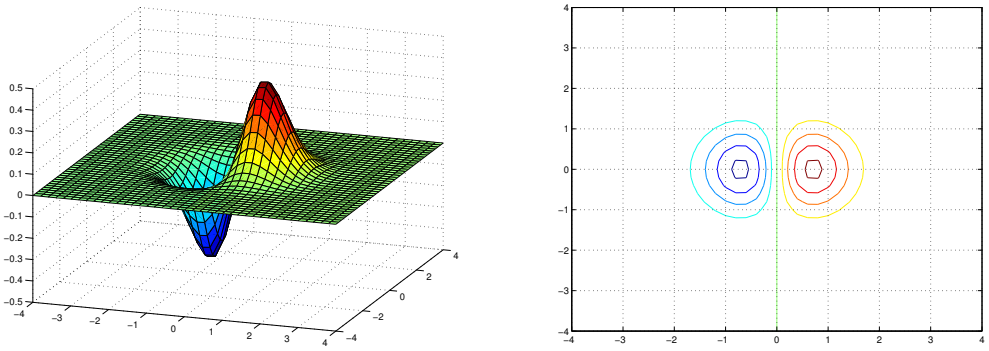
Para ejecutar dar

```
>> Newton(@Gradiente_f1, @Hessiano_f1, [-0.5;-0.1])

ans =

   -0.7071
    0.0000
```

En la figura 5.14 se muestra la función a la derecha y a la izquierda las curvas de nivel. El código en matlab utilizado para generar estas imágenes es

```
[x,y] = meshgrid(-4:0.2:4, -4:0.2:4);
z = x.*exp(-x.^2-y.^2);
surf(x,y,z)
```

```
[x,y] = meshgrid(-4:0.2:4, -4:0.2:4);
z = x.*exp(-x.^2-y.^2);
contour(x,y,z)
grid on
```



(a) ***Funsión*** **(b)** ***Curvas de Nivel***

Figura 5.14: Mínimo para la función $f(x_1, x_2) = x_1 e^{(-x_1^2 - x_2^2)}$

5.2.6. Ejemplo 2

Minimizar es $f(x) = -3.5x_1 - 2x_2 - x_2^2 + x_1^4 + 2x_1x_2 + x_2^4$, con un punto inicial $x_1^{(0)} = 1.0$, $x_2^{(0)} = 1.0$ aplicando el Método de Newton.

El vector gradiente para esta función es:

$$\nabla f(x_1, x_2) = \begin{bmatrix} -3.5 + 4x_1^3 + 2x_2 \\ -2 + 2x_1 - 2x_2 + 4x_2^3 \end{bmatrix}$$

```
function g = Gradiente_f2(x)
n = length(x);
g = zeros(n, 1);

g(1) = -3.5 + 4*x(1)^3 + 2*x(2);
g(2) = -2.0 + 2*x(1) - 2*x(2) + 4*x(2)^3;
```

El Hessiano es:

$$\nabla^2 f(x_1, x_2) = \begin{bmatrix} 12x_1^2 & 2 \\ 2 & -2 + 12x_2^2 \end{bmatrix}$$

```
function H = Hessiano_f2(x)
n = length(x);
H = zeros(n, n);
```

```
H(1,1) = 12*x(1)^2;
H(1,2) = 2;
H(2,1) = 2;
H(2,2) = -2+12*x(2)^2;
```

Primer iteración

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} - \begin{bmatrix} 12 & 2 \\ 2 & 10 \end{bmatrix}^{-1} \begin{bmatrix} 2.5000 \\ 2.0000 \end{bmatrix} = \begin{bmatrix} 0.8190 \\ 0.8362 \end{bmatrix}$$

Segunda iteración

$$\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} = \begin{bmatrix} 0.8190 \\ 0.8362 \end{bmatrix} - \begin{bmatrix} 8.0485 & 2.0000 \\ 2.0000 & 6.3909 \end{bmatrix}^{-1} \begin{bmatrix} 0.3695 \\ 0.3044 \end{bmatrix} = \begin{bmatrix} 0.7820 \\ 0.8001 \end{bmatrix}$$

Tercer iteración

$$\begin{bmatrix} x_1^{(3)} \\ x_2^{(3)} \end{bmatrix} = \begin{bmatrix} 0.7820 \\ 0.8001 \end{bmatrix} - \begin{bmatrix} 7.3385 & 2.0000 \\ 2.0000 & 5.6828 \end{bmatrix}^{-1} \begin{bmatrix} 0.0132 \\ 0.0129 \end{bmatrix} = \begin{bmatrix} 0.7807 \\ 0.7983 \end{bmatrix}$$

cuarta iteración

$$\begin{bmatrix} x_1^{(4)} \\ x_2^{(4)} \end{bmatrix} = \begin{bmatrix} 0.7807 \\ 0.7983 \end{bmatrix} - \begin{bmatrix} 7.3139 & 2.0000 \\ 2.0000 & 5.6483 \end{bmatrix}^{-1} \begin{bmatrix} 0.1610 \\ 0.3115 \end{bmatrix} \times 10^{-4} = \begin{bmatrix} 0.7807 \\ 0.7983 \end{bmatrix}$$

En la siguiente tabla se muestra como la función $f(x)$ disminuye su valor en el proceso de convergencia

k	$f(x^{(k)})$
0	-2.5000
1	-2.9296
2	-2.9422
3	-2.9423

Para ejecutar dar

```
>> Newton(@Gradiente_f2, @Hessiano_f2, [1;1])

ans =
```

```
-0.7807
 0.7983
```

En la figura 5.15 se muestras las curvas de nivel correspondientes a este ejemplo las cuales fueron determinadas utilizando el siguiente código.

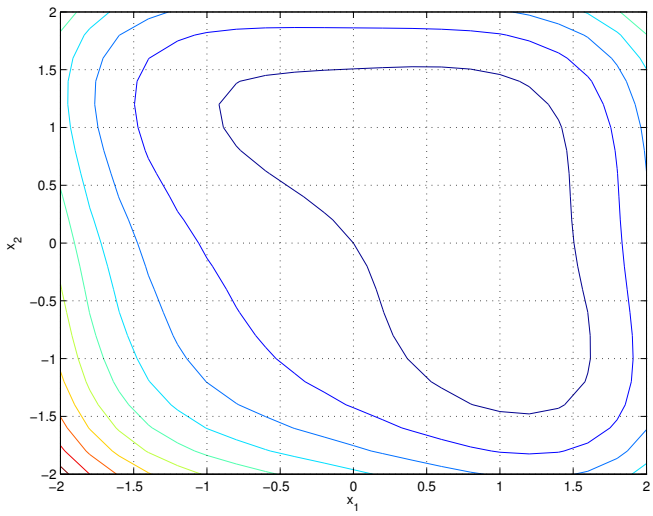


Figura 5.15: Curvas de nivel para el ejemplo2.

```
[x1, x2] = meshgrid(-2:0.2:2, -2:0.2:2);
f =-3.5*x1-2*x2-x2.^2 + x1.^4 + 2*x1.*x2 + x2.^4;
contour(x1,x2,f)
grid on
xlabel('x_1')
ylabel('x_2')
```

5.2.7. Propiedades del Método de Newton

Una propiedad importante de este método es que converge en un solo paso cuando se tiene una función estrictamente cuadrática.

Demostración:

Dada una función cuadrática

$$f(x) = \frac{1}{2}x^T Ax + x^T b + c$$

el gradiente lo podemos calcular como $\nabla f(x) = Ax + b$ y el Hessiano es $\nabla^2 f(x) = A$. Si sustituimos en la ecuación 5.13 tenemos

$$= x^{(k)} - A^{-1}(Ax^{(k)} + b)$$

$$x^{(k+1)} = x^{(k)} - x^{(k)} - A^{-1}b$$

Lo que finalmente da como resultado

$$x^{(k+1)} = -A^{-1}b$$

5.2.8. Ejemplo 3

Dada la función $f(x) = 10x_1^2 + x_2^2$ y un punto inicial $x^{(0)} = [1, 2]^T$, calcular el mínimo utilizando el algoritmo de Newton.

Para esta función el gradiente y el Hessiano es:

$$\nabla f(x) = \begin{array}{|c|}\hline 20x_1 \\ \hline 2x_2 \\ \hline \end{array}$$

y el Hessiano es:

$$A(x) = \begin{array}{|c|c|}\hline 20 & 0 \\ \hline 0 & 2 \\ \hline \end{array}$$

Aplicando la ecuación 5.13 tenemos

$$x^{(1)} = \begin{array}{|c|}\hline 1 \\ \hline 2 \\ \hline \end{array} - \begin{array}{|c|c|}\hline 20 & 0 \\ \hline 0 & 2 \\ \hline \end{array}^{-1} \begin{array}{|c|}\hline 20 \\ \hline 4 \\ \hline \end{array} = \begin{array}{|c|}\hline 0 \\ \hline 0 \\ \hline \end{array}$$

5.2.9. Ejemplo 4

Minimizar es $f(x) = x_1 e^{-x_1^2 - x_2^2}$, con un punto inicial $x_1^{(0)} = -0.5$, $x_2^{(0)} = -0.5$ aplicando el Método de Newton.

Primer iteración

$$\begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \end{bmatrix} = \begin{bmatrix} -0.5000 \\ -0.5000 \end{bmatrix} - \begin{bmatrix} 1.5163 & 0.3033 \\ 0.3033 & 0.3033 \end{bmatrix}^{-1} \begin{bmatrix} 0.3033 \\ -0.3033 \end{bmatrix} = \begin{bmatrix} -1.0000 \\ 1.0000 \end{bmatrix}$$

Segunda iteración

$$\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} = \begin{bmatrix} -1.0000 \\ 1.0000 \end{bmatrix} - \begin{bmatrix} 0.2707 & 0.2707 \\ 0.2707 & -0.2707 \end{bmatrix}^{-1} \begin{bmatrix} -0.1353 \\ 0.2707 \end{bmatrix} = \begin{bmatrix} -1.2500 \\ 1.7500 \end{bmatrix}$$

Tercer iteración

$$\begin{bmatrix} x_1^{(3)} \\ x_2^{(3)} \end{bmatrix} = \begin{bmatrix} -1.2500 \\ 1.7500 \end{bmatrix} - \begin{bmatrix} -0.0031 & 0.0729 \\ 0.0729 & -0.1256 \end{bmatrix}^{-1} \begin{bmatrix} -0.0208 \\ 0.0429 \end{bmatrix} = \begin{bmatrix} -1.3535 \\ 2.0314 \end{bmatrix}$$

Cuarta iteración

$$\begin{bmatrix} x_1^{(4)} \\ x_2^{(4)} \end{bmatrix} = \begin{bmatrix} -1.3535 \\ 2.0314 \end{bmatrix} - \begin{bmatrix} -0.0046 & 0.0280 \\ 0.0280 & -0.0507 \end{bmatrix}^{-1} \begin{bmatrix} -0.0069 \\ 0.0142 \end{bmatrix} = \begin{bmatrix} -1.4416 \\ 2.2629 \end{bmatrix}$$

Para ejecutar dar

```
>> Newton(@Gradiente_f1, @Hessiano_f1, [-0.5;-0.5])
```

En la siguiente tabla se muestra como la función $f(x)$ disminuye su valor en el proceso de convergencia

k	$x_1^{(k)}$	$x_2^{(k)}$	$f(x^{(k)})$
0	-0.5000	-0.5000	-0.3033
1	-1.0000	1.0000	-0.1353
2	-1.2500	1.7500	-0.0123
3	-1.3535	2.0314	-0.0035
4	-1.4416	2.2629	-0.0011
5	-1.5206	2.4654	-0.0003
6	-1.5933	2.6481	-0.0001
7	-1.6611	2.8161	-0.0000
8	-1.7251	2.9727	-0.0000
9	-1.7859	3.1198	-0.0000

De los datos de la tabla podemos notar que la función en lugar de disminuir crece. Esta condición se debe a que el punto inicial para el método de Newton esta muy lejos y la aproximación cuadrática, resulta in apropiada. En la figura 5.16 se muestra la función y se puede ver que esta tiene solamente un mínimo.

5.2.10. Problemas de convergencia del Método de Newton

Consideremos el caso de una función donde se tienen múltiples raíces como es el caso de la función $f(x) = (x-3)^5/5$ cuyo primer derivada es $f'(x) = (x-3)^4$. Note que en este

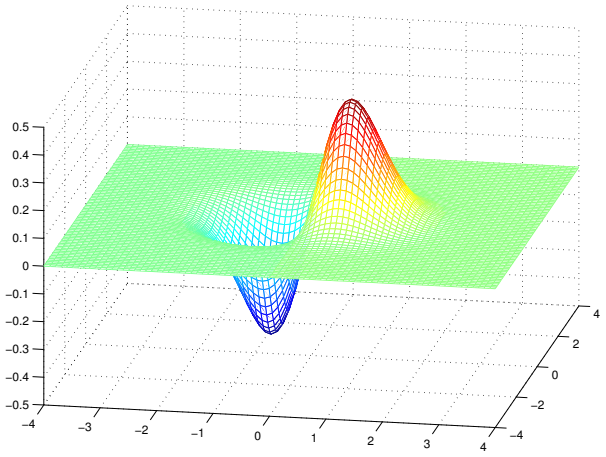


Figura 5.16: Función $f(x) = x_1 e^{-x_1^2 - x_2^2}$.

caso la solución es un polinomio de grado 4 en cuyo caso tendrá cuatro soluciones con $x^* = 3$. Si aplicamos el método de Newton tendremos una convergencia como se muestra a continuación

k	$f(x^{(k)})$
1	2.2500
2	2.4375
3	2.5781
4	2.6836
5	2.7627
6	2.8220
7	2.8665
8	2.8999
9	2.9249
10	2.9437
⋮	⋮
44	3.0000
45	3.0000

Si uno desea aplicar el método a una función arbitraria encontraremos algunas dificultades, esencialmente debido al hecho de que no encontraremos convergencia global. Si el punto inicial $x^{(0)}$ esta muy lejos de x^* el método no podrá converger.

Para comenzar, la aproximación de $f(x)$ dada por $q(x)$ es solamente valida a la vecindad de x_k, el tamaño de paso puede ser controlado a través de una formula iterativa de tipo:

$$x^{(k+1)} = x^{(k)} - \lambda^{(k)} \left[\nabla^2 f(x^{(k)})\right]^{-1} \nabla f(x^{(k)})$$

donde $\lambda^{(k)}$ es un escalar, seleccionado por ejemplo, de tal manera que $||x^{(k+1)} - x^{(k)}||$ no sea muy largo. También podemos seleccionarlo tal que $\lambda^{(k)}$ minimice $\phi(\lambda^{(k)}) = f(x^{(k)} + \lambda^{(k)} d^{(k)})$ en la dirección de:

$$d^{(k)} = -\left[\nabla^2 f(x^{(k)})\right]^{-1} \nabla f(x^{(k)})$$

5.2.11. Ejemplo 5

Minimizar $f(x) = \frac{(x-3)^5}{5}$ utilizando el método de Newton y la modificación planteada utilizando un valor $\lambda^{(k)} = 2.9$. Considere un punto inicial $x^{(0)} = 2$.

k	Newton	Newton con $\lambda = 2.9$
0	2.00000	2.00000
1	2.33333	2.96667
2	2.55556	2.99889
3	2.70370	2.99996
4	2.80247	3.00000
5	2.86831	3.00000
6	2.91221	3.00000
7	2.94147	3.00000
8	2.96098	3.00000
9	2.97399	3.00000
10	2.98266	3.00000
11	2.98844	3.00000

5.2.12. Método de Newton Modificado

Una alternativa propuesta por Ralston y Rabinowitz (1978) es la de definir una nueva función $u(x)$, que es el cociente del gradiente de $g(x)$ y su derivada (en una dimensión)

$$u'(x) = \frac{f'(x)}{f''(x)}$$

Se puede mostrar que esta función tiene las mismas raíces que $f'(x)$ y que la multiplicidad de raíces no afectará. La formulación del método de Newton es:

$$x_{k+1} = x_k - \frac{u'(x)}{u''(x)}$$

La derivada de $u'(x)$ es:

$$u''(x) = \frac{f''(x)f''(x) - f'(x)f'''(x)}{[f''(x)]^2}$$

Sustituyendo esta, tenemos la formulación final del Método de Newton modificado.

$$x^{(k+1)} = x^{(k)} - \frac{f''(x^{(k)})f'(x^{(k)})}{f''(x^{(k)})f''(x^{(k)}) - f'(x^{(k)})f'''(x^{(k)})}$$

```
function y = Newton_1d_modificado(der1f, der2f, der3f, x)
x1 = x;
x2 = x;
for iter = 1:10
    delta = der1f(x1)*der2f(x1)/(der2f(x1)^2 - der1f(x1)*der3f(x1));
    x2 = x1 - delta;
    disp([iter x2]);
    if abs(delta) < 1e-6 break;
    else
        x1 =x2;
    end;
end;
y = x2;
```

Ejemplo

Para la función $f(x) = \frac{(x-3)^5}{5}$ para un valor inicial $x_0 = 2$ la solución es:

La primer derivada es:

$$f'(x) = (x-3)^4$$

La segunda derivada

$$f''(x) = 4(x-3)^3$$

y la tercer derivada

$$f'''(x) = 12(x-3)^2$$

Evaluando en el punto inicial tenemos $f'(2) = 1$, $f''(2) = -4$ y $f'''(2) = 12$ sustituyendo en la iteración de Newton tenemos

$$x^{(1)} = 2 - \frac{(1) \times (-4)}{(4)^2 - (1) * (12)} = 3$$

Note que la solución se da en una sola iteración. Para ejecutar hacer

```
Newton_1d_modificado(@der_fun_3, @der2_fun_3, @der3_fun_3,2)
```

5.3. Optimización lineal restringida (programación lineal) Método Simplex

En programación lineal (o Pl por simplicidad) es un procedimiento de optimización que trata de cumplir con un objetivo como maximizar las utilizares o minimizar el costo, en presencia de restricciones como lo son las fuentes limitadas. El término lineal denota que las funciones matemáticas que representan ambos, la función objetivo y las restricciones son lineales.

5.3.1. Forma estándar

El problema básico de programación lineal consiste en dos partes principales: la función objetivo y un conjunto de restricciones. Para un problema de maximización la función objetivo es por lo general expresada como:

Maximizar

$$Z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

donde c_j es el pago por cada unidad de la j-esima actividad que se lleva a cabo y x_j es la magnitud o cantidad de la j-esima actividad.

Las restricciones se pueden representar en forma general como:

$$a_{i,1}x_1 + a_{i,2}x_2 + \cdots + a_{i,n}x_n \leq b_i$$

donde $a_{i,j}$ cantidad de i-ésima fuente que se consume por cada unidad de j-ésima actividad y b_i cantidad de la i-ésima fuente que ésta disponible.

Suponga que una planta procesadora de gasolina recibe cada semana una cantidad fija de materia prima. Esta última se procesa en dos tipos de gasolina de calidad regular y premium. Estas clases de gasolina son de alta demanda; es decir, se tiene garantizada su venta y se obtiene diferentes utilidades para la compañía. Si embargo su producción involucra ambas restricciones, tiempo y almacenaje en sitio. Por ejemplo, sólo una de las clases se puede producir a la vez, y las instalaciones están abiertas solo 80 horas. Además, existe un limite de almacenamiento para cada uno de los productos. Todos estos factores se en listan en la siguiente tabla:

Recurso	Regular	Premium	Disponibilidad
Materia Prima	7 m^3/tonelada	11 m^3/tonelada	77 m^3/semana
Tiempo de Producción	10 hr/tonelada	8 hr/tonelada	80 hr/semana
Almacenamiento	9 toneladas	6 toneladas	
Aprovechamiento	150 pesos/tonelada	175 pesos/tonelada	

Solución

El ingeniero que opera esta planta debe decidir la cantidad a producir de cada gasolina para maximizar las utilidades. Si las cantidades producidas cada semana de gasolina regular y premium son designadas x_1 y x_2 respectivamente, la ganancia total se puede calcular como

Ganancia total $= 150x_1 + 175x_2$

o escribirla como una función objetivo en programación lineal

Maximiza $Z = 150x_1 + 175x_2$

Las restricciones se pueden desarrollar en una forma similar. Por ejemplo el total de gasolina cruda utilizada se puede calcular como

Total de materia prima utilizada $= 7x_1 + 11x_2$

Este total no puede exceder el abastecimiento disponible de 77 m3/semana, así que la restricción se puede representar como

$$7x_1 + 11x_2 \leq 77$$

Las restricciones restantes se pueden desarrollar en una forma similar, la formulación total resultante para la PL está dada por

Maximizar $Z = 150x_1 + 175x_2$

Sujeta a

$$\begin{aligned} 7x_1 + 11x_2 &\leq 77 \\ 10x_1 + 8x_2 &\leq 80 \\ x_1 &\leq 9 \\ x_2 &\leq 6 \\ x_1 &\geq 0 \\ x_2 &\geq 0 \end{aligned}$$

5.3.2. Método Simplex

El método Simplex se basa en la suposición de que la solución óptima estará en un punto extremo. Así, el procedimiento debe debe de ser capaz de discernir si durante la solución a un problema ocurre un punto extremo. Para realizar esto, las ecuaciones con restricciones se reformulan como igualdades por medio de la introducción de las llamadas variables de holgura.

Variables de Holgura. Como lo indica su nombre, una variable de holgura mide cuanto de una fuente restringida esta disponible es decir, cuanto holgura esta disponible. Por ejemplo, recuerde la fuente restringida que se utilizo en el ejemplo:

$$7x_1 + 11x_2 \leq 77$$

Se puede definir una variable h_1 como la cantidad de gasolina cruda que no se usa para una nivel de producción en particular (x_1, x_2). Si esta cantidad se agrega al lado izquierdo de la restricción, forma la relación exacta.

$$7x_1 + 11x_2 + h_1 = 77$$

Ahora se reconoce lo que la variable de holgura nos indicaba. Si esta es positiva, significa que tiene algo de holgura para esta restricción. Esto es, se cuenta con algo más de recursos que no han sido utilizado por completo. Si es negativa nos indica que nos hemos excedido en la restricción. Finalmente, si es cero denota que se cumplió exactamente con la restricción. Es decir, se dispuso de todo el recurso puesto que esta es la condición donde las líneas de restricción se interceptan, la variable de holgura proporciona un medio para determinar puntos extremos.

Una variable de holgura diferente se desarrolla para cada ecuación restringida resultando en lo que se llama versión completamente aumentada.

$$Z = 150x_1 + 175x_2$$

Sujeta a

$$\begin{array}{rrrrrrr} 7x_1 & +11x_2 & +h_1 & & & & = 77 \\ 10x_1 & +8x_2 & & +h_2 & & & = 80 \\ x_1 & & & & +h_3 & & = 9 \\ & x_2 & & & & +h_4 & = 6 \end{array}$$

Advierta como se han formulado de igual modo las cuatro ecuaciones, de tal manera que las incógnitas están alineadas en las columnas. Se hizo así para resaltar que ahora se trata de un sistema de ecuaciones algebraicas lineales.

Los pasos del Método Simplex son los siguientes:

1. Utilizando la forma estándar, determinar una solución básica factible inicial igualando a las $n - m$ variables igual a cero (el origen).

2. Seleccionar la variable de entrada de las variables no básicas que al incrementar su valor pueda mejorar el valor en la función objetivo. Cuando no exista esta situación, la solución actual es la óptima; si no, ir al siguiente paso. Esta selección se hada tomando el j-ésimo coeficiente de la función objetivo con el coeficiente c_j mas grande.

3. Seleccionar la variable de salida de las variables básicas actuales. Tomamos la i-ésima variable tal que $b_i/a_{i,j}$ tenga el coeficiente mas pequeño no negativo.

4. Determinar la nueva solución al hacer la variable de entrada básica y la variable de salida no básica, ir al paso 2 (actualizar). La solución se obtiene haciendo una iteración de Gauss-Jordan.

La implementación en Matlab es:

```
function x = Simplex(C, A, b)
N = length(b);
B  = [A,  diag(ones(N,1)), b];
B = [B; C, zeros(1, N+1)]
M = length(B(1,:));

while 1
    [mm, m] = max(B(N+1,:)) ;
    [mn, n] = imin(B(1:N, M)./B(1:N, m));

    if mm == 0
        break;
    end;

    B(n,:) = B(n, :)./B(n,m);

    for k = 1: N+1
        if n ~= k
            B(k,:) = B(k,:) - B(n,:)*B(k,m);
        end;
    end;
end;
x=B;
```

5.3.3. Ejemplo 1

Mostrar la región de factibilidad del problema de la Gasolina planteado y resolver utilizando el método de Simplex.

$$Z = 150x_1 + 175x_2$$

Sujeta a

$$\begin{array}{rrrrrrr} 7x_1 & +11x_2 & +h_1 & & & & = 77 \\ 10x_1 & +8x_2 & & +h_2 & & & = 80 \\ x_1 & & & & +h_3 & & = 9 \\ & x_2 & & & & +h_4 & = 6 \end{array}$$

Para determinar la región de factibilidad se corrió el siguiente código en matlab y en la Figura 5.17 se muestra en verde la región de factibilidad, las restricciones en negro y en rojo la función objetivo con $Z = 1000$.

```
hold off;

[x1,x2] = meshgrid(-1:0.1:10, -1:0.1:10);
f1 = 7*x1 + 11*x2 - 77;
f2 = 10*x1 + 8*x2 - 80;
f3 = x1 - 9;
f4 = x2 - 6;
f5 = x1;
f6 = x2;

Z = 150*x1 + 175*x2 - 1000;

contour(x1, x2, f1, [0,0], 'k');
hold on;
contour(x1, x2, f2, [0,0], 'k');
contour(x1, x2, f3, [0,0], 'k');
contour(x1, x2, f4, [0,0], 'k');
contour(x1, x2, f5, [0,0], 'k');
contour(x1, x2, f6, [0,0], 'k');
contour(x1, x2, Z, [0,0], 'r');
xlabel('x_1');
ylabel('x_2');
```

Comenzamos por escribir la forma estándar del problema seleccionando como solución básica factible $x_1 = 0$, $x_2 = 0$, $h_1 = 77$, $h_2 = 80$, $h_3 = 9$ y $h_4 = 6$. Note que las variables

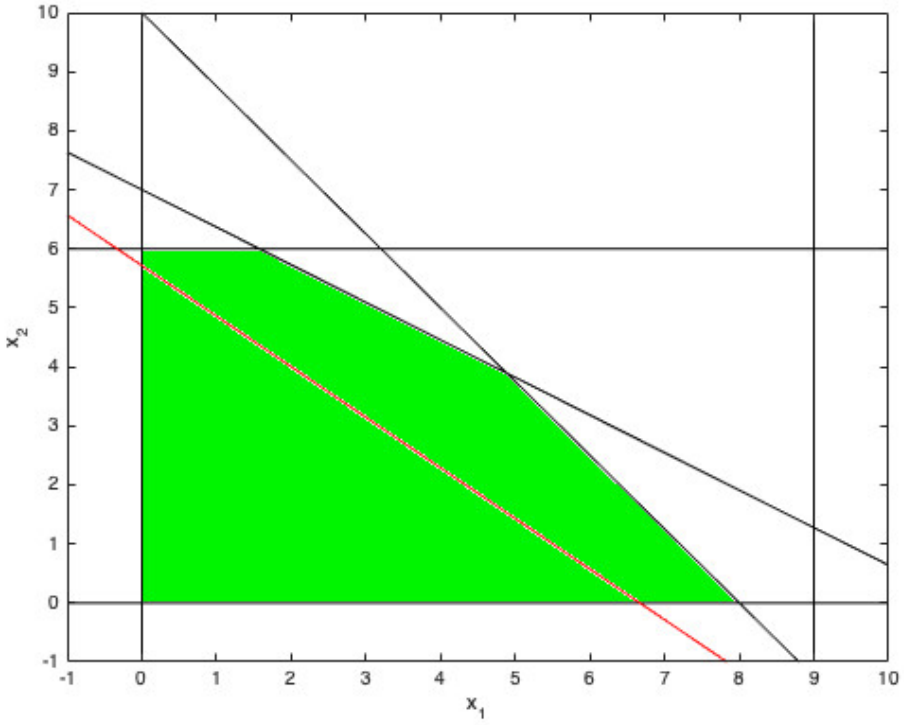


Figura 5.17: Para el ejemplo 1, en verde área de Factibilidad con la función objetivo en rojo para Z = 100.

de holgura tiene los valores que hacen las desigualdades ver como igualdades

$$\left[\begin{array}{rr|cccc|r} 7.00 & 11.00 & 1 & 0 & 0 & 0 & 77.00 \\ 10.00 & 8.00 & 0 & 1 & 0 & 0 & 80.00 \\ 1.00 & 0 & 0 & 0 & 1 & 0 & 9.00 \\ 0 & 1.00 & 0 & 0 & 0 & 1 & 6.00 \\ \hline 150.00 & 175.00 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

Note que en nuestra formulación estándar la función objetivo aparece con los coeficientes negativos, por esta razón repetiremos el procedimiento hasta que no tengamos ningún término negativo en el renglón correspondiente a la función de costo.

Primer iteración

Dado que el costo mayor es 175 comenzamos por introducir la variable x_2 por lo tanto la columna es la 2 y sacar la variable de holgura h_4 dado que esta restricción es la mas cercana del punto básico. Por lo tanto el pivote es 4, 2

$$\left[\begin{array}{rr|cccc|r|r} 7.00 & 11.00 & 1 & 0 & 0 & 0 & 77.00 & 7 \\ 10.00 & 8.00 & 0 & 1 & 0 & 0 & 80.00 & 10 \\ 1.00 & 0 & 0 & 0 & 1 & 0 & 9.00 & \infty \\ 0 & 1.00 & 0 & 0 & 0 & 1 & 6.00 & 6 \\ \hline 150.00 & 175.00 & 0 & 0 & 0 & 0 & 0 & \end{array}\right]$$

Una vez aplicado el método de Gauss Jordan tenemos

$$\left[\begin{array}{rr|ccccc|r} 7.00 & 0 & 1.00 & 0 & 0 & -11.00 & 11.00 \\ 10.00 & 0 & 0 & 1.00 & 0 & -8.00 & 32.00 \\ 1.00 & 0 & 0 & 0 & 1.00 & 0 & 9.00 \\ 0 & 1.00 & 0 & 0 & 0 & 1.00 & 6.00 \\ \hline 150.00 & 0 & 0 & 0 & 0 & -175.00 & -1050.00 \end{array}\right]$$

Segunda Iteración

La siguiente variable a introducir es x_1 dado que el mayor de los costos es 150 y sacar la variable de holgura h_1 dado que es la mas cercana de nuestro punto extremo. Por lo tanto tenemos el elemento 1, 1 como pivote

$$\left[\begin{array}{cc|ccccc|c} 7.00 & 0 & 1.00 & 0 & 0 & -11.00 & 11.00 & 1.57 \\ 10.00 & 0 & 0 & 1.00 & 0 & -8.00 & 32.00 & 3.2 \\ 1.00 & 0 & 0 & 0 & 1.00 & 0 & 9.00 & 9 \\ 0 & 1.00 & 0 & 0 & 0 & 1.00 & 6.00 & \infty \\ \hline 150.00 & 0 & 0 & 0 & 0 & -175.00 & -1050.00 & \end{array}\right]$$

Aplicando el método de Gauss-Jordan en el elemento 1,1 tenemos

$$\left[\begin{array}{cc|ccccc} 1.00 & 0 & 0.14 & 0 & 0 & -1.57 & 1.57 \\ 0 & 0 & -1.43 & 1.00 & 0 & 7.71 & 16.29 \\ 0 & 0 & -0.14 & 0 & 1.00 & 1.57 & 7.43 \\ 0 & 1.00 & 0 & 0 & 0 & 1.00 & 6.00 \\ \hline 0 & 0 & -21.43 & 0 & 0 & 60.71 & -1285.71 \end{array}\right]$$

Tercer iteración

En la función de Costo tenemos un término positivo de 60.71 corresponden a la variable de holgura h_4, por lo tanto introducimos la variable h_4 y sacamos la variable h_2. Por lo tanto el pivote para esta iteración es 2, 6

$$\left[\begin{array}{cc|ccccc|c} 1.00 & 0 & 0.14 & 0 & 0 & -1.57 & 1.57 & -1.00 \\ 0 & 0 & -1.43 & 1.00 & 0 & 7.71 & 16.29 & 2.11 \\ 0 & 0 & -0.14 & 0 & 1.00 & 1.57 & 7.43 & 4.73 \\ 0 & 1.00 & 0 & 0 & 0 & 1.00 & 6.00 & 6 \\ \hline 0 & 0 & -21.43 & 0 & 0 & 60.71 & -1285.71 & \end{array}\right]$$

Una vez eliminada la variable de holgura h_2 tenemos

$$\left[\begin{array}{cc|cccc|c} 1.00 & 0 & -0.15 & 0.20 & 0 & 0 & 4.89 \\ 0 & 0 & -0.19 & 0.13 & 0 & 1.00 & 2.11 \\ 0 & 0 & 0.15 & -0.20 & 1.00 & 0 & 4.11 \\ 0 & 1.00 & 0.19 & -0.13 & 0 & 0 & 3.89 \\ \hline 0 & 0 & -10.19 & -7.87 & 0 & 0 & -1413.89 \end{array}\right]$$

Dado que no tenemos ningún termino de costo negativo, la solución la podemos obtener de la formulación de la siguiente manera. Para x_1 la solución esta en el renglón 1 dado que hay un 1 en la posición 1,1 y es $x_1 = 4.89$. Para x_2 el 1 esta en la posición 4 por lo tanto la solución es $x_2 = 3.89$. Para h_1 no tenemos un 1 en la columna correspondiente y de igual manera para h_2, por lo tanto tenemos $h_1 = 0$ y $h_2 = 0$. Para h_3 tenemos un 1 en el renglón 3 por lo tanto $h_3 = 4.11$. Finalmente para h_4 tenemos un 1 en la posición

2 por lo tanto $h_4 = 2.11$. El costo total es $Z = 1413.89$ y lo podemos verificar haciendo $Z = 150 \times 4.89 + 175 \times 3.89 = 1413.89$. La solución en Matlab se calcula como

```
>> B = Simplex([150 175], [7 11; 10 8; 1 0; 0 1], [77; 80; 9; 6])

B =

     7.00    11.00     1.00        0        0        0    77.00
    10.00     8.00        0     1.00        0        0    80.00
     1.00        0        0        0     1.00        0     9.00
        0     1.00        0        0        0     1.00     6.00
   150.00   175.00        0        0        0        0        0


B =

     1.00        0    -0.15     0.20        0        0     4.89
        0        0    -0.19     0.13        0     1.00     2.11
        0        0     0.15    -0.20     1.00        0     4.11
        0     1.00     0.19    -0.13        0        0     3.89
        0        0   -10.19    -7.87        0        0 -1413.89
```

5.3.4. Ejemplo 2

Maximizar la función

$$
\begin{aligned}
Z &= 6x_1 + 4x_2 \\
&\text{Sujeta a} \\
2x_1 + 2x_2 &\leq 160 \\
x_1 + 2x_2 &\leq 120 \\
4x_1 + 2x_2 &\leq 280
\end{aligned}
$$

En la Figura 5.18, se muestra en verde la región de factibilidad y en rojo la función objetivo para $Z = 350$. El código para trazar la figura se muestra a continuacion:

```
hold off;

[x1,x2] = meshgrid(-10:1:120, -10:1:120);
f1 = 2*x1 + 2*x2 - 160;
f2 = x1 + 2*x2 -120;
```

```
f3 = 4*x1 + 2*x2 -280;
f4 = x1;
f5 = x2;

Z = 6*x1 + 4*x2 - 350;

contour(x1, x2, f1, [0,0], 'k');
hold on;
contour(x1, x2, f2, [0,0], 'k');
contour(x1, x2, f3, [0,0], 'k');
contour(x1, x2, f4, [0,0], 'k');
contour(x1, x2, f5, [0,0], 'k');
contour(x1, x2, Z, [0,0], 'r');
xlabel('x_1');
ylabel('x_2');
```

Su formulación estándar es:

$$\left[\begin{array}{cc|ccc|c} 2.00 & 2.00 & 1.00 & 0 & 0 & 160.00 \\ 1.00 & 2.00 & 0 & 1.00 & 0 & 120.00 \\ 4.00 & 2.00 & 0 & 0 & 1.00 & 280.00 \\ \hline 6.00 & 4.00 & 0 & 0 & 0 & 0 \end{array}\right]$$

Primer iteración

Tomamos como pivote el renglón 3 y columna 1

$$\left[\begin{array}{cc|ccc|c} 2.00 & 2.00 & 1.00 & 0 & 0 & 160.00 \\ 1.00 & 2.00 & 0 & 1.00 & 0 & 120.00 \\ 4.00 & 2.00 & 0 & 0 & 1.00 & 280.00 \\ 6.00 & 4.00 & 0 & 0 & 0 & 0 \end{array}\right] \begin{array}{c} 80.0 \\ 120.0 \\ 70 \\ \\ \end{array}$$

Realizada la eliminación de la variable x_1 tenemos

$$\left[\begin{array}{cc|ccc|c} 0 & 1.00 & 1.00 & 0 & -0.50 & 20.00 \\ 0 & 1.50 & 0 & 1.00 & -0.25 & 50.00 \\ 1.00 & 0.50 & 0 & 0 & 0.25 & 70.00 \\ \hline 0 & 1.00 & 0 & 0 & -1.50 & -420.00 \end{array}\right]$$

Segunda Iteración

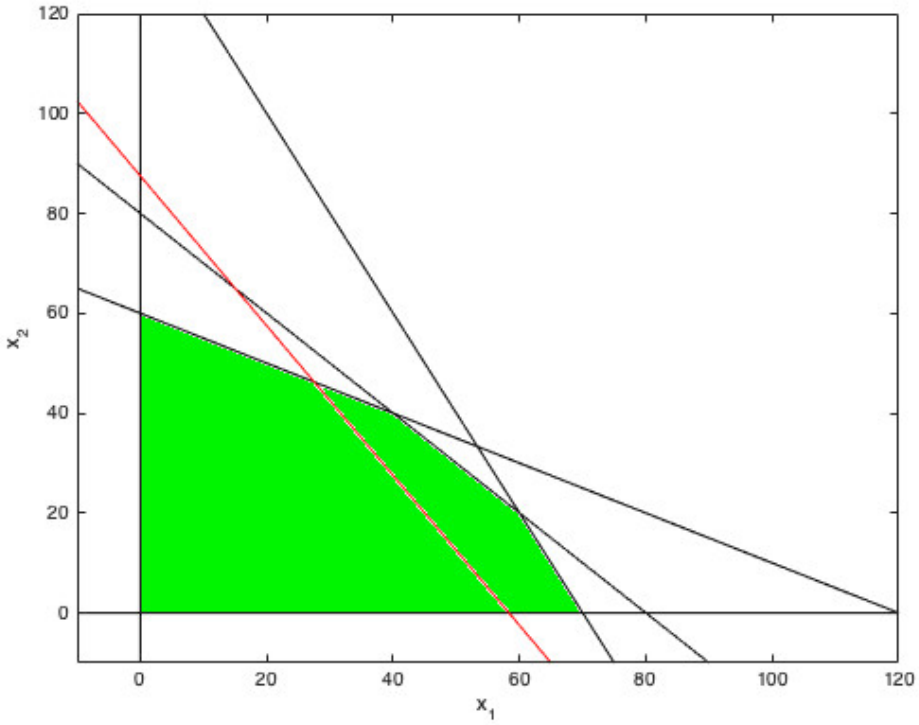


Figura 5.18: Ejemplo 2, Área de Factibilidad en verde y en rojo la función objetivo con Z=350.

Seleccionamos como pivote al renglón 1 columna 2.

$$\left[\begin{array}{cc|ccc|c} 0 & 1.00 & 1.00 & 0 & -0.50 & 20.00 \\ 0 & 1.50 & 0 & 1.00 & -0.25 & 50.00 \\ 1.00 & 0.50 & 0 & 0 & 0.25 & 70.00 \\ \hline 0 & 1.00 & 0 & 0 & -1.50 & -420.00 \end{array}\right] \begin{array}{c} 20.00 \\ 33.33 \\ 70.00 \\ \\ \end{array}$$

Realizada la eliminación de x_2 tenemos

$$\left[\begin{array}{cc|ccc|c} 0 & 1.00 & 1.00 & 0 & -0.50 & 20.00 \\ 0 & 0 & -1.50 & 1.00 & 0.50 & 20.00 \\ 1.00 & 0 & -0.50 & 0 & 0.50 & 60.00 \\ \hline 0 & 0 & -1.00 & 0 & -1.00 & -440.00 \end{array}\right]$$

Dado que no tenemos términos negativos en el renglón de costos terminamos con solución $x_1 = 60$, $x_2 = 20$, $h_1 = 0$, $h_2 = 20$, $h_3 = 0$ y costo $Z = 440$

Para resolver en Matlab hacer

```
>> Simplex([6 4], [2 2; 1 2; 4 2], [160; 120; 280])

B =

     2     2     1     0     0   160
     1     2     0     1     0   120
     4     2     0     0     1   280
     6     4     0     0     0     0


ans =

         0    1.0000    1.0000         0   -0.5000   20.0000
         0         0   -1.5000    1.0000    0.5000   20.0000
    1.0000         0   -0.5000         0    0.5000   60.0000
         0         0   -1.0000         0   -1.0000 -440.0000

>>
```

5.3.5. Ejemplo 3

Maximizar la función

$$
\begin{aligned}
Z &= 3x_1 + 4x_2 \\
&\texttt{Sujeta a} \\
2x_1 + 3x_2 &\leq 1200 \\
2x_1 + x_2 &\leq 1000 \\
4x_2 &\leq 800
\end{aligned}
$$

La formulación utilizando Simplex queda

$$
\left[\begin{array}{cc|ccc|c}
2.00 & 3.00 & 1.00 & 0 & 0 & 1200.00 \\
2.00 & 1.00 & 0 & 1.00 & 0 & 1000.00 \\
0 & 4.00 & 0 & 0 & 1.00 & 800.00 \\
\hline
3.00 & 4.00 & 0 & 0 & 0 & 0
\end{array}\right]
$$

Primer Iteración

Tomamos como pivote el renglón 3 y columna 2

$$
\left[\begin{array}{cc|ccc|c}
2.00 & 3.00 & 1.00 & 0 & 0 & 1200.00 \\
2.00 & 1.00 & 0 & 1.00 & 0 & 1000.00 \\
0 & 4.00 & 0 & 0 & 1.00 & 800.00 \\
\hline
3.00 & 4.00 & 0 & 0 & 0 & 0
\end{array}\right]
\begin{array}{c}
400.00 \\ 1000.00 \\ 200.00 \\ \\
\end{array}
$$

Reduciendo tememos como resultado

$$
\left[\begin{array}{cc|ccc|c}
2.00 & 0 & 1.00 & 0 & -0.75 & 600.00 \\
2.00 & 0 & 0 & 1.00 & -0.25 & 800.00 \\
0 & 1.00 & 0 & 0 & 0.25 & 200.00 \\
\hline
3.00 & 0 & 0 & 0 & -1.00 & -800.00
\end{array}\right]
$$

Segunda Iteración

Tomamos como pivote el renglón 1 y columna 1

$$
\left[\begin{array}{cc|ccc|c}
2.00 & 0 & 1.00 & 0 & -0.75 & 600.00 \\
2.00 & 0 & 0 & 1.00 & -0.25 & 800.00 \\
0 & 1.00 & 0 & 0 & 0.25 & 200.00 \\
\hline
3.00 & 0 & 0 & 0 & -1.00 & -800.00
\end{array}\right]
\begin{array}{c}
300.00 \\ 400.00 \\ \infty \\ \\
\end{array}
$$

Aplicando la eliminación tenemos

$$\left[\begin{array}{cc|ccc|c} 1.00 & 0 & 0.50 & 0 & -0.38 & 300.00 \\ 0 & 0 & -1.00 & 1.00 & 0.50 & 200.00 \\ 0 & 1.00 & 0 & 0 & 0.25 & 200.00 \\ \hline 0 & 0 & -1.50 & 0 & 0.12 & -1700.00 \end{array}\right]$$

Tercer iteración

Tomamos como pivote el renglón 2 la columna 5

$$\left[\begin{array}{cc|ccc|c} 1.00 & 0 & 0.50 & 0 & -0.38 & 300.00 \\ 0 & 0 & -1.00 & 1.00 & 0.50 & 200.00 \\ 0 & 1.00 & 0 & 0 & 0.25 & 200.00 \\ \hline 0 & 0 & -1.50 & 0 & 0.12 & -1700.00 \end{array}\right] \begin{array}{c} -789.47 \\ 400.00 \\ 800.00 \\ \\ \end{array}$$

$$\left[\begin{array}{cc|ccc|c} 1.00 & 0 & -0.25 & 0.75 & 0 & 450.00 \\ 0 & 0 & -2.00 & 2.00 & 1.00 & 400.00 \\ 0 & 1.00 & 0.50 & -0.50 & 0 & 100.00 \\ \hline 0 & 0 & -1.25 & -0.25 & 0 & -1750.00 \end{array}\right]$$

La solución es $x_1 = 450$, $x_2 = 100$ con un costo $Z = 1750.00$

La solución en Matlab se calcula haciendo

```
>> B = Simplex([3 4], [2 3; 2 1; 0 4], [1200; 1000; 800])

B =

     2.00     3.00     1.00        0        0  1200.00
     2.00     1.00        0     1.00        0  1000.00
        0     4.00        0        0     1.00   800.00
     3.00     4.00        0        0        0        0


B =

     1.00        0    -0.25     0.75        0   450.00
        0        0    -2.00     2.00     1.00   400.00
        0     1.00     0.50    -0.50        0   100.00
        0        0    -1.25    -0.25        0 -1750.00
```

En la Figura 5.19 se muestra la región de factibilidad y la función objetivo en rojo para un valor $Z = 1400$. El código para determinar esta figura en matlab es:

```
hold off;

[x1,x2] = meshgrid(-100:1:600, -100:1:600);
f1 = 2*x1 + 3*x2 - 1200;
f2 = 2*x1 + x2 -1000;
f3 = 4*x2 - 800;
f4 = x1;
f5 = x2;

Z = 3*x1 + 4*x2 - 1400;

contour(x1, x2, f1, [0,0], 'k');
hold on;
contour(x1, x2, f2, [0,0], 'k');
contour(x1, x2, f3, [0,0], 'k');
contour(x1, x2, f4, [0,0], 'k');
contour(x1, x2, f5, [0,0], 'k');
contour(x1, x2, Z, [0,0], 'r');
xlabel('x_1');
ylabel('x_2');
```

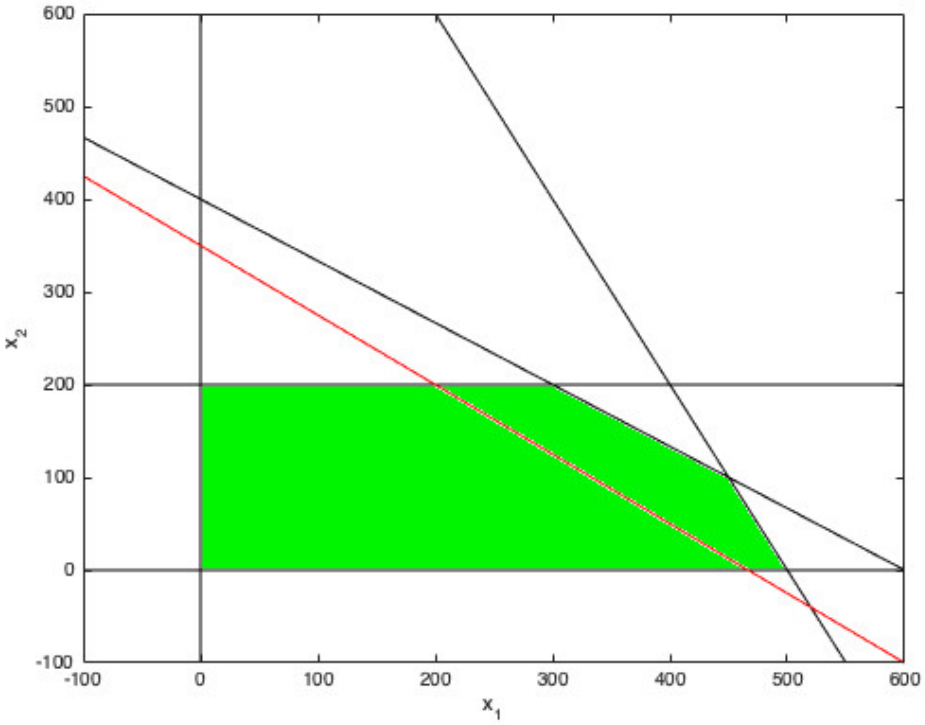


Figura 5.19: Ejemplo 3, área de Factibilidad en verde y en rojo función objetivo con $Z = 1400$.

Ajuste de curvas

6.1. Regresión lineal por el método de mínimos cuadrados

El ejemplo más simple de una aproximación por mínimos cuadrados es mediante el ajuste de un conjunto de pares de observaciones: $[x_1, y_1]$, $[x_2, y_2]$, $[x_3, y_3]$,..., $[x_n, y_n]$ a una línea recta. La expresión matemática de la línea recta es:

$$y = a_0 + a_1 x + e$$

donde a_0 y a_1, son los coeficientes que representan el cruce con el eje y y la pendiente de la línea y e representa el error de nuestra aproximación. En la figura **??**, se muestra un ejemplo de puntos a los que se les desea ajustar a una linea recta.

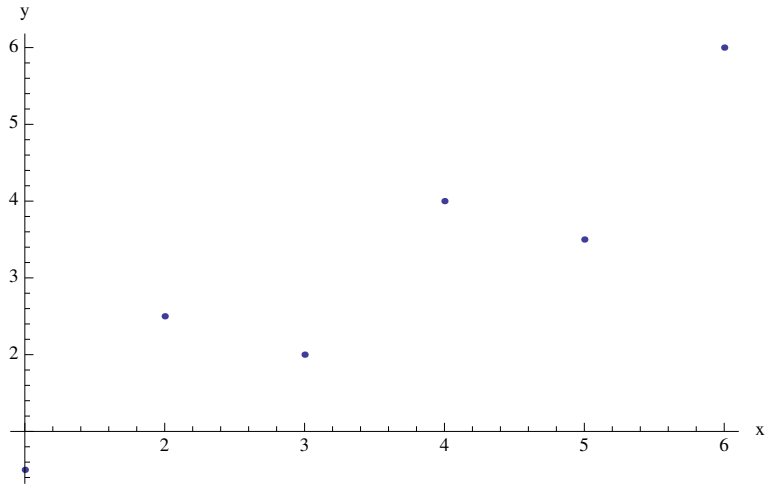


Figura 6.20: Un conjunto de puntos

Una estrategia, para ajustar a la mejor línea, es minimizar la suma al cuadrado de los errores para todos los datos disponibles

$$E(a) = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2$$

Esta misma ecuación la podemos escribir en forma matricial como

$$E(a) = [e_1, e_2, \cdots, e_n] \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{bmatrix}$$

Los elementos del vector de error e_i en forma matricial queda como

$$e_i = y_i - [1, x_i] \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

y en general

$$\begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ \vdots \\ e_n \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{bmatrix} - \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix}$$

En forma compacta tenemos que $e = y–Ma$, donde M es la matriz de coordenadas en x y a el vector de parámetros.

6.1.1. Ajuste por mínimos cuadrados

Si queremos encontrar el vector de parámetros a que minimiza nuestra suma de cuadrados, tenemos que calcular la derivada de la función de error respecto al vector de parámetros e igualar a cero.

$$\frac{\partial E(a)}{\partial a} = \frac{\partial}{\partial a}[y - Ma]^T[y - Ma] = 0$$
$$\frac{\partial E(a)}{\partial a} = -2M^T[y - Ma] = 0$$

El valor del vector de parámetros a los calculamos resolviendo el siguiente sistema de ecuaciones

$$[M^T M]a = M^T y$$

Es común encontrar la solución de este sistema como:

$$\left[\begin{array}{c|c} \sum_{i=1}^{n} 1 & \sum_{i=1}^{n} x_i \\ \hline \sum_{i=1}^{n} x_i & \sum_{i=1}^{n} x_i^2 \end{array}\right] \left[\begin{array}{c} a_0 \\ \hline a_1 \end{array}\right] = \left[\begin{array}{c} \sum_{i=1}^{n} y_i \\ \hline \sum_{i=1}^{n} x_i y_i \end{array}\right]$$

La implementación en Matlab es:

```
function a = AjustePolinomial (x, y, grado)

    N = length(x);
    M = ones(N, 1);

    for k=1:grado
        M = [M,x.^k];
    end

    a=inv(M'*M)*M'*y;
end
```

6.1.2. Ejemplo 1

Hacer el ajuste a una línea recta de los siguientes valores

x	y
1.00	0.50
2.00	2.50
3.00	2.00
4.00	4.00
5.00	3.50
6.00	6.00
7.00	5.50

La matriz M queda como

$$M = \begin{bmatrix} 1 & 1.00 \\ 1 & 2.00 \\ 1 & 3.00 \\ 1 & 4.00 \\ 1 & 5.00 \\ 1 & 6.00 \\ 1 & 7.00 \end{bmatrix}$$

$$M^T M = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1.00 & 2.00 & 3.00 & 4.00 & 5.00 & 6.00 & 7.00 \end{bmatrix} \begin{bmatrix} 1 & 1.00 \\ 1 & 2.00 \\ 1 & 3.00 \\ 1 & 4.00 \\ 1 & 5.00 \\ 1 & 6.00 \\ 1 & 7.00 \end{bmatrix} = \begin{bmatrix} 7.00 & 28.00 \\ 28.00 & 140.00 \end{bmatrix}$$

$$M^T y = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1.00 & 2.00 & 3.00 & 4.00 & 5.00 & 6.00 & 7.00 \end{bmatrix} \begin{bmatrix} 0.50 \\ 2.50 \\ 2.00 \\ 4.00 \\ 3.50 \\ 6.00 \\ 5.50 \end{bmatrix} = \begin{bmatrix} 24.00 \\ 119.50 \end{bmatrix}$$

Aplicando las formulas anteriores, tenemos que el sistema de ecuaciones a resolver es

$$\begin{bmatrix} 7.00 & 28.00 \\ 28.00 & 140.00 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \end{bmatrix} = \begin{bmatrix} 24.00 \\ 119.50 \end{bmatrix}$$

La solución es $a = [0.07142, 0.8392]$ y en la figura 6.21 se muestra el ajuste encontrado

Para ejecutar dar

```
AjustePolinomial([1,2,3,4,5,6,7]', [0.5, 2.5, 2, 4, 3.5, 6, 5.5]',1)
```

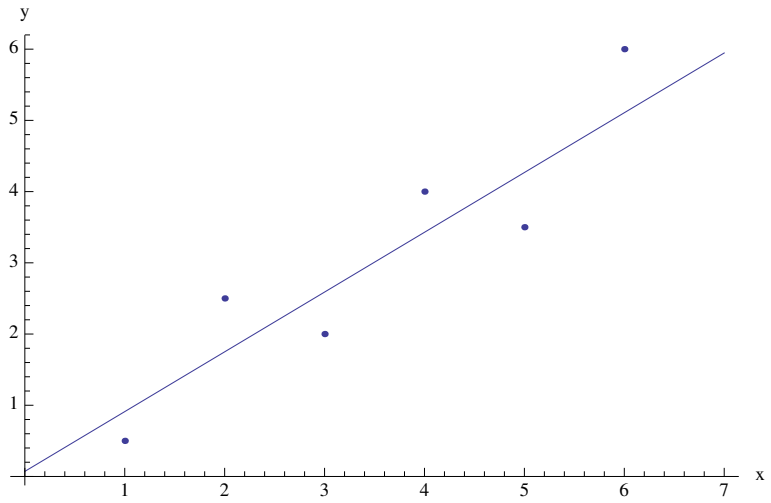


Figura 6.21: Un conjunto de puntos y la linea recta ajustada

6.1.3. Regresión polinomial

Podemos generalizar el caso de la regresión lineal y extenderla a cualquier polinomio de orden m, hacemos la siguiente representación.

$$\begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^m \\ 1 & x_2 & x_2^2 & x_2^3 & \cdots & x_2^m \\ 1 & x_3 & x_3^2 & x_3^3 & \cdots & x_3^m \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_n^2 & x_n^3 & \cdots & x_n^m \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_m \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}$$

La cual podemos escribir de manera compacta como

$$Ma = y$$

Si pre-multiplicamos ambos lados de la ecuación tenemos

$$[M^T M]a = M^T y \tag{6.14}$$

El sistema se resuelve para a o simplemente se calcula haciendo

$$a = [M^T M]^{-1}[M^T y]$$

6.1.4. Ejemplo 2

Ajustar a un polinomio de segundo orden los datos en la siguiente tabla.

x	y
0.00	2.10
1.00	7.70
2.00	13.60
3.00	27.20
4.00	40.90
5.00	61.10

En este caso por ser un polinomio de segundo grado tenemos una matriz M dada por

$$M = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \\ 1 & 4 & 16 \\ 1 & 5 & 25 \end{bmatrix}$$

El sistema de ecuaciones que debemos resolver es:

$$\begin{bmatrix} 6.00 & 15.00 & 55.00 \\ 15.00 & 55.00 & 225.00 \\ 55.00 & 225.00 & 979.00 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} = \begin{bmatrix} 152.60 \\ 585.60 \\ 2488.80 \end{bmatrix}$$

La solución del sistema es $a = [2.4785, 2.3592, 1.8607]$ y el ajuste da como resultado el polinomio $p(x) = 2.4785 + 2.3592x + 1.8607x2$. En la figura 6.22 se muestra la aproximación a un polinomio de segundo orden.

Para correr en Matlab hacer

```
AjustePolinomial([0,1,2,3,4,5]', [2.1, 7.70, 13.60, 27.20, 40.90, 61.10]',2)
```

6.2. Interpolación lineal

El modo más simple de interpolación es conectar dos puntos con una línea recta. Esta técnica, llamada interpolación lineal, la podemos representar por la siguiente formulación, la cual se calcula considerando que se tienen dos puntos $P_1 = [x_1, f(x_1)]^T$ y $P_0 = [x_0, f(x_0)]^T$

$$\frac{f(x) - f(x_0)}{x - x_0} = \frac{f(x_1) - f(x_0))}{x_1 - x_0}$$

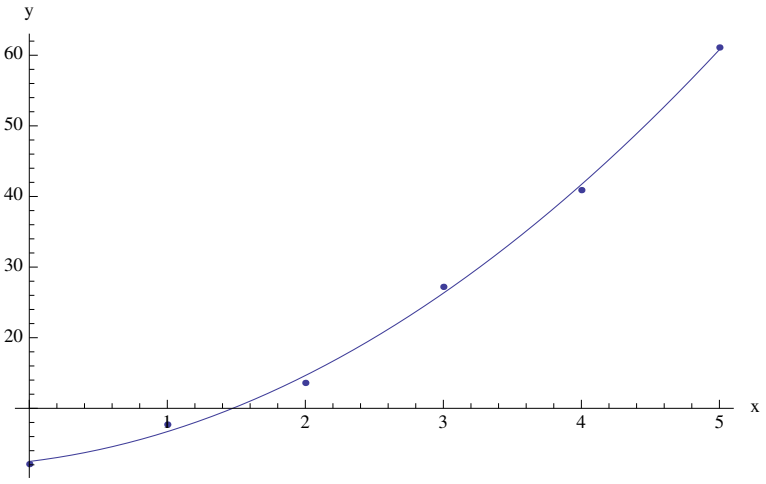


Figura 6.22: Un conjunto de puntos y su aproximación cuadrática

La cual da lugar a la siguiente ecuación

$$f(x) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x - x_0)$$

La implementación para una función de interpolación lineal para un conjunto de puntos es:

```
function y = Interpolacion_Lineal(x0, x, y)

N =length(x);

for k=1:N-1
    if x0 >= x(k) && x0 <= x(k+1) break;
    end
end

y = y(k) + (y(k+1) - y(k))/(x(k+1)-x(k))*(x0-x(k));

end
```

6.2.1. Ejemplo 1

Dados los puntos

x	y
0.00	2.10
1.00	7.70
2.00	13.60
3.00	27.20
4.00	40.90
5.00	61.10

Implementar el código correspondiente para graficar los puntos intermedios a los valores dados utilizando interpolación lineal

```
x = [0,1,2,3,4,5];
y = [2.1, 7.70, 13.60, 27.20, 40.90, 61.10];

x_nva = 0:0.01:5;

N = length(x_nva);
y_nva = zeros(N, 1);

for k=1:N
    y_nva(k) = Interpolacion_Lineal(x_nva(k), x, y);
end;

plot(x_nva, y_nva, 'r-.', x, y, 'k*');
xlabel('x');
ylabel('f(x)');
```

En la figura 6.23, se muestran los puntos correspondientes calculados de acuerdo con el código implementado

6.3. Interpolación cuadrática

El error que se observa en la figura 6.23, se debe a que suponemos que los puntos se unen por lineas. Una manera resolver este problema es suponer una función polinomial cuadrática tomando al menos tres puntos para tal propósito. Una forma de introducir tres puntos en una función cuadrática es mediante la siguiente ecuación

$$q(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1)$$

Para calcular los valores b_0, b_1 y b_2 necesitamos introducir información de tres puntos $[x_0, f(x_0)]$, $[x_1, f(x_1)]$ y $[x_2, f(x_2)]$. Si sustituimos en primer x_0 tenemos que

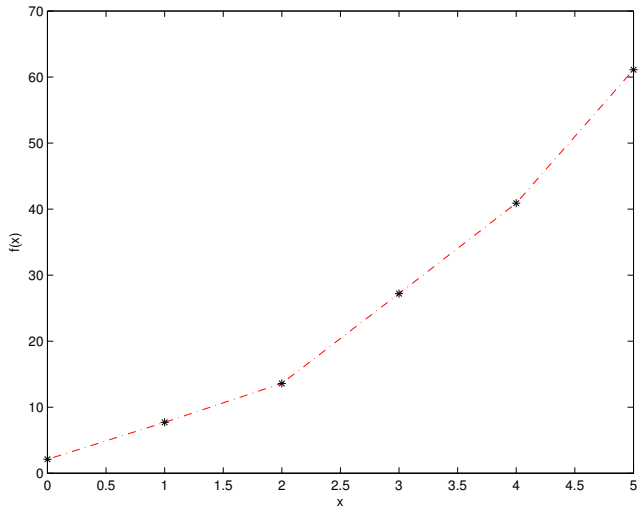


Figura 6.23: Un conjunto de puntos y su interpolación lineal

$$f(x_0) = b_0 + b_1(x_0 - x_0) + b_2(x_0 - x_0)(x - x_1)$$

por lo tanto

$$b_0 = f(x_0)$$

Para calcular b_1 utilizamos el segundo punto así tenemos

$$f(x_1) = b_0 + b_1(x_1 - x_0) + b_2(x_1 - x_0)(x_1 - x_1)$$

despejando tenemos

$$b_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0}$$

Para el calculo de b_2 sustituimos los valores de b_0, b_1 y $[x_2, f(x_2)]$ de la siguiente manera

$$f(x_2) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x_2 - x_0) + b_2(x_2 - x_0)(x_2 - x_1)$$

despejando a b_2

$$b_2 = \frac{f(x_2) - f(x_0) - \frac{f(x_1)-f(x_0)}{x_1-x_0}(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)}$$

$$b_2 = \frac{f(x_2) - f(x_1) + f(x_1) - f(x_0) - \frac{f(x_1)-f(x_0)}{x_1-x_0}(x_2 - x_0)}{(x_2 - x_0)(x_2 - x_1)}$$

$$b_2 = \frac{f(x_2) - f(x_1) + (f(x_1) - f(x_0))\left(1 - \frac{x_2-x_0}{x_1-x_0}\right)}{(x_2 - x_0)(x_2 - x_1)}$$

$$b_2 = \frac{f(x_2) - f(x_1) - (f(x_1) - f(x_0))\left(\frac{x_2-x_1}{x_1-x_0}\right)}{(x_2 - x_0)(x_2 - x_1)}$$

finalmente obtenemos

$$b_2 = \frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2 - x_0}$$

En general podemos extender el método para N puntos haciendo

k	x	$f(x)$	$b_0^{(k)}$	$b_1^{(k)}$	$b_2^{(k)}$
0	x_0	$f(x_0)$	$f(x_0)$	$\frac{f(x_1)-f(x_0)}{x_1-x_0}$	$\frac{\frac{f(x_2)-f(x_1)}{x_2-x_1}-\frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2-x_0}$
1	x_1	$f(x_1)$	$f(x_1)$	$\frac{f(x_2)-f(x_1)}{x_2-x_1}$	$\frac{\frac{f(x_3)-f(x_2)}{x_3-x_2}-\frac{f(x_2)-f(x_1)}{x_2-x_1}}{x_3-x_1}$
2	x_2	$f(x_2)$	$f(x_2)$	$\frac{f(x_3)-f(x_2)}{x_3-x_2}$	$\frac{\frac{f(x_4)-f(x_3)}{x_4-x_3}-\frac{f(x_3)-f(x_2)}{x_3-x_2}}{x_4-x_2}$
3	x_3	$f(x_3)$	$f(x_3)$	$\frac{f(x_4)-f(x_3)}{x_4-x_3}$	
4	x_4	$f(x_4)$	$f(x_4)$		

La función quadratica correspondiente al k-esimo intervalo queda representada por

$$q^{(k)}(x) = b_0^{(k)} + b_1^{(k)}(x - x_0) + b_2^{(k)}(x - x_1)(x - x_2)$$

6.3.1. Ejemplo 1

Calcular el logaritmo de $x = 2$ dados tres puntos

$$\begin{array}{ll} x_0 = 1 & f(x_0) = 0 \\ x_1 = 4 & f(x_1) = 1.386294 \\ x_2 = 6 & f(x_2) = 1.791759 \end{array}$$

Para resolver tenemos

$$b_0 = 0$$

$$b_1 = \frac{f(x_1) - f(x_0)}{x_1 - x_0} = \frac{1.386294 - 0}{4 - 1} = 0.4620981$$

El coeficiente b_2 tenemos

$$b_2 = \frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2 - x_0}$$

$$b_2 = \frac{\frac{1.791759-1.386294}{6-4} - 0.4620981}{6-1} = -0.0518731$$

En Forma Tabular

k	x	$f(x)$	$b_0^{(k)}$	$b_1^{(k)}$	$b_2^{(k)}$
0	1	0.000000	0.000000	0.4620980	−0.0518731
1	4	1.386294	1.386294	0.2027325	
2	6	1.791759	1.791759		

Finalmente para calcular el valor hacemos

$$q(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1)$$

$$q(x) = 0 + 0.4620981(x - 1) - 0.0518731(x - 1)(x - 4)$$

$$q(2) = 0 + 0.4620981(2 - 1) - 0.0518731(2 - 1)(2 - 4) = 0.5658444$$

En Matlab dar

```
Interpolacion_Newton(2, [1, 4, 6] , [0, 1.386294, 1.791759])

ans =

    0.5658
```

6.3.2. Ejemplo 2

Dados los puntos

x	y
0.00	2.10
1.00	7.70
2.00	13.60
3.00	27.20
4.00	40.90
5.00	61.10

En Forma Tabular los coeficientes para interpolar son:

k	x	$f(x)$	$b_0^{(k)}$	$b_1^{(k)}$	$b_2^{(k)}$
0	0.00	2.1000	2.1000	5.6000	0.1500
1	1.00	7.7000	7.7000	5.9000	3.8500
2	2.00	13.6000	13.6000	13.6000	0.0500
3	3.00	27.2000	27.2000	13.7000	3.2500
4	4.00	40.9000	40.9000	20.2000	
5	5.00	61.1000	61.1000		

Las funciones cuadráticas para cada intervalo quedan:

$$
\begin{aligned}
q^{(0)}(x) &= 2.1000 + 5.6000(x-0) + 0.1500(x-0)(x-1) \\
q^{(1)}(x) &= 7.7000 + 5.9000(x-1) + 3.8500(x-1)(x-2) \\
q^{(2)}(x) &= 13.6000 + 13.6000(x-2) + 0.0500(x-2)(x-3) \\
q^{(3)}(x) &= 27.2000 + 13.7000(x-3) + 3.2500(x-3)(x-4)
\end{aligned}
$$

Hacer una interpolación cuadrática de los datos y comparar con la interpolación lineal

```
x = [0,1,2,3,4,5];
y = [2.1, 7.70, 13.60, 27.20, 40.90, 61.10];

x_nva = 0:0.01:5;

grado = 2;

N = length(x_nva);
M = length(x);
y_nva = zeros(N, 1);
```

```
for n=1:N
    for m=1:M-1
        if x_nva(n) >= x(m) && x_nva(n) <= x(m+1) break;
        end;
    end;
    l = m+grado;
    if l > M
        l = M;
    end;


    y_nva(n) = Interpolacion_Newton(x_nva(n), x(m:l), y(m:l));
end;

plot(x_nva, y_nva, 'r-.', x, y, 'k*');
xlabel('x');
ylabel('f(x)');
```

En la figura 6.24, se muestran los puntos correspondientes calculados de acuerdo con el código implementado

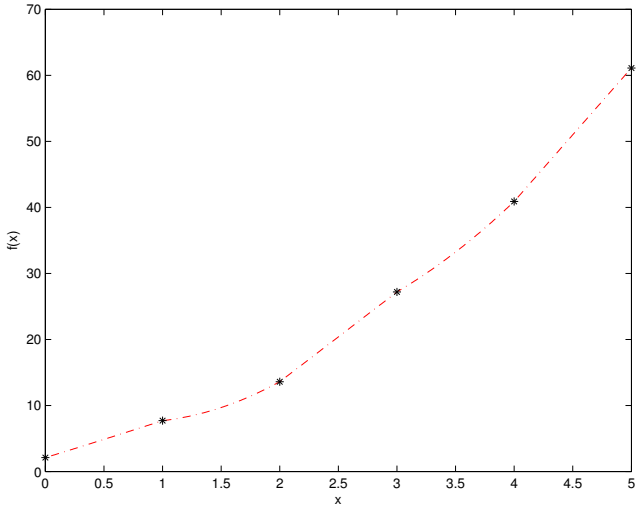


Figura 6.24: Un conjunto de puntos y su interpolación cuadrática

6.4. Formulas de interpolación de Newton

Las formulaciones anteriores pueden ser generalizadas para ajustar un polinomio de n-ésimo orden a $n+1$ datos. El polinomio de n-ésimo orden es

$$f_n(x) = b_0 + b_1(x - x_0) + b_2(x - x0)(x - x_1) + \cdots + b_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}) \quad (6.15)$$

Como se hizo antes con las interpolaciones lineales y cuadráticas, los puntos de los datos evaluaban los coeficientes $b_0, b_1, \cdots, b_n$. Para un polinomio de n-ésimo orden se requiere $n+1$ puntos: $[x_0, f(x_0)]$, $[x_1, f(x_1)]$, ..., $[x_n, f(x_n)]$. Usamos estos datos y las siguientes ecuaciones para evaluar los coeficientes

$$\begin{aligned} b_0 &= f(x_0) \\ b_1 &= f[x1, x_0] \\ b_2 &= f[x_2, x_1, x_0] \\ &\vdots \\ b_n &= f[x_n, x_{n-1}, \cdots, x_1, x_0] \end{aligned}$$

donde las evaluaciones de la función puestas entre paréntesis son diferencias divididas finitas. Por ejemplo, la primera diferencia dividida finita se representa por lo general como

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

La segunda diferencia dividida finita, la cual representa la diferencia de las dos primeras diferencias divididas, se expresa por lo general como

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

En forma similar, la n-ésima diferencia dividida finja es

$$f[x_n, x_{n-1}, \cdots, x_1, x_0] = \frac{f[x_n, x_{n-1}, \cdots, x_1] - f[x_{n-1}, x_{n-2}, \cdots, x_1, x_0]}{x_n - x_0}$$

Estas diferencias pueden usarse para evaluar los coeficientes en la ecuación 6.15, las cuales entonces se sustituirán en la siguiente ecuación, para así obtener el polinomio de interpolación

$$\begin{aligned} f_n(x) &= f(x_0) + f[x_1, x_0](x - x_0) + f[x_2, x_1, x_0](x - x0)(x - x_1) + \\ &\cdots + f[x_n, x_{n-1}, \cdots, x_1, x_0](x - x_0)(x - x_1)\cdots(x - x_{n-1}) \end{aligned}$$

En la siguiente tabla se muestra el procedimiento para el calculo de las diferencias finitas divididas.

i	x_i	$f(x_i)$	Primero	Segundo	Tercero
		b_0	b_1	b_2	b_3
0	x_0	$f(x_0)$	$f[x_1, x_0]$	$f[x_2, x_1, x_0]$	$f[x_3, x_2, x_1, x_0]$
1	x_1	$f(x_1)$	$f[x_2, x_1]$	$f[x_3, x_2, x_1]$	
2	x_2	$f(x_2)$	$f[x_3, x_2]$		
3	x_3	$f(x_3)$			

La implementación de este algoritmo en Matlab es:

```
function yint = Interpolacion_Newton(xi, x, y)
N = length(x);

f = zeros(N, N);

for n=1:N
    f(n, 1) = y(n);
end;

for m = 2:N
    for n = 1:N+1-m
        f(n,m)=(f(n+1,m-1)-f(n, m-1))/(x(n+m-1) - x(n));
    end;
end;


yint = f(1,1);
dx = 1;

for n=2:N
    dx = dx*(xi-x(n-1));
    yint = yint + f(1,n)*dx;
```

```
end;

end
```

6.4.1. Ejemplo 1

Calcular el logaritmo de 2 utilizando la interpolación de Newton y los valores $[1, 0]$, $[4, 1.386294]$, $[6, 1.791759]$ y $[5, 1.609438]$

La solución la llevaremos a cabo de forma tabular

i	x_i	$f(x_i)$	Primero	Segundo	Tercero
		b_0	b_1	b_2	b_3
0	1	0.0000	0.4621	-0.0519	0.0079
1	4	1.3863	0.2027	-0.0204	
2	6	1.7918	0.1823		
3	5	1.6094			

Para 4 puntos el polinomio de Newton de tercer orden es

$$f_3(x) = 0 + 0.462098(x-1) - 0.05187311(x-1)(x-4) + 0.00786529(x-1)(x-4)(x-6)$$

y el valor del logaritmo es $f_3(2) = 0.6287686$

6.4.2. Ejemplo 2

Dados los puntos

x	y
0.00	2.10
1.00	7.70
2.00	13.60
3.00	27.20
4.00	40.90
5.00	61.10

Hacer una interpolación utilizando polinomios de Newton de quinto grado de los datos y comparar con la interpolación lineal y cuadrática

```
x = [0,1,2,3,4,5];
y = [2.1, 7.70, 13.60, 27.20, 40.90, 61.10];

x_nva = 0:0.01:5;
```

```
grado = 5;

N = length(x_nva);
M = length(x);
y_nva = zeros(N, 1);

for n=1:N
    for m=1:M-1
        if x_nva(n) >= x(m) && x_nva(n) <= x(m+1) break;
        end;
    end;
    l = m+grado;
    if l > M
        l = M;
    end;


    y_nva(n) = Interpolacion_Newton(x_nva(n), x(m:l), y(m:l));
end;

plot(x_nva, y_nva, 'r-.', x, y, 'k*');
xlabel('x');
ylabel('f(x)');
```

En la figura 6.25, se muestran los puntos correspondientes calculados de acuerdo con el código implementado

6.5. Interpolación de Polinomios de Lagrange

La interpolación de polinomios de Lagrange es simplemente una reformulación del polinomio de Newton que evita el cálculo por diferencias divididas. Se puede expresar de manera concisa como

$$f_n(x) = \sum_{i=0}^{n} L_i(x) f(x_i)$$

donde

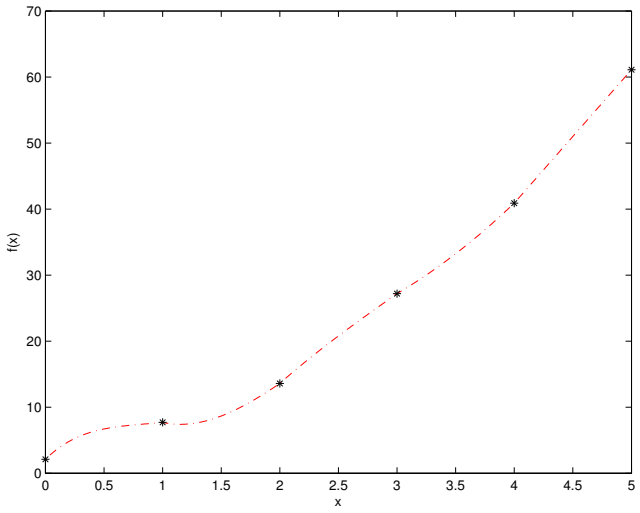


Figura 6.25: Un conjunto de puntos y su interpolación con Polinomios de Newton de Quinto grado

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

Por ejemplo la versión lineal $n = 1$ es

$$f_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1)$$

y la versión de segundo orden es

$$f_2(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f(x_0) + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f(x_1) + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} f(x_2)$$

El código en Matlab para este algoritmo es

```
function yint = Interpolacion_Lagrange(xint, x, y)

suma = 0;
```

```
N = length(x);
for n=1:N
    producto = y(n);
    for m = 1:N
        if n~=m
            producto = producto*(xint -x(m))/(x(n) - x(m));
        end;
    end;
    suma = suma + producto;
end;
yint = suma;
end
```

6.5.1. Ejemplo 1

Use una interpolación del polinomio de Lagrange de primer y segundo orden para evaluar ln(2) con base en los siguientes datos [1, 0], [4, 1.386294] y [6, 1.791759]

La solución lineal $n = 1$ es

$$f_1(x) = \frac{x-4}{1-4}0 + \frac{x-1}{4-1}1.386294 = -0.462098 + 0.462098x$$

$$f_1(2) = \frac{2-4}{1-4}0 + \frac{2-1}{4-1}1.386294 = 0.4620981$$

y la solución de segundo $n = 2$ orden es

$$f_2(x) = \frac{(x-4)(x-6)}{(1-4)(1-6)}0 + \frac{(x-1)(x-6)}{(4-1)(4-6)}1.386294 + \frac{(x-1)(x-4)}{(6-1)(6-4)}1.791760$$

La ecuación resultante de segundo grado es

$$f_2(x) = -0.66959 + 0.721463x - 0.0518731x^2$$

Para calcular el valor de $ln(2)$ resolvemos

$$f_2(2) = \frac{(2-4)(2-6)}{(1-4)(1-6)}0 + \frac{(2-1)(2-6)}{(4-1)(4-6)}1.386294 + \frac{(2-1)(2-4)}{(6-1)(6-4)}1.791760 = 0.5658444$$

Como era de esperar, ambos resultados concuerdan con los que se obtuvieron utilizando interpolación utilizando polinomios de Newton.

6.5.2. Ejemplo 2

Dados los vectores $x = [1, 2]$ y $y = [2, 4]$ calcular la interpolación de grado $n = 1$.

$$f_1(x) = L_0(x)y_0 + L_1(x)y_1$$

$$f_1(x) = 2L_0(x) + 4L_1(x)$$

Los multiplicadores $L_i(x)$ se calculan con:

$$L_0(x) = \frac{x-2}{1-2} = 2 - x$$

$$L_1(x) = \frac{x-1}{2-1} = x - 1$$

Sustituyendo en la función tenemos

$$f_1(x) = 2(2 - x) + 4(x - 1) = 2x$$

6.5.3. Ejemplo 3

Dados los puntos $x = [1, 2, 3]$ y sus correspondientes valores $y = [3, 5, 10]$ determinar:

a) El polinómio de Lagrange de grado $n = 2$

$$f_2(x) = L_0(x)y_0 + L_1(x)y_1 + L_2(x)y_2$$

$$f_2(x) = 3L_0(x) + 5L_1(x) + 10L_2(x)$$

Las funciones $L_i(x)$ son:

$$L_0(x)\frac{(x-2)(x-3)}{(1-2)(1-3)} = \frac{x^2 - 5x + 6}{2}$$

$$L_1(x) = \frac{(x-1)(x-3)}{(2-1)(2-3)} = -x^2 + 4x - 3$$

$$L_2(x) = \frac{(x-1)(x-2)}{(3-1)(3-2)} = \frac{x^2 - 3x + 2}{2}$$

Sustituyendo tenemos

$$f_2(x) = 3(x^2 - 5x + 6)/2 - 5(x^2 - 4x + 3) + 10(x^2 - 3x + 2)/2$$

$$f_2(x) = 1.5x^2 - 2.5x + 4$$

b) Corroborar que para cada valor de x_i la función $f_2(x_i) = y_i$

$$f_2(1) = 1.5(1)^2 - 2.5(1) + 4 = 3$$

$$f_2(2) = 1.5(2)^2 - 2.5(2) + 4 = 5$$

$$f_2(3) = 1.5(3)^2 - 2.5(3) + 4 = 10$$

c) Calcular el polinomio de grado 2 utilizando mínimos cuadrados para los puntos dados.

$$M = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 4 \\ 1 & 3 & 9 \end{bmatrix}$$

La solución es

$$a = (M^T M)^{-1} M^T y$$

$$a = \begin{bmatrix} 3 & 6 & 14 \\ 6 & 14 & 36 \\ 14 & 36 & 98 \end{bmatrix}^{-1} \begin{bmatrix} 18 \\ 43 \\ 113 \end{bmatrix} = \begin{bmatrix} 4 \\ -2.5 \\ 1.5 \end{bmatrix}$$

El polinomio es :

$$f_2(x) = 1.5x^2 - 2.5x + 4$$

En este caso la función calculada por mínimos cuadrados e interpolación de la Lagrange son exactamente iguales.

6.5.4. Ejemplo 4

Para los valores de $x = [0.5, 1.3, 2.1, 2.9]$ y $y = [1.25, 4.3, 3.9, 2.1]$ determinar:

a) La aproximación polinomial de grado $n = 3$

$$f_3(x) = 1.25L_0(x) + 4.3L_1(x) + 3.9L_2(x) + 2.1L_3(x)$$

Las funciones $L_i(x)$ son:

$$L_0 = \frac{(x-1.3)(x-2.1)(x-2.9)}{(0.5-1.3)(0.5-2.1)(0.5-2.9)} = -0.325521x^3 + 2.050781x^2 - 4.09831x + 2.577148$$

$$L_1 = \frac{(x-0.5)(x-2.1)(x-2.9)}{(1.3-0.5)(1.3-2.1)(1.3-2.9)} = 0.976563x^3 - 5.37109x^2 + 8.38867x - 2.97363$$

$$L_2 = \frac{(x-0.5)(x-1.3)(x-2.9)}{(2.1-0.5)(2.1-1.3)(2.1-2.9)} = -0.976563x^3 + 4.58984x^2 - 5.73242x + 1.84082$$

$$L_3 = \frac{(x-0.5)(x-1.3)(x-2.1)}{(2.9-0.5)(2.9-1.3)(2.9-2.1)} 0.3255221x^3 - 1.26953x^2 + 1.44206x - 0.444336$$

Sustituyendo tenemos

$$f_3(x) = \begin{array}{l} 1.25(-0.325521x^3 + 2.050781x^2 - 4.09831x + 2.577148) \\ +4.3(0.976563x^3 - 5.37109x^2 + 8.38867x - 2.97363) \\ +3.9(-0.976563x^3 + 4.58984x^2 - 5.73242x + 1.84082) \\ +2.1(0.3255221x^3 - 1.26953x^2 + 1.44206x - 0.444336) \end{array}$$

$$f_3(x) = 0.667321x^3 - 5.297847x^2 + 11.620281x - 3.319081$$

b) Determinar el polinomio de grado 3 de mínimos cuadrados

$$M = \begin{bmatrix} 1.0000 & 0.5000 & 0.2500 & 0.1250 \\ 1.0000 & 1.3000 & 1.6900 & 2.1970 \\ 1.0000 & 2.1000 & 4.4100 & 9.2610 \\ 1.0000 & 2.9000 & 8.4100 & 24.3890 \end{bmatrix}$$

$$a = \begin{bmatrix} 4.0000 & 6.8000 & 14.7600 & 35.9720 \\ 6.8000 & 14.7600 & 35.9720 & 93.0948 \\ 14.7600 & 35.9720 & 93.0948 & 249.6967 \\ 35.9720 & 93.0948 & 249.6967 & 685.4319 \end{bmatrix}^{-1} \begin{bmatrix} 11.5500 \\ 20.4950 \\ 42.4395 \\ 96.9382 \end{bmatrix} = \begin{bmatrix} -3.3191 \\ 11.6203 \\ -5.2979 \\ 0.6673 \end{bmatrix}$$

$$f_3(x) = 0.6673x^3 - 5.2979x^2 + 11.6203 - 3.3191$$

c) Graficar la curva calculada con los puntos dados.

La Figura 6.26 se muestra la grafica de la función $f_3(x)$ y los valores de x y y. Los puntos dados se muestran como esferas de color azul y la curva calculada en rojo.

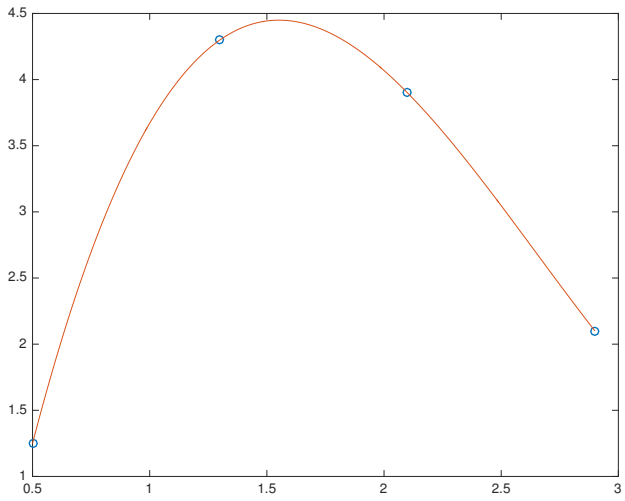


Figura 6.26: Aproximación de tercer grado para cuatro puntos utilizando interpolación de Lagrange

6.5.5. Ejemplo 5

Dados los valores en la siguiente tabla calcular:

x	$f(x)$
1	0.0000
2	0.6931
3	1.0986
4	1.3863

- El polinomio de Lagrange de cúbico en el intervalo $1 < x < 2$
- El valor de interpolación en $x = 1.4$

Para calcular el Polinomio de Lagrange hacemos

$$\begin{aligned} f_3(x) &= \frac{(x-2)(x-3)(x-4)}{(1-2)(1-3)(1-4)}0.0000 + \frac{(x-1)(x-3)(x-4)}{(2-1)(2-3)(2-4)}0.6931 + \\ & \frac{(x-1)(x-2)(x-4)}{(3-1)(3-2)(3-4)}1.0986 + \frac{(x-1)(x-2)(x-3)}{(4-1)(4-2)(4-3)}1.3863 \end{aligned}$$

Simplificando la expresión tenemos

$$f_3(x) = -1.1505 + 1.4358x - 0.3136x^2 + 0.0283x^3$$

Para calcular el valor interpolado hacemos

$$\begin{aligned} f_3(1.4) &= \frac{(1.4-2)(1.4-3)(1.4-4)}{(1-2)(1-3)(1-4)}0.0000 + \frac{(1.4-1)(1.4-3)(1.4-4)}{(2-1)(2-3)(2-4)}0.6931 + \\ & \frac{(1.4-1)(1.4-2)(1.4-4)}{(3-1)(3-2)(3-4)}1.0986 + \frac{(1.4-1)(1.4-2)(1.4-3)}{(4-1)(4-2)(4-3)}1.3863 \end{aligned}$$

obviamente que la otra alternativa para evaluar es:

$$f_3(1.4) = -1.1505 + 1.4358(1.4) - 0.3136(1.4)^2 + 0.0283(1.4)^3$$

El resultado de la interpolación es:

$$f_3(1.4) = 0.322619$$

Diferenciación e Integración

7.1. Diferenciación por diferencias divididas finitas atrás, adelante y centrales de exactitud simple

Considerando la definición de la derivada

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x) - f(x-h)}{h}$$

entonces para valores pequeños de h podemos hacer una aproximación de la derivada haciendo

$$\frac{df(x)}{dx} \approx \frac{f(x) - f(x-h)}{h}$$

En el caso de considerar la diferencia hacia adelante podemos hacer

$$\frac{df(x)}{dx} \approx \frac{f(x+h) - f(x)}{h}$$

Considerando que los incrementos se dan hacia adelante y hacia atrás, podemos dar una representación de la derivada centrada como

$$\frac{df(x)}{dx} \approx \frac{f(x+h) - f(x-h)}{2h}$$

7.2. Diferenciación por diferencias divididas finitas atrás, adelante y centrales de exactitud mejorada

7.3. Integración por el método de barras

En general todos los métodos de integración numérica hacen uso de la interpretación de la integral propia, esta interpretación nos dice que la integral en un intervalo x_0 y x_1 es el área

bajo la curva. Así podemos hacer este cálculo del área suponiendo que nuestros rectángulos son de altura $f(x_0)$. Recordemos que el área de un rectángulo esta dado como

$$A = (x_1 - x_0)f(x_0)$$

por lo que la integral la podemos aproximar por

$$I = \int_{x_0}^{x_1} f(x)dx \approx \int_{x_0}^{x_1} f(x_0)dx = f(x_0)x|_{x_0}^{x_1} = f(x_0) * (x_1 - x_0)$$

7.3.1. Ejemplo

Para la función $f(x) = 0.2 + 25x - 200x^2 + 675x^3 - 900x^4 + 400x^5$, calcular la integral utilizando el método de barras en el intervalo $x_0 = 0$ $x_1 = 0.8$.

$$I \approx f(0) * (0.8 - 0) = 0.2 \times 0.8 = 0.16$$

En Matlab

```
>> Barras(@f1, 0, 0.8, 1)

ans =

    0.1600
```

Para una mejor aproximación podemos hacer N divisiones de la región de tamaño h y aproximar la integral como:

$$\int_{x_0}^{x_1} f(x_0)dx \approx \sum_{k=0}^{N-1} f(x_0 + kh) * h$$

donde $h = (x_1 - x_0)/N$

Esta aproximación da lugar al método de barras cuya codificación en Matlab se presenta a continuación

```
function I = Barras(f, x0, x1, N)

h = abs(x1-x0)/N;

suma = 0;
for k=0:N-1
```

```
    suma = suma + f(x0+h*k);
end;

I = suma*h;

end
```

7.4. Integración utilizando la Regla Trapezoidal

Suponer que todos los rectángulos son de altura constante es una suposición fuerte. En lugar de ello supondremos, para este método, que los rectángulos varían linealmente de acuerdo con la siguiente ecuación

$$f(x) \approx f_1(x) = f(x_0) + \frac{(x - x_0)}{x_1 - x_0}(f(x_1) - f(x_0))$$

A partir de esto podemos hacer la aproximación de nuestra integral

$$I = \int_{x_0}^{x_1} f(x)dx \approx \int_{x_0}^{x_1} f_1(x)dx \approx \int_{x_0}^{x_1} \left(f(x_0) + \frac{(x - x_0)}{x_1 - x_0}(f(x_1) - f(x_0)) \right) dx$$

resolviendo la integral tenemos

$$I \approx \left(f(x_0)x + \frac{(x - x_0)^2}{2(x_1 - x_0)}(f(x_1) - f(x_0)) \right) \Bigg|_{x_0}^{x_1}$$

$$I \approx \frac{f(x_1) + f(x_0)}{2}(x_1 - x_0)$$

La cual es el área de un trapecio de altura menor $f(x_0)$ y altura mayor $f(x_1)$.

7.4.1. Ejemplo

Dados los vectores $x = [1, 2]$ y $y = [2, 4]$:

a) calcular la función de interpolación de grado $n = 1$.

$$f_1(x) = L_0(x)y_0 + L_1(x)y_1$$

$$f_1(x) = 2L_0(x) + 4L_1(x)$$

Los multiplicadores $L_i(x)$ se calculan con:

$$L_0(x) = \frac{x-2}{1-2} = 2 - x$$

$$L_1(x) = \frac{x-1}{2-1} = x - 1$$

Sustituyendo en la función tenemos

$$f_1(x) = 2(2-x) + 4(x-1) = 2x$$

b) Calcular la integral de la función del inciso a) en el intervalo $x_0 = 1$ y $x_1 = 2$

$$I = \int_1^2 f_1(x)dx = \int_1^2 2xdx = x^2\Big|_1^2 = 2^2 - 1^2 = 3$$

c) Corroborar el resultado utilizando el método de la Regla Trapezoidal

$$I = \frac{(x_1 + x_0)(y_1 - y_0)}{2} = \frac{(2+1)(4-2)}{2} = 3$$

7.4.2. Ejemplo

Para la función $f(x) = 0.2 + 25x - 200x^2 + 675x^3 - 900x^4 + 400x^5$, calcular la integral utilizando el método de la Regla Trapezoidal en el intervalo $x_0 = 0$ $x_1 = 0.8$.

$$I \approx \frac{f(x_1) + f(x_0)}{2}(x_1 - x_0) = \frac{(0.2320 + 0.2)(0.8 - 0)}{2} = 0.1728$$

En Matlab la solución es

```
>> Regla_Trapezoidal(@f1, 0, 0.8, 1)

ans =

    0.1728
```

Si aplicamos la formula para N divisiones de tamaño h tenemos que la regla trapezoidal la podemos calcular como

$$I \approx (x_1 - x_0)\frac{f(x_0) + 2\sum_{k=1}^{N-1} f(x_0 + kh) + f(x_1)}{2N}$$

La implementación en Matlab para la regla trapezoidal

```
function I = Regla_Trapezoidal(f, x0, x1, N)

h = abs(x1-x0)/N;

suma = f(x0);

for k=1:N-1
    suma = suma + 2*f(x0+h*k);
end;
suma = suma + f(x1);

I = (x1-x0)*suma/(2*N);

end
```

7.5. Integración por el método de regla Simpson 1/3

En general cuando se utiliza una función de interpolación de mayor grado la diferencia entre los valores estimados y los reales se vuelve pequeña. Para este método en lugar se suponer que la función se aproxima por lineas haremos la aproximación utilizando un polinomio de segundo orden. Pare este método utilizaremos los polinomios de Lagrange de segundo orden dado por

$$f(x) \approx f_2(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}f(x_0) + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}f(x_1) + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}f(x_2)$$

donde x_1 es un valor a la mitad del intervalo dado por $x_1 = (x_2 + x_0)/2$.

Aplicando la integración de la función de interpolación tenemos:

$$I = \int_{x_0}^{x_2} f(x) \approx \int_{x_0}^{x_2} f_2(x)dx = (x_2 - x_0)\frac{f(x_0) + 4f(x_1) + f(x_2)}{6}$$

7.5.1. Ejemplo

Dados los puntos $x = [1, 2, 3]$ y sus correspondientes valores $y = [3, 5, 10]$ determinar:

a) El polinómio de Lagrange de grado $n = 2$

$$f_2(x) = L_0(x)y_0 + L_1(x)y_1 + L_2(x)y_2$$

$$f_2(x) = 3L_0(x) + 5L_1(x) + 10L_2(x)$$

Las funciones $L_i(x)$ son:

$$L_0(x)\frac{(x-2)(x-3)}{(1-2)(1-3)} = \frac{x^2-5x+6}{2}$$

$$L_1(x) = \frac{(x-1)(x-3)}{(2-1)(2-3)} = -x^2+4x-3$$

$$L_2(x) = \frac{(x-1)(x-2)}{(3-1)(3-2)} = \frac{x^2-3x+2}{2}$$

Sustituyendo tenemos

$$f_2(x) = 3(x^2-5x+6)/2 - 5(x^2-4x+3) + 10(x^2-3x+2)/2$$

$$f_2(x) = 1.5x^2 - 2.5x + 4$$

b) Calcular la integral en los límites $x_0 = 1$ y $x_2 = 3$

$$I = \int_1^3 f_2(x)dx = \int_1^3 (1.5x^2 - 2.5x + 4)dx = (0.5x^3 - 1.25x + 4x)\Big|_1^3$$

$$I = (0.5(3)^3 - 1.25(3)^2 + 4(3)) - (0.5(1)^3 - 1.25(1)^2 + 4(1)) = 11$$

c) Corroborar el resultado utilizando la regla Simpson 1/3

$$I = (x_2 - x_0)\frac{(y_0 + 4y_1 + y_2)}{6} = (3-1)\frac{(3+4(5)+10)}{6} = 11$$

7.5.2. Ejemplo

Para la función $f(x) = 0.2 + 25x - 200x^2 + 675x^3 - 900x^4 + 400x^5$, calcular la integral utilizando el método Simpson 1/3 en el intervalo $x_0 = 0$ $x_2 = 0.8$.

Para este caso $x_1 = (0.8 - 0)/2 = 0.4$ y $f(x_1) = f(0.4) = 2.4560$.

$$I \approx (x_2 - x_0)\frac{f(x_0) + 4f(x_1) + f(x_2)}{6} = (0.8 - 0)\frac{f(0) + 4f(0.4) + f(0.8)}{6} =$$

$$I \approx (0.8 - 0)\frac{(0.2 + 4 \times 2.4560 + 0.2320)}{6} = 1.3675$$

En Matlab la solución es

```
>> Simpson13(@f1, 0, 0.8, 1)

ans =

    1.3675
```

La implementación para N divisiones de tamaño h en Matlab es:

```
function I = Simpson13(f, ini, fin, N)

h = abs(fin - ini)/N;

suma =0;

for k=0:N-1
    x0 = ini + h*k;
    x1 = x0 + h/2;
    x2 = x0 + h;
    suma = suma + f(x0) + 4*f(x1) + f(x2);
end;

I = h*suma/6;
end
```

7.6. Integración por el método de regla Simpson 3/8

Para este método utilizaremos como función de integración un polinomio de Lagrange de tercer orden dado por

$$\begin{aligned} f(x) \approx f_3(x) &= \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)} f(x_0) \\ &+ \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)} f(x_1) \\ &+ \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)} f(x_2) \\ &+ \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)} f(x_3) \end{aligned}$$

donde $h = (x_3 - x_0)$ $x_1 = x_0 + h/3$ y $x_2 = x_0 + 2 * h/3$

Resolviendo para esta función polinomial

$$I = \int_{x_0}^{x_3} f(x)dx \approx \int_{x_0}^{x_3} f_3(x)dx = (x_3 - x_0)\frac{f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)}{8}$$

7.6.1. Ejemplo

Para los valores de $x = [0.5, 1.3, 2.1, 2.9]$ y $y = [1.25, 4.3, 3.9, 2.1]$ determinar:

a) La aproximación polinomial de grado $n = 3$

$$f_3(x) = 1.25L_0(x) + 4.3L_1(x) + 3.9L_2(x) + 2.1L_3(x)$$

Las funciones $L_i(x)$ son:

$$L_0 = \frac{(x-1.3)(x-2.1)(x-2.9)}{(0.5-1.3)(0.5-2.1)(0.5-2.9)} = -0.325521x^3 + 2.050781x^2 - 4.09831x + 2.577148$$

$$L_1 = \frac{(x-0.5)(x-2.1)(x-2.9)}{(1.3-0.5)(1.3-2.1)(1.3-2.9)} = 0.976563x^3 - 5.37109x^2 + 8.38867x - 2.97363$$

$$L_2 = \frac{(x-0.5)(x-1.3)(x-2.9)}{(2.1-0.5)(2.1-1.3)(2.1-2.9)} = -0.976563x^3 + 4.58984x^2 - 5.73242x + 1.84082$$

$$L_3 = \frac{(x-0.5)(x-1.3)(x-2.1)}{(2.9-0.5)(2.9-1.3)(2.9-2.1)} 0.3255221x^3 - 1.26953x^2 + 1.44206x - 0.444336$$

Sustituyendo tenemos

$$f_3(x) = \begin{array}{l} 1.25(-0.325521x^3 + 2.050781x^2 - 4.09831x + 2.577148) \\ +4.3(0.976563x^3 - 5.37109x^2 + 8.38867x - 2.97363) \\ +3.9(-0.976563x^3 + 4.58984x^2 - 5.73242x + 1.84082) \\ +2.1(0.3255221x^3 - 1.26953x^2 + 1.44206x - 0.444336) \end{array}$$

$$f_3(x) = 0.667321x^3 - 5.297847x^2 + 11.620281x - 3.319081$$

b) Calcular la integral de la función $f_3(x)$ en el intervalo $x_0 = 0.5$ y $x_1 = 2.9$

$$I = \int_{0.5}^{2.9} f_3(x)dx = \int_{0.5}^{2.9} (0.667321x^3 - 5.297847x^2 + 11.620281x - 3.319081)dx$$

$$I = (0.166683x^4 - 1.765949x^3 + 5.810140x^2 - 3.319081x)\Big|_{0.5}^{2.9}$$

$I = 0.166683(2.9^4 - 0.5^4) - 1.765949(2.9^3 - 0.5^3) + 5.810140(2.9^2 - 0.5^2) - 3.319081(2.9 - 0.5) = 8.385$

c) Corroborar el resultado utilizando la regla Simpson 3/8

$$I = (2.9 - 0.5)\frac{(1.25 + 3 \times 4.3 + 3 \times 3.9 + 2.1)}{8} = 8.385$$

7.6.2. Ejemplo

Para la función $f(x) = 0.2 + 25x - 200x^2 + 675x^3 - 900x^4 + 400x^5$, calcular la integral utilizando la regla Simpson 3/8 en el intervalo $x_0 = 0$ $x_3 = 0.8$.

$$I \approx (0.8 - 0)\frac{f(0) + 3f(0.2667) + 3f(0.5333) + f(0.8)}{8} =$$

$$I \approx (0.8 - 0)\frac{(0.2 + 3 \times 1.4327 + 3 \times 3.4872 + 0.2320)}{8} = 1.5192$$

En Matlab la solución es

```
>> Simpson38(@f1, 0, 0.8, 1)

ans =

    1.5192
```

La implementación en Matlab de la regla de Simpson 3/8 para N intervalos es

```
function I = Simpson38(f, ini, fin, N)
h = abs(fin-ini)/N;

suma =0;

for k=0:N-1
    x0 = ini + h*k;
    x1 = x0 + h/3;
    x2 = x0 + 2*h/3;
    x3 = x0 + h;
    suma = suma + f(x0) + 3*f(x1) + 3*f(x2) + f(x3);
end;

I = h*suma/8;
end
```

7.7. Ejemplos

7.7.1. Ejemplo 1

Dada la función

$$f(x) = 0.2 + 25x - 200x^2 + 675x^3 - 900x^4 + 400x^5$$

Hacer la solución de las integral en el intervalo $0 \leq x \leq 0.8$ con $N = 1$ aplicando los métodos vistos y comparar con la solución real dada por la ecuación.

Comenzamos por calcular la integral de $f(x)$

$$I(x) = \int_0^{0.8} f(x)dx = \left(0.2x + \frac{25}{2}x^2 - \frac{200}{3}x^3 + \frac{675}{4}x^4 - \frac{900}{5}x^5 + \frac{400}{6}x^6\right)\Big|_0^{0.8} = 1.6405$$

Método de Barras

Dado $x_0 = 0$, $x_1 = 0.8$ y $f(x_0) = 0.2$ tenemos

$$f_0(x) = f(x_0) = 0.2$$

La integral de $f_0(x)$ es

$$I = \int_0^{0.8} 0.2\ dx = 0.2x|_0^{0.8} = 0.2 \times (0.8 - 0) = 0.16$$

Aplicando el método de Barras

$$MB = (x1 - x0)f(x_0) = (0.8 - 0) \times 0.2 = 0.16$$

Regla Trapezoidal

Dado $x_0 = 0$, $x_1 = 0.8$, $f(x_0) = 0.2$ y $f(x_1) = 0.232$ la aproximación del polinomio de Lagrange es :

$$f_1(x) = \frac{x - x_1}{x_0 - x_1} f(x_0) + \frac{x - x_0}{x_1 - x_0} f(x_1)$$

$$f_1(x) = \frac{x - 0.8}{0 - 0.8} 0.2 + \frac{x - 0}{0.8 - 0} 0.232$$

Reduciendo termino tenemos el polinomio

$$f_1(x) = 0.2 + 0.04x$$

cuya integral es

$$I = \int_0^{0.8} (0.2 + 0.04x)\ dx = (0.2x + 0.02x^2)|_0^{0.8} = 0.1728$$

La solución utilizando la Regla Trapezoidal es:

$$RT = (x_1 - x_0)(f(x_1) + f(x_0))/2 = (0.8 - 0)(0.232 + 0.2)/2 = 0.1728$$

Regla Simpson 1/3

Dado $x_0 = 0$, $x_1 = 0.4$, $x_2 = 0.8$, $f(x_0) = 0.2$, $f(x_1) = 2.456$ y $f(x_2) = 0.232$ la aproximación del polinomio de Lagrange es:

$$f_2(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)} f(x_0) + \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)} f(x_1) + \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)} f(x_2)$$

sustituyendo los valores

$$f_2(x) = \frac{(x-0.4)(x-0.8)}{(0-0.4)(0-0.8)}0.2 + \frac{(x-0)(x-0.8)}{(0.4-0)(0.4-0.8)}2.456 + \frac{(x-0)(x-0.4)}{(0.8-0)(0.8-0.4)}0.232$$

Reduciendo términos tenemos

$$f_2(x) = 0.2 + 11.24x - 14x^2$$

cuya integral es

$$I = \int_0^{0.8} (0.2 + 11.24x - 14x^2)\, dx = (0.2x + 5.62x^2 - 4.666667x^3)\Big|_0^{0.8} = 1.367466$$

La solución aplicando el método Simpson 1/3 es

$$RS1/3 = (x_2 - x_0)\frac{f(x_0) + 4f(x_1) + f(x_2)}{6} = (0.8-0)\frac{0.2 + 4 \times 2.456 + 0.232}{6} = 1.367466$$

Regla Simpson 3/8

Dado $x_0 = 0$, $x_1 = 0.266666$, $x_2 = 0.5333333$, $x_3 = 0.8$, $f(x_0) = 0.2$, $f(x_1) = 1.43272$, $f(x_2) = 3.48718$ y $f(x_3) = 0.232$ la aproximación del polinomio de Lagrange es:

$$f_3(x) = \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)}f(x_0) + \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x1-x_3)}f(x_1) +$$

$$\frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)}f(x_2) + \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}f(x_3)$$

sustituyendo los valores

$$f_3(x) = \frac{(x-0.266666)(x-0.533333)(x-0.8)}{(0-0.266666)(0-0.533333)(0-0.8)}0.2 + \frac{(x-0)(x-0.533333)(x-0.8)}{(0.266666-0)(0.266666-0.533333)(x1-0.8)}1.43272 +$$

$$\frac{(x-0)(x-0.266666)(x-0.8)}{(0.533333-0)(0.533333-0.266666)(0.533333-0.8)}3.48718 + \frac{(x-0)(x-0.266666)(x-0.533333)}{(0.8-0)(0.8-0.266666)(0.8-0.533333)}0.232$$

Reduciendo términos tenemos

$$f_3(x) = 0.2 - 4.582222x + 48.888888x^2 - 53.888889x^3$$

cuya integral es

$$I = \int_0^{0.8} (0.2 - 4.582222x + 48.888888x^2 - 53.888889x^3)\ dx =$$

$$I = (0.2x - 2.291111x^2 + 16.296296x^3 - 13.472222x^4)\Big|_0^{0.8} = 1.51917$$

La solución aplicando la Regla Simpson 3/8 es

$$RS3/8 = (x_3 - x_0)\frac{f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)}{8}$$

$$RS3/8 = (0.8 - 0)\frac{0.2 + 3 \times 1.43272 + 3 \times 3.48718 + 0.232}{8} = 1.51917$$

En la siguiente tabla se muestra un resumen de la la solución para cada uno de los métodos así como el error numérico respecto al valor real de la integral.

Método	Solución	error
Barras	0160000	$\|1.6405 - 0.160000\| = 1.4805$
R. Trapezoidal	0.172800	$\|1.6405 - 0.172800\| = 1.4677$
R. Simpson 1/3	1.367466	$\|1.6405 - 1.367466\| = 0.2731$
R. Simpson 3/8	1.519170	$\|1.6405 - 1.519170\| = 0.1214$

Note que el mejor desempeño lo tiene la Regla Simpson 3/8 en virtud de utilizar polinomios de grado 3 para llevar a cabo la integración.

En la Figura 7.27 se muestra de izquierda a derecha las aproximaciones polinomiales calculada por los métodos de Barras, Regla Traprezoidal, Regla Simpson 1/3 y Regla Simpson 3/8. Para cada una de las curvas en la Figura se muestra en azul la función original y en negro la aproximación de cada una de las funciones polinomiales.

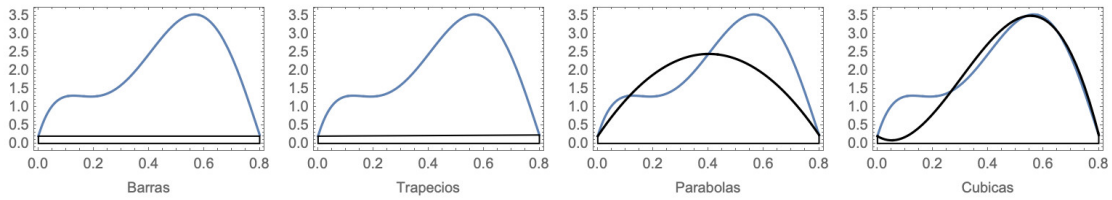


Figura 7.27: Ejemplo de las funciones aproximadas para cada uno de los métodos

7.7.2. Ejemplo 2

Para la función del Ejemplo 1 calcular los errores de integración cuando el intervalo de integración $0 \leq x \leq 0.8$ se divide en $N = 5$ intervalos del mismo tamaño $h = 0.16$. Mostrar la gráficas de las funciones aproximadas en cada intervalo. Concluir

En la siguiente tabla se muestran los

Método	Solución	error
Barras	1.5373	$\|1.6405 - 1.5373\| = 0.10318$
R. Trapezoidal	1.5399	$\|1.6405 - 1.5399\| = 0.1006$
R. Simpson 1/3	1.0852	$\|1.6405 - 1.6401\| = 0.0004$
R. Simpson 3/8	1.6405	$\|1.6405 - 1.6403\| = 0.0002$

En la Figura 7.28 se muestra de izquierda a derechas las gráficas correspondientes a cada aproximación considerando que el intervalo se divide en 5.

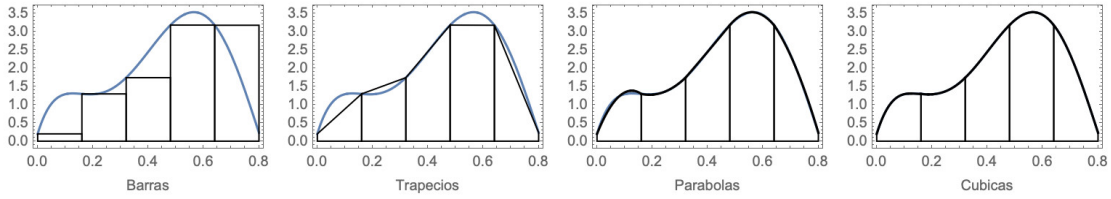


Figura 7.28: Ejemplo de las funciones aproximadas para cada uno de los métodos dividiendo el intervalo de integración en 5

7.7.3. Ejemplo 3

Para la función del Ejemplo 1 calcular los errores de integración cuando el intervalo de integración $0 \leq x \leq 0.8$ se divide en $N = 100$ intervalos del mismo tamaño $h = 0.008$.

Método	Solución	error
Barras	1.6401	0.0004
R Trapezoidal	1.6403	0.0003
Simpson 1/3	1.6405	0.0000
Simpson 3/8	1.6405	0.0000

Note que a medida que se utilizan mas intervalos la solución en mucho mejor acercando a la solución real.

Ecuaciones diferenciales ordinarias

En este capítulo se lleva a cabo la solución de ecuaciones diferenciales de la forma

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

En general los métodos que veremos hacen una formulación lineal de la forma

$$y^{(k+1)} = y^{(k)} + \phi h$$

donde ϕ es una estimación de la pendiente de la línea recta y utilizaremos esta linea para hacer una predicción de los valores de la integral.

8.1. Integración por el método de Euler

La primera derivada proporciona una estimación directa de la pendiente en $x^{(k)}$ dada por

$$\phi = f(x^{(k)}, y^{(k)})$$

donde $f(x^{(k)}, y^{(k)})$ es la ecuación diferencial evaluada en $x^{(k)}$ y $y^{(k)}$. Utilizando esta estimación tenemos

$$y^{(k+1)} = y^{(k)} + f(x^{(k)}, y^{(k)})h$$

La ecuación anterior es conocida como el método de Euler (o de Euler-Cauchy o de punto medio). Su implementación en Matlab es:

```
function [x,y] = Euler(x0, y0, h, N, f)
    D = length(y0);

    x = zeros(N+1, 1);
    y = zeros(N+1, D);

    x(1) = x0;
    y(1,:) = y0;

    for k = 1:N
        y(k+1,:) = y(k,:) + f(x(k), y(k,:))*h;
        x(k+1) = x(k) + h;
    end;
end
```

donde x_ini el el vector de valores iniciales, h es el incremento, N es el conjunto de valores donde se integra y f son el conjunto de ecuaciones diferenciales.

8.2. Integración por el método de Heun con solo uno y con varios predictores

Un método para mejorar la estimación de la pendiente involucra la determinación de dos derivadas para cada uno de los intervalos (una en el punto inicial y otra en el punto final). Las dos derivadas se promedian con el fin de obtener una estimación mejorada de las pendientes para todo el intervalo.

Recordemos que el método de Euler, la pendiente al inicio de un intervalo es

$$y'^{(k)} = f(x^{(k)}, y^{(k)})$$

y se usa para extraporlar linealmente a $y^{(k+1)}$

$$y_p^{(k+1)} = y^{(k)} + f(x^{(k)}, y^{(k)})h$$

La pendiente en el punto final del intervalo la podemos calcular como:

$$y'^{(k+1)} = f(x^{(k+1)}, y_p^{(k+1)})$$

y la pendiente promedio la podemos calcular como

$$\bar{y}' = \frac{y'^{(k+1)} + y'^{(k)}}{2}$$

De lo anterior podemos decir que el método tiene dos pasos, uno de predicción que es una iteración de Euler y un Corrector donde se utiliza la estimación de la pendiente al final del bloque. Estos pasos son:

Predicción

$$y_p^{(k+1)} = y^{(k)} + f(x^{(k)}, y^{(k)})h$$

Corrección

$$y^{(k+1)} = y^{(k)} + \frac{f(x^{(k)}, y^{(k)}) + f(x^{(k+1)}, y_p^{(k+1)})}{2}h$$

La implementación en Matlab para el Método de Heun es:

```
function [x, y] = Heun(x0, y0, h, N, f)
    D = length(y0);

    x = zeros(N+1, 1);
    y = zeros(N+1, D);

    x(1) = x0;
    y(1,:) = y0;

    for k = 1:N
        %Predictor
        k1 = f(x(k),   y(k,:)       );
        k2 = f(x(k)+h, y(k,:) + k1*h);
        %corrector
        %disp([k1, k2, y + k1]);
        x(k+1) = x(k) + h;
        y(k+1,:) = y(k,:) + (k1 + k2 )*h/2.0;
    end;
end
```

8.3. Integración por el método del punto medio

Esta técnica usa el método de Euler para predecir un valor de y en el punto medio del intervalo

$$y^{(k+1/2)} = y^{(k)} + f(x^{(k)}, y^{(k)})h/2$$

Después este valor es utilizado para calcular la pendiente en el punto medio y hacer la solución en todo el intervalo como

$$y^{(k+1)} = y^{(k)} + f(x^{(k+1/2)}, y^{(k+1/2)})h$$

La implementación en Matlab es

```
function [x,y] = Punto_Medio(x0, y0, h, N, f)
    D = length(y0);

    x = zeros(N+1, 1);
    y = zeros(N+1, D);

    x(1) = x0;
    y(1,:) = y0;

    for k = 1:N

        xm = x(k) + h/2;
        ym = y(k,:) + f(x(k), y(k,:))*h/2;

        x(k+1) = x(k) + h;
        y(k+1,:) = y(k,:) + f(xm, ym)*h;
    end;
end
```

8.4. Fórmulas de integración por algunos métodos de Runge-Kutta de segundo orden

Los algoritmos de Runge-Kutta son los métodos de integración con mejor desempeño y exactitud. En el caso del método de Runge-Kutta tenemos calcular dos valores k_1 y k_2 utilizando

$$k_1 = f(x^{(k)}, y^{(k)})$$

$$k_2 = f(x^{(k)} + \frac{3}{4}h, y^{(k)} + \frac{3}{4}k_1 h)$$

La actualización se lleva a cabo haciendo

$$y^{(k+1)} = y^{(k)} + \left(\frac{1}{3}k_1 + \frac{2}{3}k_2\right)h$$

```
function [x,y] = RK(x0, y0, h, N, f)
    D = length(y0);

    x = zeros(N+1, 1);
    y = zeros(N+1, D);

    x(1) = x0;
    y(1,:) = y0;

    for k = 1:N
        k1 = f(x(k), y(k,:));
        k2 = f(x(k) + 3*h/4, y(k,:)+3*h*k1/4);

        x(k+1) = x(k) + h;
        y(k+1,:) = y(k,:) + (k1/3 + 2*k2/3)*h;

    end;
end
```

8.5. Fórmula de integración por el método de Runge-Kutta clásico de tercer orden

En el caso del método de Runge-Kutta de tercer orden tenemos que calcular tres valores k_1, k_2 y k_3 utilizando las siguientes formulas

$$k_1 = f(x^{(k)}, y^{(k)})$$

$$k_2 = f(x^{(k)} + \frac{1}{2}h, y^{(k)} + \frac{1}{2}k_1 h)$$

$$k_3 = f(x^{(k)} + h, y^{(k)} - k_1 h + 2k_2 h)$$

La actualización se lleva a cabo haciendo

$$y^{(k+1)} = y^{(k)} + \frac{1}{6}\left(k_1 + 4k_2 + k_3\right) h$$

```
function [x, y] = RK3(x0, y0, h, N, f)
    D = length(y0);

    x = zeros(N+1, 1);
    y = zeros(N+1, D);

    x(1) = x0;
    y(1,:) = y0;

    for k = 1:N

        k1 = f(x(k),       y(k,:));
        k2 = f(x(k) + h/2, y(k,:) + k1*h/2);
        k3 = f(x(k) + h,   y(k,:) - k1*h + 2*k2*h);

        x(k+1) = x(k) + h;
        y(k+1,:) = y(k,:) + (k1 + 4*k2 + k3)*h/6;
    end;
end
```

8.6. Formula de integración por el método de Runge-Kutta clásico de cuarto orden

El más popular de los métodos RK es el de cuarto orden. En el caso del método de Runge-Kutta de cuarto orden tenemos que calcular cuatro valores k_1, k_2, k_3 y k_4 utilizando las siguientes formulas

$$k_1 = f(x^{(k)}, y^{(k)})$$

$$k_2 = f(x^{(k)} + \frac{1}{2}h, y^{(k)} + \frac{1}{2}k_1 h)$$

$$k_3 = f(x^{(k)} + \frac{1}{2}h, y^{(k)} + \frac{1}{2}k_2 h)$$

$$k_4 = f(x^{(k)} + h, y^{(k)} + k_3 h)$$

La actualización se lleva a cabo haciendo

$$y^{(k+1)} = y^{(k)} + \frac{1}{6}\left(k_1 + 2k_2 + 2k_3 + k_4\right) h$$

La implementación en Matlab para el método de Runge-Kutta de cuarto orden es:

```
function [x, y] = RK4(x0, y0, h, N, f)
    D = length(y0);

    x = zeros(N+1, 1);
    y = zeros(N+1, D);

    x(1) = x0;
    y(1,:) = y0;

    for k = 1:N

        k1 = f(x(k),       y(k,:));
        k2 = f(x(k) + h/2, y(k,:) + k1*h/2);
        k3 = f(x(k) + h/2, y(k,:) + k2*h/2);
        k4 = f(x(k) + h,   y(k,:) + k3*h);

        x(k+1) = x(k) + h;
        y(k+1,:) = y(k,:) + (k1 + 2*k2 + 2*k3 + k4)*h/6;
    end;
end
```

8.7. Ejemplo

8.7.1. Una ecuación diferencial sencilla

Consideremos la ecuación diferencial

$$f(x, y) = -2x^3 + 12x^2 - 20x + 8.5$$

La integral la podemos calcular como

$$\int \frac{dy}{dx} dx = \int f(x, y) dx = -0.5x^4 + 4x^3 - 10x^2 + 8.5x$$

La solución para y para un valor inicial $y(0) = 1$ lo cual da la siguiente expresión

$$y(x) = -0.5x^4 + 4x^3 - 10x^2 + 8.5x + 1$$

Encontrar la solución numérica para esta ecuación diferencial, suponiendo un valor inicial $x^{(0)} = 0$, $y^{(0)} = 1$ y $h = 0.5$, utilizando los métodos analizados en este capítulo.

Solución

La implementación de la función en Matlab es

```
function dy = ecuacion(x, y)
    dy = -2*x^3 + 12*x^2- 20*x + 8.5;
end
```

En la siguiente tabla se muestran los resultados al aplicar la integración numérica.

k	$x^{(k)}$	Real $y^{(k)}$	Euler $y_E^{(k)}$	Heun $y_H^{(k)}$	P. Medio $y_{PM}^{(k)}$	RK $y_{RK}^{(k)}$	RK3 $y_{RK3}^{(k)}$	RK4 $y_{RK4}^{(k)}$
0	0.0	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
1	0.5	3.2188	5.2500	3.4375	3.1094	3.2773	3.2188	3.2188
2	1.0	3.0000	5.8750	3.3750	2.8125	3.1016	3.0000	3.0000
3	1.5	2.2188	5.1250	2.6875	1.9844	2.3477	2.2188	2.2188
4	2.0	2.0000	4.5000	2.5000	1.7500	2.1406	2.0000	2.0000
5	2.5	2.7188	4.7500	3.1875	2.4844	2.8555	2.7188	2.7188
6	3.0	4.0000	5.8750	4.3750	3.8125	4.1172	4.0000	4.0000
7	3.5	4.7188	7.1250	4.9375	4.6094	4.8008	4.7188	4.7188
8	4.0	3.0000	7.0000	3.0000	3.0000	3.0312	3.0000	3.0000

Para evaluar la exactitud de los métodos calculamos el valor absoluto de la diferencia entre el valor real y el calculado.

k	$\lvert y^{(k)} - y_E^{(k)} \rvert$	$\lvert y^{(k)} - y_H^{(k)} \rvert$	$\lvert y^{(k)} - y_{PM}^{(k)} \rvert$	$\lvert y^{(k)} - y_{RK}^{(k)} \rvert$	$\lvert y^{(k)} - y_{RK3}^{(k)} \rvert$	$\lvert y^{(k)} - y_{RK4}^{(k)} \rvert$
0	0	0	0	0	0	0
1	2.0312	0.2188	0.1094	0.0586	0	0
2	2.8750	0.3750	0.1875	0.1016	0	0
3	2.9062	0.4688	0.2344	0.1289	0	0
4	2.5000	0.5000	0.2500	0.1406	0	0
5	2.0312	0.4688	0.2344	0.1367	0	0
6	1.8750	0.3750	0.1875	0.1172	0	0
7	2.4062	0.2188	0.1094	0.0820	0	0
8	4.0000	0	0	0.0312	0	0

Para reproducir estos valores utilizar el código

```
function ecuacion_simple()
    % Numero de puntos
    N = 8;

    % Valores Iniciales
    xini = 0;
    xfin = 4;
    y_ini = 1;
    h = (xfin-xini)/N;
    x = [xini:h:xfin]';


    [x1,y1] = Euler(xini, y_ini, h, N, @ecuacion);
    [x2,y2] = Heun(xini, y_ini, h, N, @ecuacion);
    [x3,y3] = Punto_Medio(xini, y_ini, h, N, @ecuacion);
    [x4,y4] = RK(xini, y_ini, h, N, @ecuacion);
    [x5,y5] = RK3(xini, y_ini, h, N, @ecuacion);
    [x6,y6] = RK4(xini, y_ini, h, N, @ecuacion);
    y2

    [x, solucion_real(x), y1, y2, y3, y4, y5, y6]


      [x, abs(solucion_real(x)-y1),...
          abs(solucion_real(x)-y2),...
          abs(solucion_real(x)-y3),...
          abs(solucion_real(x)-y4),...
          abs(solucion_real(x)-y5),...
          abs(solucion_real(x)-y6)]

    function dy = ecuacion(x, y)
        dy = -2*x^3 + 12*x^2- 20*x + 8.5;
    end

    function y = solucion_real(x)
        y = -0.5*x.^4 + 4*x.^3 - 10*x.^2 + 8.5*x +1;
    end
end
```

8.7.2. Circuito RL

Para el circuito RL que se muestra en la figura 8.29 calcular el valor de la corriente $i(t)$ en función del tiempo t, asumiendo que el interruptor se cierra en el tiempo $t = 0$

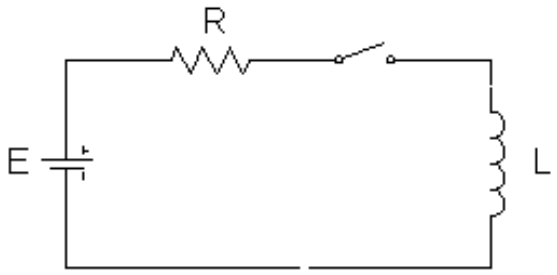


Figura 8.29: Circuito RL

Solución

La ecuación diferencial para este problema es

$$\frac{di}{dt} = \frac{V - Ri}{L}$$

La ecuación diferencial en Matlab es:

```
function di = Circuito_RL(t, i)
    %Parametros

    R = 2.0;  % Ohms
    L = 0.3;  % Henries
    V = 10.0; % Volts
    di = (V - R *i)/L;
end
```

La solución real de la ecuación diferencial, en función del tiempo, es:

$$i(t) = \frac{V}{R}\left(1 - e^{(-Rt/L)}\right)$$

La solución utilizando el Método de Runge-Kutta 4 para un valor inicial $t = 0$, $i(0) = 0$ y un incremento $h = 0.2$ es:

t	$i(t)$
0.0	0
0.2	3.5391
0.4	4.5732
0.6	4.8753
0.8	4.9636
1.0	4.9894

En la figura 8.30 se muestra la solución gráfica de los métodos estudiados en este capítulo. En negro se da la solución real de la ecuación diferencial y en amarillo la solución del Runge-Kutta de cuarto orden que es el que mejor se desempeña para el incremento dado. La peor solución es la calculada con el método de Euler la cual se muestra en azul en esta figura.

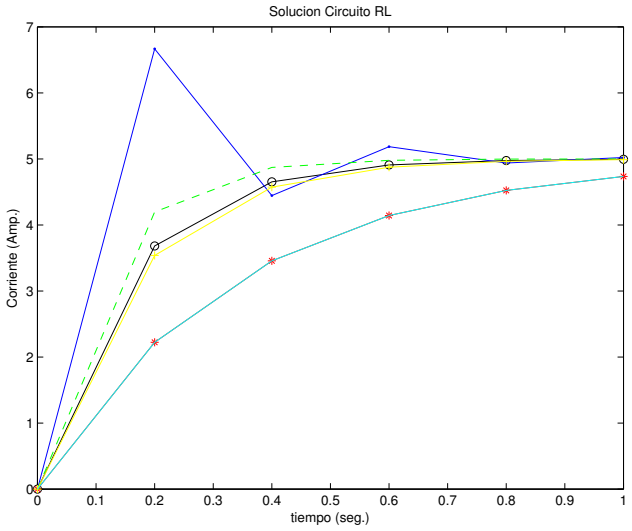


Figura 8.30: Circuito RL

El código implementado para este ejemplo es:

```
function Circuito_RL()

    % Numero de puntos
    N = 100;

    % Valores Iniciales
    tini = 0;
    tfin = 1;
```

```
i_ini = 0;
h = (tfin-tini)/N;


[t0, i0] = Solucion_Real(tini, i_ini, h, N);
[t1, i1] = Euler(tini, i_ini, h, N, @Circuito_RL);

[t2, i2] = RK(tini, i_ini, h, N, @Circuito_RL);
[t3, i3] = RK3(tini, i_ini, h, N, @Circuito_RL);
[t4, i4] = RK4(tini, i_ini, h, N, @Circuito_RL);

[t5, i5] = Heun(tini, i_ini, h, N, @Circuito_RL);



plot(t0, i0, 'ko-', t1, i1, 'b .-', ...
     t2, i2, 'r*-', t3, i3, 'g--',  ...
     t4, i4, 'y+-', t5, i5, 'c');

 xlabel('tiempo (seg.)');
 ylabel('Corriente (Amp.)');
 title('Solucion Circuito RL');

% Solucion exacta de la ecuacion Diferencial

function [t, ic] =Solucion_Real(t0, ir, h, N)
   %Parametros

    R = 2.0;  % Ohms
    L = 0.3;  % Henries
    V = 10.0; % Volts

    t = [t0:h:N*h]';
    Tau = L/R;
    ic = 5 *(1 - exp(-t/Tau));
end

% Ecuación diferencial del circuito RL

function di = Circuito_RL(t, i)
    %Parametros
```

```
        R = 2.0;  % Ohms
        L = 0.3;  % Henries
        V = 10.0; % Volts
        di = (V - R *i)/L;
    end
end
```

8.7.3. Movimiento parabólico

El movimiento parabólico lo podemos considerar como un movimiento en dos dimensiones; en una de las direcciones el objeto se desplaza con velocidad contante y en la otra con una aceleración constante

Las ecuaciones diferenciales que rigen este movimiento están dadas como

$$\frac{d^2x}{dt^2} = 0$$

$$\frac{d^2y}{dt^2} = a$$

Las ecuaciones diferenciales en el caso del tiro parabólico pueden representarse por un sistema de ecuaciones con cuatro variable el desplazamiento horizontal x, el desplazamiento vertical y, la velocidad horizontal v_x y la velocidad vertical v_y tal como se muestra a continuación:

$$\frac{d^2x}{dt^2} = 0 \equiv \begin{cases} \frac{dx}{dt} = v_x \\ \\ \frac{dv_x}{dt} = 0 \end{cases}$$

$$\frac{d^2x}{dt^2} = -g \equiv \begin{cases} \frac{dy}{dt} = v_y \\ \\ \frac{dv_y}{dt} = -g \end{cases}$$

La implementación en Matlab para este conjunto de ecuaciones es:

```
function dv = parabolico(t, v)
   g = 9.8; % m/s2

   dv(1) = v(2);
   dv(2) = 0;
   dv(3) = v(4);
```

```
   dv(4) = -g;
end
```

Para el caso de el tiro parabólico encontrar la solución del sistema de ecuaciones diferenciales, para un tiempo $t_f = 2.0$ seg, asumiendo los siguientes inicialmente $x(0) = 0$ mts, $y_0 = 5$ mts, $v_x(0) = 10$ m/s, $v_y(0) = 10$ m/s y un incremento $h = 0.001$.

La solución en matlab es:

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Datos
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


x0 = 0;    % desplazamiento horizontal inicial en mts
y0 = 5;    % desplazamiento vertical inicial en mts
vx0 = 10;  % velocidad horizontal inical en m/seg
vy0 = 10;  % velocidad vertical inicial en m/seg
t0 = 0;    % tiempo inicial seg.
tf = 2;    % tiempo final seg
h = 0.001; % intervalo de integración

N = floor((tf - t0)/h);

[t, d] = RK4(t0, [x0, vx0, y0, vy0], h, N, @parabolico);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
%  Graficas de la solución
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


plot(d(:,1), d(:,3));
grid on
title('Tiro Parabólico');
xlabel('x (mts.)');
ylabel('y (mts.)');
```

La solución es

```
>> tiro_parabolico


    1.9970   19.9700   10.0000    5.4288   -9.5706
    1.9980   19.9800   10.0000    5.4192   -9.5804
    1.9990   19.9900   10.0000    5.4096   -9.5902
    2.0000   20.0000   10.0000    5.4000   -9.6000
```

En el caso particular de un tiro parabólico, donde un objeto es lanzado con una velocidad inicial $v_x(0)$ horizontalmente y $v_y(0)$ verticalmente y esta sujeta a la acción de la gravedad la solución de analítica de las ecuaciones diferenciales es el sistema mostrada a continuación

$$x(t) = x(0) + v_x(0)t$$

$$y(t) = y(0) + v_y(0)t - \tfrac{1}{2}gt^2$$

La solución de acuerdo con la ecuaciones de movimiento es:

$$x(t) = x(0) + v_x(0)t$$

$$x(2) = 0 + 10 * 2 = 20$$

$$y(t) = y(0) + v_y(0)t - \tfrac{1}{2}gt^2$$

$$y(2) = 5 + 10 * 2 - \tfrac{1}{2}(9.8)2^2 = 5.40$$

En la figura 8.31 se muestra la gráfica del desplazamiento $x(t)$ vs $y(t)$ para la solución del tiro parabólico.

8.7.4. Sistema Masa Resorte

El sistema masa resorte como el que se muestra en la figura 8.32, esta constituido por una masa M, sujeta a un resorte de constante K, el cual esta sujeto a fuerzas de amortiguamiento C y se aplica una fuerza F. El modelo dinámico para este sistema es:

$$M\frac{dx^2}{dt^2} + C\frac{dx}{dt} + Kx = F$$

Note que tenemos una ecuación diferencial de segundo orden la cual plantearemos como un par de ecuaciones de primer orden como:

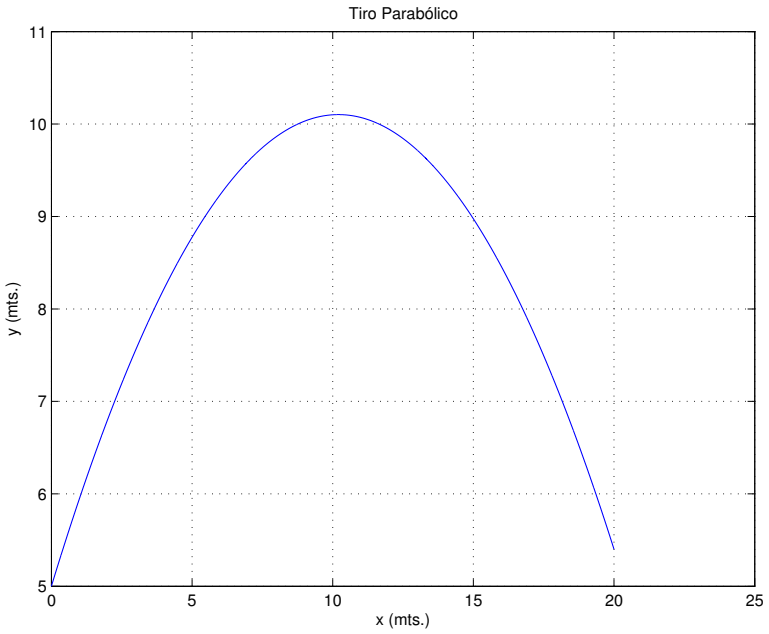


Figura 8.31: Gráfica de solución del tiro Parabólico

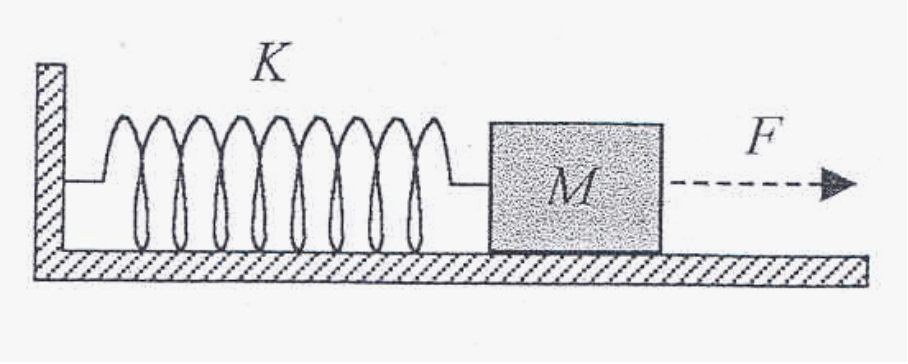


Figura 8.32: Sistema Masa Resorte

$$\frac{dx}{dt} = v$$

$$\frac{dv}{dt} = \frac{F - Cv - Kx}{M}$$

La implementación de estas funciones en Matlab es:

```
function D = Sistema_MR(t, r)
    M = 10.0;  % Masa  en Kg
    C = 1;   % Coeficiente de Amortiguamiento Dinnamica
    K = 5;   % Constante del Resorte
    F = 0;    % Fuerza Aplicada
    d = r(1);
    v = r(2);

    D(1) = v;   % derivada del desplazamiento
    D(2) = (F - C*v - K*d)/M; % derivada de la velocidad
end
```

En la figura 8.33 se muestra la solución de la ecuaciones diferenciales utilizando el método de Runge-Kutta de cuarto orden, para un desplazamiento inicial $x(0) = 2$, una velocidad inicial $v(0) = 0$, de una Masa $M = 10$, con un coeficiente de amortiguamiento $C = 1$, una constante del resorte $K = 5$ y una fuerza externa $F = 0$. En la parte superior izquierda tenemos el desplazamiento en función del tiempo a la derecha la velocidad en función del tiempo y en la parte inferior el diagrama de fase que muestra la $v(t)$ vs $x(t)$.

La implementación de este ejemplo en Matlab es:

```
function Sistema_Masa_Resorte(Metodo)

    % Numero de puntos
    N = 1000;

    % Valores Iniciales
    tini = 0;
    tfin = 40;
    d_ini = 2;
    v_ini = 0;

    h = (tfin-tini)/N;

    [t,x] = Metodo(tini, [d_ini, v_ini], h, N, @Sistema_MR);
```

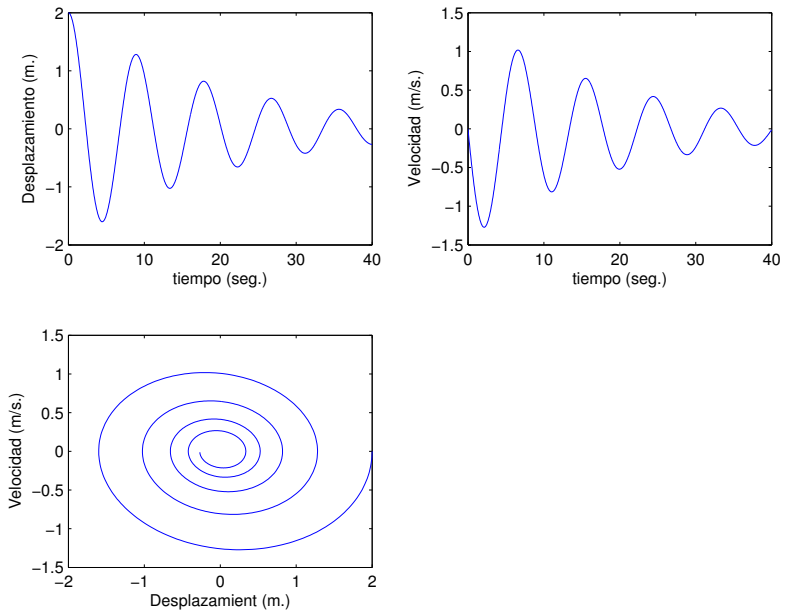


Figura 8.33: Sistema Masa Resorte

```
    title('Sistema Masa Resorte');

    subplot(2,2,1)
    plot(t, x(:,1));
    xlabel('tiempo (seg.)');
    ylabel('Desplazamiento (m.)');

    subplot(2,2,2)
    plot(t, x(:,2));
    xlabel('tiempo (seg.)');
    ylabel('Velocidad (m/s.)');

    subplot(2,2,3)
    plot(x(:,1), x(:,2));
    xlabel('Desplazamient (m.)');
    ylabel('Velocidad (m/s.)');


    % Ecuación diferencial del circuito RL

    function D = Sistema_MR(t, r)
        M = 10.0;  % Masa  en Kg
        C = 1;   % Coeficiente de Amortiguamiento Dinnamica
        K = 5;   % Constante del Resorte
        F = 0;    % Fuerza Aplicada
        d = r(1);
        v = r(2);

        D(1) = v;   % derivada del desplazamiento
        D(2) = (F - C*v - K*d)/M; % derivada de la velocidad
    end
end
```

8.7.5. Solución Circuito RC

Considere el circuito serie alimentado por una fuente V_S, con una resistencia R y un capacitor C tal como se muestra en la Figura **??**. Para este circuito la ecuación que lo rige esta dado por:

$$V_s - Ri - \frac{1}{C}\int idt = 0$$

Para resolver el circuito, vamos a sustituir el valor de la corriente como el cambio de la carga con respecto al tiempo $i = \frac{dq}{dt}$.

$$V_s - R\frac{dq}{dt} - \frac{1}{C}q = 0$$

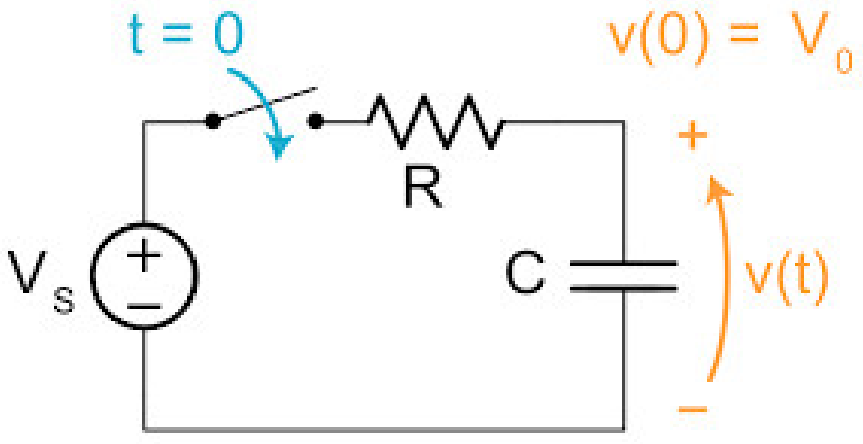


Figura 8.34: Circuito RC

Para llevar a cabo la integración numérica vamos a considerar que $V_s = 10$, $R = 2$ y $C = 0.01$. La función para llevar a cabo la integración numérica esta dada por:

```
function dq = CtoRC(t, q)
    %Parametros
    R = 2.0;    % Ohms
    C = 0.01;   % Farads
    V = 10.0;   % Volts
    dq = (V*C - q)/(R*C);
end
```

Para calcular la corriente que circula a través del circuito es necesario calcular la derivada, la cual podemos calcular mediante la siguiente expresión:

$$i = \frac{dq}{dt} \approx \frac{q(h) - q(h-1)}{h}$$

Para hacer la implementación en Matlab haremos

```
ic = (q(2:N)-q(1:N-1))/h;
tc = [t0:h:tf];
tc = tc(1:N-1);
```

El código completo para hacer la simulación se muestra a continuación y en la Figura 8.35 los resultados de la solución de la ecuación diferencial. Para esta Figura en la parte de arriba se muestra la solución de la carga con respecto al tiempo y en la parte de abajo la corriente en función del tiempo.

```
function Circuito_RC()

    % Numero de puntos
    N = 10000;

    % Valores Iniciales
    t0 = 0;
    tf = 0.13;
    q0 = 0;
    h = (tf-t0)/N;

    [t, q] = RK4(t0, q0, h, N, @CtoRC);

    subplot(2,1,1);
    plot(t, q(:,1));
    xlabel('tiempo (seg.)');
    ylabel('Carga (Coulombs.)');
    title('Solucion Circuito RC');

    % Calculo de la derivada
    ic = (q(2:N)-q(1:N-1))/h;
    tc = [t0:h:tf];
    tc = tc(1:N-1);


    subplot(2,1,2);
    plot(tc, ic);
    xlabel('tiempo (seg.)');
    ylabel('Corriente (Amp.)');
    title('Solucion Circuito RC');
```

```
    % Ecuación diferencial del circuito RL

    function dq = CtoRC(t, q)
        %Parametros
        R = 2.0;     % Ohms
        C = 0.01;    % Farads
        V = 10.0;    % Volts
        dq = (V*C - q)/(R*C);
    end
end
```

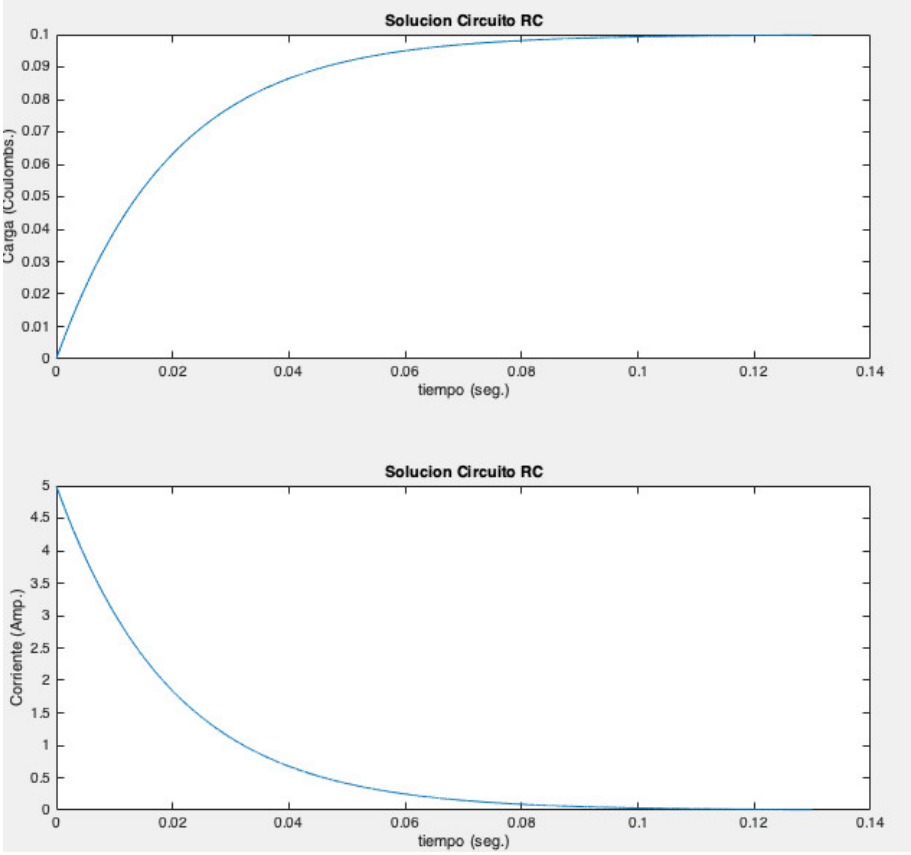


Figura 8.35: Solución numérica del circuito RC. En la parte de arriba la carga con respecto al tiemo y abajo la corriente en función del tiempo.

Ecuaciones diferenciales parciales

9.1. Fórmula de integración por el método explícito de diferencias divididas finitas

Consideremos la ecuación diferencial

$$C_1 \frac{dy}{dx} + C_2 y = C_3$$

Si hacemos la aproximación de la derivada mediante la ecuación

$$\frac{dy}{dx} \approx \frac{y(x_k) - y(x_k - h)}{h} \equiv \frac{y_k - y_{k-1}}{h}$$

Aplicándolo a la ecuación diferencial original

$$C_1 \left(\frac{y_k - y_{k-1}}{h} \right) + C_2 y_k = C_3$$

Agrupando términos semejantes tenemos

$$(C_1 + C_2 h) y_k - C_1 y_{k-1} = C_3 h$$

Lo cual da lugar a una recurrencia de la siguiente forma

$$
\begin{aligned}
(C_1 + C_2h)y_1 - C_1y_0 &= C_3h \\
(C_1 + C_2h)y_2 - C_1y_1 &= C_3h \\
(C_1 + C_2h)y_3 - C_1y_2 &= C_3h \\
(C_1 + C_2h)y_4 - C_1y_3 &= C_3h \\
&\vdots \\
(C_1 + C_2h)y_N - C_1y_{N-1} &= C_3h
\end{aligned}
$$

Lo cual da Lugar a un sistema de ecuaciones de la forma

$$
\begin{bmatrix}
(C_1 + C_2h) & 0 & 0 & \cdots & 0 \\
-C_1 & (C_1 + C_2h) & 0 & \cdots & 0 \\
0 & -C_1 & (C_1 + C_2h) & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots \\
0 & 0 & 0 & -C_1 & (C_1 + C_2h)
\end{bmatrix}
\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_N \end{bmatrix}
=
\begin{bmatrix} C_3h + C_1y_0 \\ C_3h \\ C_3h \\ \vdots \\ C_3h \end{bmatrix}
$$

de forma compacta podemos escribir $Ay = b$ cuya solución es $y = A^{-1}b$. La solución en Matlab es

```
function [x,y] = Ecuacion_Primer_Orden(x0, y0, h, N, C)
    x = [x0:h:x0+(N-1)*h]';


    A = zeros(N-1, N-1);
    b = zeros(N-1, 1);
    for k=1:N-1
        A(k,k) = C(1) + C(2)*h;
        b(k) = C(3)*h;
        if k>1
            A(k, k-1) = -C(1);
        else
            b(k) = b(k)+ C(1)*y0;
        end;
    end;
    y = [y0; A\b];
end
```

9.1.1. Ejemplo

Consideremos el caso del circuito RL planteado en el capítulo anterior, el cual tenía una resistencia $R = 2$ ohms, una inductancia $L = 0.3$ H y estaba alimentado por una fuente de voltaje de $V = 10$ volts dado por la ecuación diferencial

$$L\frac{di}{dt} + Ri = V$$

Dado $t = 0$, i(0) = 0 y h = 0.01 calcular:

- El sistema lineal de ecuaciones para calcular la solución con $N = 4$
- La solución del sistema lineal de ecuaciones y
- comparar con la solución con el método de Euler para las mismas condiciones del sistema y
- Solucionar para $N = 1000$ y graficar t vs $i(t)$

El sistema de ecuaciones a resolver por el método de diferencias es

$$\begin{bmatrix} 0.3020 & 0 & 0 & 0 \\ -0.3000 & 0.3020 & 0 & 0 \\ 0 & -0.3000 & 0.3020 & 0 \\ 0 & 0 & -0.3000 & 0.3020 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \end{bmatrix} = \begin{bmatrix} 0.0100 \\ 0.0100 \\ 0.0100 \\ 0.0100 \end{bmatrix}$$

La solución del sistema de ecuaciones es:

$$\begin{bmatrix} i_1 \\ i_2 \\ i_3 \\ i_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0.0331 \\ 0.0660 \\ 0.0987 \\ 0.1311 \end{bmatrix}$$

La solución utilizando el método de Euler es:

		Euler	Diferencias
k	t	i_E	i_D
0	0	0.0000	0.0000
1	0.0010	0.0333	0.0331
2	0.0020	0.0664	0.0660
3	0.0030	0.0993	0.0987
4	0.0040	0.1320	0.1311

La solución para $N = 100$ se muestra en la figura 9.36

```
>> [t,ic] = Ecuacion_Primer_Orden(0, 0, 0.001, 1000, [0.3, 2, 10]);
>> plot(t, ic)
```

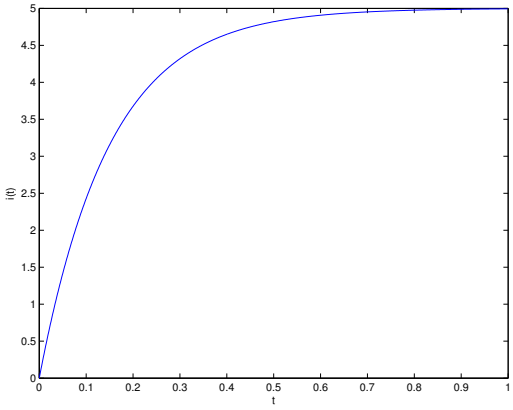


Figura 9.36: Solución para el Circuito RL

Tareas

11.1. Tarea 1

Escribir código en Matlab que permita imprimir, utilizando un solo asterisco, condicionales y ciclos: a) Un Triángulo

```
*
**
***
****
*****
******
*******
```

b Un cuadro

```
*******
*     *
*     *
*     *
*     *
*     *
*******
```

11.2. Tarea 2

Dada la función $f(x) = sen10x + cos3x$ hacer:

1.- Calcular la aproximación en serie de Taylor con $a = 0$ y mostrar la gráfica correspondiente comparada con los valores reales de la función en un intervalo $[0, 5]$

2.- Utilizando el método gráfico determinar los mínimos en el intervalo $[10, 15]$.

11.3. Tarea 3

Dada la función $f(x) = sen(10x) + cos(3x)$ la cual tiene un cruce por cero en el intervalo $x* \in [-3.6, -3.2]$, calcular el cruce por cero en esta región utilizando los métodos de Bisección, Regla Falsa, Punto Fijo y Newton Raphson. Reportar una tabla para N iteraciones como la siguiente para cada uno de los métodos.

	Biseccion		R. Falsi		P Fijo		N Raphson	
k	x_{Bk}	$f(x_{Bk})$	x_{Rk}	$f(x_{Rk})$	x_{Pk}	$f(x_{Pk})$	x_{Nk}	$f(x_{Nk})$
0	x_{B0}	$f(x_{B0})$	x_{R0}	$f(x_{R0})$	x_{P0}	$f(x_{P0})$	x_{N0}	$f(x_{N0})$
1	x_{B1}	$f(x_{B1})$	x_{R1}	$f(x_{R1})$	x_{P1}	$f(x_{P1})$	x_{N1}	$f(x_{N1})$
2	x_{B2}	$f(x_{B2})$	x_{R2}	$f(x_{R2})$	x_{P2}	$f(x_{P2})$	x_{N2}	$f(x_{N2})$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
N	x_{BN}	$f(x_{BN})$	x_{RN}	$f(x_{RN})$	x_{PN}	$f(x_{PN})$	x_{NN}	$f(x_{NN})$

donde N es el número máximo de iteraciones en el cual todos los métodos converge. Hacer una gráfica con los valores de la Tabla de k vs $f(x_Bk)$, $f(x_Rk)$, $f(x_Pk)$ y $f(x_Nk)$ con la solución de todos los métodos.

11.4. Tarea 4

Escribir el código en Matlab para realizar la división sintética de un polinomio $f_n(x) = a_n x^n + a_{n-1}x^{n-1} \cdots + a_1 x + a_0$ con un monomio $x + s$.

11.5. Tarea 5

a) Calcular las raíces del polinomio $x^4 - 1$ utilizando el método de Bairstow.

b) Dada la función $0.1 + (x-3)e^{-(x-3)^2}$ determinar, la expanción en Serie de Taylor con $N = 5$ y $a = 0$. Para el polinomio resultante calcular las raíces utilizando el Método de Bairstow.

11.6. Tarea 6

1.- Para el sistema de ecuaciones

10 x + 2 y - z = 5

2 x + 15 y = 2

-x + 20 z = 3

Resolver a mano todas las operaciones para calcular el sistema Triangular Superior con el método de Eliminación Gaussiana

2.- Hacer la implementación en Matlab del algoritmo de eliminación Gaussiana. Probar con el ejemplo dado en clases.

11.7. Tarea 7

a) Calcular el mínimo de la función $f(x) = (x-2)^4$ utilizando el algoritmo de Razón Dorada en el intervalo $[1, 5]$ b) Calcular el mínimo de la función $f(x_1, x_2) = (x_1-2)^4 + (x_1-2)^2x_2^2 + (x_2+1)^2$, utilizando el método de Newton.

En ambos casos hacer al menos tres iteraciones a mano y mostrar el resultado que da la implementación en Matlab.

11.8. Tarea 8

Buscar una aplicación en Ingeniería de los temas revisados en los capítulos 1 al 5.

11.9. Tarea 9

Dado el sistema de ecuaciones

$$\begin{aligned} 3x_1 - 0.1x_2 - 0.2x_3 &= 7.85 \\ 0.1x_1 + 7x_2 - 0.3x_3 &= -19.3 \\ 0.3x_1 - 0.2x_2 + 10x_3 &= 71.4 \end{aligned}$$

Calcular la solución del sistema y la inversa utilizando el algoritmos de Gauss-Jordan.

11.10. Tarea 10

Dadas la funciones a optimizar Z_1 y Z_2, dadas por la siguiente expresiones,

$$
\begin{aligned}
Z_1 &= 6x_1 + 4x_2 \\
&\texttt{Sujeta a} \\
2x_1 + 2x_2 &\leq 160 \\
x_1 + 2x_2 &\leq 120 \\
4x_1 + 2x_2 &\leq 280
\end{aligned}
$$

$$
\begin{aligned}
Z_2 &= 3x_1 + 4x_2 \\
&\texttt{Sujeta a} \\
2x_1 + 3x_2 &\leq 1200 \\
2x_1 + x_2 &\leq 1000 \\
4x_2 &\leq 880
\end{aligned}
$$

calcular los puntos del polígono de la región de factibilidad (para cada problema) y evaluar la función de costo correspondiente en cada uno de ellos.

11.11. Tarea 11

a) Para la implementación en Matlab del algoritmo Simplex hacer una modificación que permita hacer que esta función regrese:

```
[z, x, h] = Simplex[c, A, b]
```

donde z es el valor de la función de costo, x el valor de la variables en la solución y h el valor de las variables de holgura.

b) Optimizar utilizando el método simplex, el siguiente problema. Todos los cálculos deberán ser realizados a mano y comprobar que la solución es la correcta utilizando el algoritmo implementado en Matlab.

$$
\begin{aligned}
Z = 3x_1 + 2x_2 + 5x_3 \\
\texttt{sujeto a} \\
x_1 <= 10 \\
x_2 <= 5 \\
x_3 <= 6 \\
x_1 + x_2 + x_3 = 15
\end{aligned}
$$

11.12. Tarea 12

Dados los puntos

x	$ln(x)$
1	0.0000
2	0.6931
3	1.0986
4	1.3863
5	1.6094
6	1.7918

a) Encontrar los polinomios de mínimos cuadrados $f_k(x)$ de grado $k = 1, 2, 3, 4, 5$,

b) Realizar la interpolación utilizando interpolación de Newton de grado $k = 1, 2, 3, 4, 5$ y

c) Llenar la siguiente tabla, donde se hace la comparación del valor real de la función y los valores de los polinomios de mínimos cuadrados y la interpolación de Newton

Grado k	Polinomio $f_k(1.5)$	Polinomio Error	Interpolación $\hat{f}_k(1.5)$	Interpolación Error
1	$f_1(1.5)$	$(0.4055 - f_1(1.5))^2$	$\hat{f}_1(1.5)$	$(0.4055 - \hat{f}_1(1.5))^2$
2	$f_2(1.5)$	$(0.4055 - f_2(1.5))^2$	$\hat{f}_2(1.5)$	$(0.4055 - \hat{f}_2(1.5))^2$
3	$f_3(1.5)$	$(0.4055 - f_3(1.5))^2$	$\hat{f}_3(1.5)$	$(0.4055 - \hat{f}_3(1.5))^2$
4	$f_4(1.5)$	$(0.4055 - f_4(1.5))^2$	$\hat{f}_4(1.5)$	$(0.4055 - \hat{f}_4(1.5))^2$
5	$f_5(1.5)$	$(0.4055 - f_5(1.5))^2$	$\hat{f}_5(1.5)$	$(0.4055 - \hat{f}_5(1.5))^2$