Curso Propedeutico
Maestría en Ingeniería Eléctrica
Métodos Numéricos

Dr. Félix Calderon Solorio

7 de noviembre de 2016

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 mas 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	~=

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
```

1.6. 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)= 2x3+4x2+ 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 );
```

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. 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 3.4.

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

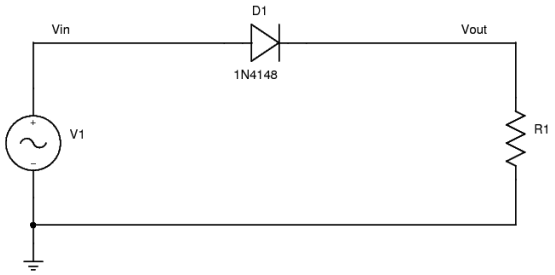

Figura 1.1: Circuito serie con diodo

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

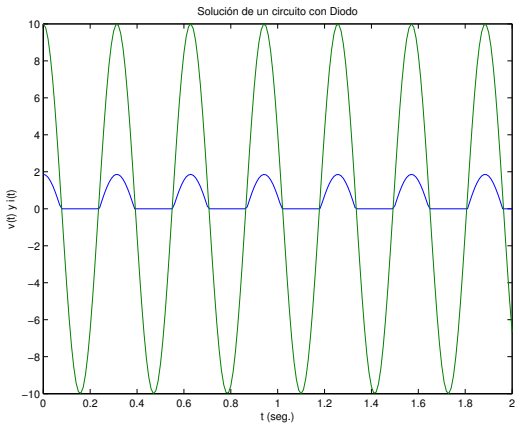

Figura 1.2: Solución gráfica del circuito con Diodo

1.6.4. 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);
```

1.6.5. 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);
```

Solución de Sistemas Lineales de Ecuaciones

2.1. Suma, resta, multiplicación y división de matrices

Una matriz se define como un arreglo bidimensional de datos, que tiene n renglones y m columnas. Un elemento del arreglo puede ser identificado con a_{ij}

$$A = \begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & ... & a_{1,N} \\ a_{2,1} & a_{2,2} & a_{2,3} & ... & a_{2,N} \\ a_{3,1} & a_{3,2} & a_{3,3} & ... & a_{3,N} \\ \\ a_{M,1} & a_{M,2} & a_{M,2} & ... & a_{M,N} \end{pmatrix}$$

Algunas de las operaciones básicas que pueden realizarse con matrices son suma, resta y multiplicación. La división de matrices como tal no existe y en su lugar se calcula la inversa.

2.1.1. Suma de matrices

Para que la sumar las matrices $A_{N\times M}$ y $B_{R\times S}$, se requiere que las matrices tengan el mismo número de renglones $N = R$ y de columnas $M = S$. Si queremos encontrar la suma $C = A + B$, cada elemento de la matriz C lo calculamos de la siguiente forma:

$$C_{n,m} = A_{n,m} + B_{n,m}$$

para todos lo n, m en la matriz C

La implementación en Matlab es:

```
function C = suma(A, B)

  [N, M] = size(A);
  [R, S] = size(B);

  if N~=R || M~=S
      fprintf('Las dimensiones tienen que ser iguales\n');
      return;
  end;

  C = zeros(N, M);

  for n=1:N
      for m = 1:M
          C(n,m) = A(n,m)+B(n,m);
      end;
  end
end
```

2.1.2. Resta de matrices

En este caso, se deben cumplir las mismas propiedades que la rsuma de matrices y el calculo de los elemento de la matriz C se calculan como:

$$C_{n,m} = A_{n,m} - B_{n,m}$$

para todos lo n, m en la matriz C

La implementación en Matlab es:

```
function C = resta(A, B)

  [N, M] = size(A);
  [R, S] = size(B);

  if N~=R || M~=S
      fprintf('Las dimensiones tienen que ser iguales\n');
      return;
  end;

  C = zeros(N, M);
```

```
  for n=1:N
      for m = 1:M
          C(n,m) = A(n,m)-B(n,m);
      end;
  end
end
```

2.1.3. Multiplicación de matrices

Para realizar el producto $C = A * B$ tenemos que considerar que el producto existe si

1.- El número de columnas de $A_{N\times M}$ es igual al número de renglones de $B_{R\times S}$, esto es $M = R$. y

2.- Las dimensiones de la matriz resultante tendrá el mismo numero de renglones que la matriz A y el mismo número de columnas que la matriz B. La matriz resultante C será de tamaño $N \times S$

3.- El cálculo de los elementos de la matriz C se lleva a cabo haciendo :

$$C_{n,m} = \sum_{k=1}^{M} a_{n,k} * b_{k,m}$$

para todos lo n, m en la matriz C

La implementación en Matlab es:

```
function C = multiplica(A, B)
   [N,M] = size(A);
   [R,S] = size(B);

   if M~=R
       fprintf('El numero de renglones no es igual al numero de columnas\n');
       return;
   end;

   C = zeros(N, S);

   for n=1:N
       for m= 1:S
           suma = 0;
```

```
            for k = 1:M
                suma = suma + A(n,k)*B(k,m);
            end;
            C(n,m) = suma;
        end;
    end;
end
```

2.2. Factorización triangular LU

Supongamos que la matriz de coeficientes A de un sistema lineal $Ax = b$ admite una factorización triangular como la siguiente.

$$\begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ l_{2,1} & 1 & 0 \\ l_{3,1} & l_{3,2} & 1 \end{bmatrix} * \begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} \\ 0 & u_{2,3} & u_{2,3} \\ 0 & 0 & u_{3,3} \end{bmatrix}$$

Entonces la solución la podemos plantear como

$$[LU]x = b$$

El cual se puede solucionar resolviendo primero $Ly = b$ y luego el sistema $Ux = y$. Estos sistemas pueden resolverse de manera muy eficiente utilizando sustitución hacia adelante y sustitución hacia atrás.

2.2.1. Sustitución hacia adelante

Comencemos por dar un sistema triangular inferior

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ l_{2,1} & 1 & 0 & 0 \\ l_{3,1} & l_{3,2} & 1 & 0 \\ l_{4,1} & l_{4,2} & l_{4,3} & 1 \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

Si despejamos comenzando con la ecuación I a y_1 y así sucesivamente para cada una de las ecuaciones tenemos:

$$\begin{aligned} y_1 &= b_1 \\ y_2 &= b_2 - l_{2,1}y_1 \\ y_3 &= b_3 - l_{3,1}y_1 - l_{3,2}y_2 \\ y_4 &= b_4 - l_{4,1}y_1 - l_{4,2}y_2 - l_{4,3}y_3 \end{aligned}$$

La formula para calcular la sustitución hacia adelante es:

$$y_k = b_k - \sum_{n=1}^{n<k} l_{k,n}x_n$$

Para todos los valores $k = 1, 2, 3, \cdots, N$

La implementación en matlab es:

```
function y = sustitucion_hacia_adelante(l,b)
     N = length(l(:,1));
     y = zeros(N,1);
     for k = 1: N
          suma = 0;
          for n = 1: k-1
               suma = suma + (l(k,n)*y(n));
          end;
          y(k) = (b(k) - suma);
     end;
end
```

2.2.2. Sustitución hacia atrás

Consideremos ahora un sistema triangular superior

$$\begin{bmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} \\ 0 & u_{2,2} & u_{2,3} & u_{2,4} \\ 0 & 0 & u_{3,3} & u_{3,4} \\ 0 & 0 & 0 & u_{4,4} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}$$

Para este caso comenzamos despejando x_4 de la última ecuación y sustituimos y despejamos en las anteriores tal como:

$$
\begin{aligned}
x_4 &= \frac{y_4}{u_{4,4}} \\
x_3 &= \frac{y_3 - u_{3,4}x_4}{u_{3,3}} \\
x_2 &= \frac{y_2 - u_{2,3}x_3 - u_{2,4}x_4}{u_{2,2}} \\
x_1 &= \frac{y_1 - u_{1,2}x_2 - u_{1,3}x_3 - u_{1,4}x_4}{u_{1,1}}
\end{aligned}
$$

La formula para calcular la sustitución hacia atrás es:

$$x_k = \frac{y_k - \sum_{n=k+1}^{N} u_{k,n}x_n}{u_{k,k}}$$

La implementación en Java es:

```
function x = sustitucion_hacia_atras(u, y)
    N = length(y);
    x = zeros(N,1);
    for k = N: -1: 1
        suma = 0;
        for n = k+1: N
            suma = suma +(u(k,n)*x(n));
        end;
        x(k) = (y(k)-suma)/u(k,k);
    end;
end
```

2.2.3. Ejemplo

Resolver el sistema de ecuaciones

$$
\begin{aligned}
x_1 + 2x_2 + 4x_3 + x_4 &= 21 \\
2x_1 + 8x_2 + 6x_3 + 4x_4 &= 52 \\
3x_1 + 10x_2 + 8x_3 + 8x_4 &= 79 \\
4x_1 + 12x_2 + 10x_3 + 6x_4 &= 82
\end{aligned}
$$

Supongamos que tenemos la descomposición triangular siguiente

$$\begin{bmatrix} 1 & 2 & 4 & 1 \\ 2 & 8 & 6 & 4 \\ 3 & 10 & 8 & 8 \\ 4 & 12 & 10 & 6 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 \\ 3 & 1 & 1 & 0 \\ 4 & 1 & 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 2 & 4 & 1 \\ 0 & 4 & -2 & 2 \\ 0 & 0 & -2 & 3 \\ 0 & 0 & 0 & -6 \end{bmatrix}$$

Primero resolvemos el sistema de ecuaciones

$$\begin{aligned} y_1 &= 21 \\ 2y_1 + y_2 &= 52 \\ 3y_1 + y_2 + y_3 &= 79 \\ 4y_1 + y_2 + 2y_3 + y_4 &= 82 \end{aligned}$$

El cual corresponde a un sistema triangular inferior

$$\begin{aligned} y_1 &= 21 \\ y_2 &= 10 \\ y_3 &= 6 \\ y_4 &= -24 \end{aligned}$$

Posteriormente solucionamos $Ux = y$

$$\begin{aligned} x_1 + 2x_2 + 4x_3 + x_4 &= 21 \\ 4x_2 - 2x_3 + 2x_4 &= 10 \\ -2x_3 + 3x_4 &= 6 \\ -6x_4 &= -24 \end{aligned}$$

Utilizando sustitución hacia atrás tenemos

$$X = [1, 2, 3, 4]T$$

2.2.4. Factorización de Crout

Consideremos un sistema más grande, por ejemplo de 4 ecuaciones con cuatro incógnitas.

$$\begin{pmatrix} a_{1,1} & a_{1,2} & a_{1,3} & a_{1,4} \\ a_{2,1} & a_{2,2} & a_{2,3} & a_{2,4} \\ a_{3,1} & a_{3,2} & a_{3,3} & a_{3,4} \\ a_{4,1} & a_{4,2} & a_{4,3} & a_{4,4} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ l_{2,1} & 1 & 0 & 0 \\ l_{3,1} & l_{3,2} & 1 & 0 \\ l_{4,1} & l_{4,2} & l_{4,3} & 1 \end{pmatrix} \times \begin{pmatrix} u_{1,1} & u_{1,2} & u_{1,3} & u_{1,4} \\ 0 & u_{2,2} & u_{2,3} & u_{2,4} \\ 0 & 0 & u_{3,3} & u_{3,4} \\ 0 & 0 & 0 & u_{4,4} \end{pmatrix}$$

Si hacemos las multiplicaciones indicadas y dejamos los valores en la misma matriz tenemos:

Para el primer renglón de la matriz ($n = 1$)

arriba de la diagonal

$$\begin{array}{ll} m = 1 & a_{1,1} = a_{1,1} \\ m = 2 & a_{1,2} = a_{1,2} \\ m = 3 & a_{1,3} = a_{1,3} \\ m = 4 & a_{1,4} = a_{1,4} \end{array}$$

Para el segundo renglón ($n = 2$)

bajo la diagonal

$$m = 1 \quad a_{2,1} = a_{2,1}/a_{1,1}$$

arriba de la diagonal

$$\begin{array}{ll} m = 2 & a_{2,2} = a_{2,2} - a_{2,1}a_{1,2} \\ m = 3 & a_{2,3} = a_{2,3} - a_{2,1}a_{1,3} \\ m = 4 & a_{2,4} = a_{2,4} - a_{2,1}a_{1,4} \end{array}$$

Tercer renglón ($n = 3$)

bajo la diagonal

$$\begin{array}{ll} m = 1 & a_{3,1} = a_{3,1}/a_{1,1} \\ m = 2 & a_{3,2} = (a_{3,2} - a_{3,1}a_{1,2})/a_{2,2} \end{array}$$

arriba de la diagonal

$$\begin{array}{ll} m=3 & a_{3,3}=a_{3,3}-a_{3,1}a_{1,3}-a_{3,2}a_{2,3} \\ m=4 & a_{3,4}=a_{3,4}-a_{3,1}a_{1,4}-a_{3,2}a_{2,4} \end{array}$$

Y finalmente para el cuarto renglón ($n=4$)

bajo la diagonal

$$\begin{array}{ll} m=1 & a_{4,1}=a_{4,1}/a_{1,1} \\ m=2 & a_{4,2}=(a_{4,2}-a_{4,1}a_{1,2})/a_{2,2} \\ m=3 & a_{4,3}=(a_{4,3}-a_{4,1}a_{1,3}-a_{4,2}a_{2,3})/a_{3,3} \end{array}$$

arriba de la diagonal

$$m=4 \quad a_{4,4}=a_{4,4}-a_{4,1}a_{1,4}-a_{4,2}a_{2,4}-a_{4,3}a_{3,4}$$

Podemos ver, que cualquier elemento bajo la diagonal se calcula como:

$$a_{n,m}=\frac{a_{n,m}-\sum_{k=1}^{m-1}a_{n,k}a_{k,m}}{a_{m,m}}$$

$$n=1,\cdots,N \qquad m=1,\cdots,n-1$$

Para los términos arriba de la diagonal

$$a_{n,m}=a_{n,m}-\sum_{k=1}^{n-1}a_{k,m}$$

$$n=1,\cdots,N \qquad m=n,...,N.$$

La implementación en Matlab es:

```
function a = Factorizacion(A)
   N = length(A);
   a = A;

   for n = 1:N
      for m = 1: n-1
        suma = 0;
```

```
        for k = 1:m-1
          suma = suma + a(n,k) * a(k,m);
        end;

        a(n,m) = (a(n,m) - suma)/ a(m,m);
      end;

      for m = n:N
        suma = 0;
        for k = 1: n-1
          suma = suma + a(n,k) * a(k,m);
        end;
        a(n,m) = a(n,m) - suma;
      end;
   end;
end
```

2.2.5. Ejemplo

Hacer la factorización LU de la siguiente matriz, utilizando eliminación Gaussiana. Corroborar utilizando Factorización de Crout.

$$\begin{pmatrix} 1 & 2 & 4 & 1 \\ 2 & 8 & 6 & 4 \\ 3 & 10 & 8 & 8 \\ 4 & 12 & 10 & 6 \end{pmatrix}$$

Paso 1.

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

Paso 2.

$$\begin{pmatrix} 1 & 2 & 4 & 1 \\ 2 & 4 & -2 & 2 \\ 3 & 1 & -2 & 3 \\ 4 & 1 & -4 & 0 \end{pmatrix}$$

Paso 3.

$$\begin{pmatrix} 1 & 2 & 4 & 1 \\ 2 & 4 & -2 & 2 \\ 3 & 1 & -2 & 3 \\ 4 & 1 & -2 & 6 \end{pmatrix}$$

La ejecución en Matlab queda

```
>> Factorizacion( [1  2  4  1; 2 8 6 4;3 10 8 8;4 12 10 6])

ans =

     1     2     4     1
     2     4    -2     2
     3     1    -2     3
     4     1     2    -6
```

2.2.6. Ejemplo

Dado el sistema lineal de ecuaciones, calcular: a) La factorización de Crout y b) Calcular la solución utilizando sustitución hacia adelante y atrás

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

a) Realizamos la factorización de LU o Crout

```
>> Af = Factorizacion([10 -1 2 1; -1 15 -3 1; 2 -3 6 -3; 1 1 -3 7])

Af =

   10.0000   -1.0000    2.0000    1.0000
```

```
   -0.1000   14.9000   -2.8000    1.1000
    0.2000   -0.1879    5.0738   -2.9933
    0.1000    0.0738   -0.5899    5.0529
```

b) Par la sustitución hacia adelante hacemos

```
>> y = sustitucion_hacia_adelante(Af, [1,2,2,1]')

y =

    1.0000
    2.1000
    2.1946
    2.0397
```

para luego realizar la sustitución hacia atrás

```
>> x = sustitucion_hacia_atras(Af, y)

x =

   -0.0508
    0.2372
    0.6707
    0.4037
```

2.2.7. Cálculo de la Matriz inversa utilizando descomposición LU

Si consideramos un sistema lineal $Ax = b$ la solución la podemos llevar a cabo utilizando el concepto de matriz inversa tal que $x = A^{-1}b$, por lo que podemos consideran que son métodos equivalentes. Consideremos un sistema de 4 ecuaciones con 4 incógnitas como el siguiente:

$$Ax = b$$

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

Si tuviéramos la matriz inversa, podemos calcular los valores de x haciendo:

$$x = A^{-1}b$$

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} a'_{1,1} & a'_{1,2} & a'_{1,3} & a'_{1,4} \\ a'_{2,1} & a'_{2,2} & a'_{2,3} & a'_{2,4} \\ a'_{3,1} & a'_{3,2} & a'_{3,3} & a'_{3,4} \\ a'_{4,1} & a'_{4,2} & a'_{4,3} & a'_{4,4} \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} a'_{1,3} \\ a'_{2,3} \\ a'_{3,3} \\ a'_{4,3} \end{bmatrix}$$

donde $a'_{n,m}$ son los elementos de la matriz inversa A^{-1}. Note que poniendo un 1 en b_3 obtuvimos la tercer columna de la matriz inversa, por lo tanto si solucionamos el sistema de ecuaciones utilizando descomposición LU obtendríamos la tercer columna.

Podemos generalizar el método si definimos

$$b^{(k)} = \begin{cases} b_n = 1 & si \quad n = k \\ b_n = 0 & si \quad n <> k \end{cases}$$

Entonces si resolvemos el sistema $Ax = b^{(k)}$ por cualquier método, calculamos la k-esima colma de la matriz inversa y la solución será $x = A_k^{-1}$. Para calcular todas las columnas de la matriz inversa solamente es necesario calcular este procedimiento para $k = 1, 2, \cdots, N$. Para hacer eficiente el procedimiento solamente factorizamos las matriz una vez y utilizando sustitución hacia adelante y atrás resolvemos para cada uno de los valores de k. El código para hacer esto es:

```
function Ainv = inversa(A)
N = length(A);

Af = Factorizacion(A);

Ainv = zeros(N,N);

for k = 1: N
    b = zeros(N,1);
    b(k) = 1;
    y = sustitucion_hacia_adelante(Af, b);
    x = sustitucion_hacia_atras(Af, y);
    Ainv(:,k) = x;
end;
end
```

2.2.8. Ejemplo

Calcular la inversa de la matriz

$$A = \begin{bmatrix} 10 & -1 & 2 & 1 \\ -1 & 15 & -3 & 1 \\ 2 & -3 & 6 & -3 \\ 1 & 1 & -3 & 7 \end{bmatrix}$$

Para comenzar hacemos la factorización de la matriz utilizando el metido de Crout o LU

$$A = \begin{bmatrix} 10 & -1 & 2 & 1 \\ -1 & 15 & -3 & 1 \\ 2 & -3 & 6 & -3 \\ 1 & 1 & -3 & 7 \end{bmatrix}$$

El resultado es

$$LU = \begin{bmatrix} 1.0000 & 0.0000 & 0.0000 & 0.0000 \\ -0.1000 & 1.0000 & 0.0000 & 0.0000 \\ 0.2000 & -0.1879 & 1.0000 & 0.0000 \\ 0.1000 & 0.0738 & -0.5899 & 1.0000 \end{bmatrix} \begin{bmatrix} 10.0000 & -1.0000 & 2.0000 & 1.0000 \\ 0.0000 & 14.9000 & -2.8000 & 1.1000 \\ 0.0000 & 0.0000 & 5.0738 & -2.9933 \\ 0.0000 & 0.0000 & 0.0000 & 5.0529 \end{bmatrix}$$

Para calcular la primer columna de la matriz inversa hacemos

$$\begin{bmatrix} 10.0000 & -1.0000 & 2.0000 & 1.0000 \\ 0.0000 & 14.9000 & -2.8000 & 1.1000 \\ 0.0000 & 0.0000 & 5.0738 & -2.9933 \\ 0.0000 & 0.0000 & 0.0000 & 5.0529 \end{bmatrix} \begin{bmatrix} y_1^{(1)} \\ y_2^{(1)} \\ y_3^{(1)} \\ y_4^{(1)} \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

La solución es $y^{(1)} = [1.0000, 0.1000, -0.1812, -0.2143]^T$

$$\begin{bmatrix} 1.0000 & 0.0000 & 0.0000 & 0.0000 \\ -0.1000 & 1.0000 & 0.0000 & 0.0000 \\ 0.2000 & -0.1879 & 1.0000 & 0.0000 \\ 0.1000 & 0.0738 & -0.5899 & 1.0000 \end{bmatrix} \begin{bmatrix} x_1^{(1)} \\ x_2^{(1)} \\ x_3^{(1)} \\ x_4^{(1)} \end{bmatrix} = \begin{bmatrix} 1.0000 \\ 0.1000 \\ -0.1812 \\ -0.2143 \end{bmatrix}$$

La primer columna de la matriz inversa es $x^{(1)} = [0.1162, -0.0016, -0.0607, -0.0424]^T$

Para la segunda columna de la matriz inversa hacemos

$$\begin{bmatrix} 10.0000 & -1.0000 & 2.0000 & 1.0000 \\ 0.0000 & 14.9000 & -2.8000 & 1.1000 \\ 0.0000 & 0.0000 & 5.0738 & -2.9933 \\ 0.0000 & 0.0000 & 0.0000 & 5.0529 \end{bmatrix} \begin{bmatrix} y_1^{(2)} \\ y_2^{(2)} \\ y_3^{(2)} \\ y_4^{(2)} \end{bmatrix} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \end{bmatrix}$$

La solución es $y^{(2)} = [0, 1.0000, 0.1879, 0.0370]^T$

$$\begin{bmatrix} 1.0000 & 0.0000 & 0.0000 & 0.0000 \\ -0.1000 & 1.0000 & 0.0000 & 0.0000 \\ 0.2000 & -0.1879 & 1.0000 & 0.0000 \\ 0.1000 & 0.0738 & -0.5899 & 1.0000 \end{bmatrix} \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \\ x_3^{(2)} \\ x_4^{(2)} \end{bmatrix} = \begin{bmatrix} 0 \\ 1.0000 \\ 0.1879 \\ 0.0370 \end{bmatrix}$$

La segunda columna de la matriz inversa es $x^{(2)} = [-0.0016, 0.0743, 0.0414, 0.0073]^T$

Para la tercercolumna de la matriz inversa hacemos

$$\begin{bmatrix} 10.0000 & -1.0000 & 2.0000 & 1.0000 \\ 0.0000 & 14.9000 & -2.8000 & 1.1000 \\ 0.0000 & 0.0000 & 5.0738 & -2.9933 \\ 0.0000 & 0.0000 & 0.0000 & 5.0529 \end{bmatrix} \begin{bmatrix} y_1^{(3)} \\ y_2^{(3)} \\ y_3^{(3)} \\ y_4^{(3)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \end{bmatrix}$$

La solución es $y^{(3)} = [0, 0, 1.0000, 0.5899]^T$

$$\begin{bmatrix} 1.0000 & 0.0000 & 0.0000 & 0.0000 \\ -0.1000 & 1.0000 & 0.0000 & 0.0000 \\ 0.2000 & -0.1879 & 1.0000 & 0.0000 \\ 0.1000 & 0.0738 & -0.5899 & 1.0000 \end{bmatrix} \begin{bmatrix} x_1^{(3)} \\ x_2^{(3)} \\ x_3^{(3)} \\ x_4^{(3)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 1.0000 \\ 0.5899 \end{bmatrix}$$

La tercer columna de la matriz inversa es $x^{(3)} = [-0.0607, 0.0414, 0.2660, 0.1168]^T$

Para la última columna de la matriz inversa hacemos

$$\begin{bmatrix} 10.0000 & -1.0000 & 2.0000 & 1.0000 \\ 0.0000 & 14.9000 & -2.8000 & 1.1000 \\ 0.0000 & 0.0000 & 5.0738 & -2.9933 \\ 0.0000 & 0.0000 & 0.0000 & 5.0529 \end{bmatrix} \begin{bmatrix} y_1^{(4)} \\ y_2^{(4)} \\ y_3^{(4)} \\ y_4^{(4)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

La solución es $y^{(4)} = [0, 0, 0, 1]^T$

$$\begin{bmatrix} 1.0000 & 0.0000 & 0.0000 & 0.0000 \\ -0.1000 & 1.0000 & 0.0000 & 0.0000 \\ 0.2000 & -0.1879 & 1.0000 & 0.0000 \\ 0.1000 & 0.0738 & -0.5899 & 1.0000 \end{bmatrix} \begin{bmatrix} x_1^{(4)} \\ x_2^{(4)} \\ x_3^{(4)} \\ x_4^{(4)} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

La última columna de la matriz inversa es $x^{(4)} = [-0.0424, 0.0073, 0.1168, 0.1979]^T$

Finalmente la matriz inversa es la concatenación de la soluciones

$$A^{-1} = [x^{(1)}, x^{(2)}, x^{(3)}, x^{(4)}] = \begin{bmatrix} 0.1162 & -0.0016 & -0.0607 & -0.0424 \\ -0.0016 & 0.0743 & 0.0414 & 0.0073 \\ -0.0607 & 0.0414 & 0.2660 & 0.1168 \\ -0.0424 & 0.0073 & 0.1168 & 0.1979 \end{bmatrix}$$

2.3. 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 $a'_{n,m}$ los calculamos como:

$$\begin{aligned} a'_{n,m} &= a_{n,m} - a_{n,k}\frac{a_{k,m}}{a_{k,k}} \\ b'_n &= b_n - a_{n,k}\frac{b_k}{a_{k,k}} \end{aligned}$$

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

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

La solución del sistema Triangular superior lo podemos realizar utilizando el método de sustitución hacia atrás. La implementación en Matlab queda

```
function x = Eliminacion_Gaussiana(A0, b0)

  a = A0;
  b = b0;

  N = length(b);

  for k =1:N-1
    for n = k+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;
  disp(a);
  disp(b);

  x = sustitucion_hacia_atras(a, b);
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
```

2.4. 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

$$\left[\begin{array}{ccc} a_{1,1} & a_{1,2} & a_{1,3} \\ a_{2,1} & a_{2,2} & a_{2,3} \\ a_{3,1} & a_{3,2} & a_{3,3} \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right] = \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{array}\right] \left[\begin{array}{c} b_1 \\ b_2 \\ b_3 \end{array}\right]$$

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

$$\left[\begin{array}{ccc|ccc|c} a_{1,1} & a_{1,2} & a_{1,3} & 1 & 0 & 0 & b_1 \\ a_{2,1} & a_{2,2} & a_{2,3} & 0 & 1 & 0 & b_2 \\ a_{3,1} & a_{3,2} & a_{3,3} & 0 & 0 & 1 & b_3 \end{array}\right] \tag{2.1}$$

Primer paso

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

$$\frac{b_1 - a_{1,2}x_2 - a_{1,3}x_3}{a_{1,1}} = x_1 \tag{2.2}$$

Sustituyendo en la ecuación 2

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

$$0x_1 + \left(\frac{a_{2,1}}{a_{1,1}}\right)b_1 + \left(a_{2,2} - \frac{a_{2,1}a_{1,2}}{a_{1,1}}\right)x_2 + \left(a_{2,3}\frac{a_{2,1}a_{1,3}}{a_{1,1}}\right)x_3 = b_2 \tag{2.3}$$

Sustituyendo en la ecuación 3

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

$$0x_1 + \left(\frac{a_{3,1}}{a_{1,1}}\right)b_1 + \left(a_{3,2} - \frac{a_{3,1}a_{1,2}}{a_{1,1}}\right)x_2 + \left(a_{3,3} - \frac{a_{3,1}a_{1,3}}{a_{1,1}}\right)x_3 = b_3 \tag{2.4}$$

En forma matricial podemos escribir las ecuaciones 2.2, 2.3 y 2.4

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

La matriz aumentada dada por (2.1) 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

$$\left[\begin{array}{c|c|c} 1 & a'_{1,2} & a'_{1,3} \\ \hline 0 & a'_{2,2} & a'_{2,3} \\ \hline 0 & a'_{3,2} & a'_{3,3} \end{array}\right]\left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right]=\left[\begin{array}{c|c|c} c'_{1,1} & 0 & 0 \\ \hline c'_{2,1} & 1 & 0 \\ \hline c'_{3,1} & 0 & 1 \end{array}\right]\left[\begin{array}{c} b'_1 \\ b'_2 \\ b'_3 \end{array}\right] \tag{2.5}$$

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.

2.4.1. 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

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

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

2.4.2. Ejemplo 2

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

$$\left[\begin{array}{ccc} 1 & 1 & 2 \\ 1 & 3 & -6 \\ 2 & -1 & 2 \end{array}\right] \left[\begin{array}{c} x_1 \\ x_2 \\ x_3 \end{array}\right] = \left[\begin{array}{c} -1 \\ 7 \\ 0 \end{array}\right]$$

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

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

2.4.3. Ejemplo 3

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

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

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

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

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

2.5. Inversa de Shiplay

Consideremos el siguiente sistema de ecuaciones

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

de la ecuación I despejamos el valor de la variable x_1

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

Reorganizando la ecuación tenemos

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

Sustituimos el valor de x_1 en la ecuación II:

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

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

y finalmente en III tenemos :

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

En forma matricial estas ecuaciones lucen como:

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

Lo que nos da un sistema equivalente como el siguiente.

$$\left[\begin{array}{ccc} a'_{1,1} & a'_{1,2} & a'_{1,3} \\ a'_{2,1} & a'_{2,2} & a'_{2,3} \\ a'_{3,1} & a'_{3,2} & a'_{3,3} \end{array}\right] \left[\begin{array}{c} b_1 \\ x_2 \\ x_3 \end{array}\right] = \left[\begin{array}{c} x_1 \\ b_2 \\ b_3 \end{array}\right]$$

Si repetimos el procedimiento suponiendo que despejamos x_2 y sustituimos en las otras dos ecuaciones tendremos un sistema.

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

Lo que nos da un sistema equivalente como el siguiente.

$$\left[\begin{array}{ccc} a''_{1,1} & a''_{1,2} & a''_{1,3} \\ a''_{2,1} & a''_{2,2} & a''_{2,3} \\ a''_{3,1} & a''_{3,2} & a''_{3,3} \end{array}\right] \left[\begin{array}{c} b_1 \\ b_2 \\ x_3 \end{array}\right] = \left[\begin{array}{c} x_1 \\ x_2 \\ b_3 \end{array}\right]$$

Finalmente si despejamos x_3 de la ecuación III y sustituimos en las otras tenemos;

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

El procedimiento para el calculo de la inversa por el método de Shiplay es:

Dado un valor de $k = 1, 2, 3, ..., N$ al que llamamos pivote hacemos:

- $a_{n,m} = a_{n,m} – a_{n,k}a_{k,m}/a_{k,k}$ para todos los valores de n y m diferentes de k
- $a_{n,k} = a_{n,k}/a_{k,k}$ para todos los n diferentes de k
- $a_{k,m} = a_{k,m}/a_{k,k}$ para todos los m diferentes de k
- $a_{k,k} = 1/a_{k,k}$

La implementación en Matlab queda

```
function a = Inversa_Shiplay(A)
   a=A;
   N=length(a);
   for k=1:N
       for n=1:N
           for m=1:N
               if ((m~=k)&(n~=k))
                   a(n,m)=a(n,m)-(a(n,k)*a(k,m))/a(k,k);
               end;
           end;
       end;
       for n=1:N
           if (n~=k)
               a(n,k)=a(n,k)/a(k,k);
               a(k,n)=-(a(k,n)/a(k,k));
           end;
       end;
       a(k,k)=1/a(k,k);
   end;
end
```

2.5.1. Ejemplo:

Determinar la matriz inversa de

$$\begin{bmatrix} 3 & -1 & -1 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{bmatrix}$$

Primer paso:

$$\begin{bmatrix} 1/3 & 1/3 & 1/3 \\ -1/3 & 2/3 & -1/3 \\ -1/3 & -1/3 & 2/3 \end{bmatrix}$$

Segundo paso:

$$\begin{bmatrix} 1/2 & 1/2 & 1/2 \\ 1/2 & 3/2 & 1/2 \\ -1/2 & -1/2 & 1/2 \end{bmatrix}$$

Tercer paso:

$$\begin{bmatrix} 1 & 1 & 1 \\ 1 & 2 & 1 \\ 1 & 1 & 2 \end{bmatrix}$$

La solución en Matlab es:

```
>> Inversa_Shiplay([ 3 -1 -1;-1  1  0 ; -1  0  1])

ans =

    1.0000    1.0000    1.0000
    1.0000    2.0000    1.0000
    1.0000    1.0000    2.0000
```

2.6. Sistemas dispersos y estrategias para conservarla

Consideremos un sistema de ecuaciones $Ax = b$ como la siguiente

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

Para ésta matriz el total de elementos es 25 de los cuales 13 son diferentes de cero y 12 son igual cero. A este tipo de matrices o sistemas de ecuaciones se les conoce como dispersos o ralos dado que tienen una cantidad de ceros igual o mayor que los diferentes de cero.

Para este tipo de sistema cuando calculamos la solución por lo métodos dados tenemos

Descomposición LU

```
>> Factorizacion(A)

ans =

   10.0000   -1.0000   -1.0000   -1.0000    1.0000
    0.1000    4.1000    0.1000    0.1000   -0.1000
    0.1000    0.0244    4.0976    0.0976   -0.0976
    0.1000    0.0244    0.0238    4.0952   -0.0952
    0.1000    0.0244    0.0238    0.0233    2.9070
```

Eliminación Gaussiana

```
>> Eliminacion_Gaussiana(A,b)
   10.0000   -1.0000   -1.0000   -1.0000    1.0000
         0    4.1000    0.1000    0.1000   -0.1000
         0         0    4.0976    0.0976   -0.0976
         0         0         0    4.0952   -0.0952
         0         0         0         0    2.9070
```

Inversa de Shiplay

```
>> Inversa_Shiplay(A)

ans =

    0.0960    0.0240    0.0240    0.0240   -0.0320
   -0.0240    0.2440   -0.0060   -0.0060    0.0080
   -0.0240   -0.0060    0.2440   -0.0060    0.0080
   -0.0240   -0.0060   -0.0060    0.2440    0.0080
```

```
   -0.0320   -0.0080   -0.0080   -0.0080    0.3440
```

Note que en todos los casos la dispersidad se pierde y tenemos una matriz con números diferentes de cero ahí donde existían ceros.

En este caso una opción es el manejo de Matrices Ralas del Matlab para no almacenar los valores iguales a cero y utilizar métodos iterativo como los de la siguiente sección.

La forma de representar la matriz dispersa en Matlab es

```
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);
```

2.7. Método iterativo de Jacobi y Gauss-Seidel

2.7.1. Método 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;
```

2.7.2. 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
```

2.7.3. 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
```

Solución de Ecuaciones No lineales

3.1. Introducción

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

3.2. Métodos de punto fijo

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])
```

3.3. El método de Biseccciones

Para este método debemos considerar una función unidimensional $f : \mathbb{R} \to \mathbb{R}$ 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;
```

3.3.1. 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	0.0000	0.5000	1.0000	-1.0000	-0.3776	0.4597
1	0.5000	0.7500	1.0000	-0.3776	0.0183	-0.3776
2	0.5000	0.6250	0.7500	-0.3776	-0.1860	-0.3776
3	0.6250	0.6875	0.7500	-0.1860	-0.0853	-0.1860
4	0.6875	0.7188	0.7500	-0.0853	-0.0339	-0.0853
5	0.7188	0.7344	0.7500	-0.0339	-0.0079	-0.0339
6	0.7344	0.7422	0.7500	-0.0079	0.0052	-0.0079
7	0.7344	0.7383	0.7422	-0.0079	-0.0013	-0.0079
8	0.7383	0.7402	0.7422	-0.0013	0.0019	-0.0013
9	0.7383	0.7393	0.7402	-0.0013	0.0003	-0.0013
10	0.7383	0.7388	0.7393	-0.0013	-0.0005	-0.0013
11	0.7388	0.7390	0.7393	-0.0005	-0.0001	-0.0005
12	0.7390	0.7391	0.7393	-0.0001	0.0001	-0.0001
13	0.7390	0.7391	0.7391	-0.0001	0.0000	-0.0001

Para ver los resultados correr

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

3.4. Método de 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 3.3 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.

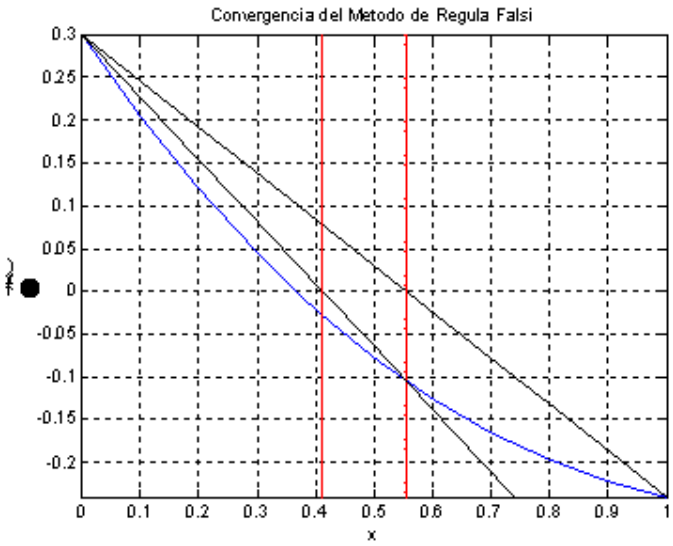

Figura 3.3: Método Regula Falsi

Para calcula la ecuación de la línea secante hacemos

$$\begin{aligned} p1 &= [a, f(a)] \\ p2 &= [b, f(b)] \end{aligned}$$

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)
```

3.4.1. 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	-0.0893
2	0.7363	0.7389	1.0000	-0.0047	-0.0002	-0.0047
3	0.7389	0.7391	1.0000	-0.0002	0.0000	-0.0002
4	0.7391	0.7391	1.0000	0.0000	0.0000	0.0000
5	0.7391	0.7391	1.0000	0.0000	0.0000	0.0000
6	0.7391	0.7391	1.0000	0.0000	0.0000	0.0000

3.4.2. 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 3.4.

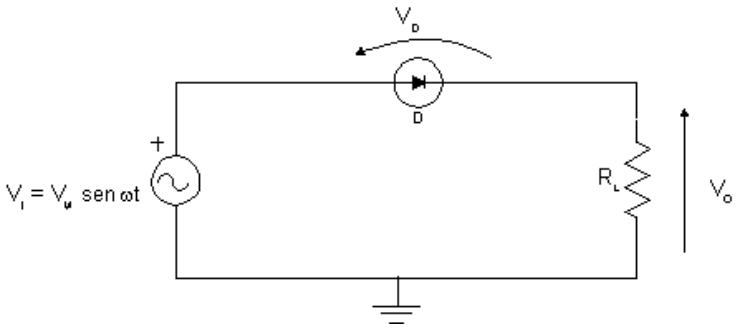

Figura 3.4: Circuito de corriente alterna con un diodo

La ecuación que modela al diodo esta dada por (3.6) y la curva se muestra en la figura 3.5:

$$i_d = I_s(e^{v_d/V_t} - 1) \tag{3.6}$$

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. 3.6

```
function Simula_Circuito_Diodo()
    % Paramentros del Cicuito

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

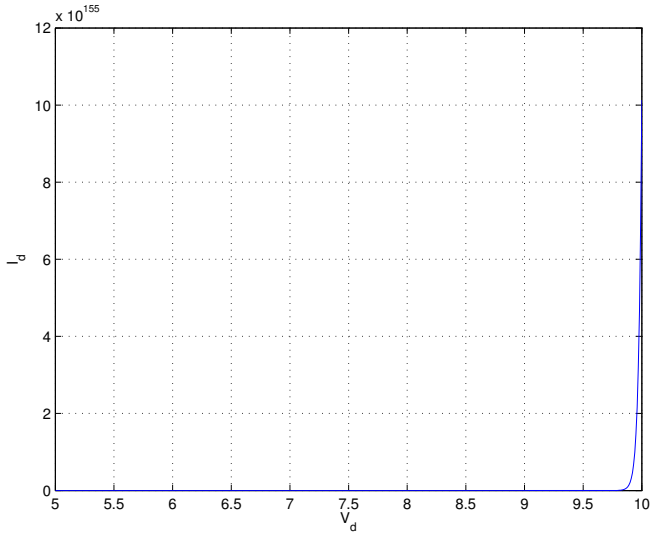

Figura 3.5: Comportamiento del diodo

```
    %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
```

3.5. El método Newton Raphson

Consideremos el sistema de ecuaciones no lineales

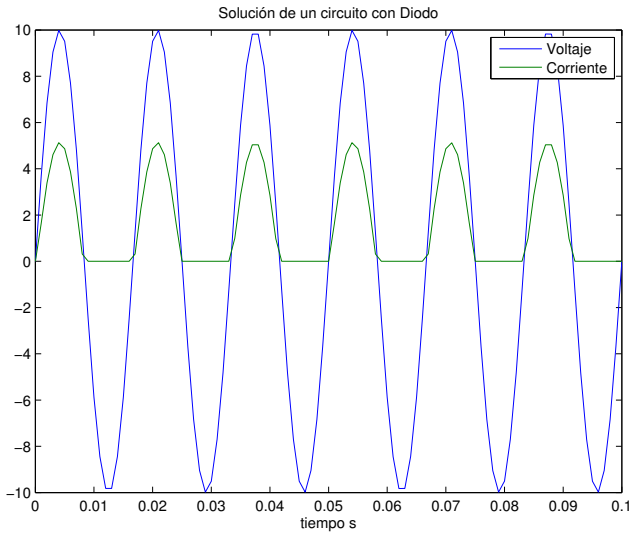

Figura 3.6: Solución al circuito con diodo

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

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;
```

3.5.1. Ejemplo 1

Resolver el siguiente sistema de ecuaciones dado por y cuya gráfica se muestra en la Figura 3.7

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

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

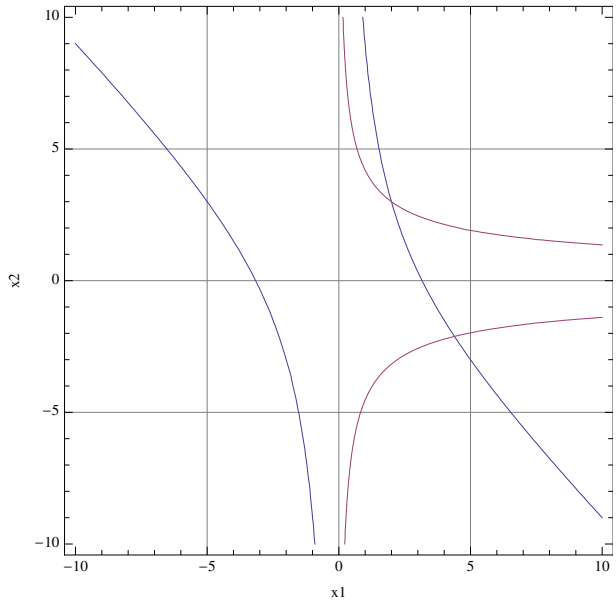

Figura 3.7: Curvas con la solución del sistema del ejemplo 1

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

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
```

3.5.2. Ejemplo 2

Resolver el siguiente sistema de ecuaciones dado por las ecuaciones y cuya solución gráfica se muestra en la figura 3.8

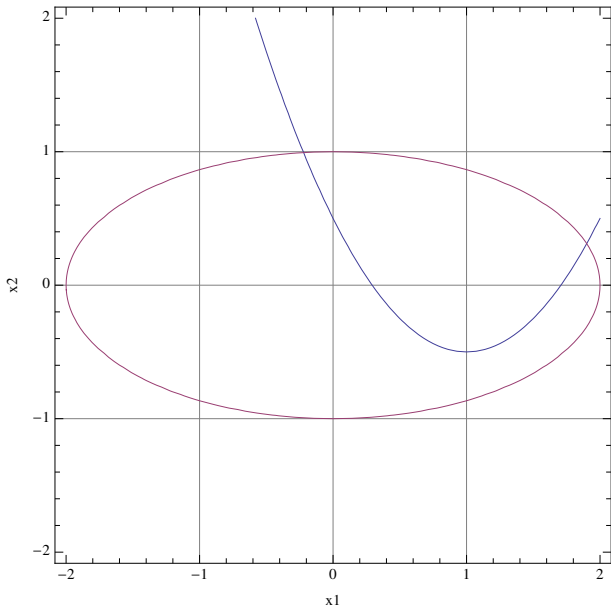

Figura 3.8: Curvas con la solución del sistema del ejemplo 2

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

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
```

3.5.3. Ejemplo

Para el circuito que se muestra en la figura 3.9, 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.

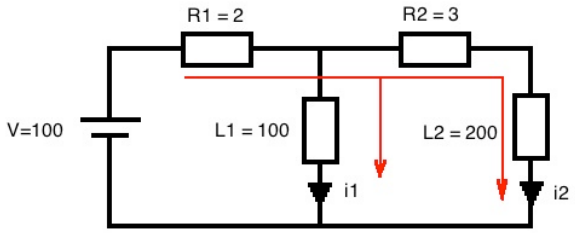

Figura 3.9: Circuito eléctrico de dos mallas

La ecuaciones de Voltaje para cada una de las mallas es:

$$V_1 - R_1(i_1 + i_2) - L_1/i_1 = 0$$
$$V_1 - R_1(i_1 + i_2) - R_2 i_2 - L_2/i_2 = 0$$

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.

3.6. Convergencia del método de Newton Raphson

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

Ajuste de curvas

4.1. 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.

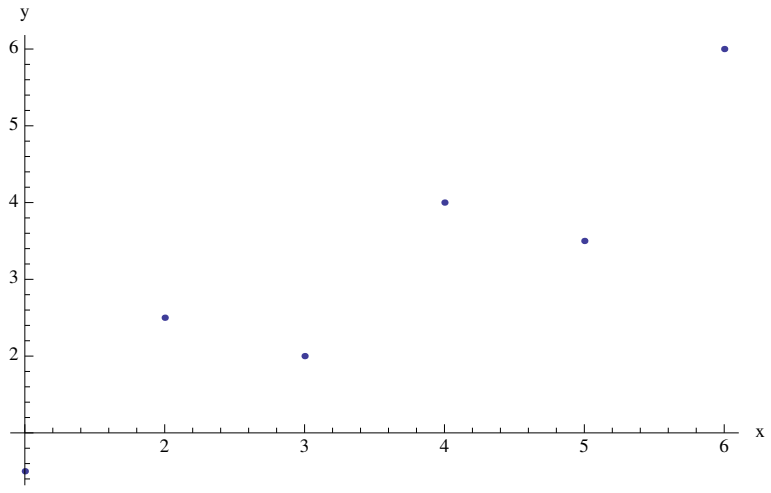

Figura 4.10: Un conjunto de puntos

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.

4.1.1. 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
```

4.1.2. 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 4.11 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)
```

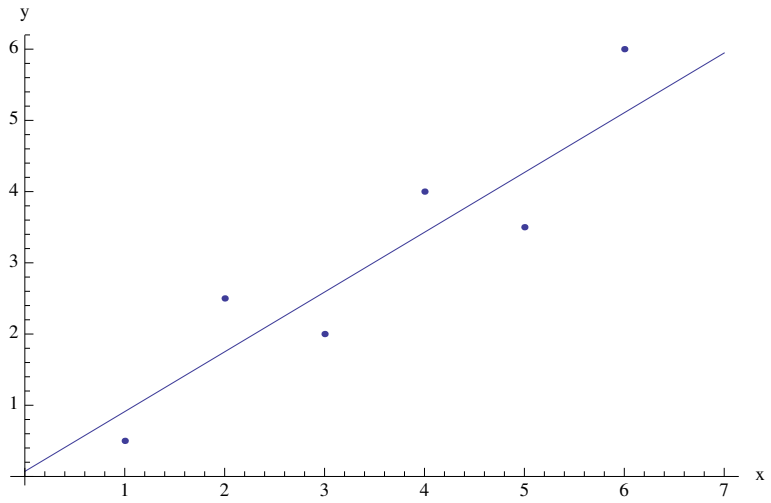

Figura 4.11: Un conjunto de puntos y la linea recta ajustada

4.1.3. 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

$$[M^T M]a = M^T y \tag{4.7}$$

El sistema se resuelve para a o simplemente se calcula haciendo

$$a = [M^T M]^{-1}[M^T y]$$

4.1.4. 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 4.12 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)
```

4.2. 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 $P_0 = [x_0, f(x_0)]^T$

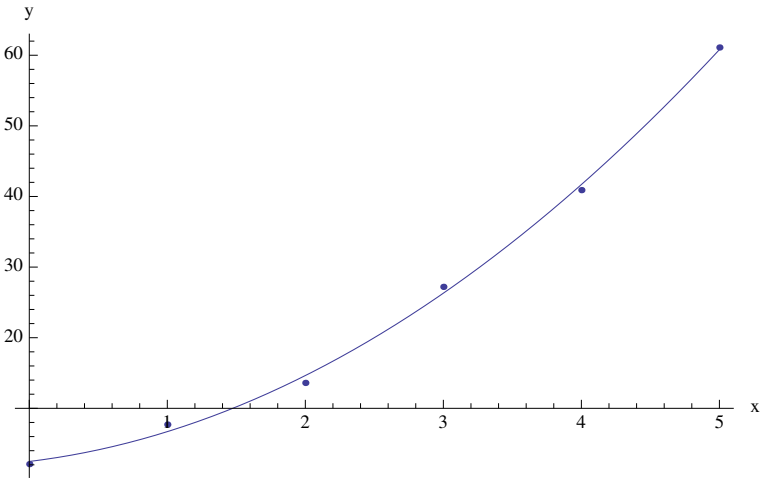

Figura 4.12: Un conjunto de puntos y su aproximación cuadrática

$$\frac{f(x) - f(x_0)}{x - x_0} = \frac{f(x_1) - f(x_0))}{x_1 - x_0}$$

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
```

4.2.1. 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 4.13, se muestran los puntos correspondientes calculados de acuerdo con el código implementado

4.3. Interpolación cuadrática

El error que se observa en la figura 4.13, 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

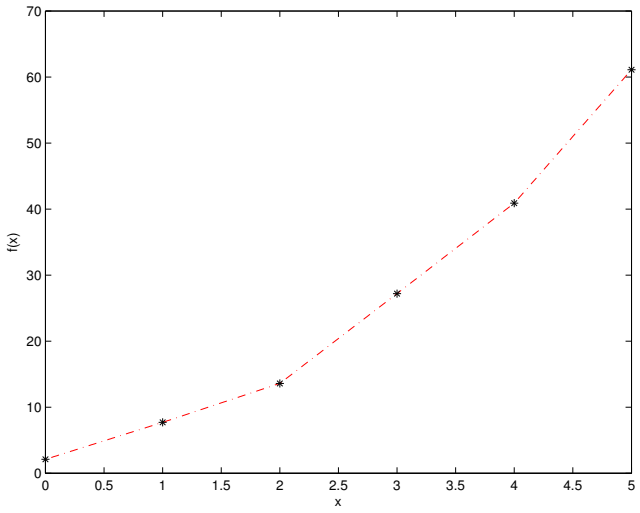

Figura 4.13: Un conjunto de puntos y su interpolación lineal

$$q(x) = b_0 + b_1(x - x_0) + b_2(x - x_0)(x - x_1)$$

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

$$f(x_0) = b_0 + b_1(x_0 - x_0) + b_2(x_0 - x_0)(x - x_1)$$

por lo tanto

$$b_0 = f(x_0)$$

Para calcular b_1 utilizamos el segundo punto así tenemos

$$f(x_0) = b_0 + b_1(x_1 - x_0) + b_2(x_1 - x_0)(x_1 - x_1)$$

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

$$f(x_0) = f(x_0) + \frac{f(x_1) - f(x_0)}{x_1 - x_0}(x_2 - x_0) + b_2(x_2 - x_0)(x_2 - x_1)$$

despejando tenemos

$$b_2 = \frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2 - x_0}$$

4.3.1. 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

$$b_2 = \frac{\frac{f(x_2)-f(x_1)}{x_2-x_1} - \frac{f(x_1)-f(x_0)}{x_1-x_0}}{x_2 - x_0}$$

$$b_2 = \frac{\frac{1.791759-1.386294}{6-4} - 0.4620981}{6 - 1} = -0.0518731$$

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
```

4.3.2. 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

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 4.14, se muestran los puntos correspondientes calculados de acuerdo con el código implementado

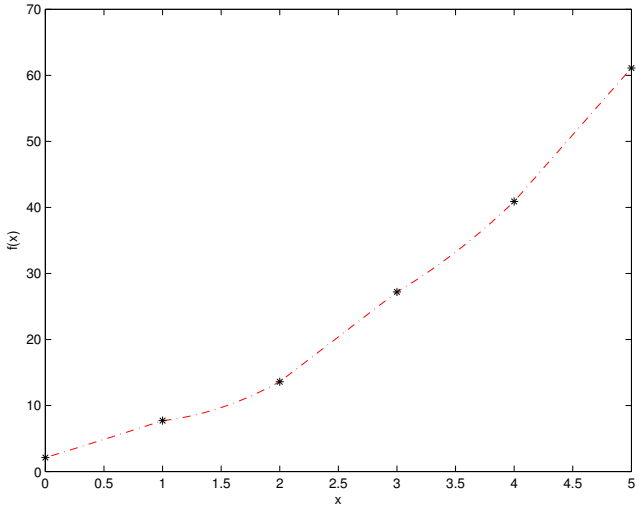

Figura 4.14: Un conjunto de puntos y su interpolación cuadrática

4.4. 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

$$f_n(x) = b_0 + b_1(x - x_0) + b_2(x - x0)(x - x_1) + \cdots + b_n(x - x_0)(x - x_1)\cdots(x - x_{n-1}) \quad (4.8)$$

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 4.8, 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
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
```

4.4.1. 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
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 $f_3(2) = 0.6287686$

4.4.2. 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 4.15, se muestran los puntos correspondientes calculados de acuerdo con el código implementado

4.5. 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

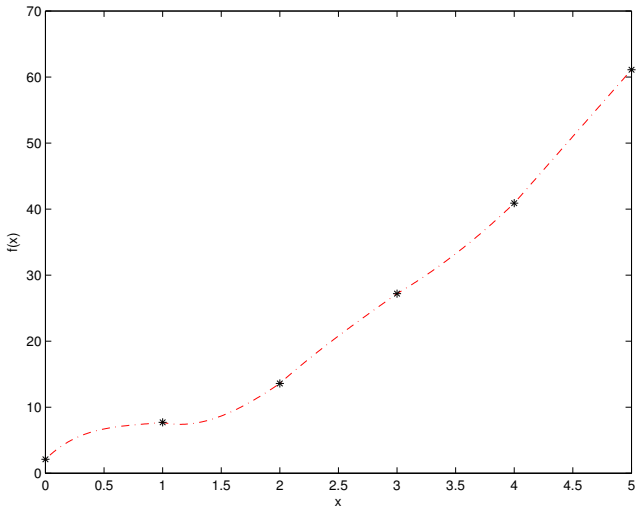

Figura 4.15: Un conjunto de puntos y su interpolación con Polinomios de Newton de Quinto grado

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

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
```

4.5.1. 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

$$f_1(x) = \frac{2-4}{1-4}0 + \frac{2-1}{4-1}1.386294 = 0.4620981$$

y la solución de segundo orden es

$$f_2(x) = \frac{(2-4)(2-6)}{(1-4)(1-6)}0 + \frac{(2-1)(2-6)}{(4-1)(4-6)}1.386294 + \frac{(2-1)(2-4)}{(6-1)(6-4)}1.791760 = 0.5658444$$

Como era de esperar, ambos resultados concuerdan con los que se obtuvieron utilizando interpolación utilizando polinomios de Newton.

Diferenciación e Integración

5.1. 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

$$\frac{df(x)}{dx} \approx \frac{f(x+h) - f(x-h)}{2h}$$

5.2. 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 a y b es el área

bajo la curva. Así podemos hacer este cálculo del área suponiendo que nuestros rectángulos son de altura $f(a)$. Recordemos que el área de un rectángulo esta dado como

$$A = (b-a)f(a)$$

por lo que la integral la podemos aproximar por

$$I = \int_a^b f(x)dx \approx \int_a^b f(a)dx = f(a)x|_a^b = f(a) * (b-a)$$

5.2.1. 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 $a = 0$ $b = 0.8$.

$$I \approx f(0) * (0.8 - 0) = 0.2 \times 0.8 = 0.16$$

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:

$$\int_a^b f(x)dx \approx \sum_{k=0}^{N-1} f(a+kh) * h$$

Esta aproximación da lugar al método de barras cuya codificación en Matlab se presenta a continuación

```
function I = Barras(f, a, b, N)

h = abs(b-a)/N;

suma = 0;
for k=0:N-1
```

```
    suma = suma + f(a+h*k);
end;

I = suma*h;

end
```

5.3. Integración por el método de trapezoides

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

$$f(x) = f(a) + \frac{(x-a)}{b-a}(f(b) - f(a))$$

A partir de esto podemos hacer la aproximación de nuestra integral

$$I = \int_a^b f(x)dx \approx \int_a^b \left(f(a) + \frac{(x-a)}{b-a}(f(b) - f(a)) \right) dx$$

resolviendo la integral tenemos

$$I \approx \left(f(a)x + \frac{(x-a)^2}{2(b-a)}(f(b) - f(a)) \right) \Bigg|_a^b$$

$$I \approx \frac{f(b) + f(a)}{2}(b-a)$$

La cual es el área de un trapecio de altura menor $f(a)$ y altura mayor $f(b)$.

5.3.1. Ejemplo

Para la función $f(x) = 0.2 + 25x - 200x^2 + 675x^3 - 900x^4 + 400x^5$, calcular la integral utilizando el método de la Regla Trapezoidal en el intervalo $a = 0$ $b = 0.8$.

$$I \approx \frac{f(b) + f(a)}{2}(b-a) = \frac{(0.2320 + 0.2)(0.8 - 0)}{2} = 0.1728$$

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

$$I \approx (b-a)\frac{f(a) + 2\sum_{k=1}^{N-1} f(a+kh) + f(b)}{2N}$$

La implementación en Matlab para la regla trapezoidal

```
function I = Regla_Trapezoidal(f, a, b, N)

h = abs(b-a)/N;

suma = f(a);

for k=1:N-1
    suma = suma + 2*f(a+h*k);
end;
suma = suma + f(b);

I = (b-a)*suma/(2*N);

end
```

5.4. 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

$$f(x) = \frac{(x-x_1)(x-x_2)}{(x_0-x_1)(x_0-x_2)}f(x_0) + \frac{(x-x_0)(x-x_2)}{(x_1-x_0)(x_1-x_2)}f(x_1) + \frac{(x-x_0)(x-x_1)}{(x_2-x_0)(x_2-x_1)}f(x_2)$$

Aplicando la integración de la función de interpolación tenemos:

$$I \approx \int_a^b f(x)dx = (b-a)\frac{f(x_0)+4f(x_1)+f(x_2)}{6}$$

5.4.1. 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 $a = 0$ $b = 0.8$.

$$I \approx (b-a)\frac{f(x_0)+4f(x_1)+f(x_2)}{6} = (0.8-0)\frac{f(0)+4f(0.4)+f(0.8)}{6} =$$

$$I \approx (0.8-0)\frac{(0.2+4\times 2.4560+0.2320)}{6} = 1.3675$$

En Matlab la solución es

```
>> Simpson13(@f1, 0, 0.8, 2)

ans =

    1.3675
```

La implementación para N divisiones de tamaño h es:

$$I \approx (b-a)\frac{f(x_a)+4\sum_{k=1,3,5,\cdots}^{N-1} f(a+k*h)+2\sum_{k=2,4,6,\cdots}^{N-1} f(a+k*h)+f(b)}{3N}$$

y la implementación en Matlab es:

```
function I = Simpson13(f, a, b, N)

h = abs(b-a)/N;

s_par = 0;
s_impar = 0;

for k=1:N-1
    if mod(k,2) == 1
        s_impar = s_impar + f(a+h*k);
    else
```

```
        s_par = s_par + f(a+h*k);
    end;
end;

I = (b-a)*(f(a) + 4*s_impar + 2*s_par + f(b))/(3*N);

end
```

5.5. 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

$$\begin{aligned} f(x) &= \frac{(x-x_1)(x-x_2)(x-x_3)}{(x_0-x_1)(x_0-x_2)(x_0-x_3)}f(x_0) \\ &+ \frac{(x-x_0)(x-x_2)(x-x_3)}{(x_1-x_0)(x_1-x_2)(x_1-x_3)}f(x_1) \\ &+ \frac{(x-x_0)(x-x_1)(x-x_3)}{(x_2-x_0)(x_2-x_1)(x_2-x_3)}f(x_2) \\ &+ \frac{(x-x_0)(x-x_1)(x-x_2)}{(x_3-x_0)(x_3-x_1)(x_3-x_2)}f(x_3) \end{aligned}$$

Resolviendo para esta función polinomial

$$I \approx \int_a^b f(x)dx = (b-a)\frac{f(x_0)+3f(x_1)+3f(x_2)+f(x_3)}{8}$$

5.5.1. 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 $a = 0$ $b = 0.8$.

$$I \approx (0.8-0)\frac{f(0)+3f(0.2667)+3f(0.5333)+f(0.8)}{8} =$$

$$I \approx (0.8-0)\frac{(0.2+3\times 1.4327+3\times 3.4872+0.2320)}{8} = 1.5192$$

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, a, b, N)
h = abs(b-a)/N;

suma =0;

for k=0:N-1
    x0 = a + 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
```

5.6. Ejemplos

5.6.1. 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 $a = 0$ $b = 0.8$ con $N = 1$ aplicando los métodos vistos y comparar con la solución real dada por la ecuación.

Comenzamos por calcular la integral de $f(x)$

$$I(x) = \int_a^b f(x)dx = 0.2x + \frac{25}{2}x^2 - \frac{200}{3}x^3 + \frac{675}{4}x^4 - \frac{900}{5}x^5 + \frac{400}{6}x^6$$

La solución para cada uno de los métodos es

Método	Solución	error
Barras	01600	1.4805
R Trapezoidal	0.1728	1.4677
Simpson 1/3	1.3675	0.2731
Simpson 3/8	1.5192	0.1214

Note que el mejor desempeño lo tiene el método de Simpson 3/8 en virtud de utilizar polinomios de grado 3 para llevar a cabo la integración.

5.6.2. Ejemplo 2

Repetir el ejemplo anterior para $N = 100$.

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.

6.1. 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 solucion = Euler(xini, h, N, f)
    D = length(xini);
    x = xini(1);
    y = xini(2:D);

    solucion = zeros(N+1,D);
    solucion(1,:) = [x,y];

    for k = 2:N+1
        x = x + h;
        y = y + f(x, y)*h;
        solucion(k,:)=[x, y];
    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.

6.2. 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 solucion = Heun(xini, h, N, f)
    D = length(xini);
    x = xini(1);
    y = xini(2:D);

    solucion = zeros(N+1,D);
    solucion(1,:) = [x,y];

    for k = 2:N+1
        %Predictor
        k1 = f(x,   y       );
        k2 = f(x+h, y + k1*h);
        %corrector
        x = x + h;
        y = y + (k1 + k2 )*h/2.0;
        solucion(k,:) = [x,y];
    end;
end
```

6.3. 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 solucion = Punto_Medio(xini, h, N, f)
    D = length(xini);
    x = xini(1);
    y = xini(2:D);

    solucion = zeros(N+1,D);
    solucion(1,:) = [x,y];

    for k = 2:N+1

        xm = x + h/2;
        ym = y + f(x, y)*h/2;

        x = x + h;
        y = y + f(xm, ym)*h;
        solucion(k,:) = [x,y];
    end;
end
```

6.4. Fórmulas de integración por algunos métodos de Runge-Kutta de segundo orden clásicos

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

$$k_1 = f(x^{(k)}, y^{(k)})$$

$$k_2 = f(x^{(k)} + \frac{3}{4}h, \frac{3}{4}k_1 h)$$

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 solucion = RK(xini, h, N, f)
    D = length(xini);
    x = xini(1);
    y = xini(2:D);

    solucion = zeros(N+1,D);
    solucion(1,:) = [x,y];

    for k = 2:N+1
        k1 = f(x, y);
        k2 = f(x + 3*h/4, y+3*h*k1/4);

        x = x + h;
        y = y + (k1/3 + 2*k2/3)*h;
        solucion(k,:) = [x,y];
    end;
end
```

6.5. 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

$$k_1 = f(x^{(k)}, y^{(k)})$$

$$k_2 = f(x^{(k)} + \frac{1}{2}h, \frac{1}{2}k_1 h)$$

$$k_3 = f(x^{(k)} + h, y^{(k)} - k_1 h + 2k_2 h)$$

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 solucion = RK3(xini, h, N, f)
    D = length(xini);
    x = xini(1);
    y = xini(2:D);

    solucion = zeros(N+1,D);
    solucion(1,:) = [x,y];

    for k = 2:N+1

        k1 = f(x,       y);
        k2 = f(x + h/2, y + k1*h/2);
        k3 = f(x + h,   y - k1*h + 2*k2*h);

        x = x + h;
        y = y + (k1 + 4*k2 + k3)*h/6;
        solucion(k,:) = [x,y];
    end;
end
```

6.6. 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 solucion = RK4(xini, h, N, f)
    D = length(xini);
    x = xini(1);
    y = xini(2:D);

    solucion = zeros(N+1,D);
    solucion(1,:) = [x,y];

    for k = 2:N+1

        k1 = f(x,       y);
        k2 = f(x + h/2, y + k1*h/2);
        k3 = f(x + h/2, y + k2*h/2);
        k4 = f(x + h,   y + k3*h);

        x = x + h;
        y = y + (k1 + 2*k2 + 2*k3 + k4)*h/6;
        solucion(k,:) = [x,y];
    end;
end
```

6.7. Ejemplo

6.7.1. 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 = Euler([xini, y_ini], h, N, @ecuacion);
    x2 = Heun([xini, y_ini], h, N, @ecuacion);
    x3 = Punto_Medio([xini, y_ini], h, N, @ecuacion);
    x4 = RK([xini, y_ini], h, N, @ecuacion);
    x5 = RK3([xini, y_ini], h, N, @ecuacion);
    x6 = RK4([xini, y_ini], h, N, @ecuacion);

    [x, solucion_real(x)- x1(:,2), x2(:,2), x3(:,2), x4(:,2), x5(:,2), x6(:,2)]
    [x, abs(solucion_real(x)-x1(:,2)),...
        abs(solucion_real(x)-x2(:,2)),...
        abs(solucion_real(x)-x3(:,2)),...
        abs(solucion_real(x)-x4(:,2)),...
        abs(solucion_real(x)-x5(:,2)),...
        abs(solucion_real(x)-x6(:,2))]

    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
```

6.7.2. Circuito RL

Para el circuito RL que se muestra en la figura 6.16 calcular el valor de la corriente $i(t)$ en función del tiempo t, asumiendo que el interruptor se cierra en el tiempo $t = 0$

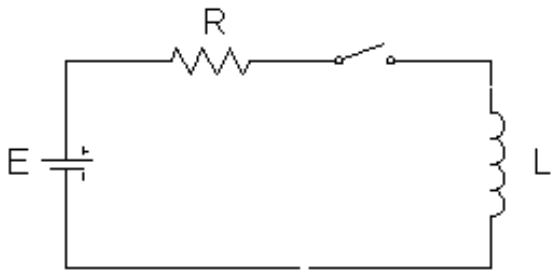

Figura 6.16: Circuito RL

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 6.17 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.

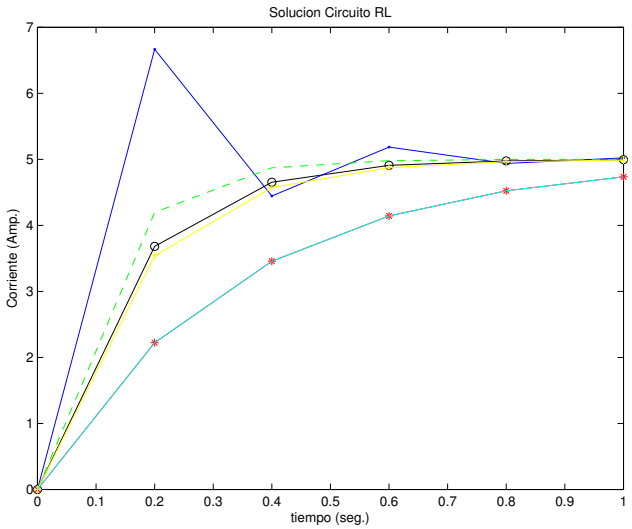

Figura 6.17: Circuito RL

El código implementado para este ejemplo es:

```
function Circuito_RL()

    % Numero de puntos
    N = 5;

    % Valores Iniciales
    tini = 0;
    tfin = 1;
    i_ini = 0;
    h = (tfin-tini)/N;


    x0 = Solucion_Real([tini, i_ini], h, N);
    x1 = Euler([tini, i_ini], h, N, @Circuito_RL);
    x2 = RK([tini, i_ini], h, N, @Circuito_RL);
    x3 = RK3([tini, i_ini], h, N, @Circuito_RL);
    x4 = RK4([tini, i_ini], h, N, @Circuito_RL);

    x5 = Heun([tini, i_ini], h, N, @Circuito_RL);


    plot(x0(:,1), x0(:,2), 'ko-', x1(:,1), x1(:,2), 'b .-', ...
         x2(:,1), x2(:,2), 'r*-', x3(:,1), x3(:,2), 'g--', ...
         x4(:,1), x4(:,2), 'y+-', x5(:,1), x5(:,2), 'c');

    xlabel('tiempo (seg.)');
    ylabel('Corriente (Amp.)');
    title('Solucion Circuito RL');

    % Solucion exacta de la ecuacion Diferencial

    function x =Solucion_Real(xini, h, N)
       %Parametros

        R = 2.0;  % Ohms
        L = 0.3;  % Henries
        V = 10.0; % Volts

        t = [xini:h:N*h]';
        Tau = L/R;
        i = 5 *(1 - exp(-t/Tau));
```

```
        x = [t, i];
    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
```

6.7.3. Sistema Masa Resorte

El sistema masa resorte como el que se muestra en la figura 6.18, 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:

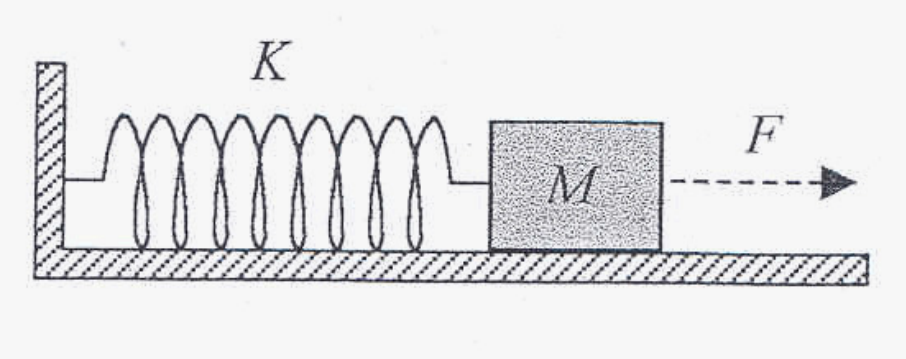

Figura 6.18: Sistema Masa Resorte

$$M\frac{dx^2}{dt^2} + C\frac{dx}{dt} + Kx = F$$

Note que tenemos una ecuación diferencial de segundo orden la cual plantearemos como un par de ecuaciones de primer orden como:

$$\frac{dx}{dt} = v$$

$$\frac{dv}{dt} = \frac{F - Cv - Kx}{M}$$

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 6.19 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;

    x = Metodo([tini, d_ini, v_ini], h, N, @Sistema_MR);
```

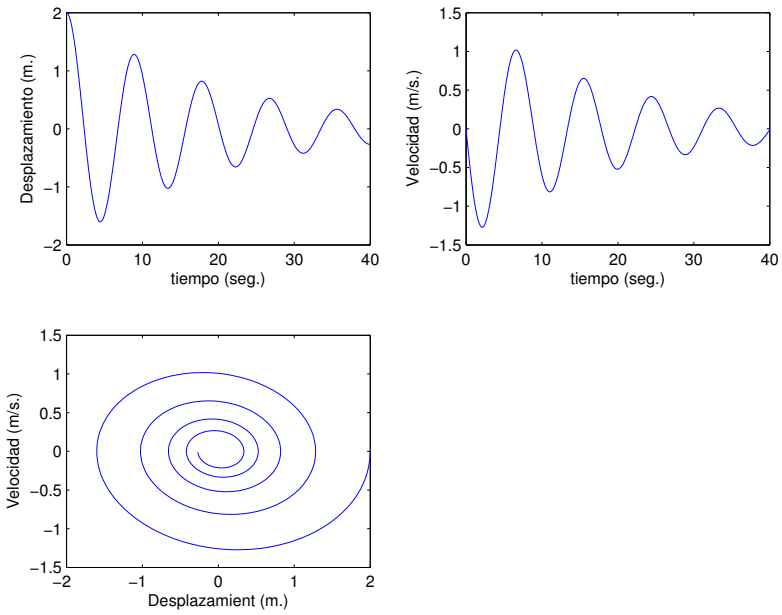

Figura 6.19: Sistema Masa Resorte

```
    title('Sistema Masa Resorte');

    subplot(2,2,1)
    plot(x(:,1), x(:,2));
    xlabel('tiempo (seg.)');
    ylabel('Desplazamiento (m.)');

    subplot(2,2,2)
    plot(x(:,1), x(:,3));
    xlabel('tiempo (seg.)');
    ylabel('Velocidad (m/s.)');

    subplot(2,2,3)
    plot(x(:,2), x(:,3));
    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
```