Universidad de las Fuerzas Armadas

Métodos Numéricos

Actividad 3

Apellidos: Guzmán Cajas

Nombres: Diego Alejandro

NRC: 3657

GitHub: https://github.com/GuzmanDiegoEspe/MetodosNumericos-GuzmanDiego

1) Para el sistema lineal.

$$\begin{cases} 6x_1 = 12 & (1) \\ 6x_2 + 3x_1 = -12 & (2) \\ 7x_3 - 2x_2 + 4x_1 = 14 & (3) \\ 21x_4 + 9x_3 - 3x_2 + 5x_1 = -2 & (4) \end{cases}$$

Encuentre los valores de x_1, x_2, x_3, x_4 utilizando sustitución progresiva. Encuentre la descomposición LU del sistema anterior. Que puede concluir?

Con nuestra primer ecuación encontramos: x_1

$6x_1 = 12$

$x_1 = \frac{12}{6} = 2$

Con x_1 encontramos: x_2

$6x_2 + 3x_1 = -12$

$x_2 = \frac{-12-3(2)}{6}$

x_2 = -3

Teniendo los valores de x_1, x_2 encontramos: x_3

$7x_3 - 2x_2 + 4x_1$ = 14

$x_3 = \frac{14-4(2)+2(-3)}{7}$

x_3 = 0

Encontramos x_4 con los valores obtenidos anteriormente.

$x_4 = \frac{-2-5(2)+3(-3)-9(0)}{21}$

x_4 = -1

En forma matricial lo representamos así:

$$\begin{pmatrix} 6 & 0 & 0 & 0 \\ 3 & 6 & 0 & 0 \\ 4 & -2 & 7 & 0 \\ 5 & -3 & 9 & 21 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 12 \\ -12 \\ 14 \\ -2 \end{pmatrix}$$

Sustitución Progresiva:

In [2]:

```
import numpy as np
A = np.array([[6,0,0,0],[3,6,0,0],[4,-2,7,0],[5,-3,9,21]])
b = np.array([12,-12,14,-2])
n = np.size(b)
x = np.zeros(n)

for i in range (n):
    sumj=0
    for j in range (i):
        sumj+=A[i,j]*x[j]
    x[i]=(b[i]-sumj)*1/A[i,i]
print(x)
```

```
[ 2. -3.  0. -1.]
```

Podemos concluir:

Descomposición LU:

L : Matriz Triangular Inferior

U : Matriz Triangular Superior

A = L.U

A = L

$$\begin{pmatrix} 6 & 0 & 0 & 0 \\ 3 & 6 & 0 & 0 \\ 4 & -2 & 7 & 0 \\ 5 & -3 & 9 & 21 \end{pmatrix} = L.U$$

A = LU

L = LU

L L^{-1} = U

I =

L = LU

$$\begin{pmatrix} 6 & 0 & 0 & 0 \\ 3 & 6 & 0 & 0 \\ 4 & -2 & 7 & 0 \\ 5 & -3 & 9 & 21 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

U = I

$$\mathbf{A} = L * I$$

2) Resuelva las siguientes matrices usando la descomposición de Cholesky

$$A = \begin{pmatrix} 4 & 6 & 10 \\ 6 & 25 & 19 \\ 10 & 19 & 62 \end{pmatrix}$$

$$B = \begin{pmatrix} 4 & 6 & 10 \\ 6 & 3 & 19 \\ 10 & 19 & 62 \end{pmatrix}$$

Analice el caso y escriba sus conclusiones

Debemos determinar si nuestra matriz cumple con las condiciones para aplicar cholesky. Esta deber ser:

- Simetrica
- Definida positiva

Simetrica: A = A^T

$$\begin{pmatrix} 4 & 6 & 10 \\ 6 & 25 & 19 \\ 10 & 19 & 62 \end{pmatrix} = \begin{pmatrix} 4 & 6 & 10 \\ 6 & 25 & 19 \\ 10 & 19 & 62 \end{pmatrix} EsSimetrica$$

Matriz definida Positiva:

$$\begin{pmatrix} 4 & 6 & 10 \\ 6 & 25 & 19 \\ 10 & 19 & 62 \end{pmatrix}$$

Se debe calcular determinantes de las submatrices principales de A.

Encontrar menores principales:

Menor principal 1: A1 = |4| = 4 > 0

Menor principal 2: A2 =

$$\begin{vmatrix} 4 & 6 \\ 6 & 25 \end{vmatrix} = 100 - 36 = 64 > 0$$

Menor principal 3: A3 = \begin{equation*}\begin{vmatrix}

- & - & + \ 4 & 6 & 10\ 6 & 25 & 19 \ 10 & 19 & 62 \end{vmatrix} \end{equation*}

$$A_3 = 4[(25)(62) - 19^2] - 6[(6)(62) - 190] + 10[(6)(19) - 250]$$
$$A_3 = 4756 - 1092 - 1360$$
$$A_3 = 2304 > 0$$

A es simetrica y definida positiva. Ya se puede realizar metodo de Cholesky

Entonces U = L^T

A = LU

A = L L^T

$$\begin{pmatrix} 4 & 6 & 10 \\ 6 & 25 & 19 \\ 10 & 19 & 62 \end{pmatrix} = \begin{pmatrix} L_{11} & 0 & 0 \\ L_{21} & L_{22} & 0 \\ L_{31} & L_{32} & L_{33} \end{pmatrix} \begin{pmatrix} L_{11} & L_{21} & L_{31} \\ 0 & L_{22} & L_{32} \\ 0 & 0 & L_{33} \end{pmatrix}$$

Hacemos el producto:

$$\begin{pmatrix} 4 & 6 & 10 \\ 6 & 25 & 19 \\ 10 & 19 & 62 \end{pmatrix} = \begin{pmatrix} L_{11}^2 & L_{11} * L_{21} & L_{11} * L_{31} \\ L_{11} * L_{21} & L_{21}^2 + L_{22}^2 & L_{21} * L_{31} + L_{22} * L_{32} \\ L_{31} * L_{11} & L_{31} * L_{21} + L_{32} * L_{22} & L_{31}^2 + L_{32}^2 + L_{33}^2 \end{pmatrix}$$

Igualar cada elemento:

Fila 1 :

L_{11}^2 = 4

L_{11} = 2

$L_{11}L_{21}$ = 6

L_{21} = 3

$L_{11}L_{31}$ = 10

L_{31} = 5

Fila 2 :

$L_{11}L_{21}$ = 6

6 = 6

$L_{21}^2 + L_{22}^2$ = 25

L_{22}^2 = 25 - 9

L_{22} = 4

$L_{21}L_{31} + L_{22} * L_{32}$ = 19

(3)(5) + (4) L_{32} = 19

L_{32} = 1

Fila 3:

$L_{31}L_{11}$ = 10

10 = 10

$L_{31}L_{21} + L_{32}L_{22}$ = 19

(5)(3) + (1)(2) = 19

19 = 19

$L_{31}^2 + L_{32}^2 + L_{33}^2$ = 62

25 + 1 + L_{33}^2 = 62

L_{33}^2 = 36

L_{33} = 6

$$L = \begin{pmatrix} 2 & 0 & 0 \\ 3 & 4 & 0 \\ 5 & 1 & 6 \end{pmatrix}$$

$$\text{A} = LL^T$$

$$\begin{pmatrix} 4 & 6 & 10 \\ 6 & 25 & 19 \\ 10 & 19 & 62 \end{pmatrix} = \begin{pmatrix} 2 & 0 & 0 \\ 3 & 4 & 0 \\ 5 & 1 & 6 \end{pmatrix} \begin{pmatrix} 2 & 3 & 5 \\ 0 & 4 & 1 \\ 0 & 0 & 6 \end{pmatrix}$$

Esta lista la descomposición de Cholesky

Matriz B debe ser:

- Simetrica
- Definida positiva

Simetrica: B = B^T

$$\begin{pmatrix} 4 & 6 & 10 \\ 6 & 3 & 19 \\ 10 & 19 & 62 \end{pmatrix} = \begin{pmatrix} 4 & 6 & 10 \\ 6 & 3 & 19 \\ 10 & 19 & 62 \end{pmatrix} EsSimetrica$$

Matriz definida Positiva:

$$\begin{pmatrix} 4 & 6 & 10 \\ 6 & 6 & 19 \\ 10 & 19 & 62 \end{pmatrix}$$

Se debe calcular determinantes de las submatrices principales de B.

Encontrar menores principales:

Menor principal 1: B_1 = |4| = 4 > 0

Menor principal 2: B_2 =

$$\begin{vmatrix} 4 & 6 \\ 6 & 3 \end{vmatrix} = 12 - 36 = -24 < 0$$

Conclusiones:

La matriz A si pudimos realizar la descomposición de Cholesky, debido a que si era simetrica y además era definida positiva. Pero al analizar la matriz B pudimos comprobar que esta no era

posible descomponer por el metodo de cholesky, esta si es simetrica pero al no ser definida positiva ya no se pudo continuar con la descomposición.

3) Utilice el método de Jacobi, Gauss-Seidel y SOR (ω = 1.1) para resolver el siguiente sistema lineal con una precisión de cuatro cifras decimales.

$$\begin{pmatrix} 7 & 1 & -1 & 2 \\ 1 & 8 & 0 & -2 \\ -1 & 0 & 4 & -1 \\ 2 & -2 & -1 & 6 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 3 \\ -5 \\ 4 \\ -3 \end{pmatrix}$$

Compare el número de iteraciones necesario en cada algoritmo. Analice el error cometido si la solución exacta es: $x = (1, -1, 1, -1)^T$

Metodo de Jacobi:

Donde: Ax = b

A = D - L - U

Formula:

$$x^{(k)} = D^{-1}(L+U)x^{k-1} + D^{-1}(b)$$

$$D = \begin{pmatrix} 7 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 6 \end{pmatrix} L = \begin{pmatrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ -2 & 2 & 1 & 0 \end{pmatrix} U = \begin{pmatrix} 0 & -1 & 1 & -2 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

Valor de inicio:

$$x^0 = \begin{pmatrix} 0 & 0 & 0 & 0 \end{pmatrix}$$

D = Matriz Diagonal

L = Matriz Triangular Inferior

U = Matriz Triangular Superior

$$D^{-1} = \begin{pmatrix} \frac{1}{7} & 0 & 0 & 0 \\ 0 & \frac{1}{8} & 0 & 0 \\ 0 & 0 & \frac{1}{4} & 0 \\ 0 & 0 & 0 & \frac{1}{6} \end{pmatrix}$$

Algoritmo para: $L + U$

In [7]:

```
import numpy as np

A = np.array([[7,1,-1,2],[1,8,0,-2],[-1,0,4,-1],[2,-2,-1,6]])
b = np.array([3,-5,4,-3])

n = A.shape[1]
D = np.eye(n)
D[np.arange(n),np.arange(n)] = A[np.arange(n),np.arange(n)]
```

```
LU = D-A
L = np.tril(LU)
U = np.triu(LU)
L_U = L+U

print(L_U)
```

```
[[ 0. -1.  1. -2.]
 [-1.  0.  0.  2.]
 [ 1.  0.  0.  1.]
 [-2.  2.  1.  0.]]
```

$$(L+U) = \begin{pmatrix} 0 & 1 & 1 & -1 \\ -1 & 0 & 0 & 2 \\ 1 & 0 & 0 & 1 \\ -2 & 2 & 1 & 0 \end{pmatrix}$$

Calcular: $D^{-1}(L+U)$

In [16]:

```
import numpy as np

A = np.array([[7,1,-1,2],[1,8,0,-2],[-1,0,4,-1],[2,-2,-1,6]])
b = np.array([3,-5,4,-3])

n = A.shape[1]
D = np.eye(n)
D[np.arange(n),np.arange(n)] = A[np.arange(n),np.arange(n)]
LU = D-A
L = np.tril(LU)
U = np.triu(LU)
L_U = L+U
D_inv = np.linalg.inv(D)
Resultado = np.dot(D_inv,L_U)

print(Resultado.round(decimals=4))
```

```
[[ 0.     -0.1429  0.1429 -0.2857]
 [-0.125   0.      0.      0.25  ]
 [ 0.25    0.      0.      0.25  ]
 [-0.3333  0.3333  0.1667  0.    ]]
```

$$D^{-1}(L+U) = \begin{pmatrix} 0 & -0.1429 & 0.1429 & -0.2857 \\ -0.125 & 0 & 0 & 0.25 \\ 0.25 & 0 & 0 & 0.25 \\ -0.3333 & 0.3333 & 0.1667 & 0 \end{pmatrix}$$

Calcular: D^{-1}(b)

In [15]:

```
import numpy as np

A = np.array([[7,1,-1,2],[1,8,0,-2],[-1,0,4,-1],[2,-2,-1,6]])
b = np.array([3,-5,4,-3])

n = A.shape[1]
D = np.eye(n)
D[np.arange(n),np.arange(n)] = A[np.arange(n),np.arange(n)]
LU = D-A
L = np.tril(LU)
U = np.triu(LU)
L_U = L+U
```

```
D_inv = np.linalg.inv(D)
Resultado = np.dot(D_inv,L_U)
Total = np.dot(D_inv,b)

print(Total.round(decimals=4))
```

```
[ 0.4286 -0.625   1.     -0.5   ]
```

$$\begin{pmatrix} 0.4286 \\ -0.625 \\ 1 \\ -0.5 \end{pmatrix}$$

$D^{-1}(L+U)X$

$$D^{-1}(L+U)X = \begin{pmatrix} 0 & -0.1429 & 0.1429 & -0.2857 \\ -0.125 & 0 & 0 & 0.25 \\ 0.25 & 0 & 0 & 0.25 \\ -0.3333 & 0.3333 & 0.1667 & 0 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

$$D^{-1}(L+U)X = \begin{pmatrix} -0.1425x_2+ & 0.1429x_3 & -0.2857x_4 \\ -0.125x_1+ & 0.25x_4 & \\ 0.25x_1+ & 0.25x_4 & \\ -0.3333x_1+ & 0.3333x_2+ & 0.1667x_3 \end{pmatrix}$$

$D^{-1}(L+U)X + D^{-1}b$

$$\begin{pmatrix} -0.1425x_2+ & 0.1429x_3 & -0.2857x_4 \\ -0.125x_1+ & 0.25x_4 & \\ 0.25x_1+ & 0.25x_4 & \\ -0.3333x_1+ & 0.3333x_2+ & 0.1667x_3 \end{pmatrix} + \begin{pmatrix} 0.4286 \\ -0.625 \\ 1 \\ -0.5 \end{pmatrix}$$

$K_1^k = -0.1425x_2 + 0.1429x_3 - 0.2857x_4 + 0.4286$

$K_2^k = -0.125x_1 + 0.25x_4 - 0.625$

$K_3^k = 0.25x_1 + 0.25x_4 + 1$

$K_4^k = -0.3333x_1 + 0.3333x_2 + 0.1667x_3 - 0.5$

$$x^0 = [0, 0, 0, 0]$$

Primera iteración:

x_1^1 = 0.4286

x_2^1 = -0.625

x_3^1 = 1

x_4^1 = -0.5

Segunda iteración:

x^1 = [0.4286, -0.625, 1, -0.5]

x_1^2 = -(0.1425)(0.4286) + (0.1429)(1)

Para evitar realizar todo el proceso manual se utilizara un algoritmo.

In [20]:

```
import numpy as np

A = np.array([[7,1,-1,2],[1,8,0,-2],[-1,0,4,-1],[2,-2,-1,6]])
b = np.array([3,-5,4,-3])
k = 15
n = A.shape[1]
D = np.eye(n)
D[np.arange(n),np.arange(n)] = A[np.arange(n),np.arange(n)]
LU = D-A
L = np.tril(LU)
U = np.triu(LU)
L_U = L+U
x = np.zeros(n)
print(" ",0,"Vector inicial: ",x)
for i in range(k):
    D_inv=np.linalg.inv(D)
    x = np.dot(np.dot(D_inv,L+U),x)+np.dot(D_inv,b)
    print(" ",i+1," La respuesta de la iteracion es: ", x.round(decimals=4))
```

```
 0 Vector inicial:  [0. 0. 0. 0.]
 1  La respuesta de la iteracion es:  [ 0.4286 -0.625   1.     -0.5   ]
 2  La respuesta de la iteracion es:  [ 0.8036 -0.8036  0.9821 -0.6845]
 3  La respuesta de la iteracion es:  [ 0.8793 -0.8966  1.0298 -0.872 ]
 4  La respuesta de la iteracion es:  [ 0.9529 -0.9529  1.0018 -0.9203]
 5  La respuesta de la iteracion es:  [ 0.9708 -0.9742  1.0081 -0.9683]
 6  La respuesta de la iteracion es:  [ 0.9884 -0.9884  1.0006 -0.9803]
 7  La respuesta de la iteracion es:  [ 0.9928 -0.9936  1.002  -0.9922]
 8  La respuesta de la iteracion es:  [ 0.9971 -0.9971  1.0002 -0.9951]
 9  La respuesta de la iteracion es:  [ 0.9982 -0.9984  1.0005 -0.9981]
 10  La respuesta de la iteracion es:  [ 0.9993 -0.9993  1.     -0.9988]
 11  La respuesta de la iteracion es:  [ 0.9996 -0.9996  1.0001 -0.9995]
 12  La respuesta de la iteracion es:  [ 0.9998 -0.9998  1.     -0.9997]
 13  La respuesta de la iteracion es:  [ 0.9999 -0.9999  1.     -0.9999]
 14  La respuesta de la iteracion es:  [ 1.     -1.      1.     -0.9999]
 15  La respuesta de la iteracion es:  [ 1. -1.  1. -1.]
```

Metodo de Gauss Seidel:

Ax = b

Formula:

x^k= $(D-L)^{-1}(U)\ X^{k-1}$ + $(D-L)^{-1}b$

$$D = \begin{pmatrix} 7 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 6 \end{pmatrix} L = \begin{pmatrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ -2 & 2 & 1 & 0 \end{pmatrix} U = \begin{pmatrix} 0 & -1 & 1 & -2 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$D-L$

$(D-L)^{-1}$

In [23]:

```
import numpy as np

A = np.array([[7,1,-1,2],[1,8,0,-2],[-1,0,4,-1],[2,-2,-1,6]])
b = np.array([3,-5,4,-3])
```

```
k = 15
n = A.shape[1]
D = np.eye(n)
D[np.arange(n),np.arange(n)] = A[np.arange(n),np.arange(n)]
LU = D-A
L = np.tril(LU)
U = np.triu(LU)
Resultado = D-L
r_inv = np.linalg.inv(Resultado)
print(Resultado)
print ("")
print(r_inv.round(decimals=4))
```

```
[[ 7.  0.  0.  0.]
 [ 1.  8.  0.  0.]
 [-1.  0.  4.  0.]
 [ 2. -2. -1.  6.]]

[[ 0.1429  0.      0.      0.    ]
 [-0.0179  0.125   0.      0.    ]
 [ 0.0357  0.      0.25    0.    ]
 [-0.0476  0.0417  0.0417  0.1667]]
```

$$D - L = \begin{pmatrix} 7 & 0 & 0 & 0 \\ 1 & 8 & 0 & 0 \\ -1 & 0 & 4 & 0 \\ 2 & -2 & -1 & 6 \end{pmatrix}$$

$$(D - L)^{-1} = \begin{pmatrix} 0.1429 & 0 & 0 & 0 \\ -0.0179 & 0.125 & 0 & 0 \\ 0.0357 & 0 & 0.25 & 0 \\ -0.0476 & 0.0417 & 0.0417 & 0.1667 \end{pmatrix}$$

$(D - L)^{-1}U$

In [25]:

```
import numpy as np

A = np.array([[7,1,-1,2],[1,8,0,-2],[-1,0,4,-1],[2,-2,-1,6]])
b = np.array([3,-5,4,-3])
k = 15
n = A.shape[1]
D = np.eye(n)
D[np.arange(n),np.arange(n)] = A[np.arange(n),np.arange(n)]
LU = D-A
L = np.tril(LU)
U = np.triu(LU)
Resultado = D-L
r_inv = np.linalg.inv(Resultado)
T = np.dot(r_inv,U)
print(T.round(decimals=4))
```

```
[[ 0.     -0.1429  0.1429 -0.2857]
 [ 0.      0.0179 -0.0179  0.2857]
 [ 0.     -0.0357  0.0357  0.1786]
 [ 0.      0.0476 -0.0476  0.2202]]
```

$$(D - L)^{-1}U = \begin{pmatrix} 0 & -0.1429 & 0.1429 & -0.2857 \\ 0 & 0.0179 & -0.0179 & 0.2857 \\ 0 & -0.0357 & 0.0357 & 0.1786 \\ 0 & 0.0476 & -0.0476 & 0.2202 \end{pmatrix}$$

$(D-L)^{-1}Ux^{k-1}$

$$\begin{pmatrix} 0 & -0.1429 & 0.1429 & -0.2857 \\ 0 & 0.0179 & -0.0179 & 0.2857 \\ 0 & -0.0357 & 0.0357 & 0.1786 \\ 0 & 0.0476 & -0.0476 & 0.2202 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

$$-0.1425x_2 + 0.1429x_3 - 0.2857x_4$$

$$0.0179x_2 - 0.0279x_2 + 0.2857x_4$$

$$0.0357x_2 + 0.0357x_3 + 0.1786x_4$$

$$0.0476x_2 - 0.0476x_3 + 0.2202x_4$$

$(D-L)^{-1}b$

In [26]:

```
import numpy as np

A = np.array([[7,1,-1,2],[1,8,0,-2],[-1,0,4,-1],[2,-2,-1,6]])
b = np.array([3,-5,4,-3])
k = 15
n = A.shape[1]
D = np.eye(n)
D[np.arange(n),np.arange(n)] = A[np.arange(n),np.arange(n)]
LU = D-A
L = np.tril(LU)
U = np.triu(LU)
Resultado = D-L
r_inv = np.linalg.inv(Resultado)
T = np.dot(r_inv,U)
RT = np.dot(r_inv,b)
print(RT.round(decimals=4))
```

```
[ 0.4286 -0.6786  1.1071 -0.6845]
```

$$\begin{pmatrix} 0.4286 \\ -0.6786 \\ 1.1071 \\ -0.6845 \end{pmatrix}$$

$x^k = (D-L)^{-1}Ux^{k-1} + (D-L)^{-1}b$

$x_1^1 = -0,1425x_2 + 0,1429x_3 - 0,2857x_4 + 0,4286$

$x_2^1 = 0,0179x_2 - 0,0179x_3 + 0,2857x_4 - 0,6786$

$x_3^1 = 0,0357x_2 + 0,0357x_3 + 0,1786x_4 + 1,1071$

$x_4^1 = 0,0476x_2 - 0,0476x_3 + 0,2202x_4 - 0,6845$

$$x^0 = [0,0,0,0]$$

K = 1

x_1^1 = 0.4286

x_2^1 = -0.6786

x_3^1 = 1.1071

x_4^1 = -0.6845

K = 2

x^1 = [0.4286, -0.6786, 1.1071, -0.6845]

x_2^2 = -0.1425(-0.6786)+0.1429(1.107)-0.2857(-0.6845)+0.4286

Para evitar realizar todo el proceso manual se utilizara un algoritmo.

In [30]:

```
import numpy as np

A = np.array([[7,1,-1,2],[1,8,0,-2],[-1,0,4,-1],[2,-2,-1,6]])
b = np.array([3,-5,4,-3])
k = 8
n = A.shape[1]
D = np.eye(n)
D[np.arange(n),np.arange(n)] = A[np.arange(n),np.arange(n)]
LU = D-A
L = np.tril(LU)
U = np.triu(LU)
Resultado = D-L
x = np.zeros(n)
print(" ",0, "Vector inicial: ",x)
for i in range (k):
    Resultado_inv = np.linalg.inv(Resultado)
    x = np.dot(np.dot(Resultado_inv,U),x)+np.dot(Resultado_inv,b)
    print(" ",i+1," Las iteraciones son: ",x.round(decimals=4))
```

```
 0 Vector inicial:  [0. 0. 0. 0.]
 1  Las iteraciones son:  [ 0.4286 -0.6786  1.1071 -0.6845]
 2  Las iteraciones son:  [ 0.8793 -0.906   1.0487 -0.9203]
 3  Las iteraciones son:  [ 0.9708 -0.9764  1.0126 -0.9803]
 4  Las iteraciones son:  [ 0.9928 -0.9942  1.0031 -0.9951]
 5  Las iteraciones son:  [ 0.9982 -0.9986  1.0008 -0.9988]
 6  Las iteraciones son:  [ 0.9996 -0.9996  1.0002 -0.9997]
 7  Las iteraciones son:  [ 0.9999 -0.9999  1.     -0.9999]
 8  Las iteraciones son:  [ 1. -1.  1. -1.]
```

Metodo SOR:

Formula:

$x^k = (D - wL)^{-1}[(1 - w)D + wU]x^{k-1} + w(D - wL)^{-1}b$

$$D = \begin{pmatrix} 7 & 0 & 0 & 0 \\ 0 & 8 & 0 & 0 \\ 0 & 0 & 4 & 0 \\ 0 & 0 & 0 & 6 \end{pmatrix} L = \begin{pmatrix} 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ -2 & 2 & 1 & 0 \end{pmatrix} U = \begin{pmatrix} 0 & -1 & 1 & -2 \\ 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$D - wL$

$(D - wL)^{-1}$

In [32]:

```
import numpy as np

A = np.array([[7,1,-1,2],[1,8,0,-2],[-1,0,4,-1],[2,-2,-1,6]])
b = np.array([3,-5,4,-3])
```

```
k = 8
w = 1.1
n = A.shape[1]
D = np.eye(n)
D[np.arange(n),np.arange(n)] = A[np.arange(n),np.arange(n)]
LU = D-A
L = np.tril(LU)
U = np.triu(LU)
Resultado = D-w*L
r_inv = np.linalg.inv(Resultado)
print(Resultado)
print(" ")
print(r_inv.round(decimals=4))
```

```
[[ 7.   0.   0.   0. ]
 [ 1.1  8.   0.   0. ]
 [-1.1  0.   4.   0. ]
 [ 2.2 -2.2 -1.1  6. ]]

[[ 0.1429  0.      0.      0.    ]
 [-0.0196  0.125   0.      0.    ]
 [ 0.0393  0.      0.25    0.    ]
 [-0.0524  0.0458  0.0458  0.1667]]
```

$$D - wL = \begin{pmatrix} 7 & 0 & 0 & 0 \\ 1.1 & 8 & 0 & 0 \\ -1.1 & 0 & 4 & 0 \\ 2.2 & -2.2 & -1.1 & 6 \end{pmatrix}$$

$$(D - wL)^{-1} = \begin{pmatrix} 0.1429 & 0 & 0 & 0 \\ -0.0196 & 0.125 & 0 & 0 \\ 0.0393 & 0 & 0.25 & 0 \\ -0.0524 & 0.0458 & 0.0458 & 0.1667 \end{pmatrix}$$

$[(1 - w)D + wU]$

In [33]:

```
import numpy as np

A = np.array([[7,1,-1,2],[1,8,0,-2],[-1,0,4,-1],[2,-2,-1,6]])
b = np.array([3,-5,4,-3])
k = 8
w = 1.1
n = A.shape[1]
D = np.eye(n)
D[np.arange(n),np.arange(n)] = A[np.arange(n),np.arange(n)]
LU = D-A
L = np.tril(LU)
U = np.triu(LU)
Resultado = D-w*L
r_inv = np.linalg.inv(Resultado)
F = (1-w)*D+w*U
print(F.round(decimals=4))
```

```
[[-0.7 -1.1  1.1 -2.2]
 [ 0.  -0.8  0.   2.2]
 [ 0.   0.  -0.4  1.1]
 [ 0.   0.   0.  -0.6]]
```

$$[(1-w)D+wU] = \begin{pmatrix} -0.7 & -1.1 & 1.1 & -2.2 \\ 0 & -0.8 & 0 & 2.2 \\ 0 & 0 & -0.4 & 1.1 \\ 0 & 0 & 0 & -0.6 \end{pmatrix}$$

$(D-wL)^{-1}\,[(1-w)D+wU]$

In [34]:

```
import numpy as np

A = np.array([[7,1,-1,2],[1,8,0,-2],[-1,0,4,-1],[2,-2,-1,6]])
b = np.array([3,-5,4,-3])
k = 8
w = 1.1
n = A.shape[1]
D = np.eye(n)
D[np.arange(n),np.arange(n)] = A[np.arange(n),np.arange(n)]
LU = D-A
L = np.tril(LU)
U = np.triu(LU)
Resultado = D-w*L
r_inv = np.linalg.inv(Resultado)
F = (1-w)*D+w*U
P = np.dot(r_inv,F)
print(P.round(decimals=4))
```

```
[[-0.1    -0.1571  0.1571 -0.3143]
 [ 0.0138 -0.0784 -0.0216  0.3182]
 [-0.0275 -0.0432 -0.0568  0.1886]
 [ 0.0367  0.021  -0.076   0.1665]]
```

$$\begin{pmatrix} -0.1 & -0.1571 & 0.1571 & -0.3143 \\ 0.0138 & -0.0784 & -0.0216 & 0.3182 \\ -0.0275 & -0.0432 & -0.0568 & 0.1886 \\ 0.0367 & 0.021 & -0.076 & 0.1665 \end{pmatrix}$$

$(D-wL)^{-1}\,[(1-w)D+wU]x^{k-1}$

$$\begin{pmatrix} -0.1 & -0.1571 & 0.1571 & -0.3143 \\ 0.0138 & -0.0784 & -0.0216 & 0.3182 \\ -0.0275 & -0.0432 & -0.0568 & 0.1886 \\ 0.0367 & 0.021 & -0.076 & 0.1665 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix}$$

$$-0.1x_1 - 0.1571x_2 + 0.1571x_3 - 0.3143x_4$$

$$0.0138x_1 - 0.0784x_2 - 0.0216x_3 + 0.3182x_4$$

$$-0.0275x_1 - 0.0432x_2 - 0.0568x_3 + 0.1886x_4$$

$$0.0367x_1 + 0.021x_2 - 0.076x_3 + 0.1665x_4$$

$w(D-wL)^{-1}b$

In [35]:

```
import numpy as np

A = np.array([[7,1,-1,2],[1,8,0,-2],[-1,0,4,-1],[2,-2,-1,6]])
b = np.array([3,-5,4,-3])
k = 8
w = 1.1
```

```
n = A.shape[1]
D = np.eye(n)
D[np.arange(n),np.arange(n)] = A[np.arange(n),np.arange(n)]
LU = D-A
L = np.tril(LU)
U = np.triu(LU)
Resultado = D-w*L
r_inv = np.linalg.inv(Resultado)
F = (1-w)*D+w*U
P = np.dot(r_inv,F)
I = w*np.dot(r_inv,b)
print(I.round(decimals=4))
```

```
[ 0.4714 -0.7523  1.2296 -0.7733]
```

$$\begin{pmatrix} 0.4714 \\ -0.7523 \\ 1.2296 \\ -0.7733 \end{pmatrix}$$

$$x_1^k = -0.1x_1 - 0.1571x_2 + 0.1571x_3 - 0.3143x_4 + 0.4714$$

$$x_2^k = 0.0138x_1 - 0.0784x_2 - 0.0216x_3 + 0.3182x_4 - 0.7523$$

$$x_3^k = -0.0275x_1 - 0.0432x_2 - 0.0568x_3 + 0.1886x_4 + 1.2296$$

$$x_4^k = 0.0367x_1 + 0.021x_2 - 0.076x_3 + 0.1665x_4 - 0.7733$$

$$x^0 = [0, 0, 0, 0]$$

k = 1

x_1^1 = 0.4714

x_2^1 = -0.7523

x_3^1 = 1.2296

x_4^1 = -0.7733

Para evitar realizar todo el proceso manual se utilizara un algoritmo.

In [38]:

```
import numpy as np

A = np.array([[7,1,-1,2],[1,8,0,-2],[-1,0,4,-1],[2,-2,-1,6]])
b = np.array([3,-5,4,-3])
k = 5
w = 1.1
n = A.shape[1]
D = np.eye(n)
D[np.arange(n),np.arange(n)] = A[np.arange(n),np.arange(n)]
LU = D-A
L = np.tril(LU)
U = np.triu(LU)
Resultado = D-w*L
x = np.zeros(n)
print(" ",0, "Vector inicial: ",x)
for i in range (k):
    Resultado_inv = np.linalg.inv(Resultado)
    x = np.dot(np.dot(Resultado_inv,(1-w)*D+w*U),x)+w*np.dot(Resultado_inv,b)
```

```
    print(" ",i+1," Las iteraciones son: ",x.round(decimals=4))
```

```
 0 Vector inicial:  [0. 0. 0. 0.]
 1  Las iteraciones son:  [ 0.4714 -0.7523  1.2296 -0.7733]
 2  Las iteraciones son:  [ 0.9788 -0.9595  1.0335 -0.9939]
 3  Las iteraciones son:  [ 0.9991 -1.0022  0.9981 -1.0015]
 4  Las iteraciones son:  [ 1.0006 -1.0003  1.     -1.0002]
 5  Las iteraciones son:  [ 1. -1.  1. -1.]
```

Analisis del error:

$|x_1^1 - x_1^2|$ = 0.4714 - 0.9788 = |0.5074|

$|x_2^1 - x_2^2|$ = -0.7523 + 0.9595 = |0.2072|

$|x_3^1 - x_3^2|$ = 1.2296 - 1.0335 = |0.1961|

$|x_4^1 - x_4^2|$ = -0.7733 + 0.9939 = |0.2206|

Conclusion:

Luego de realizar todos los metodos, se puede concluir que el mas eficiente de estos es: El metodo SOR.

- Numero de iteraciones del metodo Jacobi: 15 iteraciones
- Numero de iteraciones del metodo Gauss Seidel: 8 iteraciones
- Numero de iteraciones del metodo SOR: 5 iteraciones.

4) Programe un algoritmo para encontrar la norma de Frobenius para una matriz cuadrada de cualquier dimensión.

Matriz de Ejemplo:

$$A = \begin{pmatrix} 5 & 2 & 4 & 3 \\ 2 & 4 & 6 & 1 \\ -1 & 4 & 1 & 0 \\ 3 & -2 & 0 & 4 \end{pmatrix}$$

Resolución:

$||A||_F$ = $\sqrt{(5)^2 + (2)^2 + (4)^2 + (3)^2 + (2)^2 + (4)^2 + (6)^2 + (1)^2 + (-1)^2 + (4)^2 + (1)^2 + (0)^2 + (3)^2 +$

$||A||_F = \sqrt{158}$

$||A||_F$ = 12,5698

Algoritmo:

In [44]:

```
import numpy as np
from math import sqrt
A = np.array([[5,2,4,3],[2,4,6,1],[-1,4,1,0],[3,-2,0,4]])
m = A.shape[0]
n = A.shape[1]
suma_raiz = 0
for i in range (m):
    for j in range (n):
```

```
        suma_raiz += pow(A[i][j],2)
    Respuesta = sqrt(suma_raiz)
print(round(Respuesta,4))
```

```
12.5698
```