Universidad de las Fuerzas Armadas ESPE
Métodos numéricos NRC 3657

Lizeth Abigail Iza Moreno

23 de julio de 2021

Descripción de la actividad.

1. Para el sistema lineal

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

Encuentre los valores de x_1, x_2, x_3, x_4 utilizando sustitución progresiva.

Primero encontramos el valor de x_1 en la primera ecuación

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

Reemplazamos x_1 y encontramos el valor de x_2

$$6x_2 + 3x_1 = -12$$
$$6x_2 + 3(2) = -12$$
$$x_2 = \frac{-12 - 6}{6}$$
$$x_2 = \frac{-18}{6}$$
$$x_2 = -3$$

Reemplazamos x_1, x_2 y encontramos el valor de x_3

$$7x_3 - 2x_2 + 4x_1 = 14$$
$$7x_3 - 2(-3) + 4(2) = 14$$
$$x_3 = \frac{14 - 8 - 6}{7}$$
$$x_3 = \frac{14 - 14}{7}$$
$$x_3 = 0$$

Reemplazamos x_1, x_2, x_3 y encontramos el valor de x_4

$$21x_4 + 9x_3 - 3x_2 + 5x_1 = -2$$
$$21x_4 + 9(0) - 3(-3) + 5(2) = -2$$
$$x_4 = \frac{-2 - 10 - 9}{21}$$
$$x_4 = \frac{-21}{21}$$
$$x_4 = -1$$

Programa de Sustitución Progresiva en Python.

Código:

```python
import numpy as np #Importamos la libreria numpy
#Definimos la matriz del sistema de ecuaciones
A=np.array([[6,0,0,0],[3,6,0,0],[4,-2,7,0],[5,-3,9,21]])
#Escribimps los terminos indepnedientes
b=np.array([12,-12,14,-2])
#Numero de ecuaciones
n=np.size(b)
#Vector para almacenar la solucion
x=np.zeros(n)
#Metodo de la Sustitucion Progresiva
for i in range(n):
    suma=0
    for j in range(i):
        suma+=A[i,j]*x[j]
    x[i]=(b[i]-suma)*1/A[i,i]
print("******Programa de Sustitución Progresiva*****\n")
print("La matriz Ingresada es:")
print(A)
print("\n",b)
print("----------------------------------------------------------------------")
print("Las soluciones del sistema de ecuaciones (x1,x2,x3,x4) es:")
print(x)
```

Figura 1: Código del método de Sustitución Progresiva

Corrida de Escritorio:

```
******Programa de Sustitución Progresiva*****

La matriz Ingresada es:
[[ 6  0  0  0]
 [ 3  6  0  0]
 [ 4 -2  7  0]
 [ 5 -3  9 21]]

 [ 12 -12  14  -2]
----------------------------------------------------------------------
Las soluciones del sistema de ecuaciones (x1,x2,x3,x4) es:
[ 2. -3.  0. -1.]
```

Figura 2: Ejecución del método de Sustitución Progresiva

Encuentre la descomposición LU del sistema anterior.

Escribimos el Sistema de ecuaciones en forma de Matriz Ax=b

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

La descomposición LU

$$\begin{aligned} A &= LU \\ L &= LU \\ LL^{-1} &= U \\ I &= U \end{aligned}$$

Comprobamos A = L*I

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

¿Qué puede concluir?

Por lo tanto, se puede concluir que el método de sustitución progresiva nos a permitido despejar los valores de x1, x2 , x3 y x4 de la matriz dada y la descomposición LU esta compuesta de A=LU en donde L(Matriz triangular inferior) U (Matriz Triangular superior) se concluye que la matriz dada A es equivalente a A=L y U=I en donde I es una matriz identidad multiplicando A=L*I nos da como resultado la misma matriz A.

2. Resuelva las siguientes matrices usando la descomposición de Cholesky

Matriz A:

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

Para resolver por el método de cholesky se debe cumplir las condiciones que A sea una matriz simétrica y definida positiva.

Primero verificamos si la matriz es simétrica $A = A^T$:

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

Por lo tanto la matriz si es simétrica

Ahora verificamos si la matriz es definida positiva:
Calculamos las determinantes de cada una de las subdeterminantes:

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

Calculamos:

$A_1 = |4| = 4 > 0$

$A_2 = \begin{vmatrix} 4 & 6 \\ 6 & 25 \end{vmatrix} = 100 - 36 = 64 > 0$

$A_3 = \begin{vmatrix} 4 & 6 & 10 \\ 6 & 25 & 19 \\ 10 & 19 & 62 \end{vmatrix}$

$A_3 = 4(25)(62) + (6)(19)(10) + (10)(6)(19) - (10)(25)(10) - (19)(19)(4) - 62(6)(6)$

$A_3 = 6200 + 1140 + 1140 - 2500 - 1444 - 2232$

$A_3 = 8480 - 6176$

$A_3 = 2304 > 0$

Por lo tanto la matriz si es definida positiva

Resolvemos por el método de Cholesky:
Consideramos que: $U = L^T$

$$A = LU$$
$$A = LL^T$$

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

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

Igualamos:

$$
\begin{aligned}
{L_{11}}^2 &= 4 \\
L_{11} &= 2
\end{aligned}
$$

$$
\begin{aligned}
L_{11} * L_{21} &= 6 \\
2 * L_{21} &= 6 \\
L_{21} &= \frac{6}{2} \\
L_{21} &= 3
\end{aligned}
$$

$$
\begin{aligned}
L_{11} * L_{31} &= 10 \\
2 * L_{31} &= 10 \\
L_{31} &= \frac{10}{2} \\
L_{31} &= 5
\end{aligned}
$$

$$
\begin{aligned}
{L_{21}}^2 + {L_{22}}^2 &= 25 \\
9 + {L_{22}}^2 &= 25 \\
{L_{22}}^2 &= 25 - 9 \\
L_{22} &= 4
\end{aligned}
$$

$$
\begin{aligned}
L_{21} * L_{31} + L_{22} * L_{32} &= 19 \\
3 * 5 + 4 * L_{32} &= 19 \\
15 + 4 * L_{32} &= 19 \\
L_{32} &= \frac{19 - 15}{4} \\
L_{32} &= \frac{4}{4} \\
L_{32} &= 1
\end{aligned}
$$

$$
\begin{aligned}
{L_{31}}^2 + {L_{32}}^2 + {L_{33}}^2 &= 62 \\
25 + 1 + {L_{33}}^2 &= 62 \\
{L_{33}}^2 &= 62 - 26 \\
{L_{33}}^2 &= 36 \\
L_{33} &= 6
\end{aligned}
$$

La solución de L es:

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

Por lo tanto la descomposición de Cholesky de la matriz A nos quedaría: $A = LL^T$

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

Programa del Método de Cholesky en Python para la Matriz A:

Código:

```python
import numpy as np #Importamos la matriz numpy
MAT_A=np.array([[4, 6, 10],  #Definimos la matriz A
                [6, 25, 19],
                [10, 19, 62]])
def cholesky(MAT_A): #Metodo de descomposición de  cholesky
    MAT_A = np.array(MAT_A,float)
    L = np.zeros_like(MAT_A)
    n,_= np.shape(MAT_A)
    for j in range(n):
        for i in range(j, n):
            if i == j:
                L[i, j] = np.sqrt(MAT_A[i, j] - np.sum(L[i, :j] ** 2))
            else:
                L[i, j] = (MAT_A[i, j] - np.sum(L[i, :j] * L[j, :j])) / L[j, j]
    return L
l=cholesky(MAT_A)
Tl=np.transpose(l) #Traspuesta
print("\t******Programa Descomposición de Cholesky******")
print("-----------------------------------------------------------------")
print("La Matriz A:")
print(MAT_A)
print("\nLa Matriz L es :")
print(l)
print("\nLa Matriz Transpuesta de L :")
print(Tl)
```

Figura 3: Código del Método de Cholesky

Corrida de Escritorio:

```
    ******Programa Descomposición de Cholesky******
-----------------------------------------------------------------
La Matriz A:
[[ 4  6 10]
 [ 6 25 19]
 [10 19 62]]

La Matriz L es :
[[2. 0. 0.]
 [3. 4. 0.]
 [5. 1. 6.]]

La Matriz Transpuesta de L :
[[2. 3. 5.]
 [0. 4. 1.]
 [0. 0. 6.]]
```

Figura 4: Ejecución del Método de Cholesky

Matriz B:

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

Primero verificamos si la matriz es simétrica $B = B^T$:

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

Por lo tanto la matriz B si es simétrica

Ahora verificamos si la matriz es definida positiva:
Calculamos las determinantes de cada una de las subdeterminantes B:

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

Calculamos:

$B_1 = |4| = 4 > 0$

$B_2 = \begin{vmatrix} 4 & 6 \\ 6 & 3 \end{vmatrix} = 12 - 36 = -24 < 0$

$B_3 = \begin{vmatrix} 4 & 6 & 10 \\ 6 & 3 & 19 \\ 10 & 19 & 62 \end{vmatrix}$

$B_3 = (4)(3)(62) + (6)(19)(10) + (10)(6)(19) - (10)(3)(10) - (19)(19)(4) - (62)(6)(6)$

$B_3 = 744 + 1140 + 1140 - 300 - 1444 - 2232$

$B_3 = 3024 - 3976$

$B_3 = -952 < 0$

Por lo tanto la matriz B No es definida positiva

Programa del Método de Cholesky para la Matriz B:

Código:

```python
import numpy as np #Importamos la matriz numpy
MAT_B=np.array([[4, 6, 10],  #Definimos la matriz B
                [6, 3, 19],
                [10, 19, 62]])
def cholesky(MAT_B): #Método de descomposición de  cholesky
    MAT_B = np.array(MAT_B,float)
    L = np.zeros_like(MAT_B)
    n,_= np.shape(MAT_B)
    for j in range(n):
        for i in range(j, n):
            if i == j:
                L[i, j] = np.sqrt(MAT_B[i, j] - np.sum(L[i, :j] ** 2))
            else:
                L[i, j] = (MAT_B[i, j] - np.sum(L[i, :j] * L[j, :j])) / L[j, j]
    return L
l=cholesky(MAT_B)
Tl=np.transpose(l) #Matriz Transpuesta
print("\t******Programa Descomposición de Cholesky******")
print("La Matriz B:")
print(MAT_B)
print("\nLa Matriz L es :")
print(l)
print("\nLa Matriz Transpuesta de L :")
print(Tl)
```

Figura 5: Código del Método de Cholesky

Corrida de Escritorio:

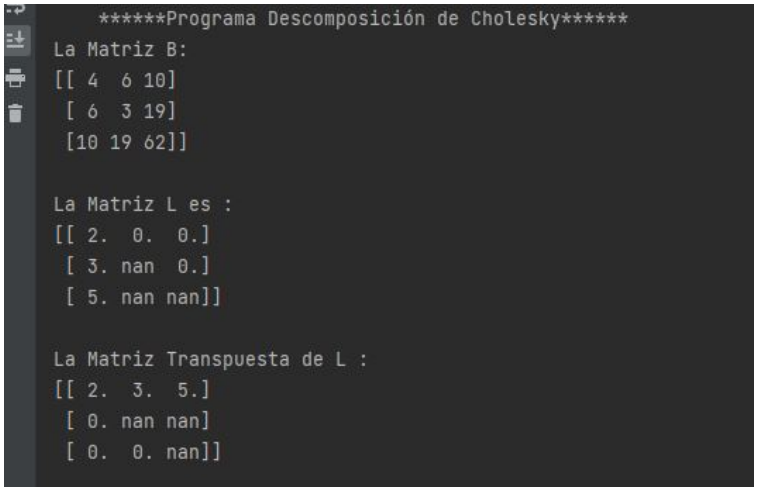

Figura 6: Ejecución del Método de Cholesky

Analice el caso y escriba sus conclusiones:

Podemos concluir que en el primero caso de la matriz A si se logro realizar el método de Cholesky porque cumplía con las condiciones de ser simétrica y matriz definida positiva,lo que no pasa con el caso de la Matriz B, a pesar de ser simétrica no cumple con la condición de ser definida positiva ya que en sus subdeterminantes dan valores < 0 y no se puede usar el método de Cholesky.

El método de descomposición de Cholesky tiene la ventaja de estabilidad numérica y la eficiencia frente a la descomposición LU.

3. Utilice el método de Jacobi, Gauss-Seidel y SOR ($\omega = 1{,}1$) 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 $x = (1, -1, 1, -1)^T$.

Método de Jacobi

Código:

```python
import numpy as np #Importamos la libreria numpy
A=np.array([[7,1,-1,2], #Definimos la  matriz
            [1,8,0,-2],
            [-1,0,4,-1],
            [2,-2,-1,6]])
b=np.array([3,-5,4,-3])
print("*****Programa Metodo Jacobi*****")
print("La matriz ingresada es:")
print(A)
print("\n",b)
k=15 #Numero de iteraciones
n=A.shape[1] #numero de columnas
D=np.eye(n) #Matriz identidad
D[np.arange(n),np.arange(n)]=A[np.arange(n),np.arange(n)]
LU=D-A
L=np.tril(LU) #Matriz tridiagonal inferior
U=np.triu(LU)#Matriz tridiagonal  superior
X=np.zeros(n) #Almacenar las respuestas
print("\t\t\tSoluciones del método jacobi")
print("Iteración",0,"es:",X)
#Metodo Jacobi
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("Iteración",i+1, "es:",X.round(decimals=4))
```

Figura 7: Código del Método de Jacobi

Corrida de Escritorio:

```
*****Programa Metodo Jacobi*****
La matriz ingresada es:
[[ 7  1 -1  2]
 [ 1  8  0 -2]
 [-1  0  4 -1]
 [ 2 -2 -1  6]]

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

Figura 8: Ejecución del Método de Jacobi

Método Gauss-Seidel

Código:

```python
import numpy as np #Importamos la libreria numpy
A=np.array([[7,1,-1,2], #Definimos la  matriz
            [1,8,0,-2],
            [-1,0,4,-1],
            [2,-2,-1,6]])
b=np.array([3,-5,4,-3])
print("*****Programa Método Gauss Seidel*****")
print("-----------------------------------------------")
print("La matriz ingresada es:")
print(A)
print("\n",b)
k=8 #Numero de iteraciones
n=A.shape[1] #numero de columnas
D=np.eye(n) #Matriz identidad
D[np.arange(n),np.arange(n)]=A[np.arange(n),np.arange(n)] #Matriz diagonal
LU=D-A
L=np.tril(LU)  #Matriz tridiagonal inferior
U=np.triu(LU) #Matriz tridiagonal  superior
DL=D-L
X=np.zeros(n) #almacenar las respuestas
print("\t\t\tSoluciones del Gauss Seidel")
print("-----------------------------------------------")
print("Iteración",0,"es:",X)
# Método Gauss Seidel
for i in range(k):
    DL_inv=np.linalg.inv(DL)
    X=np.dot(np.dot(DL_inv,U),X)+np.dot(DL_inv,b)
    print("Iteración",i+1, "es:",X.round(decimals=4))
```

Figura 9: Código del Método de Gauss-Seidel

Corrida de Escritorio:

```
*****Programa Método Gauss Seidel*****
-----------------------------------------------
La matriz ingresada es:
[[ 7  1 -1  2]
 [ 1  8  0 -2]
 [-1  0  4 -1]
 [ 2 -2 -1  6]]

 [ 3 -5  4 -3]
            Soluciones del Gauss Seidel
-----------------------------------------------
Iteración 0 es: [0. 0. 0. 0.]
Iteración 1 es: [ 0.4286 -0.6786  1.1071 -0.6845]
Iteración 2 es: [ 0.8793 -0.906   1.0487 -0.9203]
Iteración 3 es: [ 0.9708 -0.9764  1.0126 -0.9803]
Iteración 4 es: [ 0.9928 -0.9942  1.0031 -0.9951]
Iteración 5 es: [ 0.9982 -0.9986  1.0008 -0.9988]
Iteración 6 es: [ 0.9996 -0.9996  1.0002 -0.9997]
Iteración 7 es: [ 0.9999 -0.9999  1.     -0.9999]
Iteración 8 es: [ 1. -1.  1. -1.]
```

Figura 10: Ejecución del Método de Gauss-Seidel

Método SOR

Código:

```python
import numpy as np #Importamos la libreria numpy
A=np.array([[7,1,-1,2], #Definimos la  matriz
            [1,8,0,-2],
            [-1,0,4,-1],
            [2,-2,-1,6]])
b=np.array([3,-5,4,-3])
print("*****Programa Método SOR*****")
print("---------------------------------------------")
print("La matriz ingresada es:")
print(A)
print("\n",b)
k=5 #Numero de iteraciones
w=1.1
n=A.shape[1] #numero de columnas
D=np.eye(n) #Matriz identidad
D[np.arange(n),np.arange(n)]=A[np.arange(n),np.arange(n)] #Matriz diagonal
LU=D-A
L=np.tril(LU)  #Matriz tridiagonal inferior
U=np.triu(LU) #Matriz tridiagonal  superior
D_wL=D-w*L
X=np.zeros(n) #almacenar las respuestas
print("\t\t\tSoluciones del SOR")
print("---------------------------------------------")
print("Iteración",0,"es:",X)
# Método SOR
for i in range(k):
    D_wL_inv=np.linalg.inv(D_wL)
    X=np.dot(np.dot(D_wL_inv,(1-w)*D+w*U),X)+w*np.dot(D_wL_inv,b)
    print("Iteración", i + 1, "es:", X.round(decimals=4))
```

Figura 11: Código del Método de SOR

Corrida de Escritorio:

```
*****Programa Método SOR*****
---------------------------------------------
La matriz ingresada es:
[[ 7  1 -1  2]
 [ 1  8  0 -2]
 [-1  0  4 -1]
 [ 2 -2 -1  6]]

 [ 3 -5  4 -3]
           Soluciones del SOR
---------------------------------------------
Iteración 0 es: [0. 0. 0. 0.]
Iteración 1 es: [ 0.4714 -0.7523  1.2296 -0.7733]
Iteración 2 es: [ 0.9788 -0.9595  1.0335 -0.9939]
Iteración 3 es: [ 0.9991 -1.0022  0.9981 -1.0015]
Iteración 4 es: [ 1.0006 -1.0003  1.     -1.0002]
Iteración 5 es: [ 1. -1.  1. -1.]
```

Figura 12: Ejecución del Método de SOR

Comparar número de iteraciones:

Como se puede evidenciar en el algoritmo del método de Jacobi fue necesario 15 iteraciones, el algoritmo Gauss-Seidel necesita 8 iteraciones y para el algoritmo de SOR 5

iteraciones por lo tanto en este caso el método que converge mas rápido es el método de SOR para llegar a la solución donde $x = (1, -1, 1, -1)$.

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

   Para el siguiente algoritmo de la norma de Frobenius usare la matriz de 4x4 siguiente:

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

   Código:

Norma de Frobenius.py

```
import numpy as np #importamos la matriz numpy
from math import sqrt
A=np.array([[6,1,-1,5], # definimos la matriz
            [1,8,4,4],
            [8,-2,4,-1],
            [1,6,4,3]])
print("\n\t*****Programa de la Norma de Frobenius*****")
print("-----------------------------------------------------")
print("La matriz ingresada es:\n",A)
filas=A.shape[0] #Numero de filas
columnas=A.shape[1]#Numero de columnas
sumar=0
#Norma de Frobenius
for i in range(filas):
    for j in range(columnas):
        sumar+=pow(A[i][j],2)
    Res=sqrt(sumar) #Resultado
print("-----------------------------------------------------")
print("El resultado de la norma de Frobenius es:",Res)
```

Figura 13: Código de la norma de Frobenius

   Corrida de Escritorio:

```
    *****Programa de la Norma de Frobenius*****
-----------------------------------------------------
La matriz ingresada es:
 [[ 6  1 -1  5]
 [ 1  8  4  4]
 [ 8 -2  4 -1]
 [ 1  6  4  3]]
-----------------------------------------------------
El resultado de la norma de Frobenius es: 17.52141546793523

Process finished with exit code 0
```

Figura 14: Ejecución de la norma de Frobenius