Universidad de las Fuerzas Armadas

Métodos Numéricos

Actividad 3

Apellidos: Jaramillo Salgado
Nombres: Ricardo Alejandro
NRC: 3657

Fecha límite de entrega: 23/07/2021

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. Encuentre la descomposición LU del sistema anterior. ¿Qué puede concluir?

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

A = L*U

$$\begin{pmatrix} 6 & 0 & 0 & 0 \\ 3 & 6 & 0 & 0 \\ 4 & -2 & 7 & 0 \\ 5 & -3 & 9 & 21 \end{pmatrix} \cdot \left(-\frac{1}{2}\right) \quad F_2 \to F_2 - \left(\frac{1}{2}\right) \cdot F_1 \begin{pmatrix} 6 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 4 & -2 & 7 & 0 \\ 5 & -3 & 9 & 21 \end{pmatrix} \cdot \left(-\frac{2}{3}\right) \quad F_3 \to F_3 - \left(\frac{2}{3}\right) \cdot F_1 \begin{pmatrix} 6 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & -2 & 7 & 0 \\ 5 & -3 & 9 & 21 \end{pmatrix} \cdot \left(-\frac{5}{6}\right) \quad F_4 \to F_4 - \left(\frac{5}{6}\right) \cdot F_1 \begin{pmatrix} 6 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & -2 & 7 & 0 \\ 0 & -3 & 9 & 21 \end{pmatrix} \cdot \left(\frac{1}{7}\right) \quad F_3 \to F_3 - \left(-\frac{1}{3}\right) \cdot F_2$$

$$\begin{pmatrix} 6 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 7 & 0 \\ 0 & -3 & 9 & 21 \end{pmatrix} \cdot \left(\frac{3}{7}\right) \quad F_4 \to F_4 - \left(-\frac{1}{2}\right) \cdot F_2 \begin{pmatrix} 6 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 9 & 21 \end{pmatrix} \cdot \left(-\frac{9}{7}\right) \quad F_4 \to F_4 - \left(-\frac{9}{7}\right) \cdot F_3 \begin{pmatrix} 6 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 21 \end{pmatrix}$$

Por lo tanto:

$$L = \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 \\ \frac{2}{3} & -\frac{1}{3} & 1 & 0 \\ \frac{5}{6} & -\frac{1}{2} & \frac{9}{7} & 1 \end{pmatrix}; \; U = \begin{pmatrix} 6 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 21 \end{pmatrix}$$

Entonces:

$$L \cdot \left(U \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} \right) = \begin{pmatrix} 12 \\ -12 \\ 14 \\ -2 \end{pmatrix} \rightarrow L \begin{pmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{pmatrix} = \begin{pmatrix} 12 \\ -12 \\ 14 \\ -2 \end{pmatrix} \rightarrow \begin{pmatrix} 1 & 0 & 0 & 0 \\ \frac{1}{2} & 1 & 0 & 0 \\ \frac{2}{3} & -\frac{1}{3} & 1 & 0 \\ \frac{5}{6} & -\frac{1}{2} & \frac{9}{7} & 1 \end{pmatrix}$$

$$\begin{cases} z1 = 12 \\ \frac{1}{2}z1 + z2 = -12 \\ \frac{2}{3}z1 - \frac{1}{3}z2 + z3 = 14 \\ \frac{5}{6}z1 - \frac{1}{2}z2 + \frac{9}{7}z3 + z4 = -2 \end{cases}$$

Para $z_1 = 12 \rightarrow z_1 = 12$ *Para* $\frac{1}{2}z_1 + z_2 = -12 \rightarrow \frac{1}{2}(12) + z_2 = -12 \Rightarrow z_2 = -12 - 8 = -18$ *Para* $\frac{2}{3}z_1 - \frac{1}{3}z_2 + z_3 = 14 \rightarrow \frac{2}{3}(12) - \frac{1}{3}(-18) + z_3 = 14 \Rightarrow 2 \cdot 4 + 6 + z_3 = 14 \Rightarrow 8 + 6 + z_3 = 14 = 0$
Para $\frac{5}{6}z_1 - \frac{1}{2}z_2 + \frac{9}{7}z_3 + z_4 = -2 \rightarrow \frac{5}{6}(12) - \frac{1}{2}(-18) + \frac{9}{7}(0) + z_4 = -2 \Rightarrow 5 \cdot 2 + 9 + z_4 = -2 \Rightarrow 19 + z_4 = -2 \Rightarrow z_4 = -2 - 19 = -21$

Finalmente

$$U = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 12 \\ -18 \\ 0 \\ -21 \end{pmatrix} \rightarrow \begin{pmatrix} 6 & 0 & 0 & 0 \\ 0 & 6 & 0 & 0 \\ 0 & 0 & 7 & 0 \\ 0 & 0 & 0 & 21 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{pmatrix} = \begin{pmatrix} 12 \\ -18 \\ 0 \\ -21 \end{pmatrix}$$

$$\begin{pmatrix} = 6x_1 = 12 \rightarrow x_1 = 2 \\ = 6x_2 = -18 \rightarrow x_2 = -3 \\ = 7x_3 = 0 \rightarrow x_3 = 0 \\ = 21x_4 = -21 \rightarrow x_4 = -1 \end{pmatrix}$$

$R. / x_1 = 2\,,\ x_2 = -3\,,\ x_3 = 0\,,\ x_4 = -1$

Código en Python

In [ ]:

```
## DECLARAMOS VARIABLES Y LIBRERÍAS A UTILIZAR EN NUESTRO EJERCICIO
import numpy as np
## DEFINIMOS LAS DIMENSIONES DEL SISTEMA LINEAL
x = np.array([[6,0,0,0],[3,6,0,0],[4,-2,7,0],[5,-3,9,21]])
## DEFINIMOS LOS TERMINOS INDEPENDIENTES
y = np.array([12,-12,14,-2])
num_ec=np.size(y)
## LA SOLUCIÓN SE LEERA EN LA VARIABLE SOL
sol = np.zeros(num_ec)
## Aplicamos sustitución progresiva
for i in range(num_ec):
    sumj=0
    for j in range (i):
        sumj+=x[i,j]*sol[j]
    sol[i]=(y[i]-sumj)*1/x[i,i]
print("\x1b[1;34m"+"LAS SOLUCIONES SON: ")
print(sol)
```

Salida de Escritorio

```
LAS SOLUCIONES SON:
[ 2. -3.  0. -1.]
```

¿Que podemos concluir de este sistema?

En definitiva gracias a la sustitución progresiva podemos despejar las incógnitas de un sistema en donde AB recibe sus matrices aumentadas y "n" sus incógnitas en donde A debe ser triangular inferior, por otro lado gracias a la descomposición LU podemos separar la matriz A en dos matrices triangulares A = LU, en donde L es triangular inferior y U es triangular superior, de esta forma la resolución de estas dos subecuaciones es trivial.

2. Resuelva las siguientes matrices usando la descomposición de Cholesky

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

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

Análisis de A

Fórmulas

$$l_{ki} = \frac{a_{ki} - \sum_{j=1}^{i-1} l_{ij} \cdot l_{kj}}{l_{ii}}$$

$$l_{kk} = \sqrt{a_{kk} - \sum_{j=1}^{k-1} l_{kj}^2}$$

$$\begin{pmatrix} 4 & 6 & 10 \\ 6 & 25 & 19 \\ 10 & 19 & 62 \end{pmatrix} = \begin{pmatrix} I_{1,1} & 0 & 0 \\ I_{2,1} & I_{2,2} & 0 \\ I_{3,1} & I_{3,2} & I_{3,3} \end{pmatrix} \cdot \begin{pmatrix} I_{1,1} & I_{2,1} & I_{3,1} \\ 0 & I_{2,2} & I_{3,2} \\ 0 & 0 & I_{3,3} \end{pmatrix} = \begin{pmatrix} {l_{1,1}}^2 & l_{1,1}l_{2,1} & l_{1,1}l_{3,1} \\ l_{1,1}l_{2,1} & {l_{2,1}}^2 + {l_{2,2}}^2 & l_{2,1}l_{3,1} + l_{2,2}l_{3,2} \\ l_{1,1}l_{3,1} & l_{2,1}l_{3,1} + l_{2,2}l_{3,2} & {l_{3,1}}^2 + {l_{3,2}}^2 + {l_{3,3}}^2 \end{pmatrix}$$

Por lo tanto:

Si ${l_{1,1}}^2 = 4 \rightarrow \ l_{1,1} = \sqrt{4}$

Si $l_{1,1}l_{2,1} = 6 \rightarrow \ I_{2,1} = \frac{6}{I_{1,1}} = \frac{6}{2} = 3$

Si $\ l_{1,1}l_{3,1} = 10 \rightarrow \ I_{3,1} = \frac{10}{I_{1,1}} = \frac{10}{2} = 5$

Si $l_{2,1^2} + l_{2,2^2} = 25 \rightarrow \ I_{2,2} = \sqrt{25 - I_{2,1^2}} = \sqrt{25 - 3^2} = \sqrt{16}$

Si $l_{2,1} l_{3,1} + l_{2,2} l_{3,2} = 19 \rightarrow \ I_{3,2} = \frac{19 - I_{3,1} I_{2,1}}{I_{2,2}} = \frac{19 - 5 \cdot 3}{4} = 1$

Si $l_{3,1^2} + l_{3,2^2} + l_{3,3^2} = 62 \rightarrow \ I_{3,3} = \sqrt{62 - \left(I_{3,1^2} + I_{3,2^2}\right)} = \sqrt{62 - \left(5^2 + 1^2\right)} = \sqrt{36}$

Entonces

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

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

Conclusión de A:

Se puede concluir que la matriz es simétrica es decir ($A = A^T$) en donde L es una matriz triangular inferior con entradas diagonales reales y positivas.

Análisis de B

$$\begin{pmatrix} 4 & 6 & 10 \\ 6 & 3 & 19 \\ 10 & 19 & 62 \end{pmatrix} = \begin{pmatrix} I_{1,1} & 0 & 0 \\ I_{2,1} & I_{2,2} & 0 \\ I_{3,1} & I_{3,2} & I_{3,3} \end{pmatrix} . \begin{pmatrix} I_{1,1} & I_{2,1} & I_{3,1} \\ 0 & I_{2,2} & I_{3,2} \\ 0 & 0 & I_{3,3} \end{pmatrix} = \begin{pmatrix} I_{1,1^2} & I_{1,1} I_{2,1} & I_{1,1} I_{3,1} \\ I_{1,1} I_{2,1} & I_{2,1^2 + I_{2,2^2}} & I_{2,1} I_{3,1} + I_{2,2} I_{3,2} \\ I_{1,1} I_{3,1} & I_{2,1} I_{3,1} + I_{2,2} I_{3,2} & I_{3,1^2} + I_{3,2^2} + I_{3,3^2} \end{pmatrix}$$

Por lo tanto:

Si $l_{1,1^2} = 4 \rightarrow \ l_{1,1} = \sqrt{4} = 2$

Si $l_{1,1} l_{2,1} = 6 \rightarrow \ I_{2,1} = \frac{6}{I_{1,1}} = \frac{6}{2} = 3$

Si $l_{1,1} l_{3,1} = 10 \rightarrow \ I_{3,1} = \frac{10}{I_{1,1}} = \frac{10}{2} = 5$

Si $l_{2,1^2} + l_{2,2^2} = 3 \rightarrow \ I_{2,2} = \sqrt{3 - I_{2,1^2}} = \sqrt{3 - 3^2} = \sqrt{-6}$

Entonces

La matriz no es positiva, por lo tanto para esta matriz es imposible calcular la factorización de Cholesky

Conclusión de B:

En definitiva la descomposición de Cholesky($A = L \cdot L^T$), en donde cada matriz definida positiva simétrica puede descomponerse en un producto de una matriz triangular inferior única L y su transpuesta, que en otras palabras en este caso para la matriz "B" no se cumple.

Código en Python

In [ ]:

```
## DECLARAMOS VARIABLES Y LIBRERÍAS A UTILIZAR EN NUESTRO EJERCICIO
import math
MAX = 100
## INICIO DEL ALGORITMO
def Cholesky(matriz, n):
    infer = [[0 for x in range(n + 1)]
             for y in range(n + 1)]
## DESCOMPONEMOS LA MATRIZ EN SU DIAGONAL INFERIOR
    for i in range(n):
        for j in range(i + 1):
            sum1 = 0
## REALIZAMOS SUMATORIA DE LAS DIAGONALES
            if (j == i):
                for k in range(j):
    ## POW NOS AYUDA A ELEVAR UN NÚMERO PARA OPERARLO
                    sum1 += pow(infer[j][k], 2)
                infer[j][j] = int(math.sqrt(matriz[j][j] - sum1))
            else:
    # EVALUAMOS Y SEGUIDAMENTE USAMOS L(i, j)
                for k in range(j):
                    sum1 += (infer[i][k] * infer[j][k])
                if (infer[j][j] > 0):
                    infer[i][j] = int((matriz[i][j] - sum1) /
                                      infer[j][j])

## MOSTRAMOS EN PANTALLA LA DIAGONAL INFERIOR Y SU TRANSPUESTA
    print("\x1b[1;34m"+"DIAGONAL INFERIOR\t\tTRANSPUESTA")
    for i in range(n):
## DIAG. INFER.
        for j in range(n):
            print(infer[i][j], end="\t")
        print("\x1b[1;34m"+"", end="\t")
## TRANSPUESTA
        for j in range(n):
            print(infer[j][i], end="\t")
        print("")

## INGRESAMOS DATOS EN PANTALLA
n = 3
matriz = [[4, 6, 10],
          [6, 25, 19],
          [10, 19, 62]]
Cholesky(matriz, n)
```

Salida de Escritorio

```
DIAGONAL INFERIOR       TRANSPUESTA
2   0   0       2   3   5
3   4   0       0   4   1
5   1   6       0   0   6

Process finished with exit code 0
```

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 $x = (1, -1, 1, -1)^T$.

1. Gauss Seidel

El total de ecuaciones son 4:

$$\begin{pmatrix} 7a + b - c + 2d = 3 \\ a + 8b - 0c - 2d = -5 \\ -a - 0b + 4c - d = 4 \\ 2a - 2b - c + 6d = -3 \end{pmatrix}$$

De las ecuaciones anteriores

$$\begin{pmatrix} a_{k+1} = \frac{1}{7}\left(3 - b_k + c_k - 2d_k\right) \\ b_{k+1} = \frac{1}{8}\left(-5 - a_{k+1} - 0c_k + 2d_k\right) \\ c_{k+1} = \frac{1}{4}\left(4 + a_{k+1} - 0b_{k+1} + d_k\right) \\ d_{k+1} = \frac{1}{6}\left(-3 - 2a_{k+1} + 2b_{k+1} + c_{k+1}\right) \end{pmatrix}$$

Los valores iniciales dados serán (0,1,2,3)

Programa en Python

In [ ]:

```
## DECLARAMOS VARIABLES Y LIBRERÍAS A UTILIZAR EN NUESTRO EJERCICIO
import numpy as np
## DEFINIMOS UNA VARIABLE CONSTANTE PARA NUESTRO LIMITE ITERATIVO
LIMITE_ITER = 1000
## DECLARAMOS NUESTRA MATRIZ
A = np.array([[7., 1., -1., 2.],
          [1., 8., 0., -2.],
          [-1., -0., 4., -1.],
          [2., -2., -1., 6.]
          ])
## Y SU VECTOR B
b = np.array([3., -5., 4., -3.])
## MOSTRAMOS EL SISTEMA EN PANTALLA
print("\x1b[1;34m"+"EL SISTEMA ES:")
for i in range(A.shape[0]):
    fila = ["{0:3g}*x{1}".format(A[i, j], j + 1) for j in range(A.shape[1])]
    print("[{0}] = [{1:3g}]".format(" + ".join(fila), b[i]))

x = np.zeros_like(b)
## MÉTODO GAUSS SEIDEL
for it_count in range(1, LIMITE_ITER):
    x_new = np.zeros_like(x)
    print("ITERACIÓN {0}: {1}".format(it_count, x))
    for i in range(A.shape[0]):
        s1 = np.dot(A[i, :i], x_new[:i])
        s2 = np.dot(A[i, i + 1:], x[i + 1:])
        x_new[i] = (b[i] - s1 - s2) / A[i, i]
## DEFINIMOS LA PRECISIÓN DE 4 CIFRAS DECIMALES
    if np.allclose(x, x_new, rtol=1e-4):
        break
    x = x_new
```

Salida de Escritorio

```
EL SISTEMA ES:
[  7*x1 +   1*x2 +  -1*x3 +   2*x4] = [  3]
[  1*x1 +   8*x2 +   0*x3 +  -2*x4] = [ -5]
[ -1*x1 +  -0*x2 +   4*x3 +  -1*x4] = [  4]
[  2*x1 +  -2*x2 +  -1*x3 +   6*x4] = [ -3]
ITERACIÓN 1: [0. 0. 0. 0.]
ITERACIÓN 2: [ 0.42857143 -0.67857143  1.10714286 -0.68452381]
ITERACIÓN 3: [ 0.8792517  -0.90603741  1.04868197 -0.92031604]
ITERACIÓN 4: [ 0.9707645  -0.97642457  1.01261211 -0.98029434]
ITERACIÓN 5: [ 0.99280362 -0.99417404  1.00312732 -0.995138  ]
ITERACIÓN 6: [ 0.99822534 -0.99856267  1.00077183 -0.9988007 ]
ITERACIÓN 7: [ 0.99956227 -0.99964546  1.00019039 -0.99970418]
ITERACIÓN 8: [ 0.99989203 -0.99991255  1.00004696 -0.99992703]

Process finished with exit code 0
```

2. Gauss Jacobi

El total de ecuaciones son 4:

$$\begin{pmatrix} 7a + b - c + 2d = 3 \\ a + 8b - 0c - 2d = -5 \\ -a - 0b + 4c - d = 4 \\ 2a - 2b - c + 6d = -3 \end{pmatrix}$$

De las ecuaciones anteriores

$$\begin{pmatrix} a_{k+1} = \frac{1}{7}\left(3 - b_k + c_k - 2d_k\right) \\ b_{k+1} = \frac{1}{8}\left(-5 - a_{k+1} - 0c_k + 2d_k\right) \\ c_{k+1} = \frac{1}{4}\left(4 + a_{k+1} - 0b_{k+1} + d_k\right) \\ d_{k+1} = \frac{1}{6}\left(-3 - 2a_{k+1} + 2b_{k+1} + c_{k+1}\right) \end{pmatrix}$$

Los valores iniciales dados serán (0,1,2,3)

Programa en Python

In [ ]:

```
## DECLARAMOS VARIABLES Y LIBRERÍAS A UTILIZAR EN NUESTRO EJERCICIO
import numpy as np
import copy
import functools
import math
## PEDIMOS DATOS POR PANTALLA
a = input("\x1b[1;34m"+"INTRODUZCA EL NÚMERO DE LA VARIABLE INDEPENDIENTE X Y EL NÚMERO DE ECUACIONES: ")
## LOS GUARDAMOS RESPECTIVAMENTE EN ARREGLOS
mu, nu = [int(i) for i in a.split(" ")]
b = input("DIGITE LA PRESICIÓN DEL ERROR REQUERIDO (EN DECIMALES, EJ: 0.0001): ")
e = float(b)
## INICIALIZAMOS LA MATRIZ LDU
L, D, U = [], [], []
for p in range(nu):
    L.append([]), D.append([]), U.append([])
## INGRESAMOS LOS DATOS DEL SISTEMA POR TECLADO
    for q in range(mu):
        x_in = float(input("DIGITE EL COEFICIENTE EN LA FILA% d Y LA COLUMNA% d: " % (p + 1, q + 1)))
        if p < q:
            L[p].append(x_in), D[p].append(0), U[p].append(0)
        elif p == q:
            L[p].append(0), D[p].append(x_in), U[p].append(0)
        else:
            L[p].append(0), D[p].append(0), U[p].append(x_in)
L, D, U = np.array(L), np.array(D), np.array(U)
## ESTA ES NUESTRA VARIABLE INDEPENDIENTE X MATRIZ
X_Current = []
for q in range(mu):
    x_in = float(input("DIGITE EL VALOR INICIAL PARA X% d: " %q))
    X_Current.append(x_in)
## TRANSPONEMOS EL VECTOR DE FILA X A UN VECTOR DE COLUMNA PARA FACILITAR EL CÁLCULO DE LA MATRIZ SUBSIGUIENTE
X_Current = np.array(X_Current).T
## MATRIZ Y VARIABLE DEPENDIENTE
b_Const = []
for p in range(nu):
    y_in = float(input("DIGITE EL VALOR DE LA ECUACIÓN B/Y% d: " % (p + 1)))
    b_Const.append(y_in)
## REALIZAMOS LO MISMO DE ARRIBA PARA B
b_Const = np.array(b_Const).T
L_U = copy.deepcopy(L)
for p in range(nu):
    for q in range(mu):
## SUMA L Y U POR FILA Y SUMA LAS POSICIONES CORRESPONDIENTES
        L_U[p][q] = L[p][q] + U[p][q]
## AQUI ENCONTRAMOS LA INVERSA DE LA MATRIZ
G1 = np.dot(-np.linalg.inv(D), L_U)
d1 = np.dot(np.linalg.inv(D), b_Const)
## COPIAMOS SISTEMA
X_New = copy.deepcopy(X_Current)

## MÉTODO DE JACOBI
## INICIALIZAMOS ITERACIONES IGUAL A CERO
super = 0
while 1:
## AUXILIAR PARA LA PRESICIÓN DEL ERROR
    band = 0
    X_Current = X_New
```

```
    X_New = np.dot(G1, X_Current)
    for p in range(mu):
        X_New[p] = X_New[p] + d1[p]
        if math.fabs(X_New[p]-X_Current[p]) < e:
            band += 1
    super += 1
##  ESTABLECEMOS EL LÍMITE SUPERIOR DE ITERACIÓN EN 50 VECES
    if super > 50:
        print("\x1b[1;34m"+"LA ITERACIÓN NO CONVERGE")
        super = 0
        break
## MOSTRAMOS RESULTADOS EN PANTALLA
    if band == mu:
        print("\x1b[1;34m"+"LOS RESULTADOS SON: ")
        print(X_New)
        break
```

Salida de Escritorio

```
INTRODUZCA EL NÚMERO DE LA VARIABLE INDEPENDIENTE X Y EL NÚMERO DE ECUACIONES: 4 4
DIGITE LA PRESICIÓN DEL ERROR REQUERIDO (EN DECIMALES, EJ: 0.0001): 0.0001
DIGITE EL COEFICIENTE EN LA FILA 1 Y LA COLUMNA 1:7
DIGITE EL COEFICIENTE EN LA FILA 1 Y LA COLUMNA 2:1
DIGITE EL COEFICIENTE EN LA FILA 1 Y LA COLUMNA 3:-1
DIGITE EL COEFICIENTE EN LA FILA 1 Y LA COLUMNA 4:2
DIGITE EL COEFICIENTE EN LA FILA 2 Y LA COLUMNA 1:1
DIGITE EL COEFICIENTE EN LA FILA 2 Y LA COLUMNA 2:8
DIGITE EL COEFICIENTE EN LA FILA 2 Y LA COLUMNA 3:0
DIGITE EL COEFICIENTE EN LA FILA 2 Y LA COLUMNA 4:-2
DIGITE EL COEFICIENTE EN LA FILA 3 Y LA COLUMNA 1:-1
DIGITE EL COEFICIENTE EN LA FILA 3 Y LA COLUMNA 2:0
DIGITE EL COEFICIENTE EN LA FILA 3 Y LA COLUMNA 3:4
DIGITE EL COEFICIENTE EN LA FILA 3 Y LA COLUMNA 4:-1
DIGITE EL COEFICIENTE EN LA FILA 4 Y LA COLUMNA 1:2
DIGITE EL COEFICIENTE EN LA FILA 4 Y LA COLUMNA 2:-2
DIGITE EL COEFICIENTE EN LA FILA 4 Y LA COLUMNA 3:-1
DIGITE EL COEFICIENTE EN LA FILA 4 Y LA COLUMNA 4:6
DIGITE EL VALOR INICIAL PARA X 0:0
DIGITE EL VALOR INICIAL PARA X 1:1
DIGITE EL VALOR INICIAL PARA X 2:2
DIGITE EL VALOR INICIAL PARA X 3:3
DIGITE EL VALOR DE LA ECUACIÓN B/Y 1:3
DIGITE EL VALOR DE LA ECUACIÓN B/Y 2:-5
DIGITE EL VALOR DE LA ECUACIÓN B/Y 3:4
DIGITE EL VALOR DE LA ECUACIÓN B/Y 4:-3
LOS RESULTADOS SON:
[ 0.99997754 -0.99997515  0.99999645 -0.99994881]
```

3. SOR

El total de ecuaciones son 4:

$$\begin{pmatrix} 7a + b - c + 2d = 3 \\ a + 8b - 0c - 2d = -5 \\ -a - 0b + 4c - d = 4 \\ 2a - 2b - c + 6d = -3 \end{pmatrix}$$

De las ecuaciones anteriores

$$\begin{pmatrix} a_{k+1} = \frac{1}{7}\left(3 - b_k + c_k - 2d_k\right) \\ b_{k+1} = \frac{1}{8}\left(-5 - a_{k+1} - 0c_k + 2d_k\right) \\ c_{k+1} = \frac{1}{4}\left(4 + a_{k+1} - 0b_{k+1} + d_k\right) \\ d_{k+1} = \frac{1}{6}\left(-3 - 2a_{k+1} + 2b_{k+1} + c_{k+1}\right) \end{pmatrix}$$

Los valores iniciales dados serán (0,1,2,3)

1era Aproximación

$$\begin{pmatrix} a_1 = (1-11)\cdot 0 + 1.1\cdot\frac{1}{7}[3-(1)+(2)-2(3)] = (-0,1)\cdot 0 + 1.1\cdot\frac{1}{7}[-2] = 0 + 0.3143 = -0.3143 \\ b_1 = (1-1.1)\cdot 1 + 1.1\cdot\frac{1}{8}[-5-(-0.3143)-0(2)+2(3)] = (-0.1)\cdot 1 + 1.1\cdot\frac{1}{8}[1.3143] = -0.1 + 0.1807 = 0.0807 \\ c_1 = (1-1.1)\cdot 2 + 1.1\cdot\frac{1}{4}[4+(-0.3143)-0(0.0807)+(3)] = (-0.1)\cdot 2 + 1.1\cdot\frac{1}{4}[6.6857] = -0.2 + 1.8386 = 1.6386 \\ d_1 = (1-1.1)\cdot 3 + 1.1\cdot\frac{1}{6}[-3-2(-0.3143)+2(0.0807)+(1.6386)] = (-0.1)\cdot 3 + 1.1\cdot\frac{1}{6}[-0.5714] = -0.3 \pm 0.1048 = -0.4048 \end{pmatrix}$$

2da Aproximación

$$\left\langle \begin{array}{c} a_2 = (1 - 1.1) \cdot -0.3143 + 1.1 \cdot \frac{1}{7}[3 - (0.0807) + (1.6386) - 2(-0.4048)] = (-0.1) \cdot -0.3143 + 1.1 \cdot \frac{1}{7}[5.3674] = 0.0314 + 0.8434 = 0.8749 \\ b_2 = (1 - 1.1) \cdot 0.0807 + 1.1 \cdot \frac{1}{8}[-5 - (0.8749) - 0(1.6386) + 2(-0.4048)] = (-0.1) \cdot 0.0807 + 1.1 \cdot \frac{1}{8}[-6.6844] = -0.0081 \pm 0.9191 = -0.9272 \\ c_2 = (1 - 1.1) \cdot 1.6386 + 1.1 \cdot \frac{1}{4}[4 + (0.8749) - 0(-0.9272) + (-0.4048)] = (-0.1) \cdot 1.6386 + 1.1 \cdot \frac{1}{4}[4.4701] = -0.1639 + 1.2293 = 1.0654 \\ d_2 = (1 - 1.1) \cdot -0.4048 + 1.1 \cdot \frac{1}{6}[-3 - 2(0.8749) + 2(-0.9272) + (1.0654)] = (-0.1) \cdot -0.4048 + 1.1 \cdot \frac{1}{6}[-5.5387] = 0.0405 \pm 1.0154 = -0.9749 \end{array} \right\rangle$$

3ra Aproximación

$$\left\langle \begin{array}{c} a_3 = (1 - 1.1) \cdot 0.8749 + 1.1 \cdot \frac{1}{7}[3 - (-0.9272) + (1.0654) - 2(-0.9749)] = (-0.1) \cdot 0.8749 + 1.1 \cdot \frac{1}{7}[6.9425] = -0.0875 + 1.091 = 1.0035 \\ b_3 = (1 - 1.1) \cdot -0.9272 + 1.1 \cdot \frac{1}{8}[-5 - (1.0035) - 0(1.0654) + 2(-0.9749)] = (-0.1) \cdot -0.9272 + 1.1 \cdot \frac{1}{8}[-7.9534] = 0.0927 \pm 1.0936 = -1.0009 \\ c_3 = (1 - 1.1) \cdot 1.0654 + 1.1 \cdot \frac{1}{4}[4 + (1.0035) - 0(-1.0009) + (-0.9749)] = (-0.1) \cdot 1.0654 + 1.1 \cdot \frac{1}{4}[4.0285] = -0.1065 + 1.1078 = 1.0013 \\ d_3 = (1 - 1.1) \cdot -0.9749 + 1.1 \cdot \frac{1}{6}[-3 - 2(1.0035) + 2(-1.0009) + (1.0013)] = (-0.1) \cdot -0.9749 + 1.1 \cdot \frac{1}{6}[-6.0074] = 0.0975 \pm 1.1014 = -1.0039 \end{array} \right\rangle$$

4ta. Aproximación

$$\left\langle \begin{array}{c} a_4 = (1 - 1.1) \cdot 1.0035 + 1.1 \cdot \frac{1}{7}[3 - (-1.0009) + (1.0013) - 2(-1.0039)] = (-0.1) \cdot 1.0035 + 1.1 \cdot \frac{1}{7}[7.0099] = -0.1003 + 1.1016 = 1.0012 \\ b_4 = (1 - 1.1) \cdot -1.0009 + 1.1 \cdot \frac{1}{8}[-5 - (1.0012) - 0(1.0013) + 2(-1.0039)] = (-0.1) \cdot -1.0009 + 1.1 \cdot \frac{1}{8}[-8.0089] = 0.1001 \pm 1.1012 = -1.0011 \\ c_4 = (1 - 1.1) \cdot 1.0013 + 1.1 \cdot \frac{1}{4}[4 + (1.0012) - 0(-1.0011) + (-1.0039)] = (-0.1) \cdot 1.0013 + 1.1 \cdot \frac{1}{4}[3.9973] = -0.1001 + 1.0993 = 0.9991 \\ d_4 = (1 - 1.1) \cdot -1.0039 + 1.1 \cdot \frac{1}{6}[-3 - 2(1.0012) + 2(-1.0011) + (0.9991)] = (-0.1) \cdot -1.0039 + 1.1 \cdot \frac{1}{6}[-6.0056] = 0.1004 \pm 1.101 = -1.0006 \end{array} \right\rangle$$

5ta. Aproximación

$$\begin{pmatrix} a_5 = (1 - 1.1) \cdot 1.0012 + 1.1 \cdot \frac{1}{7}[3 - (-1.0011) + (0.9991) - 2(-1.0006)] = (-0.1) \cdot 1.0012 + 1.1 \cdot \frac{1}{7}[7.0015] = -0.1001 + 1.1002 = 1.0001 \\ b_5 = (1 - 1.1) \cdot -1.0011 + 1.1 \cdot \frac{1}{8}[-5 - (1.0001) - 0(0.9991) + 2(-1.0006)] = (-0.1) \cdot -1.0011 + 1.1 \cdot \frac{1}{8}[-8.0014] = 0.1001 \pm 1.1002 = -1.0001 \\ c_5 = (1 - 1.1) \cdot 0.9991 + 1.1 \cdot \frac{1}{4}[4 + (1.0001) - 0(-1.0001) + (-1.0006)] = (-0.1) \cdot 0.9991 + 1.1 \cdot \frac{1}{4}[3.9995] = -0.0999 + 1.0999 = 0.9999 \\ d_5 = (1 - 1.1) \cdot -1.0006 + 1.1 \cdot \frac{1}{6}[-3 - 2(1.0001) + 2(-1.0001) + (0.9999)] = (-0.1) \cdot -1.0006 + 1.1 \cdot \frac{1}{6}[-6.0005] = 0.1001 \pm 1.1001 = -1 \end{pmatrix}$$

6ta. Aproximación

$$\begin{pmatrix} a_6 = (1 - 1.1) \cdot 1.0001 + 1.1 \cdot \frac{1}{7}[3 - (-1.0001) + (0.9999) - 2(-1)] = (-0.1) \cdot 1.0001 + 1.1 \cdot \frac{1}{7}[7.0001] = -0.1 + 1.1 = 1 \\ b_6 = (1 - 1.1) \cdot -1.0001 + 1.1 \cdot \frac{1}{8}[-5 - (1) - 0(0.9999) + 2(-1)] = (-0.1) \cdot -1.0001 + 1.1 \cdot \frac{1}{8}[-8] = 0.1 \pm 1.1 = -1 \\ c_6 = (1 - 1.1) \cdot 0.9999 + 1.1 \cdot \frac{1}{4}[4 + (1) - 0(-1) + (-1)] = (-0.1) \cdot 0.9999 + 1.1 \cdot \frac{1}{4}[4] = -0.1 + 1.1 = 1 \\ d_6 = (1 - 1.1) \cdot -1 + 1.1 \cdot \frac{1}{6}[-3 - 2(1) + 2(-1) + (1)] = (-0.1) \cdot -1 + 1.1 \cdot \frac{1}{6}[-6] = 0.1 \pm 1.1 = -1 \end{pmatrix}$$

La solución es:

$$\begin{pmatrix} a = 1 \to 1 \\ b = -1 \to -1 \\ c = 1 \to 1 \\ d = -1 \to -1 \end{pmatrix}$$

Comparación del número de interaciones:

Realicé una tabla con el fin de comparar de mejor manera los resultados y la comparación es la siguiente:

Gauss Seidel

Iteration	a	b	c	d
1	-0.2857	0.1607	1.6786	-0.0714
2	0.6658	-0.7261	1.1486	-0.7725
3	0.9171	-0.9328	1.0361	-0.9439
4	0.9795	-0.9834	1.0089	-0.9862
5	0.995	-0.9959	1.0022	-0.9966
6	0.9988	-0.999	1.0005	-0.9992
7	0.9997	-0.9998	1.0001	-0.9998

Gauss Jacobi

Iteration	a	b	c	d
1	-0.2857	0.125	1.75	0.1667
2	0.6131	-0.5476	0.9702	-0.0714
3	0.6658	-0.7195	1.1354	-0.7252
4	0.9008	-0.8895	0.9852	-0.7725
5	0.9171	-0.9307	1.0321	-0.9326
6	0.9754	-0.9728	0.9961	-0.9439
7	0.9795	-0.9829	1.0079	-0.9834
8	0.9939	-0.9933	0.999	-0.9862
9	0.995	-0.9958	1.0019	-0.9959
10	0.9985	-0.9983	0.9998	-0.9966
11	0.9988	-0.999	1.0005	-0.999
12	0.9996	-0.9996	0.9999	-0.9992

SOR

Iteration	a	b	c	d
1	-0.3143	0.0807	1.6386	-0.4048
2	0.8749	-0.9272	1.0654	-0.9749
3	1.0035	-1.0009	1.0013	-1.0039
4	1.0012	-1.0011	0.9991	-1.0006
5	1.0001	-1.0001	0.9999	-1
6	1	-1	1	-1

Análisis del error cometido:

En conclusión al analizar los errores puedo decir que el mejor método y el más cercano al valor exacto es el SOR puesto que gracias al parámetro w es posible acelerar la convergencia del algoritmo, en segundo lugar queda el método Gauss Seidel pues es otro que en un menor número de interaciones ha sido capaz de acercarse bastante al valor exacto con un menor error cometido y por último el peor método es el Gauss Seidel que como podemos observar le tomó muchas más iteraciones para llegar a estar un tanto cerca del valor cometido y sin embargo puedo notar que el error cometido es mayor en comparación a los otros 2 métodos.

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

(Para demostrar de mejor manera la norma de Frobenius lo he hecho de 2 maneras diferentes con el fin de interpretar el algoritmo mucho mejor).

Forma I

In [ ]:

```
## DECLARAMOS VARIABLES Y LIBRERÍAS A UTILIZAR EN NUESTRO EJERCICIO
from math import sqrt
import numpy as np
## DECLARAMOS SUS DIMENSIONES
fila = 2
col = 2
## DEFINIMOS MATRIZ CUADRADA
x = np.array([[2,4],[6,8]])
print("\x1b[1;34m"+"LA SIGUIENTE MATRIZ CUADRARA DE DIMENSIÓN 2 ESTÁ REPRESENTADA POR:")
print(x)
def frobenius(mat):
## AUXILIAR QUE ALMACENA LA SUMA DE LOS CUADRADOS DE LA MATRIZ OBTENIDA
    sum = 0
    for i in range(fila):
        for j in range(col):
            sum += pow(mat[i][j], 2)
## RETORNA LA RAÍZ CUADRADA DE LA SUMA DE CUADRADOS
    res = sqrt(sum)
## RETORNAMOS VALOR CON PRECISIÓN DE 4 CIFRAS DECIMALES
    return round(res, 4)
```

```
#3 MATRIZ OBTENIDA
mat = [[2, 4], [6, 8]]

print("\x1b[1;34m"+"LA NORMA DE FROBENIUS ES: ")
print(frobenius(mat))
```

Salida de Escritorio

```
LA SIGUIENTE MATRIZ CUADRARA DE DIMENSIÓN 2 ESTÁ REPRESENTADA POR:
[[2 4]
 [6 8]]
LA NORMA DE FROBENIUS ES:
10.9545

Process finished with exit code 0
```

Forma II

In [ ]:

```
## DECLARAMOS VARIABLES Y LIBRERÍAS A UTILIZAR EN NUESTRO EJERCICIO
import numpy as np
## DECLARAMOS SUS DIMENSIONES
x = np.arange(1, 10).reshape((3, 3))
print("\x1b[1;34m"+"LA SIGUIENTE MATRIZ CUADRARA DE DIMENSIÓN 3 ESTÁ REPRESENTADA POR:")
print(x)
print("\x1b[1;34m"+"LA NORMA ES: ")
## ESTA FUNCIÓN PUEDE DEVOLVER UNA DE OCHO NORMAS MATRICIALES DIFERENTES, O UNA DE UN NÚMERO INFINITO DE NORMAS VECTORIALES.
print(np.linalg.norm(x, "fro"))
print("\x1b[1;34m"+"EL NÚMERO DE CONDICIÓN DE FROBENIUS ES: ")
## ESTA FUNCIÓN ES CAPAZ DE DEVOLVER EL NÚMERO DE CONDICIÓN USANDO UNA DE SIETE NORMAS DIFERENTES, DEPENDIENDO DEL VALOR DE PARÁMETRO.
print(np.linalg.cond(x, 'fro'))
```

Salida de Escritorio

```
LA SIGUIENTE MATRIZ CUADRARA DE DIMENSIÓN 3 ESTÁ REPRESENTADA POR:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
LA NORMA ES:
16.881943016134134
EL NÚMERO DE CONDICIÓN DE FROBENIUS ES:
4.56177073661e+17
```

Muchas Gracias. 😃