SistemasLineales

July 24, 2021

1 1. Para el sistema lineal

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

2 Encuentre los valores x1, x2, x3, x4 utilizando sustitución progresiva. Encuentre al descomposición LU del sistema anterior. ¿Qué puede concluir?

$$Sustitucinprogresiva$$

$$x_1 = \frac{b_1}{a_{11}}$$

$$x_i = \frac{\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j\right)}{a_{ii}} \quad i = 1, 2, ..., n$$

[1]:
```python
import numpy as np
n = 4
A = np.array([[6,0,0,0], # Matriz de coeficientes
             [3,6,0,0],
             [4,-2,7,0],
             [5,-3,9,21]])
print('\t Matriz A \n\n',A,'\n') # Visualización de los datos de la matriz␣
 ↪decoeficientes
b = ([12, # Vector de términos independientes
      -12,
      14,
      -2])
bT = np.array([b])
#print(bT.shape)
#print(bT.T.shape)
```

```
print('\t Vector b \n\n',bT.T,'\n') # Visualización de manera vertical del␣
 ↪vector de términos independientes
x = np.zeros(n) # Vector donde se guardarán las soluciones
#print(b[0]) # Comprobación de que el almacenamiento inició en la posición [0]
#print(A[0,0]) # Comprobación de que el almacenamiento inició en la posición␣
 ↪[0,0]

# Sustitución regresiva
x[0] = b[0]/A[0,0] # Cálculo directo de la primera solución
#print(x)

# Cálculo de las soluciones restantes
for i in range(1,n,1):
    x[i] = b[i]
    #print(b[i])
    for j in range(0,i,1):
        #print(i,j)
        x[i] = x[i] - A[i,j]*x[j]
        #print(x[i])
    #print(x)
    x[i] = x[i]/A[i,i]
xT = np.array([x]).T
print('\t Vector x \n\n',xT)
```

```
        Matriz A

[[ 6  0  0  0]
 [ 3  6  0  0]
 [ 4 -2  7  0]
 [ 5 -3  9 21]]

        Vector b

[[ 12]
 [-12]
 [ 14]
 [ -2]]

        Vector x

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

[2]:
```
L = np.zeros([n,n])
U = np.zeros([n,n])
```

```python
for k in range(0,n,1):
    L[k,k] = 1
    for j in range(k,n,1):
        U[k,j] = A[k,j]
        for s in range(1,k-1,1):
            U[k,j] = U[k,j] - L[k,s]*U[s,j]
        #print(U[k,j])
    for i in range(k+1,n,1):
        L[i,k] = A[i,k]
        for s in range(1,k-1,1):
            L[i,k] = L[i,k] - L[i,s]*U[s,k]
        L[i,k] = L[i,k]/U[k,k]
print('\t Matriz L \n\n',L,'\n')
print('\t Matriz U \n\n',U,'\n')
print('\t L * U = A \n\n',np.dot(L,U))
```

```
        Matriz L

[[ 1.          0.          0.          0.        ]
 [ 0.5         1.          0.          0.        ]
 [ 0.66666667 -0.33333333  1.          0.        ]
 [ 0.83333333 -0.5         1.28571429  1.        ]]

        Matriz U

[[ 6.  0.  0.  0.]
 [ 0.  6.  0.  0.]
 [ 0.  0.  7.  0.]
 [ 0.  0.  0. 21.]]

        L * U = A

[[ 6.  0.  0.  0.]
 [ 3.  6.  0.  0.]
 [ 4. -2.  7.  0.]
 [ 5. -3.  9. 21.]]
```

3 Con los resultados obtenidos, vemos que la matriz dada puede ser decompuesta en las matrices L y U, además, la matriz U tiene 4 pivotes, por lo que la matriz A sería una matriz invertible, por tanto, el producto LU es único.

4 2. Resuelva las siguientes matrices utilizando la descomposición de Cholesky

$$A = \begin{pmatrix} 4 & 6 & 10 \\ 6 & 25 & 19 \\ 10 & 19 & 62 \end{pmatrix} \tag{1}$$

$$B = \begin{pmatrix} 4 & 6 & 10 \\ 6 & 3 & 19 \\ 10 & 19 & 62 \end{pmatrix} (1)$$

Analice el caso y escriba sus conclusiones

5 Podemos ver que A y B son matrices simétricas ya que son iguales a sus respectivas matrices transpuestas. Ahora veremos si las matrices son definidas positivas, para ello emplearemos el méto de Cholesky.

[3]:
```
n = 3
A = np.array([[4,6,10],
             [6,25,19],
             [10,19,62]])
B = np.array([[4,6,10],
             [6,3,19],
             [10,19,62]])
def cholesky(M):
    A = M

    L = np.zeros([n,n])
    k = 0
    for k in range(0,n,1):
    #while k < n:
        #print('k',k)
        L[k,k] = A[k,k]
        #print('l',k,k,'=',L[k,k],'antes de restar la sumatoria')
        for s in range(0,k,1):
            L[k,k] = L[k,k] - L[k,s]**2
            #print('l',k,k,'=',L[k,k],'después de restar la sumatoria',s)
        if L[k,k] > 0:
            L[k,k] = L[k,k]**0.5
            #print('l',k,k,'=',L[k,k],'final')
            for i in range(k+1,n,1):
```

```
                L[i,k] = A[i,k]
                #print('l',i,k,'=',L[i,k],'antes de restar la sumatoria')
                for s in range(0,k,1):
                    L[i,k] = L[i,k] - L[i,s]*L[k,s]
                    #print('l',i,k,'=',L[i,k],'después de la restar la␣
 ↪sumatoria',s)
                L[i,k] = L[i,k]/L[k,k]
                #print('l',i,k,'=',L[i,k],'final')
            k +=1
            if k == n:
                print('\t Matriz L \n\n',L,'\n')
                print('\t Matriz L^T \n\n',L.T,'\n')
                print('Matriz definida positiva L * L^T \n\n',np.dot(L,L.T),'\n')
        else:
            print('Matriz no definida positiva \n\n',M,'\n')
            break
    return
cholesky(A)
cholesky(B)
```

```
         Matriz L

 [[2. 0. 0.]
 [3. 4. 0.]
 [5. 1. 6.]]

         Matriz L^T

 [[2. 3. 5.]
 [0. 4. 1.]
 [0. 0. 6.]]

Matriz definida positiva L * L^T

 [[ 4.  6. 10.]
 [ 6. 25. 19.]
 [10. 19. 62.]]

Matriz no definida positiva

 [[ 4  6 10]
 [ 6  3 19]
 [10 19 62]]
```

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

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

7 Método de Jacobi

$$x_i^{k+1} = \frac{1}{a_{i_i}} \begin{pmatrix} b_i - \sum_{j=1} \\ j \neq i^n a_{ij} x_j^k \end{pmatrix} \quad \| x^k - x^{k-1} \| < \epsilon$$

[4]:
```
n = 4
A = np.array([[7,1,-1,2],
             [1,8,0,-2],
             [-1,0,4,-1],
             [2,-2,-1,6]])
b = ([3,
      -5,
      4,
      -3])
x = np.zeros(n)
y = np.zeros(n)
k = 0
iMax = 50
tol = 1e-4
sumas = 0
while k < iMax:
    for i in range(0,n,1):
        y[i] = x[i]
        #print('x^k ',i,' = ',y,'\n')
        x[i] = b[i]
        for j in range(0,n,1):
            if j != i:
                #print('i =',i,' j =',j)
                x[i] = x[i] - A[i,j]*x[j]
                sumas += 1
        #print('i =',i,' j =',j)
        #print('sumatoria ',i,' = ','\t',sumAx)
        x[i] = x[i]/A[i,i]
        #print('x^k+1 ',i,' = ',x,'\n')
    k += 1
    #print(k)
```

```
    if k == iMax:
        print('Se alcanzó el número máximo de iteraciones \n')
    elif np.absolute(x[i] - y[i]) < tol:
        #print(np.absolute(x[i] - y[i]),'error (anterior - actual) \n')
        break
print(k,'fueron las iteraciones realizadas con la tolerancia␣
 ↪de',tol,'con',sumas,'sumatorias\n')
xT = np.array([x]).T
print('\t Vector x \n',xT,'\n')
print('El error para el método de Jacobi \n')
xR = ([1,-1,1,-1])
n = 4
e = np.zeros(n)
for i in range(0,n,1):
    e[i] = (xR[i]-x[i])/xR[i]
    print('El error de x',i+1,'es =',(100*abs(e[i])).round(decimals=4),'%')
```

```
8 fueron las iteraciones realizadas con la tolerancia de 0.0001 con 96
sumatorias

         Vector x
 [[ 0.99997337]
 [-0.99997843]
 [ 1.00001158]
 [-0.999982  ]]

El error para el método de Jacobi

El error de x 1 es = 0.0027 %
El error de x 2 es = 0.0022 %
El error de x 3 es = 0.0012 %
El error de x 4 es = 0.0018 %
```

8 Método de Gauss-Seidel

$$x_i^{k+1} = \frac{1}{a_{i_i}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij} x_j^k \right) \quad i = 1, 2, ..., n \parallel x^k - x^{k-1} \parallel < \epsilon$$

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

```python
x = ([0.4285, # x^0 = Vector inicial
      -0.625,
      1,
      -0.5])
y = np.zeros(n)
k = 0
iMax = 50
tol = 1e-4
sumas = 0
while k < iMax:
    for i in range(0,n,1):
        y[i] = x[i]
        #print('x^k ',i,' = ',y,'\n')
        x[i] = b[i]
        for j in range(i+1,n,1):
            #print('i =',i,' j =',j)
            x[i] = x[i] - A[i,j]*x[j]
            sumas += 1
        for j in range(0,i-1,1):
            #print('i =',i,' j =',j)
            x[i] = x[i] - A[i,j]*x[j]
            sumas += 1
        #print('sumatoria ',i,' = ','\t',x[i])
        x[i] = x[i]/A[i,i]
        #print('x^k+1 ',i,' = ',x,'\n')
    k += 1
    #print(k)
    if k == iMax:
        print('Se alcanzó el número máximo de iteraciones \n')
    elif np.absolute(x[i] - y[i]) < tol:
        #print(np.absolute(x[i] - y[i]),'error (anterior - actual) \n')
        break
print(k,'fueron las iteraciones realizadas con la tolerancia␣
 ↪de',tol,'con',sumas,'sumatorias\n')
xT = np.array([x]).T
print('\t Vector x \n',xT,'\n')
print('El error para el método de Gauss-Seidel \n')
xR = ([1,-1,1,-1])
n = 4
e = np.zeros(n)
for i in range(0,n,1):
    e[i] = (xR[i]-x[i])/xR[i]
    print('El error de x',i+1,'es =',(100*abs(e[i])).round(decimals=4),'%')
```

```
7 fueron las iteraciones realizadas con la tolerancia de 0.0001 con 63
sumatorias
```

```
        Vector x
 [[ 1.02451435]
 [-0.91131073]
 [ 0.96981786]
 [-1.14527503]]

El error para el método de Gauss-Seidel

El error de x 1 es = 2.4514 %
El error de x 2 es = 8.8689 %
El error de x 3 es = 3.0182 %
El error de x 4 es = 14.5275 %
```

9 Método Sor (Sobre relajación sucesiva)

$$x_i^{k+1} = (1-\omega)x_i^k + \frac{\omega}{a_{i_i}}\left(b_i - \sum_{j<i} a_{ij}x_j^{k+1} - \sum_{j>i} a_{ij}x_j^k\right) \quad i = 1,2,...,n \parallel x^k - x^{k-1} \parallel < \epsilon$$

[6]:

```
n = 4
A = np.array([[7,1,-1,2],
             [1,8,0,-2],
             [-1,0,4,-1],
             [2,-2,-1,6]])
b = ([3,
      -5,
      4,
      -3])
x = ([0.4285, # x^0 = Vector inicial
      -0.625,
      1,
      -0.5])
y = np.zeros(n)
k = 0
iMax = 50
tol = 1e-4
w = 1.1
sumas = 0
while k < iMax:
    for i in range(0,n,1):
        y[i] = x[i]
        #print('x^k ',i,' = ',y,'\n')
        x[i] = b[i]
        for j in range(0,i-1,1):
            #print('i =',i,' j =',j)
            x[i] = x[i] - A[i,j]*x[j]
            sumas += 1
        for j in range(i+1,n,1):
```

```
            #print('i =',i,' j =',j)
            x[i] = x[i] - A[i,j]*x[j]
            sumas += 1
        #print('sumatoria ',i,' = ','\t',sumAx)
        x[i] =  (1-w)*y[i] + w*x[i]/A[i,i]
        #print('x^k+1 ',i,' = ',x,'\n')
    k += 1
    #print(k)
    if k == iMax:
        print('Se alcanzó el número máximo de iteraciones \n')
    elif np.absolute(x[i] - y[i]) < tol:
        #print(np.absolute(x[i] - y[i]),'error (anterior - actual) \n')
        break
print(k,'fueron las iteraciones realizadas con la tolerancia␣
 ↪de',tol,'con',sumas,'sumatorias\n')
xT = np.array([x]).T
print('\t Vector x \n',xT,'\n')
print('El error para el método de SOR \n')
xR = ([1,-1,1,-1])
n = 4
e = np.zeros(n)
for i in range(0,n,1):
    e[i] = (xR[i]-x[i])/xR[i]
    print('El error de x',i+1,'es =',(100*abs(e[i])).round(decimals=4),'%')
```

```
5 fueron las iteraciones realizadas con la tolerancia de 0.0001 con 45
sumatorias

         Vector x
 [[ 1.02451442]
 [-0.9113107 ]
 [ 0.96981854]
 [-1.14527506]]

El error para el método de SOR

El error de x 1 es = 2.4514 %
El error de x 2 es = 8.8689 %
El error de x 3 es = 3.0181 %
El error de x 4 es = 14.5275 %
```

10 3. Compare el número de iteraciones necesario en cada algoritmo. Analice el error cometido si la solución exacta es

$$x = \begin{pmatrix} 1 \\ -1 \\ 1 \\ -1 \end{pmatrix} \tag{3}$$

(3)

El número de iteraciones con la tolerancia de 0.0001 para los métodos fue # Jacobi 8 iteraciones # Gauss-Seidel 7 iteraciones # SOR 5 iteraciones # Por lo que se puede concluir que el método iterativo más rápido es SOR.

[ ]:

```

```

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

$$\| A \|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} | a_{i_j} |^2 paraunamatrizcuadradam = n}$$

[7]:

```
m = int(input('Digite el tamaño de la matriz cuadrada \t'))
n = m
A = np.zeros([m,n])
norma = 0
for i in range(0,m,1):
    for j in range(0,n,1):
        print('a',i+1,j+1,'= ',end='')
        A[i,j] = float(input())
        norma = norma + abs(A[i,j])**2
print('\n La norma de la matriz dada \n\t\n\t',norma**0.5)
```

```
Digite el tamaño de la matriz cuadrada  4
a 1 1 = 7
a 1 2 = 1
a 1 3 = -1
a 1 4 = 2
a 2 1 = 1
a 2 2 = 8
a 2 3 = 0
a 2 4 = -2
a 3 1 = -1
a 3 2 = 0
a 3 3 = 4
a 3 4 = -1
a 4 1 = 2
```

```
a 4 2 = -2
a 4 3 = -1
a 4 4 = 6

 La norma de la matriz dada

         13.674794331177344
```

[ ]:

```

```