Taller 2 Análisis Numérico

Natalia Gaona, Diego Alejandro Cardozo, Esteban Rojas

Desarrollo

Condiciones a tener en cuenta:

En el método de Jacobi la división de A = M − N es la siguiente:

$$M = D, N = D - A = L + U$$

donde D es la matriz diagonal principal de A, L es una matriz triangular inferior de entradas lij = −aij si i > j, lij = 0 si i ≤ j y U es una matriz triangular superior de entradas uij = −aij si j > i, uij = 0 si j ≤ i.

Entonces el método de Jacobi escrito en forma matricial está dado por

$$Dx^{(k+1)} = Mx^{(k+1)} = Nx^{(k)} + b = (L + U)x^{(k)} + b,$$

donde la matriz de iteración BJ y el vector c del método de Jacobi son, respectivamente BJ = D −1 (L + U) = I − D −1A, c = D −1 b.

También podemos encontrar una ecuación para calcular cada componente del vector de aproximación. En este caso, las componentes del vector x (k+1) se calculan mediante la ecuaci´on x (k+1) i = 1 aii " bi − Xn j=1,j6=i aijx (k) j # , para cada i ∈ {1, ..., n}.

Método de Gauss-Seidel:

Este método se diferencia al de Jacobi en que en la etapa (k + 1) − ésima los valores disponibles de x ^(k+1) se utilizan para actualizar la solución. La división de A = M − N correspondiente es

$$M = D - L, N = U,$$

Donde D es la matriz diagonal principal de A, L es una matriz triangular inferior y U es una matriz triangular superior. De esta forma el m´etodo de Gauss-Seidel escrito en forma matricial es dado por

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

donde la matriz de iteración asociada BGS y el vector c son, respectivamente

$$BGS = (D - L)^{-1}U, c = (D - L)^{-1} b$$

Método SOR:

Una generalización del método de Gauss-Seidel es el método de sobrerrelajación sucesiva (SOR), en el que después de introducir un parámetro w de relajación dado por la siguiente

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

En este caso la matriz de iteración B(w) y el vector c est´an son, respectivamente

$$B(w) = (D - wL)^{-1} [(1 - w)D + wU], c = w(D - wL)^{-1} b.$$

Este método es consistente para cualquier w 6= 0. Cuando w = 1 coincide con el método de Gauss-Seidel. En particular cuando 0 < w < 1 el método se llama sub-relajación mientras que si w > 1 se llama sobre relajación. Tal como en los métodos anteriores, para encontrar las componentes del vector de aproximación. Podemos notar que el método SOR converge dependiendo del valor del parámetro w, y que de la elección de este.

1.i

i.
$$u - 8v - 2w = 1$$
$$u + v + 5w = 4$$
$$3u - v + w = -2$$

a)Es la matriz A de coeficientes diagonal dominante? se puede reorganizar con operaciones entre filas para que sea diagonal dominante?

Implementación en python

```
import numpy as np

def es_diagonal_dominante(x):
    abs_x = np.abs(x)
    return np.all( 2*np.diag(abs_x) >= np.sum(abs_x, axis=1) )

A=np.array([[ 1, 8,   2],
            [ 1, 1,  5],
            [3,-1, 1]])

print(es_diagonal_dominante(A))
```

```
PS D:\Carpetas Windows\documentos\Universidad\Analisis Numerico>
False
```

Podemos notar que no es la matriz dominante debido a que si empezamos a sumar y le sacamos valor absoluto los valores que están al lado de los números de los coeficientes de la diagonal podemos notar que estos siempre tienden a ser mayores.

$$\left(\begin{array}{ccc|c} 1 & -8 & -2 & 1 \\ 1 & 1 & 5 & 4 \\ 3 & -1 & 1 & -2 \end{array}\right) \times(-1) \underset{F_2 - 1\cdot F_1 \to F_2}{\sim} \left(\begin{array}{ccc|c} 1 & -8 & -2 & 1 \\ 0 & 9 & 7 & 3 \\ 3 & -1 & 1 & -2 \end{array}\right) \times(-3) \underset{F_3 - 3\cdot F_1 \to F_3}{\sim} \left(\begin{array}{ccc|c} 1 & -8 & -2 & 1 \\ 0 & 9 & 7 & 3 \\ 0 & 23 & 7 & -5 \end{array}\right) \times\left(\frac{1}{9}\right) \underset{F_2/(9) \to F_2}{\sim} \left(\begin{array}{ccc|c} 1 & -8 & -2 & 1 \\ 0 & 1 & \frac{7}{9} & \frac{1}{3} \\ 0 & 23 & 7 & -5 \end{array}\right) \times(-23) \underset{F_3 - 23\cdot F_2 \to F_3}{\sim}$$

$$a_{3,3} = 7 - \frac{23}{1}\cdot\left(\frac{7}{9}\right) = \frac{-98}{9}$$

$$\left(\begin{array}{ccc|c} 1 & -8 & -2 & 1 \\ 0 & 1 & \frac{7}{9} & \frac{1}{3} \\ 0 & 0 & \frac{-98}{9} & \frac{-38}{3} \end{array}\right) \times\left(\frac{-9}{98}\right) \underset{F_3/\left(\frac{-98}{9}\right) \to F_3}{\sim} \left(\begin{array}{ccc|c} 1 & -8 & -2 & 1 \\ 0 & 1 & \frac{7}{9} & \frac{1}{3} \\ 0 & 0 & 1 & \frac{57}{49} \end{array}\right) \times\left(\frac{-7}{9}\right) \underset{F_2 - \left(\frac{7}{9}\right)\cdot F_3 \to F_2}{\sim} \left(\begin{array}{ccc|c} 1 & -8 & -2 & 1 \\ 0 & 1 & 0 & \frac{-4}{7} \\ 0 & 0 & 1 & \frac{57}{49} \end{array}\right) \times(2) \underset{F_1 - (-2)\cdot F_3 \to F_1}{\sim} \left(\begin{array}{ccc|c} 1 & -8 & 0 & \frac{163}{49} \\ 0 & 1 & 0 & \frac{-4}{7} \\ 0 & 0 & 1 & \frac{57}{49} \end{array}\right) \times(8) \underset{F_1 - (-8)\cdot F_2 \to F_1}{\sim}$$

$$\left(\begin{array}{ccc|c} 1 & 0 & 0 & \frac{-61}{49} \\ 0 & 1 & 0 & \frac{-4}{7} \\ 0 & 0 & 1 & \frac{57}{49} \end{array}\right)$$

$$\begin{cases} u = \frac{-61}{49} \\ v = \frac{-4}{7} \\ w = \frac{57}{49} \end{cases} \quad (1)$$

Se puede detallar que cada vez que se hace una operación entre filas no se encuentra la matriz dominante, solo hasta que llegamos a la matriz identidad.

b)Encuentre la matriz de transición por el método Jacobi y determine si converge o no

Código en python:

```python
import numpy as np

A= np.array([[1,-8,-2],
             [1,1,5],
             [3,-1,1]])
B= np.array([[1],
            [4],
            [-2]])

#encontramos la matriz triangular superior
U=(np.triu(A))
#encontramos la matriz triangular inferior
L=(np.tril(A))
print('matriz triangular inferior')
print(L)
print('matriz triangular superior')
print(U)


D=np.array([[1,0,0],
             [0,1,0],
             [0,0,1]])
DI=np.linalg.inv(D)

sum=L+U
print('Suma de L y U')
print(sum)
print('Matriz de transicion')
T=np.matmul(DI,sum)
print(T)
#encontrar valores propios
VP, vectarCaracteristicas=np.linalg.eig(T)
print('Valores propios de la matriz de transicion')
print(VP)
print('Vectores propios de la matriz de transicion')
print(vectarCaracteristicas)

print('valor maximo de los valores propios')
Max=np.amax(VP)
print(Max)
absMax=abs(Max)

if(absMax<1):
    print('CONVERGE')
else:
    print('NO CONVERGE')
```

En la siguiente imagen podemos ver que por medio del método Jacobi no converge.

```
C:\Users\esroj\PycharmProjects\TareaAnalisis\venv\Scripts\python.exe C:/Users/esroj/PycharmProjects/TareaAnalisis/Jaccobi.py
matriz triangular inferior
[[ 1  0  0]
 [ 1  1  0]
 [ 3 -1  1]]
matriz triangular superior
[[ 1 -8 -2]
 [ 0  1  5]
 [ 0  0  1]]
Suma de L y U
[[ 2 -8 -2]
 [ 1  2  5]
 [ 3 -1  2]]
Matriz de transicion
[[ 2. -8. -2.]
 [ 1.  2.  5.]
 [ 3. -1.  2.]]
Valores propios de la matriz de transicion
[ 3.82519395+5.38460759j  3.82519395-5.38460759j -1.6503879 +0.j        ]
Vectores propios de la matriz de transicion
[[ 0.76046196+0.j          0.76046196-0.j         -0.7435195 +0.j        ]
 [-0.22610386-0.41326059j -0.22610386+0.41326059j -0.46049141+0.j        ]
 [ 0.21042015-0.39435226j  0.21042015+0.39435226j  0.48489835+0.j        ]]
valor maximo de los valores propios
(3.825193948805311+5.384607585726803j)
Valor Abosluto del Max
6.6050062527028235
NO CONVERGE
```

c)Compare la solución entre la solución de Jacobi y Gauss Seidel. Utilizar la tolerancia 10-6, genere varias iteraciones

Método Gauss Seidel:

Código:

```
# int(input())input as number of variable to be solved
n = 3
a = []
b = []
# initial solution depending on n(here n=3)
x = [0, 0, 0]
a = [[1,-8,-2], [1,1,5], [3,-1,1]]
b = [1,4,-2]
print(x)

# loop run for m times depending on m the error value
for i in range(0, 100):
    x = seidel(a, x, b)
    # print each time the updated solution
    print(x)
```

```
def seidel(a, x, b):
    # Finding length of a(3)
    n = len(a)
    # for loop for 3 times as to calculate x, y , z
    for j in range(0, n):
        # temp variable d to store b[j]
        d = b[j]

        # to calculate respective xi, yi, zi
        for i in range(0, n):
            if (j != i):
                d -= a[j][i] * x[i]
        # updating the value of our solution
        x[j] = d / a[j][j]
    # returning our updated solution
    return x
```

Solución:

```
C:\Users\Tatis\MetodoGauus\venv\Scripts\python.exe C:/Users/Tatis/Downloads/Gauss_Seidel.py
[0, 0, 0]
[1.0, 3.0, -2.0]
[21.0, -7.0, -72.0]
[-199.0, 563.0, 1158.0]
[6821.0, -12607.0, -33072.0]
[-166999.0, 332363.0, 833358.0]
[4325621.0, -8492407.0, -21469272.0]
[-110877799.0, 218224163.0, 550857558.0]
[2847508421.0, -5601796207.0, -14144321472.0]
[-73103012599.0, 143824619963.0, 363133657758.0]
[1876864275221.0, -3692532564007.0, -9323125389672.0]
[-48186511291399.0, 94802138239763.0, 239361672113958.0]
[1237140450146021.0, -2433948810715807.0, -6145370161153872.0]
[-3.17623308080342e+16, 6.248918161380357e+16, 1.577617403790618e+17]
[8.154658009862409e+17, -1.6043466711757716e+18, -4.050744074134494e+18]
[-2.093626151767516e+19, 4.1189981888347636e+19, 1.039987664413731e+20]
[5.375173879895273e+20, -1.0575112201963928e+21, -2.670063384164975e+21]
[-1.3800216529901093e+22, 2.715053345072597e+22, 6.855118304042925e+22]
[3.5430663368666626e+23, -6.970625488888125e+23, -1.7599824499488113e+24]
[-9.096465291008123e+24, 1.7896377540752178e+25, 4.518577341377655e+25]
[2.3354256715357052e+26, -4.594714342224533e+26, -1.1600991356831648e+27]
[-5.995969745145956e+27, 1.179646542356178e+28, 2.978437465899965e+28]
[1.5394047270649353e+29, -3.0286234600149177e+29, -7.646837641209723e+29]
```

Método Jacobi:

Código:

```python
import numpy as np
def jacobi(A,b,x0,eps=1e-6,n=400):
    D=np.diag(np.diag(A))
    LU=A-D
    x=x0
    for i in range(n):
        D_inv=np.linalg.inv(D)
        x_anterior=x
        x=(D_inv @ (-LU) @ x)+D_inv @ b
        print("Iteracion: ",i," -x: ", x)
        if(np.linalg.norm(x-x_anterior)<eps):
            return(x)
    print("No converge")
    return(x)
A = np.array([[1,-8,-2],
              [1,1,5],
              [3,-1,1]])
b = np.array([1,4,-2])
D=np.diag(np.diag(A))
LU=A-D
x0=np.random.rand(3)
x=jacobi(A,b,x0)
print(x)
```

Solución:

Primeras iteraciones:

```
Iteracion:  0  -x:  [ 6.07517945 -0.78327247 -2.57788559]
Iteracion:  1  -x:  [-10.4219509   10.81424848 -21.00881081]
Iteracion:  2  -x:  [ 45.49636621 119.46600494  40.08010117]
Iteracion:  3  -x:  [1036.88824185 -241.89687206  -19.02309369]
Iteracion:  4  -x:  [-1972.22116387  -937.77277342 -3354.56159762]
Iteracion:  5  -x:  [-14210.30538261  18749.02915196   4976.89071818]
Iteracion:  6  -x:  [159947.01465202 -10670.14820831  61377.9452998 ]
Iteracion:  7  -x:  [  37395.70493314 -466832.74115101 -490513.19216438]
Iteracion:  8  -x:  [-4715687.31353688  2415174.25588877  -579021.85595043]
Iteracion:  9  -x:  [18163351.33520927  7610800.59328904 16562234.1964994 ]
Iteracion:  10  -x:  [ 9.40108741e+07 -1.00974518e+08 -4.68792554e+07]
Iteracion:  11  -x:  [-9.01554656e+08  1.40385407e+08 -3.83007143e+08]
Iteracion:  12  -x:  [3.57068971e+08 2.81659037e+09 2.84504937e+09]
Iteracion:  13  -x:  [ 2.82228217e+10 -1.45823158e+10  1.74538346e+09]
Iteracion:  14  -x:  [-1.13167760e+11 -3.69497390e+10 -9.92507811e+10]
Iteracion:  15  -x:  [-4.94099474e+11  6.09421665e+11  3.02553540e+11]
Iteracion:  16  -x:  [ 5.48048040e+12 -1.01866823e+12  2.09172009e+12]
Iteracion:  17  -x:  [-3.96590564e+12 -1.59390808e+13 -1.74601094e+13]
```

Últimas iteraciones:

```
Iteracion:  86  -x:  [-1.56088371e+65 -5.33694498e+64 -1.38402783e+65]
Iteracion:  87  -x:  [-7.03761165e+65  8.48102288e+65  4.14895664e+65]
Iteracion:  88  -x:  [ 7.61460963e+66 -1.37071715e+66  2.95938578e+66]
Iteracion:  89  -x:  [-5.04696567e+66 -2.24115385e+67 -2.42145460e+67]
Iteracion:  90  -x:  [-2.27721400e+68  1.26119696e+68 -7.27064151e+66]
Iteracion:  91  -x:  [9.94416284e+68 2.64074608e+68 8.09283897e+68]
Iteracion:  92  -x:  [ 3.73116466e+69 -5.04083577e+69 -2.71917424e+69]
Iteracion:  93  -x:  [-4.57650346e+70  9.86470656e+69 -1.62343297e+70]
Iteracion:  94  -x:  [4.64489930e+70 1.26936683e+71 1.47159810e+71]
Iteracion:  95  -x:  [ 1.30981309e+72 -7.82248045e+71 -1.24102957e+70]
Iteracion:  96  -x:  [-6.28280495e+72 -1.24776161e+72 -4.71168731e+72]
Iteracion:  97  -x:  [-1.94054675e+73  2.98412415e+73  1.76006533e+73]
Iteracion:  98  -x:  [ 2.73931238e+74 -6.85977988e+73  8.80576440e+73]
Iteracion:  99  -x:  [-3.72667102e+74 -7.14219458e+74 -8.90391514e+74]
No converge
[-3.72667102e+74 -7.14219458e+74 -8.90391514e+74]
[ 7.12187159e+75 -5.53884413e+75 -1.29417336e+75]
```

Podemos darnos cuenta de que ambos sistemas son equivalentes a la hora de solucionar un ejercicio. Ambos métodos son iterativos, se puede observar que el método Gauss-Seidel converge más rápido debido a que toma un valor inmediato que se calcula en la última iteración. En ambos observamos que se presenta una divergencia debido al sistema dado.

d)Evalúe la matriz de transición del método SOR y determine varias soluciones aproximadas, para 10 valores de w, utilice una tolerancia de 10-16

Código:

```python
import numpy as np
def sor_solver(A, b, omega, initial_guess, convergence_criteria):
    phi = initial_guess[:]
    residual = np.linalg.norm(np.matmul(A, phi) - b) #Initial residual
    while residual > convergence_criteria:
        for i in range(A.shape[0]):
            sigma = 0
            for j in range(A.shape[1]):
                if j != i:
                    sigma += A[i][j] * phi[j]
            phi[i] = (1 - omega) * phi[i] + (omega / A[i][i]) * (b[i] - sigma)
        residual = np.linalg.norm(np.matmul(A, phi) - b)
        print('Residual: {0:10.6g}'.format(residual))
    return phi

# An example case that mirrors the one in the Wikipedia article
residual_convergence = 1e-8
omega = 0.5 #Relaxation factor

A = np.array([[1,-8,-2],
              [1,1,5],
              [3,-1,1]])

b = np.array([[1],[4],[-2]])

initial_guess = np.zeros(3)

phi = sor_solver(A, b, omega, initial_guess, residual_convergence)
print(phi)
```

Solución:

Primeros valores con w=0.5 :

```
SOR ×
C:\Users\esroj\PycharmProjects\TareaAnalisis\venv\Scripts\python.exe C:/Users/esroj/PycharmProjects/TareaAnalisis/venv/SOR.py
Residual:    24.1771
Residual:    86.8119
Residual:    477.305
Residual:       2794
Residual:    17183.7
Residual:     105983
Residual:     655517
Residual: 4.05333e+06
Residual: 2.50704e+07
Residual: 1.5505e+08
Residual: 9.58962e+08
Residual: 5.93094e+09
Residual: 3.66816e+10
Residual: 2.26868e+11
Residual: 1.40313e+12
Residual: 8.67803e+12
Residual: 5.36718e+13
Residual: 3.31948e+14
Residual: 2.05303e+15
Residual: 1.26975e+16
Residual: 7.85314e+16
Residual:  4.857e+17
Residual: 3.00395e+18
Residual: 1.85787e+19
Residual: 1.14905e+20
Residual: 7.10665e+20
Residual: 4.39531e+21
```

Últimos valores para w=0.5:

```
SOR ×
Residual: 6.34418e+134
Residual: 3.92374e+135
Residual: 2.42675e+136
Residual: 1.50089e+137
Residual: 9.28268e+137
Residual: 5.74113e+138
Residual: 3.55076e+139
Residual: 2.19607e+140
Residual: 1.35822e+141
Residual: 8.4003e+141
Residual: 5.19541e+142
Residual: 3.21324e+143
Residual: 1.98732e+144
Residual: 1.22912e+145
Residual: 7.60181e+145
Residual: 4.70155e+146
Residual: 2.90781e+147
Residual: 1.79842e+148
Residual: 1.11228e+149
Residual: 6.87921e+149
Residual: 4.25464e+150
Residual: 2.6314e+151
Residual: 1.62747e+152
Residual: 1.00655e+153
Residual: 6.22531e+153
Residual:        inf
Residual:        inf
Residual:        inf
```

Primeros valores con w=0.6 :

```
C:\Users\esroj\PycharmProjects\TareaAnalisis\venv\Scripts\python.exe C:/Users/esroj/PycharmProjects/TareaAnalisis/venv/SOR.py
Residual:    27.3815
Residual:    155.969
Residual:    1253.25
Residual:    11481.2
Residual:     104234
Residual:     955656
Residual: 8.73806e+06
Residual: 7.99766e+07
Residual: 7.3176e+08
Residual: 6.69611e+09
Residual: 6.12717e+10
Residual: 5.60664e+11
Residual: 5.13032e+12
Residual: 4.69446e+13
Residual: 4.29564e+14
Residual: 3.93069e+15
Residual: 3.59676e+16
```

Últimos valores para w=0.6:

```
Residual: 2.92559e+143
Residual: 2.67704e+144
Residual: 2.44961e+145
Residual: 2.2415e+146
Residual: 2.05107e+147
Residual: 1.87682e+148
Residual: 1.71737e+149
Residual: 1.57147e+150
Residual: 1.43796e+151
Residual: 1.3158e+152
Residual: 1.20401e+153
Residual: 1.10172e+154
Residual:        inf
Residual:        inf
Residual:        inf
Residual:        inf
```

Primeros valores con w=0.9 :

```
C:\Users\esroj\PycharmProjects\TareaAnalisis\venv\Scripts\python.exe C:/Users/esroj/PycharmProjects/TareaAnalisis/venv/SOR.py
Residual:    35.5419
Residual:    532.607
Residual:    9799.69
Residual:     207848
Residual: 4.30932e+06
Residual: 8.98054e+07
Residual: 1.86958e+09
Residual: 3.89295e+10
Residual: 8.10578e+11
Residual: 1.68777e+13
Residual: 3.51425e+14
Residual: 7.31732e+15
Residual: 1.5236e+17
Residual: 3.17241e+18
Residual: 6.60554e+19
Residual: 1.37539e+21
Residual: 2.86382e+22
```

Últimos valores para w=0.9:

```
Residual: 1.32937e+141
Residual: 2.76799e+142
Residual: 5.76347e+143
Residual: 1.20006e+145
Residual: 2.49874e+146
Residual: 5.20284e+147
Residual: 1.08333e+149
Residual: 2.25569e+150
Residual: 4.69675e+151
Residual: 9.77949e+152
Residual:          inf
Residual:          inf
Residual:          inf
```

Primeros valores con w=1.2 :

```
C:\Users\esroj\PycharmProjects\TareaAnalisis\venv\Scripts\python.exe C:/Users/esroj/PycharmProjects/TareaAnalisis/venv/SOR.py
Residual:    41.7623
Residual:    1123.25
Residual:    36608.5
Residual: 1.37201e+06
Residual: 5.04841e+07
Residual: 1.86283e+09
Residual: 6.87088e+10
Residual: 2.53442e+12
Residual: 9.34845e+13
Residual: 3.44827e+15
Residual: 1.27193e+17
Residual: 4.69165e+18
Residual: 1.73057e+20
```

Últimos valores para w=1.2:

```
Residual: 2.08788e+139
Residual: 7.70135e+140
Residual: 2.84072e+142
Residual: 1.04783e+144
Residual: 3.86503e+145
Residual: 1.42566e+147
Residual: 5.25868e+148
Residual: 1.93972e+150
Residual: 7.15486e+151
Residual: 2.63914e+153
Residual:        inf
Residual:        inf
Residual:        inf
```

Primeros valores con w=1.5 :

```
Residual:    47.6501
Residual:    1711.98
Residual:    85570.2
Residual: 5.00719e+06
Residual: 2.88116e+08
Residual: 1.66111e+10
Residual: 9.57478e+11
Residual: 5.51914e+13
Residual: 3.18136e+15
Residual: 1.83381e+17
Residual: 1.05705e+19
Residual: 6.09307e+20
Residual: 3.51218e+22
Residual: 2.0245e+24
Residual: 1.16697e+26
```

Últimos valores para w=1.5:

```
Residual: 6.27287e+145
Residual: 3.61582e+147
Residual: 2.08424e+149
Residual: 1.20141e+151
Residual: 6.92518e+152
Residual:        inf
Residual:        inf
Residual:        inf
```

Este método nos va permitir mejorar la convergencia usando relajación. La relajación representa una ligera modificación del método de Gauss-Seidel y ésta permite mejorar la convergencia en algunos casos. Después de que se calcula cada nuevo valor de x, ése valor se modifica mediante un promedio ponderado de los resultados de las iteraciones anterior y actual.

e)Construya una función f(w) que determine el valor óptimo de w para que el método SOR converja

Mientras mayor es w el número de iteraciones es cada menor; por eso es importante conocer el valor óptimo de w. El valor óptimo es con el que se consiguen menos iteraciones para la convergencia.

```
import numpy as np
 #Creamos la matriz A de coeficientes
A = [[1, -8, -2], [1, 1, 5], [3, -1, 1]]
# Creamos el vector de términos independientes

b = [1, 4, -2]
# Valores iniciales
tol = 0.00001  # valor de la tolerancia establecida
error = 1000000000000000000000000  # valor inicial de la norma
kmax = 100  # número máximo de iteraciones
n = len(A)  # determina tamaño de A
x = np.zeros(n)  # vector x inicial de ceros
x1 = np.copy(x)  # vector auxiliar de x
k = 1  # contador inicial de iteraciones
error1 = [0]
lambd = 1.4

while k < kmax and error > tol:
    print("iteración: ", k)
    for i in range(n):
        sumatoria = 0
        for j in range(n):
```

```
while k < kmax and error > tol:
    print("iteración: ", k)
    for i in range(n):
        sumatoria = 0
        for j in range(n):
            if i != j:
                sumatoria += (A[i][j] * x[j])
        x[i] = (lambd * ((b[i] - sumatoria) / A[i][i])) + (1 - lambd) * x[i]
        if(i==2):
            print("Valor w",  x[i])

    error = np.linalg.norm(x - x1)  # cálculo de la norma vectorial
    error1.append(error)  # se genera el vector error para ser graficado
    x1 = np.copy(x)  # actualiza vector solución iteración anterior

   # print("  error: ", error1[k])
    print()

    k += 1
```

```
iteración:  99
Valor w 1.0112994900374896e+167
```

1.ii

$$\text{ii.}\quad \begin{aligned} u + 4v &= 5 \\ v + w &= 2 \\ 2u + 3w &= 0 \end{aligned}$$

a)Es la matriz A de coeficientes diagonales dominante? se puede reorganizar con operaciones entre filas para que sea diagonal dominante?

```
MetodoGauus C:\Users\Tatis\MetodoGauus
venv library root
GaussJordanContadorDeOperaciones.py
xternal Libraries
cratches and Consoles
```

```
import numpy as np
def es_diagonal_dominante(x):
    abs_x=np.abs(x)
    return np.all(2*np.diag(abs_x)>=np.sum(abs_x,axis=1))

A=np.array([[1,4,0], [0,1,1], [2,0,3]])
print(es_diagonal_dominante(A))
```

```
NUEVO
C:\Users\Tatis\MetodoGauus\venv\Scripts\python.exe C:/Users/Tatis/Downloads/NUEVO.py
False
```

Podemos notar que no es la matriz dominante debido a que si empezamos a sumar y le sacamos valor absoluto los valores que están al lado de los números de los coeficientes de la diagonal podemos notar que estos siempre tienden a ser mayores.

$$\left(\begin{array}{ccc|c} 1 & 4 & 0 & 5 \\ 0 & 1 & 1 & 2 \\ 2 & 0 & 3 & 0 \end{array}\right) \times(-2) \underset{F_3 - 2\cdot F_1 \to F_3}{\sim} \left(\begin{array}{ccc|c} 1 & 4 & 0 & 5 \\ 0 & 1 & 1 & 2 \\ 0 & -8 & 3 & -10 \end{array}\right) \times(8) \underset{F_3 - (-8)\cdot F_2 \to F_3}{\sim} \left(\begin{array}{ccc|c} 1 & 4 & 0 & 5 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 11 & 6 \end{array}\right) \times\left(\frac{1}{11}\right) \underset{F_3/(11) \to F_3}{\sim} \left(\begin{array}{ccc|c} 1 & 4 & 0 & 5 \\ 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & \frac{6}{11} \end{array}\right) \times(-1)$$

$$\underset{F_2 - 1\cdot F_3 \to F_2}{\sim} \left(\begin{array}{ccc|c} 1 & 4 & 0 & 5 \\ 0 & 1 & 0 & \frac{16}{11} \\ 0 & 0 & 1 & \frac{6}{11} \end{array}\right) \times(-4) \underset{F_1 - 4\cdot F_2 \to F_1}{\sim} \left(\begin{array}{ccc|c} 1 & 0 & 0 & \frac{-9}{11} \\ 0 & 1 & 0 & \frac{16}{11} \\ 0 & 0 & 1 & \frac{6}{11} \end{array}\right)$$

$$\left\{\begin{array}{lcl} u & = & \frac{-9}{11} \\ v & = & \frac{16}{11} \\ w & = & \frac{6}{11} \end{array}\right. \quad (1)$$

En la imagen anterior podemos darnos cuenta que al igual que en el ejercicio anterior necesitaríamos obtener la matriz identidad para que se diera la diagonal dominante.

b)Encuentre la matriz de transición por el método Jacobbi y determine si converge o no
Utilizamos el código del punto anterior y este fue nuestro resultado:

```
matriz triangular inferior
[[1 0 0]
 [0 1 0]
 [2 0 3]]
matriz triangular superior
[[1 4 0]
 [0 1 1]
 [0 0 3]]
Suma de L y U
[[2 4 0]
 [0 2 1]
 [2 0 6]]
Matriz de transicion
[[2. 4. 0.]
 [0. 2. 1.]
 [2. 0. 6.]]
Valores propios de la matriz de transicion
[6.41113886+0.j        1.79443057+1.3309139j 1.79443057-1.3309139j]
```

```
Valores propios de la matriz de transicion
[6.41113886+0.j        1.79443057+1.3309139j 1.79443057-1.3309139j]
Vectores propios de la matriz de transicion
[[-0.19657085+0.j         -0.87074495+0.j         -0.87074495-0.j        ]
 [-0.21677533+0.j          0.04474964-0.28972164j  0.04474964+0.28972164j]
 [-0.9562261 +0.j          0.3763954 +0.11911582j  0.3763954 -0.11911582j]]
valor maximo de los valores propios
(6.411138860801188+0j)
Valor Abosluto del Max
6.411138860801188
NO CONVERGE
```

Deducimos que el sistema de ecuación no converge.

c)Compare la solución entre la solución de Jacobi y Gauss Seidel. Utilizar la tolerancia 10-6, genere varias iteraciones

Para esto se utilizan los códigos que se mostraron en el anterior ejercicio

Método Jacobi:

Primeras iteraciones:

```
Iteracion:  0  -x:  [[ 3.47784632  4.31485728  4.44413339]
 [ 0.47784632  1.31485728  1.44413339]
 [-1.52215368 -0.68514272 -0.55586661]]
Iteracion:  1  -x:  [[ 3.08861472 -0.25942913 -0.77653355]
 [ 3.52215368  2.68514272  2.55586661]
 [-2.31856421 -2.87657152 -2.96275559]]
Iteracion:  2  -x:  [[-9.08861472 -5.74057087 -5.22346645]
 [ 4.31856421  4.87657152  4.96275559]
 [-2.05907648  0.17295275  0.51768903]]
Iteracion:  3  -x:  [[-12.27425685 -14.50628608 -14.85102237]
 [  4.05907648   1.82704725   1.48231097]
 [  6.05907648   3.82704725   3.48231097]]
Iteracion:  4  -x:  [[-11.23630592  -2.30818899  -0.92924386]
 [ -4.05907648  -1.82704725  -1.48231097]
 [  8.1828379    9.67085739   9.90068158]]
Iteracion:  5  -x:  [[21.23630592 12.30818899 10.92924386]
 [-6.1828379  -7.67085739 -7.90068158]
 [ 7.49087061  1.53879266  0.61949591]]
Iteracion:  6  -x:  [[ 29.73135161  35.68342956  36.60272631]
 [ -5.49087061   0.46120734   1.38050409]
```

Últimas iteraciones:

```
Iteracion:  95  -x:  [[ 5.43836293e+13  9.86968387e+13  5.11119961e+13]
 [-5.89573412e+13 -5.15718063e+13 -5.95026134e+13]
 [-1.03874101e+13  1.91547295e+13 -1.25684990e+13]]
Iteracion:  96  -x:  [[ 2.35829365e+14  2.06287225e+14  2.38010454e+14]
 [ 1.03874101e+13 -1.91547295e+13  1.25684990e+13]
 [-3.62557529e+13 -6.57978925e+13 -3.40746640e+13]]
Iteracion:  97  -x:  [[-4.15496406e+13  7.66189179e+13 -5.02739959e+13]
 [ 3.62557529e+13  6.57978925e+13  3.40746640e+13]
 [-1.57219576e+14 -1.37524817e+14 -1.58673636e+14]]
Iteracion:  98  -x:  [[-1.45023012e+14 -2.63191570e+14 -1.36298656e+14]
 [ 1.57219576e+14  1.37524817e+14  1.58673636e+14]
 [ 2.76997604e+13 -5.10792786e+13  3.35159973e+13]]
Iteracion:  99  -x:  [[-6.28878306e+14 -5.50099267e+14 -6.34694543e+14]
 [-2.76997604e+13  5.10792786e+13 -3.35159973e+13]
 [ 9.66820077e+13  1.75461047e+14  9.08657708e+13]]
No converge
[[-6.28878306e+14 -5.50099267e+14 -6.34694543e+14]
 [-2.76997604e+13  5.10792786e+13 -3.35159973e+13]
 [ 9.66820077e+13  1.75461047e+14  9.08657708e+13]]
```

Método Gauss Seidel:
Primeras iteraciones:

```
[0, 0, 0]
[5.0, 2.0, -3.3333333333333335]
[-3.0, 5.333333333333334, 2.0]
[-16.333333333333336, 0.0, 10.888888888888891]
[5.0, -8.888888888888891, -3.3333333333333335]
[40.555555555555564, 5.333333333333334, -27.03703703703704]
[-16.333333333333336, 29.03703703703704, 10.888888888888891]
[-111.14814814814817, -8.888888888888891, 74.09876543209877]
[40.555555555555564, -72.09876543209877, -27.03703703703704]
[293.3950617283951, 29.03703703703704, -195.59670781893007]
[-111.14814814814817, 197.59670781893007, 74.09876543209877]
[-785.3868312757203, -72.09876543209877, 523.5912208504802]
[293.3950617283951, -521.5912208504802, -195.59670781893007]
[2091.364883401921, 197.59670781893007, -1394.2432556012807]
[-785.3868312757203, 1396.2432556012807, 523.5912208504802]
[-5579.973022405123, -521.5912208504802, 3719.9820149367483]
[2091.364883401921, -3717.9820149367483, -1394.2432556012807]
[14876.928059746993, 1396.2432556012807, -9917.952039831329]
[-5579.973022405123, 9919.952039831329, 3719.9820149367483]
[-39674.808159325316, -3717.9820149367483, 26449.872106216877]
[14876.928059746993, -26447.872106216877, -9917.952039831329]
```

Últimas iteraciones:

```
[-2.3854364300054006e+17, -2.236346653130063e+16, 1.5902909533369338e+17]
[8.945386612520253e+16, -1.5902909533369338e+17, -5.963591075013501 6e+16]
[6.361163813347735e+17, 5.9635910750135016e+16, -4.2407758755651565e+17]
[-2.3854364300054006e+17, 4.2407758755651565e+17, 1.5902909533369338e+17]
[-1.6963103502260626e+18, -1.5902909533369338e+17, 1.1308735668173751e+18]
[6.361163813347735e+17, -1.1308735668173751e+18, -4.2407758755651565e+17]
[4.5234942672695004e+18, 4.2407758755651565e+17, -3.0156628448463334e+18]
[-1.6963103502260626e+18, 3.0156628448463334e+18, 1.1308735668173751e+18]
[-1.2062651379385334e+19, -1.1308735668173751e+18, 8.041767586256889e+18]
[4.5234942672695004e+18, -8.041767586256889e+18, -3.0156628448463334e+18]
[3.2167070345027555e+19, 3.0156628448463334e+18, -2.1444713563351704e+19]
[-1.2062651379385334e+19, 2.1444713563351704e+19, 8.041767586256889e+18]
[-8.577885425340681e+19, -8.041767586256889e+18, 5.7185902835604546e+19]
[3.2167070345027555e+19, -5.7185902835604546e+19, -2.1444713563351704e+19]
[2.2874361134241818e+20, 2.1444713563351704e+19, -1.5249574089494544e+20]
[-8.577885425340681e+19, 1.5249574089494544e+20, 5.7185902835604546e+19]
[-6.099829635797818e+20, -5.7185902835604546e+19, 4.0665530905318785e+20]
[2.2874361134241818e+20, -4.0665530905318785e+20, -1.5249574089494544e+20]
[1.6266212362127514e+21, 1.5249574089494544e+20, -1.0844141574751676e+21]
[-6.099829635797818e+20, 1.0844141574751676e+21, 4.0665530905318785e+20]
[-4.3376566299006704e+21, -4.0665530905318785e+20, 2.891771086600447e+21]
[1.6266212362127514e+21, -2.891771086600447e+21, -1.0844141574751676e+21]
```

El método de Gauss-Seidel utiliza los valores obtenidos de las incógnitas de iteraciones pasadas para despejar las actuales y en el de Jacobi obtiene los valores en una iteración y los utiliza hasta en la siguiente para volver a obtener los valores más aproximados. Es por esto que el método de Gauss Seidel converge más rápido.

d)Evalúe la matriz de transición del método SOR y determine varias soluciones aproximadas, para 10 valores de w, utilice una tolerancia de 10-16

Valores con w=0.5 :

```
C:\Users\esroj\PycharmProjects\TareaAnalisis\venv\Scripts\python.exe C:/Users/esroj/PycharmProjects/TareaAnalisis/venv/SOR.py
Residual:    10.2103
Residual:    14.3047
Residual:    15.5614
Residual:    13.9617
Residual:    10.9363
Residual:    8.23374
Residual:    6.59362
Residual:    6.66524
Residual:    8.92945
Residual:    11.5801
Residual:    12.8125
Residual:    12.1371
Residual:    10.2185
Residual:    8.17127
Residual:    6.84438
Residual:    6.83561
Residual:    8.24697
```

Valores con w=0.6 :

```
C:\Users\esroj\PycharmProjects\TareaAnalisis\venv\Scripts\python.exe C:/Users/esroj/PycharmProjects/TareaAnalisis/venv/SOR.py
Residual:         12
Residual:    16.8524
Residual:    15.8819
Residual:    11.7139
Residual:    9.31535
Residual:    7.37351
Residual:    8.76541
Residual:    14.4784
Residual:    16.5821
Residual:     13.634
Residual:    10.0536
Residual:    8.17461
Residual:    6.94278
Residual:    11.1925
Residual:    15.6704
Residual:    15.2227
Residual:    11.4833
Residual:    8.87078
```

Valores con w=0.9 :

```
C:\Users\esroj\PycharmProjects\TareaAnalisis\venv\Scripts\python.exe C:/Users/esroj/PycharmProjects/TareaAnalisis/venv/SOR.py
Residual:    18.4046
Residual:    25.7945
Residual:     19.325
Residual:    39.7423
Residual:    36.7487
Residual:    96.6214
Residual:    55.5554
Residual:    180.059
Residual:    107.111
Residual:    362.369
Residual:     258.03
Residual:    662.555
Residual:     617.69
Residual:    1197.23
Residual:    1480.21
Residual:    2036.86
Residual:    3341.42
Residual:    3340.64
Residual:    7262.85
Residual:    5314.53
Residual:    15105.4
Residual:    8948.45
Residual:    30263.9
Residual:    18124.7
```

Valores con w=1.2 :

```
C:\Users\esroj\PycharmProjects\TareaAnalisis\venv\Scripts\python.exe C:/Users/esroj/PycharmProjects/TareaAnalisis/venv/SOR.py
Residual:    26.4658
Residual:    43.4416
Residual:    105.068
Residual:    154.226
Residual:    565.949
Residual:    628.554
Residual:     2697.3
Residual:    3093.78
Residual:    12291.4
Residual:    17562.2
Residual:    52704.3
Residual:     100912
Residual:     213273
Residual:     552258
Residual:     826002
Residual: 2.84992e+06
Residual: 3.28929e+06
Residual: 1.38756e+07
Residual: 1.54962e+07
Residual: 6.37954e+07
Residual: 8.66812e+07
Residual: 2.76745e+08
Residual: 5.00425e+08
Residual: 1.1319e+09
Residual: 2.76832e+09
```

Valores con w=1.5 :

```
C:\Users\esroj\PycharmProjects\TareaAnalisis\venv\Scripts\python.exe C:/Users/esroj/PycharmProjects/TareaAnalisis/venv/SOR.py
Residual:     36.773
Residual:    87.3162
Residual:    328.478
Residual:    752.124
Residual:    3290.16
Residual:    7900.05
Residual:    28932.7
Residual:    90702.1
Residual:     238732
Residual:     972000
Residual: 2.15533e+06
Residual: 9.31205e+06
Residual: 2.36311e+07
Residual: 8.01638e+07
Residual: 2.69146e+08
Residual: 6.63368e+08
Residual: 2.81295e+09
Residual: 6.28079e+09
Residual: 2.62264e+10
Residual: 7.0749e+10
Residual: 2.21447e+11
Residual: 7.94532e+11
Residual: 1.85835e+12
Residual: 8.08754e+12
Residual: 1.85079e+13
Residual: 7.34324e+13
```

e)Construya una función f(w) que determine el valor óptimo de w para que el método SOR converja

Como lo explicamos en el ejercicio 1.i para determinar el valor de w debemos hallar el valor que tenga menos iteraciones(el código se encuentra en el punto anterior). Para este caso w es:

```
iteración:  9
Valor w 1 0.5715695318317555
```

8.i

Dados los sistemas del punto 1, evaluar el error hacia atrás, hacia delante y el número de condición cuando el sistema se soluciona por el método de <u>Gauss con pivoteo parcial</u>:

Debemos enfatizar una vez más que si los errores con que conocemos los datos de entrada son grandes, por mucho que utilicemos un algoritmo perfecto no vamos a obtener un resultado con una precisión mejor. Por ello, es importante estudiar cómo se propagan los errores y cuáles son sus posibles fuentes durante la computación, con el objeto de minimizarlos o al menos acotarlos

Codigo:

```python
import numpy as np
# Método de Gauss-Jordan
# Solución a Sistemas de Ecuaciones
# de la forma A.X=B
import struct

import numpy as np

# INGRESO
A = np.array([[1, -8,  -2],
              [1, 1, 5],
              [3, -1, 1]])

B = np.array([[1],
              [4],
              [-2]])

# PROCEDIMIENTO
casicero = 1e-15  # Considerar como 0

# Evitar truncamiento en operaciones
A = np.array(A, dtype=float)

# Matriz aumentada
AB = np.concatenate((A, B), axis=1)
AB0 = np.copy(AB)

# Pivoteo parcial por filas
tamano = np.shape(AB)
n = tamano[0]
m = tamano[1]

contadorOperaciones = 0

# Para cada fila en AB
for i in range(0, n - 1, 1):
    # columna desde diagonal i en adelante
    columna = abs(AB[i:, i])
    dondemax = np.argmax(columna)

    # dondemax no está en diagonal
    if (dondemax != 0):
        # intercambia filas
        temporal = np.copy(AB[i, :])
        AB[i, :] = AB[dondemax + i, :]
        AB[dondemax + i, :] = temporal
        contadorOperaciones += 3

AB1 = np.copy(AB)

# eliminacion hacia adelante
for i in range(0, n - 1, 1):
    pivote = AB[i, i]
    adelante = i + 1
    for k in range(adelante, n, 1):
        factor = AB[k, i] / pivote
        AB[k, :] = AB[k, :] - AB[i, :] * factor
        contadorOperaciones += 3
AB2 = np.copy(AB)

# elimina hacia atras
ultfila = n - 1
ultcolumna = m - 1
for i in range(ultfila, 0 - 1, -1):
    pivote = AB[i, i]
    atras = i - 1
    for k in range(atras, 0 - 1, -1):
        factor = AB[k, i] / pivote
        AB[k, :] = AB[k, :] - AB[i, :] * factor
        contadorOperaciones += 3
    # diagonal a unos
    AB[i, :] = AB[i, :] / AB[i, i]
X = np.copy(AB[:, ultcolumna])
X = np.transpose([X])

print('solución de X: ')
print(X)
print('-------------------- ')
aux=np.copy(X)

resulExact = np.array([[-61/49],
                       [-4/7],
                       [57/49]])

resta=np.subtract(resulExact,aux)
errorAdelante=np.linalg.norm(resta)
print('Error hacia adelante : ', errorAdelante)

multiMatri=np.matmul(A,aux)
resta2=np.subtract(B,multiMatri)
errorAtras=np.linalg.norm(resta2)
print("Error hacia atras: ", errorAtras)

#Numero de condicion proceso
# creacion de la matriz
m = np.copy(AB)
# funcion que nos permite encontrar el nuemro de condicion
numero_condicion = np.linalg.cond(m)
print("Numero de condicion: " + str(numero_condicion))
```

Por medio de una calculadora pudimos hallar la solución exacta:

Resultado:

$$\begin{cases} x_1 = -\dfrac{61}{49} \\ x_2 = -\dfrac{4}{7} \\ x_3 = \dfrac{57}{49} \end{cases}$$

Y así encontrar los errores y el número de condición:

```
C:\Users\esroj\PycharmProjects\TareaAnalisis\venv\Scripts\python.exe C:/Users/esroj/PycharmProjects/TareaAnalisis/errorA_NumCond.py
solución de X:
[[-1.24489796]
 [-0.57142857]
 [ 1.16326531]]
-------------------
Error hacia adelante :  2.482534153247273e-16
Error hacia atras:  9.930136612989092e-16
Numero de condicion: 2.0565718352279188

Process finished with exit code 0
```

9. Dado un sistema cualquiera que está asociada a una matriz dispersa con n=10000 implemente el método del gradiente conjugado para resolver el problema.
Código:

```python
from numpy import *
import numpy
import numpy as np

def f(A,x,b):
    return 0.5*np.dot(np.dot(x,A),x)+np.dot(b,x)
def  g(A,x,b):
    return np.dot(A,x)+b
#
def minimize_cg(A,b, x0, jac=None,gtol=1e-5,maxiter=None,disp=False):
    if maxiter is None:
        maxiter = len(x0) * 200
    gfk = np.dot(A, x0) + b
    k = 0
    xk = x0
    warnflag = 0
    pk = -gfk

    gnorm = numpy.amax(numpy.abs(gfk))
    while (gnorm > gtol) and (k < maxiter):
        deltak = numpy.dot(gfk, gfk)
        alpha_k = -np.dot(gfk, pk) / (np.dot(np.dot(pk,A.T ),pk))
        xk = xk + alpha_k * pk
        gfkp1=np.dot(A, xk) + b
```

```
gnorm = numpy.amax(numpy.abs(gfk))
while (gnorm > gtol) and (k < maxiter):
    deltak = numpy.dot(gfk, gfk)
    alpha_k = -np.dot(gfk, pk) / (np.dot(np.dot(pk,A.T),pk))
    xk = xk + alpha_k * pk
    gfkp1=np.dot(A, xk) + b
    beta_k = max(0, numpy.dot(gfkp1, gfkp1) / deltak)
    pk = -gfkp1 + beta_k * pk
    gfk = gfkp1
    gnorm=numpy.amax(numpy.abs(gfk))#Valor máximo como norma
    k += 1
if warnflag == 0:
    # msg = _status_message['success']
    if disp:
        print('success')
        print("         Vlor de la función Actual: ", xk )
        print("         Iteraciones: %d" % k)
        print("         Valor de la función: ", f(A,xk,b))
        print("         Valor actual de x*: ",-np.dot(np.linalg.inv(A),b))
        print("         Valor de la función actual f(x*): ",f(A,-np.dot(np.linalg.inv(A),b),b))
```

Solución:

```
C:\Users\Tatis\pythonProject2222\venv\Scripts\python.exe C:/Users/Tatis/
success
         Valor de la función Actual:  [-0.66666667 -0.33333333]
         Iteraciones: 2
         Valor de la función:  -0.3333333333333333
         Valor actual de x*:  [-0.66666667 -0.33333333]
         Valor de la función actual f(x*):  -0.3333333333333333
```