ANÁLISIS NUMÉRICO

Taller 2

David Andrés Ramírez
Juan Felipe Arias C.

PUNTO 1 II

1. Dado el sistema:

i. $$\begin{aligned} u - 8v - 2w &= 1 \\ u + v + 5w &= 4 \\ 3u - v + w &= -2 \end{aligned}$$

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

iii. $$\begin{aligned} u + 3v - w &= 18 \\ 4u - v + w &= 27.34 \\ u + v + 7w &= 16.2 \end{aligned}$$

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

b) Encuentre la matriz de transición por el método de Jacobi y determine si el método converge.

c) Compare la solución entre la solución de Jacobi y Gauss Seidel. Utilice una tolerancia de 10^{-6}, genere varias iteraciones

d) Evalue la matriz de transición del métdo **SOR** y determine varias soluciones aproximadas, para 10 valores de ω. Utilice una tolerancia de 10^{-16}

e) Construya una función $f(\omega)$ que determine el valor óptimo de ω para que el método**SOR** converja

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

```python
def convergencia(soluciones_temp, soluciones, indice, cantidad_ecu, decimales):

    # Se calcula si la incognita actual cumple con la convergencia.
    flag = (abs((soluciones_temp[indice] - soluciones[indice])) / soluciones_temp[indice]) < (1 * 10 ** (0 - decimales))

    # Si la incognita actual es la ultima en el sistema de ecuaciones, solo retorna el resultado booleano.
    if indice == (cantidad_ecu - 1):
        return flag

    # Si hay mas incognitas en el sistema de ecuaciones, calcula su convergencia y decide si ambas convergen o no.
    else:
        flag_temp = convergencia(soluciones_temp, soluciones, indice + 1, cantidad_ecu, decimales)
        flag = flag_temp and flag
        return flag
```

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

```python
def revisar_matriz(matriz):
    '''
    Funcion encargada de revisar si la matriz del sistema de ecuaciones es diagonalmente dominante.
    param matriz: list, es el arreglo de dos dimensiones que contiene los valores ceficientes del sistema de ecuaciones.
    return bool, indica si la matriz del sistema de ecuaciones es dominante.
    '''
    flag = True
    for i in range(len(matriz)):
        for j in range(len(matriz)):
            if i != j:
                if abs(matriz[i][i]) < abs(matriz[i][j]):
                    flag = False
    return flag
```

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

```python
# Inicio del programa.
print("Bienvenido a GaussSei. Este programa resuelve un sistema de ecuaciones utilizando el metodo de Gauss-Seidel.")

# Se obtiene un numero correcto de ecuaciones.
cantidad_ecu = 0
while cantidad_ecu < 1:
    cantidad_ecu = int(input("\nEscribe el numero de ecuaciones (debe ser igual al numero de incognitas): "))
    if cantidad_ecu < 1:
        print("El valor debe ser mayor a 0. Intentalo de nuevo.")

# Inicializa dos vectores vacios. El primero contendra la matriz de coeficientes del sistema de ecuaciones, mientra que el segundo contendra las igualdades de cada ecuacion.
matriz = []
vector = []

# Se obtiene toda la informacion del sistema de ecuaciones y la almacena de forma correcta en ambos vectores.
for i in range(cantidad_ecu):
    print("\nEn la ecuacion " + str(i + 1) + ": ")
    fila = []
    for j in range(cantidad_ecu):
        fila.append(float(input("\tEscribe el coeficiente de la incognita " + str(j + 1) + ": ")))
    matriz.append(fila)
    vector.append(float(input("\tEscribe la igualdad de la ecuacion: ")))
```

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

```python
# Se revisa si la matriz del sistema de ecuaciones es diagonalmente dominante.
try:
    if revisar_matriz(matriz) == False:

        # Si la matriz no es diagonalmente independiente, se intercambian las filas para intentar obtener una matriz diagonalmente dominante.
        for i in range(len(matriz)):
            for j in range(i, len(matriz)):

                # El intercambio se hace buscando el valor mas alto para cada variable en cada fila de la matriz
                if abs(matriz[i][i]) < abs(matriz[j][i]):
                    fila = matriz[i][:]
                    matriz[i] = matriz[j][:]
                    matriz[j] = fila[:]

                    # Ademas de hacer el intercambio de filas en la matriz, se intercambian los valores en el vector de soluciones.
                    temp = vector[i]
                    vector[i] = vector[j]
                    vector[j] = temp

        # Si despues del arreglo de la matriz, esta continua siendo no diagonalmente dominante, entonces termina el programa.
        # El motivo es debido a que en esta instancia no se puede asegurar la convergencia del sistema.
        if revisar_matriz(matriz) == False:
            raise ValueError
except ValueError:
    print(
        "\nError: El sistema no es diagonalmente dominante. El programa ha terminado porque no se puede asegurar convergencia del metodo en esta instancia.")
```

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

```python
except ValueError:
    print(
        "\nError: El sistema no es diagonalmente dominante. El programa ha terminado porque no se puede asegurar convergencia del metodo en esta instancia.")

# Si la matriz del sistema de ecuaciones es diagonalmente dominante, continua con el programa.
else:

    # Se obtiene el numero de decimales deseado.
    decimales = int(input("\nEscribe el numero de aproximacion de decimales: "))

    # Inicializa dos vectores con la solucion trivial '0' para cada incognita en el sistema de ecuaciones.
    soluciones = []
    for i in range(cantidad_ecu):
        soluciones.append(0)
    soluciones_temp = []
    for i in range(cantidad_ecu):
        soluciones_temp.append(0)

    # Inicializa las variables para controlar las iteraciones del programa.
    iteraciones = 0
    convergio = False

    # Itera el siguiente codigo hasta que los valores de cada incognita convergan en la precision deseada.
    while convergio == False:

        # Se elige una incognita con la cual trabajar, y se inicializa la suma del despeje.
        for i in range(cantidad_ecu):
            suma = 0

            # El siguiente codigo realiza el despeje de la incognita actual.
            for j in range(cantidad_ecu):
                if (i != j):

                    # Calcula la suma dependiendo del signo del coeficiente de la incognita actual.
                    if matriz[i][i] < 0:
                        suma = suma + (matriz[i][j] * soluciones_temp[j])

                    else:
                        suma = suma + (-1 * (matriz[i][j]) * soluciones_temp[j])

            # Termina de calcular el despeje, y toma en cuenta los signos del coeficiente de la incognita actual.
```

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

```
        # Termina de calcular el despeje, y toma en cuenta los signos del coeficiente de la incognita actual.
        if matriz[i][i] < 0:
            suma = (suma + (-1 * vector[i])) / (-1 * matriz[i][i])
        else:
            suma = (suma + vector[i]) / (matriz[i][i])

        # Asigna el resultado del despeje al espacio correspondiente de la incognita actual en el vector de soluciones temporal.
        soluciones_temp[i] = suma

    # Calcula el error relativo entre el valor de las incognitas en la iteracion actual y la iteracion anteriror.
    # Si es menor al criterio de paro, termina las iteraciones.
    iteraciones = iteraciones + 1
    indice = 0
    if convergencia(soluciones_temp, soluciones, indice, cantidad_ecu, decimales) == True:
        convergio = True

    # En caso de que no haya convergencia, asigna los valores del vector de soluciones temporal al vector de soluciones de la iteracion.
    else:
        soluciones = soluciones_temp[:]

# Transforma el valor de cada incognita a una cadena solo con los decimales deseados e imprime los resultados.
for i in soluciones_temp:
    incognita_lista = [digito for digito in str(i)]
    incognita_cadena = ''
    for digito in incognita_lista[:incognita_lista.index('.') + 1 + decimales]:
        incognita_cadena = incognita_cadena + str(digito)
    print("\nEl resultado para la incognita " + str(soluciones_temp.index(i) + 1) + " es:", incognita_cadena)
print("\nLas iteraciones realizadas fueron:", (iteraciones))
```

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

```
C:\Users\juanf\PycharmProjects\MetodoJacobi\venv\Scripts\python.exe C:/Users/juanf/PycharmProjects/MetodoJacobi/A.py
Bienvenido a GaussSei. Este programa resuelve un sistema de ecuaciones utilizando el metodo de Gauss-Seidel.

Escribe el numero de ecuaciones (debe ser igual al numero de incognitas): 3

En la ecuacion 1:
    Escribe el coeficiente de la incognita 1: 1
    Escribe el coeficiente de la incognita 2: 4
    Escribe el coeficiente de la incognita 3: 0
    Escribe la igualdad de la ecuacion: 5

En la ecuacion 2:
    Escribe el coeficiente de la incognita 1: 1
    Escribe el coeficiente de la incognita 2: 1
    Escribe el coeficiente de la incognita 3: 0
    Escribe la igualdad de la ecuacion: 2

En la ecuacion 3:
    Escribe el coeficiente de la incognita 1: 2
    Escribe el coeficiente de la incognita 2: 3
    Escribe el coeficiente de la incognita 3: 0
    Escribe la igualdad de la ecuacion: 0

Error: El sistema no es diagonalmente dominante. El programa ha terminado porque no se puede asegurar convergencia del metodo en esta instancia.

Process finished with exit code 0
```

b) Encuentre la matriz de transición por el método de Jacobi y determine si el método converge.

```python
def matrizTransicionGS(a):
    #d (diagonal).
    d = np.diag(a)
    d = np.diag(d)
    if(not(np.linalg.det(d))): #Evaluando que tenga inversa...
        return
    dInv = np.linalg.inv(d)
    #l (lower): matriz inferior triangular.
    l = np.tril(a)
    #u (upper): matriz superior triangular.
    u = np.triu(a)
    ident = np.identity(len(a))
    dInv_l = np.dot(dInv, l)
    _dInv_u = np.dot(-dInv, u)
    if(not(np.linalg.det(_dInv_u))): #Evaluando que tenga inversa...
        return
    else:
        if(np.linalg.det(ident + dInv_l)): #Evaluando que tenga inversa...
            t = np.dot(np.linalg.inv(ident + dInv_l), _dInv_u)
            return(t)
        return
```

```python
def jacobi(A, b, x, n):
    y=[]
    D = np.diag(A)
    R = A-np.diagflat(D)
    for i in range(n):
        y=x
        x = (b-np.dot(R,x))/D
        print("({})->{}".format(i+1, x))
        if(i > 0):
            e1 = (x-y)
            print("Vector error: ",abs(e1))
    return x
```

b) Encuentre la matriz de transición por el método de Jacobi y determine si el método converge.

```
-------------- JACOBI -------------
(1)->[-1.4365  -0.35625  1.2445 ]
(2)->[0.3306875 1.03225   0.6554375]
Vector error:  [1.7671875 1.3885    0.5890625]
(3)->[-0.7106875  -0.44040625  1.0025625 ]
Vector error:  [1.041375   1.47265625 0.347125  ]
(4)->[0.39380469 0.42740625 0.63439844]
Vector error:  [1.10449219 0.8678125  0.36816406]
(5)->[-0.25705469 -0.49300391  0.85135156]
Vector error:  [0.65085937 0.92041016 0.21695312]
(6)->[0.43325293 0.04937891 0.62124902]
Vector error:  [0.69030762 0.54238281 0.23010254]
(7)->[ 0.02646582 -0.52587744  0.75684473]
Vector error:  [0.40678711 0.57525635 0.1355957 ]
(8)->[ 0.45790808 -0.18688818  0.61303064]
Vector error:  [0.43144226 0.33898926 0.14381409]
(9)->[ 0.20366614 -0.5464234   0.69777795]
Vector error:  [0.25424194 0.35953522 0.08474731]
(10)->[ 0.47331755 -0.33455511  0.60789415]
Vector error:  [0.26965141 0.21186829 0.0898838 ]
(11)->[ 0.31441634 -0.55926463  0.66086122]
Vector error:  [0.15890121 0.22470951 0.05296707]
(12)->[ 0.48294847 -0.42684695  0.60468384]
Vector error:  [0.16853213 0.13241768 0.05617738]
(13)->[ 0.38363521 -0.56729039  0.63778826]
Vector error:  [0.09931326 0.14044344 0.03310442]
(14)->[ 0.48896779 -0.48452934  0.6026774 ]
Vector error:  [0.10533258 0.08276105 0.03511086]
(15)->[ 0.42689701 -0.57230649  0.62336766]
Vector error:  [0.06207079 0.08777715 0.02069026]
(16)->[ 0.49272987 -0.52058084  0.60142338]
Vector error:  [0.06583286 0.05172566 0.02194429]
(17)->[ 0.45393563 -0.57544156  0.61435479]
Vector error:  [0.03879424 0.05486072 0.01293141]
(18)->[ 0.49508117 -0.54311302  0.60063961]
Vector error:  [0.04114554 0.03232853 0.01371518]
(19)->[ 0.47083477 -0.57740097  0.60872174]
Vector error:  [0.0242464  0.03428795 0.00808213]
(20)->[ 0.49655073 -0.55719564  0.60014976]
Vector error:  [0.02571596 0.02020533 0.00857199]
(21)->[ 0.48139673 -0.57862561  0.60520109]
```

```
(20)->[ 0.49655073 -0.55719564  0.60014976]
Vector error:  [0.02571596 0.02020533 0.00857199]
(21)->[ 0.48139673 -0.57862561  0.60520109]
Vector error:  [0.015154   0.02142997 0.00505133]
(22)->[ 0.49746921 -0.56599728  0.5998436 ]
Vector error:  [0.01607248 0.01262833 0.00535749]
(23)->[ 0.48799796 -0.57939101  0.60300068]
Vector error:  [0.00947125 0.01339373 0.00315708]
(24)->[ 0.49804325 -0.5714983   0.59965225]
Vector error:  [0.0100453  0.00789271 0.00334843]
(25)->[ 0.49212372 -0.57986938  0.60162543]
Vector error:  [0.00591953 0.00837108 0.00197318]
(26)->[ 0.49840203 -0.57493644  0.59953266]
Vector error:  [0.00627831 0.00493294 0.00209277]
(27)->[ 0.49470233 -0.58016836  0.60076589]
Vector error:  [0.00369971 0.00523193 0.00123324]
(28)->[ 0.49862627 -0.57708527  0.59945791]
Vector error:  [0.00392394 0.00308309 0.00130798]
(29)->[ 0.49631395 -0.58035523  0.60022868]
Vector error:  [0.00231232 0.00326995 0.00077077]
(30)->[ 0.49876642 -0.5784283   0.59941119]
Vector error:  [0.00245247 0.00192693 0.00081749]
RESULTADO -> [ 0.499      -0.58066667  0.59933333]
```

No converge

c) Compare la solución entre la solución de Jacobi y Gauss Seidel. Utilice una tolerancia de 10^{-6}, genere varias iteraciones

```python
def seidel(a, x, b):
    n = len(a)
    for j in range(0, n):
        d = b[j]
        for i in range(0, n):
            if(j != i):
                d-=a[j][i] * x[i]
        x[j] = d / a[j][j]
        para=(b[j]-d)/ a[j][j]
        if x[j]==para:
         print("FIN.")
    return x
```

```python
def jacobi(A, b, x, n):
  y=[]
  D = np.diag(A)
  R = A-np.diagflat(D)
  for i in range(n):
      y=x
      x = (b-np.dot(R,x))/D
      print("({})->{}".format(i+1, x))
      if(i > 0):
        e1 = (x-y)
        print("Vector error: ",abs(e1))
  return x
```

c) Compare la solución entre la solución de Jacobi y Gauss Seidel. Utilice una tolerancia de 10^{-6}, genere varias iteraciones

Seidel

```
--------------- SIDEL -------------
0 [0.7986652720263248, -0.8054154812223759, 0.543146129694406]
1 [0.939134675763985, -0.9248144743993872, 0.5132963814001532]
2 [1.013759046499617, -0.9882451895246743, 0.4974387026188315]
3 [1.0534032434529215, -1.0219427569349833, 0.48901431076625423]
4 [1.0744642230843646, -1.03984458962171, 0.4845388525945726]
5 [1.0856528685135687, -1.0493549382365335, 0.4821612654408667]
6 [1.0915968363978334, -1.0544073109381582, 0.4808981722654605]
7 [1.094754569336349, -1.0570913839358966, 0.4802271540160259]
8 [1.0964321149599354, -1.0585172977159452, 0.47987067557101376]
9 [1.0973233110724656, -1.0592748144115958, 0.4796812963971011]
10 [1.0977967590072475, -1.0596772451561605, 0.47958068871095993]
11 [1.0980482782226004, -1.0598910364892105, 0.47952724087769744]
12 [1.0981818978057567, -1.0600046131348932, 0.47949884671627674]
13 [1.0982528832093084, -1.060064950727912, 0.47948376231802203]
14 [1.098290594204945, -1.0600970050742031, 0.47947574873144927]
15 [1.098310628171377, -1.0601140339456705, 0.4794714915135824]
16 [1.098321271216044, -1.0601230805336375, 0.4794692298665907]
17 [1.0983269253335235, -1.0601278865334949, 0.47946802836662633]
18 [1.0983299290834343, -1.0601304397209192, 0.47946739006977024]
19 [1.0983315248255745, -1.0601317961017385, 0.47946705097456543]
20 [1.0983323725635865, -1.0601325166790485, 0.4794668708302379]
21 [1.0983328229244054, -1.0601328994857446, 0.4794667751285639]
22 [1.0983330621785903, -1.0601331028518017, 0.4794667242870496]
23 [1.0983331892823762, -1.0601332108900199, 0.4794666972774951]
24 [1.0983332568062625, -1.060133268285323, 0.4794666829286693]
```

Jacobi

```
(20)->[ 0.49655073 -0.55719564  0.60014976]
Vector error:  [0.02571596 0.02020533 0.00857199]
(21)->[ 0.48139673 -0.57862561  0.60520109]
Vector error:  [0.015154   0.02142997 0.00505133]
(22)->[ 0.49746921 -0.56599728  0.5998436 ]
Vector error:  [0.01607248 0.01262833 0.00535749]
(23)->[ 0.48799796 -0.57939101  0.60300068]
Vector error:  [0.00947125 0.01339373 0.00315708]
(24)->[ 0.49804325 -0.5714983   0.59965225]
Vector error:  [0.0100453  0.00789271 0.00334843]
(25)->[ 0.49212372 -0.57986938  0.60162543]
Vector error:  [0.00591953 0.00837108 0.00197318]
(26)->[ 0.49840203 -0.57493644  0.59953266]
Vector error:  [0.00627831 0.00493294 0.00209277]
(27)->[ 0.49470233 -0.58016836  0.60076589]
Vector error:  [0.00369971 0.00523193 0.00123324]
(28)->[ 0.49862627 -0.57708527  0.59945791]
Vector error:  [0.00392394 0.00308309 0.00130798]
(29)->[ 0.49631395 -0.58035523  0.60022868]
Vector error:  [0.00231232 0.00326995 0.00077077]
(30)->[ 0.49876642 -0.5784283   0.59941119]
Vector error:  [0.00245247 0.00192693 0.00081749]
RESULTADO -> [ 0.499      -0.58066667  0.59933333]
```

d) Evalue la matriz de transición del métdo **SOR** y determine varias soluciones aproximadas, para 10 valores de ω. Utilice una tolerancia de 10^{-16}

SOR

```python
def sor(a,b,w,x):
  for i in range(len(a)):
    sigma=0
    for j in range(len(a)):
        if j != i:
          sigma+=a[i][j]*x[j]
    if(a[i][i]!=0):
          x[i] += w*(((b[i]-sigma)/a[i][i])-x[i])
  return x
```

```python
print("--------------- SOR -------------  ")
w=0
for j in range(0,15):
  w+=0.125
  print("W = ",w)
  for i in range(0, 25):
      x = sor(a,b,w,x)
      print(i,x)
  print(" ")

print('\n\n')
```

d) Evalue la matriz de transición del métdo **SOR** y determine varias soluciones aproximadas, para 10 valores de ω. Utilice una tolerancia de 10^{-16}

```
--------------- SOR -------------
W =  0.125
0 [0.0079375, -0.045275390625, 0.09164764404296875]
1 [0.019127380371093752, -0.08307641983032227, 0.1706580504179001]
2 [0.03246237218379975, -0.11493340066820384, 0.23879662534478124]
3 [0.04711708197346894, -0.14205280747766666, 0.29757039694300651]
4 [0.0624823974278165, -0.1653861063973472, 0.3482682815002136]
5 [0.0781145452240907, -0.1856839479155562, 0.39199462294032583]
6 [0.0936955971881627, -0.2035388346956162, 0.429697206488547]
7 [0.1090029132923564, -0.2194187157770555, 0.4621907208094457]
8 [0.1238855537349108, -0.233693436942276, 0.490176460803818]
9 [0.1382461192313853, -0.246655566602314, 0.514258916747019]
10 [0.15202681369642923, -0.2585367934127213, 0.5349597773594943]
11 [0.165198786366818, -0.269520837415536, 0.552729779020322]
12 [0.1777540165786724, -0.2797536161984596, 0.56795875613658]
13 [0.18969916602494397, -0.28935124985922256, 0.5809841850614068]
14 [0.201050949946128, -0.298406364401100, 0.592098463041196]
15 [0.21183267786546522, -0.30699305543081273, 0.6015551221788341]
16 [0.22207169207892077, -0.315170797066271, 0.6095741444981588]
17 [0.23179749279401862, -0.32298752036685924, 0.616346516424424]
18 [0.24104038622915935, -0.33048203789172226, 0.622038138187255]
19 [0.249830529002863, -0.3376859534309236, 0.626793184869132]
20 [0.25819727101165457, -0.34462516638224044, 0.630737000311045]
21 [0.26616872148353277, -0.35132105696382143, 0.633978592242045]
22 [0.27377148038844945, -0.357791420122197, 0.636612786332971]
23 [0.28103049097634925, -0.36405120156304976, 0.6387220879925044]
24 [0.2879689797508415, -0.370113077969544, 0.640378293306893]
```

```
W =  0.25
0 [0.3012479369324207, -0.38310765332030616, 0.6424644916476507]
1 [0.3136436376968729, -0.3950473913304151, 0.643282906777587]
2 [0.32517911414710754, -0.4061139457267948, 0.6432050584752658]
3 [0.33590570043410467, -0.4164299619717866, 0.6425019212332127]
4 [0.3458848931952885, -0.42608201887588076, 0.641371314745167]
5 [0.35517904843569403, -0.4351343785670303, 0.6399575873984358]
6 [0.36384698230808887, -0.44363724389563713, 0.6383658628053495]
7 [0.3719422199614985, -0.45163173273917445, 0.6366724138078137]
8 [0.3795126148597191, -0.4591528889775898, 0.634932254794760]
9 [0.3866006278280874, -0.4662315185262862, 0.633184721188177]
10 [0.39324388059474424, -0.4728953214319681, 0.6314575833016354]
11 [0.39947578321455224, -0.4791696014703524, 0.6297700873843295]
12 [0.40532613768660525, -0.48507772145748224, 0.6281352079471545
13 [0.41082167603823183, -0.490641404853583, 0.6265613181570169]
14 [0.4159865204387207, -0.4958809438376338, 0.6250534296279106]
15 [0.42084256729859687, -0.5008153498949679, 0.6236141128524975]
16 [0.4254098035792541, -0.505462468539055, 0.6222441803556822]
17 [0.4297065655355134, -0.5098390711699698, 0.6209431933186386]
18 [0.4337497499960044, -0.5139609319193132, 0.6197098367440219]
19 [0.4375549872318745, -0.51784289424896, 0.6185421966674565]
20 [0.4411367830955859, -0.5214989302254264, 0.6174379643615032]
21 [0.4445086367389569, -0.5249421942850303, 0.616394586128313]
22 [0.4476831389826609, -0.5281850726400021, 0.6154093725562346]
23 [0.45067205535699606, -0.5312392290746737, 0.6144795776000089]
24 [0.4534863969692483, -0.5341156476377388, 0.613602455222648]
```

```
W =  0.375
0 [0.4574615240038942, -0.5385518532225587, 0.6121997982745402]
1 [0.46119366122127853, -0.5425056443943457, 0.6109524697596177]
2 [0.4646382507492088, -0.546062491729716, 0.6098394350001001]
3 [0.46779148251723834, -0.5492767147577869, 0.608842454866652]
4 [0.47066625259890155, -0.5521876001233217, 0.6079464467800136]
5 [0.4732816704089977, -0.5548264904939804, 0.6071390457536979]
6 [0.47565849445705555, -0.5572199725853755, 0.6064100311661823]
7 [0.4778171763252966, -0.5593913732855198, 0.6057508282333465]
8 [0.4797770589398628, -0.561361515983341, 0.605154125522396]
9 [0.4815560882077483, -0.5631491480303358, 0.6046135958236541]
10 [0.48317075301337464, -0.564771217195503, 0.6041236957777053]
11 [0.4846361254695946, -0.5662430745563536, 0.603679521621407]
12 [0.4859659431374711, -0.5675786379531278, 0.6032767037052741]
13 [0.48717270638523663, -0.568790531419183, 0.6029113274952479]
14 [0.4882677784524182, -0.5698902078740526, 0.6025798726963375]
15 [0.4892614824973387, -0.5708880588083778, 0.6022791649219256]
16 [0.49016319310069295, -0.5717935131033756, 0.6020063362227622]
17 [0.4909814212482575, -0.5726151263947983, 0.6017587920397139]
18 [0.491723892578698, -0.5733606620307845, 0.6015341829594352]
19 [0.4923976190578444, -0.5740371644768121, 0.6013303801799459]
20 [0.49300896442025616, -0.5746510258993347, 0.6011454539344036]
21 [0.49356370379684, -0.5752080465735974, 0.6009776543427274]
22 [0.49406707797185423, -0.5757134896934517, 0.6008253943054436]
23 [0.4945238427086922, -0.5761721311040917, 0.6006872341498937]
24 [0.4949383135659584, -0.5765883044289306, 0.6005618678034713]
```

d) Evalue la matriz de transición del métdo **SOR** y determine varias soluciones aproximadas, para 10 valores de ω. Utilice una tolerancia de 10^{-16}

```
W =  0.5
0 [0.4954397709438282, -0.577138832842967, 0.6003885797963648]
1 [0.49589694778802673, -0.5776071993674479, 0.6002433899772515]
2 [0.4963011736568063, -0.5780111160578698, 0.6001203054813921]
3 [0.49665475535010434, -0.57836105310005, 0.6000150211031898]
4 [0.49696277258757093, -0.5786646886324653, 0.5999244244725368]
5 [0.49723064453096, -0.5789282829562756, 0.599846176866734]
6 [0.49746342837408336, -0.5791571550100774, 0.5997784440571073]
7 [0.4976656473158207, -0.579355889741333, 0.599719735810887]
8 [0.4978412823109102, -0.5795284587608969, 0.5996688105603314]
9 [0.4979938131907914, -0.5796783080069539, 0.5996246167792965]
10 [0.4981262720980034, -0.5798084289428161, 0.5995862547717963]
11 [0.49824129690255775, -0.5799214189633928, 0.599552950015474]
12 [0.49834118056255117, -0.5800195334407188, 0.5995240333276473]
13 [0.4984279153215451, -0.5801047307999829, 0.5994989253138258]
14 [0.49850323171076616, -0.5801787116273005, 0.5994771237035004]
15 [0.4985686327156208, -0.5802429526190704, 0.5994581927743664]
16 [0.4986254235899618, -0.5802987360589751, 0.5994417543798114]
17 [0.4986747378170966, -0.5803471754134224, 0.5994274802632279]
18 [0.49871755968858167, -0.5803892375570259, 0.5994150854369857]
19 [0.49875474392817554, -0.5804257620719555, 0.5994043224594985]
20 [0.49878703274107106, -0.5804574780064421, 0.599394976478944]
21 [0.49881507062295133, -0.5804850184269599, 0.599386860936102]
22 [0.49883941722158565, -0.5805089330545619, 0.5993798138362307]
23 [0.49886055850625355, -0.5805296992375972, 0.5993736945134157]
24 [0.4988789164672257, -0.5805477314798313, 0.5993683808217289]
```

```
W =  0.625
0 [0.4988988428063806, -0.5805691723670325, 0.5993617096257995]
1 [0.4989163655994392, -0.5805864688833431, 0.5993565053466525]
2 [0.49893104438885677, -0.5806006489281158, 0.5993523381099767]
3 [0.4989431958308756, -0.5806123135640825, 0.5993489527968534]
4 [0.498953220419742, -0.5806219157837766, 0.599346182957605]
5 [0.49896148068104856, -0.580629821401032, 0.5993439090151906]
6 [0.49896828403712695, -0.5806363303841667, 0.5993420392581704]
7 [0.49897388638150075, -0.5806416895013019, 0.5993405007372355]
8 [0.49897849934679805, -0.5806461018916068, 0.5993392343558998]
9 [0.49898229751673995, -0.580649734802215, 0.5993381918206163]
10 [0.4989854247573158, -0.5806527259338511, 0.5993373335505567]
11 [0.4989879995654861, -0.580655188661271, 0.599336626836264]
12 [0.49899011952202804, -0.580657216330761, 0.5993360450119176]
13 [0.4989918649758047, -0.5806588857983317, 0.5993355659734798]
14 [0.49899330208389475, -0.5806602603428438, 0.5993351715614856
15 [0.49899448531716856, -0.5806613920645071, 0.5993348468254779
16 [0.4989954595241759, -0.5806623238596667, 0.5993345794564813]
17 [0.4989962616307847, -0.5806630910467301, 0.5993343593201289]
18 [0.498996922039699, -0.5806637227048626, 0.5993341780724136]
19 [0.4989974657827914, -0.5806642427761923, 0.5993340288433752]
20 [0.4989979134698869, -0.5806646709733043, 0.5993339059766869]
21 [0.498998282069944, -0.580665023526418, 0.5993338048152548]
22 [0.4989985855542375, -0.580665313798572, 0.5993337215246937]
23 [0.49899883542591966, -0.580665552792131, 0.5993336529479897]
24 [0.4989990411560313, -0.5806657495658154, 0.5993335964858375]
```

```
W =  0.75
0 [0.498999244419779, -0.580665963036485, 0.5993335310521185]
1 [0.49899941531296754, -0.5806661248003933, 0.5993334843629559]
2 [0.4989995490284631, -0.5806662492105545, 0.59933344936376]
3 [0.4989996524380527, -0.5806663450433384, 0.5993334226453141]
4 [0.49899973219639104, -0.5806664188753081, 0.5993334021222083]
5 [0.498999793664586, -0.580666475758296, 0.5993333863258716]
6 [0.49899984103065614, -0.5806665195832172, 0.5993333741596147]
7 [0.4989998775232237, -0.5806665533476899, 0.5993333647872119]
8 [0.4989999056388815, -0.5806665793611911, 0.5993333575665796]
9 [0.4989999273003904, -0.5806665994030337, 0.5993333520035761]
10 [0.49899994398930403, -0.5806666148440714, 0.5993333477176306]
11 [0.4989999568471162, -0.580666626740465, 0.5993333444155704]
12 [0.4989999667532906, -0.5806666359059227, 0.5993333418715321]
13 [0.49899997438540417, -0.5806666429673584, 0.5993333399115034]
14 [0.49899998026549014, -0.5806666484077709, 0.5993333384014188]
15 [0.4989999847957437, -0.5806666525992825, 0.5993333372379892]
16 [0.4989999882860323, -0.5806666558285909, 0.5993333363416365]
17 [0.49899999097509046, -0.5806666583165793, 0.5993333356510505]
18 [0.49899999304684844, -0.5806666602334252, 0.5993333351189954]
19 [0.49899999464301376, -0.58066666171024, 0.5993333347090789]
20 [0.49899999587276345, -0.5806666628480371, 0.5993333343932628]
21 [0.49899999682021173, -0.5806666637246417, 0.5993333341499454]
22 [0.4989999975501639, -0.5806666644000129, 0.599333333962484]
23 [0.49899999811254825, -0.5806666649203458, 0.5993333338180562]
24 [0.49899999854583155, -0.5806666653212311, 0.5993333337067832]
```

d) Evalue la matriz de transición del métdo **SOR** y determine varias soluciones aproximadas, para 10 valores de ω. Utilice una tolerancia de 10^{-16}

```
W =  0.875
0 [0.49899999893528685, -0.580666665718077, 0.5993333335875186]
1 [0.498999992443989, -0.580666659966266, 0.5993333335116778]
2 [0.498999994658361, -0.580666661933538, 0.5993333334591636]
3 [0.49899999622618, -0.5806666663323202, 0.5993333334222004]
4 [0.4989999973341236, -0.5806666664304856, 0.5993333333961064]
5 [0.498999998116827, -0.580666664998292, 0.5993333333776758]
6 [0.498999998669733, -0.580666665488134, 0.5993333333646566]
7 [0.4989999990603044, -0.5806666665834156, 0.59933333335546]
8 [0.498999999336203, -0.580666666078584, 0.5993333333489635]
9 [0.498999999531096, -0.580666666251248, 0.5993333333443744]
10 [0.49899999996687683, -0.5806666666373217, 0.5993333333411327]
11 [0.49899999997660194, -0.5806666666459375, 0.5993333333388429]
12 [0.498999999834717, -0.580666666520236, 0.5993333333372253]
13 [0.49899999998832445, -0.5806666666563228, 0.5993333333360825]
14 [0.498999999917524, -0.580666666593597, 0.5993333333352755]
15 [0.498999999941739, -0.580666666661505, 0.5993333333347053]
16 [0.498999999958844, -0.580666666630205, 0.5993333333343025]
17 [0.498999999970927, -0.58066666664091, 0.599333333334018]
18 [0.49899999999794625, -0.5806666666648472, 0.599333333333817]
19 [0.49899999999854927, -0.5806666666653815, 0.599333333333675]
20 [0.49899999999897526, -0.5806666666657587, 0.5993333333335747]
21 [0.498999999992761, -0.580666666660253, 0.5993333333335038]
22 [0.498999999994886, -0.580666666662137, 0.5993333333334537]
23 [0.498999999996388, -0.580666666663467, 0.5993333333334184]
24 [0.49899999999974487, -0.5806666666664406, 0.5993333333333934]
```

```
W =  1.0
0 [0.49899999999983047, -0.5806666666665246, 0.5993333333333689]
1 [0.498999999998934, -0.5806666666665778, 0.599333333333556]
2 [0.498999999999334, -0.5806666666666112, 0.5993333333333473]
3 [0.49899999999995837, -0.580666666666632, 0.599333333333342]
4 [0.498999999999974, -0.580666666666645, 0.599333333333388]
5 [0.498999999999838, -0.5806666666666531, 0.5993333333333368]
6 [0.49899999999998984, -0.5806666666666582, 0.5993333333333355]
7 [0.49899999999999367, -0.5806666666666613, 0.5993333333333347]
8 [0.498999999999996, -0.580666666666633, 0.599333333333342]
9 [0.498999999999975, -0.5806666666666647, 0.5993333333333339]
10 [0.498999999999985, -0.5806666666666653, 0.5993333333333337]
11 [0.498999999999999, -0.5806666666666658, 0.5993333333333336]
12 [0.49899999999999933, -0.5806666666666661, 0.5993333333333335]
13 [0.49899999999999956, -0.5806666666666662, 0.5993333333333335]
14 [0.49899999999999967, -0.5806666666666664, 0.5993333333333335]
15 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
16 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
17 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
18 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
19 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
20 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
21 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
22 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
23 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
24 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
```

```
W =  1.125
0 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
1 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
2 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
3 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
4 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
5 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
6 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
7 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
8 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
9 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
10 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
11 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
12 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
13 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
14 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
15 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
16 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
17 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
18 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
19 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
20 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
21 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
22 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
23 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
24 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
```

d) Evalue la matriz de transición del métdo **SOR** y determine varias soluciones aproximadas, para 10 valores de ω. Utilice una tolerancia de 10^{-16}

```
W =  1.375
0 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
1 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
2 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
3 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
4 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
5 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
6 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
7 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
8 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
9 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
10 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
11 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
12 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
13 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
14 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
15 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
16 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
17 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
18 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
19 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
20 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
21 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
22 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
23 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
24 [0.49899999999999983, -0.5806666666666664, 0.5993333333333335]
```

e) Construya una función $f(\omega)$ que determine el valor óptimo de ω para que el método**SOR** converja

No se pudo desarrollar

PUNTO 5

5. Sea I una imagen en blanco y negro, digamos con valores en una gama de 0 a 1 de 800×600 píxeles. Se considera la transformación de desenfoque que consiste en que el valor de gris de cada píxel se cambia por una combinacion lineal de los valores de los píxeles adyacentes y el mismo, segun la caja

a_{11}	a_{12}	a_{13}
a_{21}	a_{22}	a_{23}
a_{31}	a_{32}	a_{33}

Figura 2: *I*

Donde se supone que a_{22} (la ponderación del propio pıxel) es mayor que la suma de todos los demás valores a_{ij} en valor absoluto. Se pide:

a) Si se desea realizar la operación inversa (enfocar), ¿se puede utilizar el algoritmo de Gauss-Seidel o el de Jacobi? ¿Piensas que es mejor usar uno de estos (si es que se puede) o, por ejemplo, la factorización **LU**? ¿Por qué?

b) ¿Qué condiciones se han de dar para que la matriz de la transformación sea simétrica? ¿Y definida positiva?

Main

main.py | desenfoque.py | gauss.py

```python
import numpy as np
import desenfoque as des
import jacobi as jac
import gauss as gau

A = np.random.rand(800, 600)
x = des.igual(A)
d = des.desenfoque(A)
x0 = np.zeros_like(x)

j = jac.jacobi(d, x, x0, n=10)
g = gau.gauss(d)
```

Desenfoque

.py | desenfoque.py | gauss.py | jacobi

```python
def igual(A):
    x = []
    for i in range(len(A)):
        for j in range(len(A[i])):
            x.append(A[i][j])
    return x


def desenfoque(A):
    d = []
    for i in range(len(A)):
        for j in range(len(A[i])):
            B = []
            if i-1 < 0:
                B.append(0.)
            else:
                B.append(A[i-1, j])
            if i+1 > len(A[i]) - 1:
                B.append(0.)
            else:
                B.append(A[i+1, j])
            if j-1 < 0:
                B.append(0.)
            else:
                B.append(A[i, j-1])
            if j+1 > len(A[i]) - 1:
                B.append(0.)
            else:
                B.append(A[i, j+1])
            d.append(B)
    return d
```

Gauss

```python
import numpy as np


def gauss(A):
    D = np.diag(np.diag(A))
    try:
        LU = A - D
    except:
        print("● La matriz no es simetrica")
    else:
        BJ = np.dot(np.linalg.inv(D), -LU)
        print(BJ)
```

Jacobi

```python
import numpy as np


def jacobi(A, b, x0, eps=1e-16, n=500):
    D = np.diag(np.diag(A))
    try:
        LU = A - D
    except:
        print("● La matriz no es simetrica")
    else:
        x = x0
        for i in range(0, n):
            D_inv = np.linalg.inv(D)
            xTemp = x
            x = np.dot(D_inv, np.dot(-(LU), x) + b)
            print(f"Pasos: {i} - x: {x}")
            if np.linalg.norm(x - xTemp) < eps:
                return x
```

output

```
C:\Users\juanf\PycharmProjects\Meto
● La matriz no es simetrica
● La matriz no es simetrica


Process finished with exit code 0
```

10. **fecha de entrega 21 septiembre**: Dado un sistema de ecuaciones no lineales, implemente el método de Newton Multivariado (es decir para varias variables) para resolver el problema:

Determinar numericamente la interseccion entre la circunferencia $x^2 + y^2 = 1$ y la recta $y = x$. Usamos una aproximacion inicial $(1, 1)$.

No se pudo desarrollar

Bibliografia

1. HTTPS://GITHUB.COM/SALVADOR04/SCIENTIFIC_COMPUTATION/BLOB/3BACB6A56D147BDBC6A1A5E6DE6811F1FADE8DF6/6-GAUSSSEI.PY

Muchas gracias por su atención