Taller 2 Grupo 2

David Alejandro Castillo Chíquiza
Sergio Esteban Triana Bobadilla
David Alejandro Antolínez Socha

20 de septiembre de 2021

1. Punto 2

1.1. Enunciado

Dado el sistema lineal de la forma $AX = b$ donde la matriz de coeficientes esta dado por:

a) Si $A = \begin{bmatrix} 4 & 3 & 0 \\ 3 & 4 & -1 \\ 0 & -1 & 4 \end{bmatrix}$, es diagonalmente dominante.

b) Calcule el radio espectral $\rho(\lambda)$ de la matriz de transición por el método de Gauss-Seidel.

c) Utilice el método Gauss-Seidel para aproximar la solución con una tolerancia de 10^{-16}, determine el numero de iteraciones. Tenga en cuenta que

$$b = \begin{bmatrix} 0{,}254 \\ -1{,}425 \\ 2{,}978 \end{bmatrix}$$

d) Que pasa con la solución anterior si $a_{13} = -2$ explique su respuesta.

e) Evalué matriz de transición del método **SOR** y determine varias soluciones aproximadas, para 10 valores de w. Utilice una tolerancia de 10^{-5}.

1.2. Solución

b) El radio espectral es supremo de los valores absolutos dentro del espectro de una matriz El proceso que se aplica para hallar este valor es el siguiente: Suponiendo una matriz A, a la cual se descompondrá en 3 matrices, la matriz con su diagonal, la matriz con la triangular inferior y la matriz con la triangular superior.
lo siguiente que se realiza es hallar una matriz U restando la matriz de la diagonal a la matriz de la triangular superior, con esa matriz U se hace

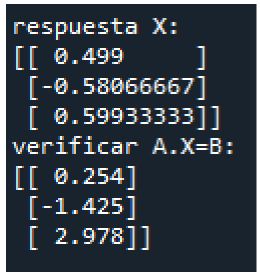
producto punto con la inversa de la matriz de la diagonal inferior y U y ese resultado se halla el máximo valor absoluto del espectro de la matriz.

```
import numpy as np

# INGRESO
A = np.array([[4,3,0],
              [3,4,-1],
              [0,-1,4]])

B = np.array([0.254,-1.425,2.978])

diagonal = np.array([[4,0.,0.],
                     [0.,4,0.],
                     [0.,0.,4]])

superior = np.array([[4.,3,0.],
                     [0.,4,-1],
                     [0.,0.,4]])

inferior = np.array([[4,0.,0.],
                     [3,4,0.],
                     [0.,-1,4]])

U = superior - diagonal

M = np.dot(np.linalg.inv(inferior),U);M

np.linalg.eigvals(M)

print(np.max(abs(np.linalg.eigvals(M))))
```

```
EL resultado del radio espectral es:  0.625
```

c) Al aproximar la solución por el método de Gauss-Sediel la solución es la siguiente:

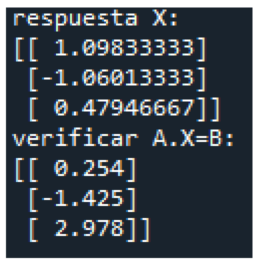
```python
# Gauss-Seidel
tamano = np.shape(A)
n = tamano[0]
m = tamano[1]
# valores iniciales
X = np.copy(X0)
diferencia = np.ones(n, dtype=float)
errado = 2*tolera



itera = 0
while not(errado<=tolera or itera>iteramax):
    # por fila
    for i in range(0,n,1):
        # por columna
        suma = 0
        for j in range(0,m,1):
            # excepto diagonal de A
            if (i!=j):
                suma = suma-A[i,j]*X[j]

        nuevo = (B[i]+suma)/A[i,i]
        diferencia[i] = np.abs(nuevo-X[i])
        X[i] = nuevo
    errado = np.max(diferencia)
    itera = itera + 1



# Respuesta X en columna
X = np.transpose([X])



# revisa si NO converge
if (itera>iteramax):
    X=0
# revisa respuesta
verifica = np.dot(A,X)



# SALIDA
print('respuesta X: ')
print(X)
print('verificar A.X=B: ')
print(verifica)
```



```
respuesta X:
[[ 0.499     ]
 [-0.58066667]
 [ 0.59933333]]
verificar A.X=B:
[[ 0.254]
 [-1.425]
 [ 2.978]]
```

Al final se hace la verificación, con el fin de corroborar que la solución si está correcta.

d) Al aplicar el cambio dado $A_{13} = -2$, el resultado es el siguiente:



```
respuesta X:
[[ 1.09833333]
 [-1.06013333]
 [ 0.47946667]]
verificar A.X=B:
[[ 0.254]
 [-1.425]
 [ 2.978]]
```

podemos ver que con el cambio tenemos que la solución varía pasando de (0.499,-0.58066667,0.59933333) a (1.0983333,-1.06013333,0.47946667)

2. Punto 7

2.1. Enunciado

Dada la matriz A (del punto 1) Verificar si:

i. Se puede descomponer de la forma LU, entonces utilice el resultado para resolver el sistema, teniendo en cuenta que la máquina admite cuatro dígitos significativos; ¿cómo afecta esto la respuesta?

2.2. Solución

Implementación del método para descomponer de la forma LU:

```python
import numpy as np

# funcion LU
def descomposicionLu(A):
    n = len(A[0])
    L = np.zeros([n,n])
    U = np.zeros([n,n])
    for i in range(n):
        L[i][i]=1
        if i == 0:
            U[0][0]=A[0][0]
            for j in range(1,n):
                U[0][j]=A[0][j]
                L[j][0]=A[j][0]/U[0][0]
        else:
            for j in range (i, n):
                temp=0
                for k in range (0,i):
                    temp = temp + L[i][k]*U[k][j]
                U[i][j]= A[i][j]-temp
            for j in range (i+1, n):
                temp=0
                for k in range (0,i):
                    temp = temp + L[j][k]*U[k][i]
                L[j][i] = (A[j][i]-temp)/U[i][i]
    return L,U

#main
A = [[1,-8,-2],[1,1,5],[3,-1,1]]
L,U=descomposicionLu(A)
print(L,'\n',U)
```

Resultados de la implementación:

```
[[1.        0.        0.        ]
 [1.        1.        0.        ]
 [3.        2.55555556 1.        ]]

 [[  1.         -8.          -2.        ]
 [  0.          9.          7.        ]
 [  0.          0.        -10.88888889]]
```

3. Punto 10

3.1. Enunciado

Dado un sistema de ecuaciones no lineales, implemente el método de Newton Multivariado (es decir para varias variables) para resolver el problema: Determinar la intersección dela circunferencia $x^2 + y^2 = 1$ y la recta $x = y$. Usamos una aproximación lineal $(1, 1)$.

3.2. Solución

Implementación del metodo de Newton Multivariado:

```
import numpy as np
import sympy as sp

def newton(F, V, U):
    n=len(F)
    J=np.zeros([n,n],dtype=sp.Symbol)
    T=list(np.copy(F))

    for i in range(n):
        for j in range(n):
            J[i][j]=sp.diff(F[i],V[j])
    for i in range(n):
        for j in range(n):
            for k in range(n):
                J[i][j]=J[i][j].subs(V[k],float(U[k]))
    for i in range(n):
        for j in range(n):
            T[i]=T[i].subs(V[j],float(U[j]))
    J=np.array(J,float)
    T=np.array(T,float)
    U=U-np.dot(np.linalg.inv(J),T)
    return U
```

Resultados de la implementación:

```
[x,y]=sp.symbols('x,y')
f=x**2+y**2-1
g=y-x
F=[f,g]
V=[x,y]
U=[1,1]


U=newton(F,V,U);print("Iteración 1: ");print(U)
U=newton(F,V,U);print("Iteración 2: ");print(U)
U=newton(F,V,U);print("Iteración 3: ");print(U)
U=newton(F,V,U);print("Iteración 4: ");print(U)
```

```
1.6s
```

```
Iteración 1:
[0.75 0.75]
Iteración 2:
[0.70833333 0.70833333]
Iteración 3:
[0.70710784 0.70710784]
Iteración 4:
[0.70710678 0.70710678]
```

Verficación en WolframAlpha:

WolframAlpha computational intelligence.

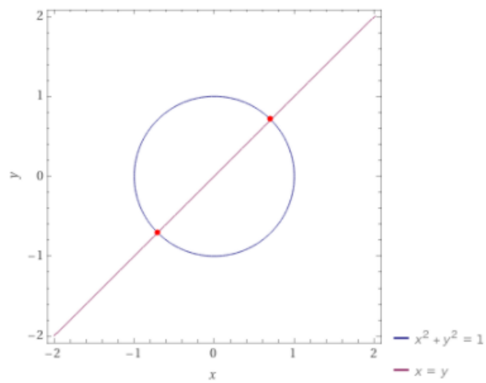
x^2+y^2=1, x=y

NATURAL LANGUAGE | MATH INPUT | EXTENDED KEYBOARD | EXAMPLES | UPLOAD | RANDOM

Input

$\{x^2 + y^2 = 1, x = y\}$

Plot of solution set



Solutions | Approximate forms

$x = -\frac{1}{\sqrt{2}}, \quad y = -\frac{1}{\sqrt{2}}$

$x = \frac{1}{\sqrt{2}}, \quad y = \frac{1}{\sqrt{2}}$

WolframAlpha computational intelligence.

1/sqrt(2)

NATURAL LANGUAGE | MATH INPUT | EXTENDED KEYBOARD | EXAMPLES | UPLOAD | RANDOM

Input

$\frac{1}{\sqrt{2}}$

Decimal approximation | More digits

0.7071067811865475244008443621048490392848359376884740365883398689
…