Universidad Nacional de Ingeniería
Escuela Profesional de Matemática
Ciclo 2021-3

[[**Análisis y Modelamiento Numérico I - CM4F1**]
[**Los profesores**] UNI, 17 de febrero de 2022

Práctica Calificada 3

1. (7 puntos) Dado el sistema no lineal

$$f_1(x_1, x_2) = x_1^2 - x_2^2 + 2x_2 = 0,$$
$$f_2(x_1, x_2) = 2x_1 + x_2^2 - 6 = 0,$$

tiene dos soluciones,

$$P_1 = (0{,}625204094; 2{,}179355825) \quad \wedge \quad P_2 = (2{,}109511920; -1{,}334532188).$$

Implemente el método de continuación de homotopía para aproximar la solución del sistema no lineal comenzando en el punto inicial:

a) $P_0 = (0, 0)$

b) $P_0 = (3, -2)$

Solución: El método de continuación de homotopía se aplica para obtener soluciones de la ecuación:

$$F(x, y) = (x^2 - y^2 + 2y, 2x + y^2 - 6) = (0, 0).$$

Dado el punto inicial x_0 definimos la homotopía $G : [0, 1] \times \mathbb{R}^2 \to \mathbb{R}^2$

$$G(t, x) = tF(x) + (1 - t)[F(x) - F(x_0)].$$

Buscamos una función diferenciable $t \to x(t)$ que empiece su recorrido en el punto $x(0) = x_0$, el cuál es solución de $G(0, x) = 0$ y llegue al punto $x(1)$ (por determinar) que será una solución de $G(1, x) = 0$, es decir una solución de $F(x) = 0$. Como F es diferenciable, obtenemos el determinante de su matriz jacobiana:

$$det(JF(x, y)) = \begin{vmatrix} 2x & -2y + 2 \\ 2 & 2y \end{vmatrix} = 4xy + y - 2.$$

Claramente la matriz jacobiana no es singular para todo $(x, y) \in \mathbb{R}^2$, por lo que no se puede justificar la existencia y unicidad de $x(t)$ tal que $G(t, x(t)) = 0$ para todo $t \in [0, 1]$. Y que cumpla además

$$\left\{ \begin{array}{lcl} x'(t) & = & -[JF(x(t)]^{-1}F(x_0), \quad 0 \leq t \leq 1 \\ x(0) & = & x_0 \end{array} \right. .$$

Consulte el Teorema 10.10 del Burden, pag 675. Sin embargo, al menos podemos verificar las condiciones de este Teorema localmente. De todas maneras aplicamos el método de Continuación y resolvemos el sistema EDO usando el método de Runge-Kutta de orden 4. Obtenemos los siguientes resultados:

```
Item a
Punto inicial:  [0, 0]
1 [ 2.30398796 -2.00109948] error = 2.70e+00
la solucion aproximada es: [ 2.30398796 -2.00109948]
1 [ 0.42489283 -0.08659096] error = 5.14e+00
2 [ 0.81515371 -0.29026545] error = 4.29e+00
3 [ 1.14894385 -0.52327943] error = 3.43e+00
4 [ 1.43426458 -0.74854607] error = 2.57e+00
5 [ 1.68374278 -0.95838808] error = 1.71e+00
6 [ 1.90672959 -1.15311636] error = 8.57e-01
7 [ 2.10953755 -1.33461778] error = 2.91e-04
la solución aproximada es: [ 2.10953755 -1.33461778]
Item b
Punto inicial:  [3, -2]
1 [ 2.10944599 -1.33456326] error = 4.23e-04
la solución aproximada es: [ 2.10944599 -1.33456326]
1 [ 2.88703763 -1.91167365] error = 3.43e+00
2 [ 2.77013123 -1.82123046] error = 2.86e+00
3 [ 2.64887498 -1.72857291] error = 2.29e+00
4 [ 2.52279513 -1.63361423] error = 1.71e+00
5 [ 2.39133479 -1.53629019] error = 1.14e+00
6 [ 2.2538351  -1.4365787 ] error = 5.71e-01
7 [ 2.10951188 -1.33453222] error = 3.03e-07
la solución aproximada es: [ 2.10951188 -1.33453222]
```

En cada ítem hemos aplicado el método con un mallado de $N+1$ puntos, para $N=1$ y $N=7$. Se observa que partiendo de los puntos $(0,0)$ y $(3,-2)$ obtenemos soluciones aproximadas de P_2. El segundo punto converge más rápidamente, posiblemente porque está más cerca a P_2. Si utilizamos el punto $(1,1)$ obtenemos una solución aproximada para P_1.

```python
# Solucion 1
# Metodo de Continuacion de homotopia

import numpy as np
from numpy import *


def f(x):
    ft = np.zeros((2,1))
    ft[0,0] = x[0]**2 - x[1]**2 + 2*x[1]
    ft[1,0] = 2*x[0] + x[1]**2-6.0
    return ft
def Jaco(x):
    Jacof = np.zeros((2,2))
    Jacof[:,0] = np.transpose([2*x[0], 2.0])
    Jacof[:,1] = np.transpose([-2*x[1]+2.0, 2*x[1]])
    return Jacof

def homo(n,f,x0,Jaco):
    b = -1*f(x0)/n
    j = 1
    for j in range(1,n+1):
        A = Jaco(x0)
        #Paso 1
        k1 = np.transpose(np.dot(np.linalg.inv(A), b))
        A1 = x0 + (1/2*k1)
        A1 = A1.ravel().tolist()
        A2 = Jaco(A1)
```

```
        #Paso 2
        k2 = np.transpose(np.dot(np.linalg.inv(A2), b))
        A3 = x0 + (1/2*k2)
        A3 = A3.ravel().tolist()
        A4 = Jaco(A3)
        #Paso3
        k3 = np.transpose(np.dot(np.linalg.inv(A4), b))
        A5 = x0 + k3
        A5 = A5.ravel().tolist()
        A6 = Jaco(A5)
        #Paso4
        k4 = np.transpose(np.dot(np.linalg.inv(A6), b))
        x = x0 + (k1 + 2*k2 + 2*k3 + k4)/6.0
        magnitud=np.linalg.norm(f(x[0]),np.inf)

        print(f'{j} {x[0]} error = {magnitud:.2e}')

        x0 = x.ravel().tolist()


    print(f'la solucion aproximada es: {x[0]} ')


print('Item a')
x0 = [0, 0]
print('Punto inicial: ',x0)
homo(1,f,x0,Jaco)
homo(7,f,x0,Jaco)

print('Item b')
x0 = [3,-2]
print('Punto inicial: ',x0)
homo(1,f,x0,Jaco)
homo(7,f,x0,Jaco)
```

2. (6 puntos) Usando el método de potencia inverso desplazado, calcule el valor propio mas pequeño de la matriz de Pascal 20×20. Implemente el algoritmo.

Solución: Teóricamente la matriz triangular inferior de Pascal tiene como único valor propio a $\lambda = 1$ y los vectores propios son múltiplos del vector canónico e_n, donde la n-ésima componente es 1. Debido a que ésta matriz está mal condicionada (Véase Proposición 1 de [2]) es necesario aplicar el método lo más cerca posible del valor propio, porque convergerá más rápido. Por tanto, buscamos el valor propio más cercano a $q = 0{,}9999999$, obteniendo los siguientes resultados:

```
MATRIZ DE PASCAL 20x20
 1: autovalor: 1.0000, autovector :
 [6.86e-32 -6.86e-25  7.73e-25 -1.21e-24  2.53e-24 -3.67e-24  2.70e-24
  6.39e-25 -2.74e-24 -1.21e-24  7.68e-24 -5.34e-25 -2.20e-23  4.82e-24
  8.67e-23  1.04e-24 -5.60e-22  2.92e-17 -5.26e-09  1.00e+00],
 error: 1.00e+00
 2: autovalor: 1.0000, autovector :
 [-2.03e-32 -6.44e-26  5.99e-25 -9.54e-25  1.75e-24 -2.52e-24  1.73e-24
   1.01e-24 -2.56e-24 -1.33e-24  7.10e-24 -6.32e-26 -2.00e-23  1.69e-24
   8.10e-23  1.25e-23 -5.22e-22  6.45e-18 -3.21e-09  1.00e+00],
 error: 2.05e-09
 3: autovalor: 1.0000, autovector :
 [-7.04e-33  2.76e-27  3.78e-25 -5.87e-25  1.15e-24 -1.75e-24  8.57e-25
```

```
 1.91e-24 -3.26e-24 -1.16e-24  7.26e-24  2.66e-25 -1.99e-23 -3.33e-24
 8.90e-23  2.65e-23 -5.57e-22 -2.02e-17  2.62e-10  1.00e+00],
 error: 3.47e-09

Valor propio mas pequeño: 1.0000
```

El vector propio calculado es aproximadamente el vector canónico.

```python
# Solucion 2
# Metodo de la potencia inversa modificado

import numpy as np
from scipy.linalg import pascal

def potInversaDesplazada(A, x, TOL, N, q):
    xp = max(x)
    x = x/xp
    yp = 0.1
    k = 0

    while k<N:
        y = np.linalg.solve(A -q* np.identity(len(A)), x)

        yp = np.linalg.norm(y,np.inf)
        u = yp

        ERR = np.linalg.norm(x-(y/yp) , np.inf)
        x = y/yp

        if  ERR < TOL:
            break
        k +=1
        np.set_printoptions(precision = 2)
        print(f' {k}: autovalor: {1/u+q:.4f}, autovector : {x}, error: {ERR:.2e↩
            }')
    print(f'\nValor propio mas pequeno: {1/u+q:.4f}')

dim=20
A = pascal(dim, kind='lower')
x = np.ones(dim)
iter = 15
TOL = 1e-10
q = 0.9999999

print(f'MATRIZ DE PASCAL {dim}x{dim}')
potInversaDesplazada(A, x, TOL, iter, q)
#print(A)
```

Véase algoritmo 9.3 de [1], pag 594.

Referencias

[1] R. L. Burden, J. D. Faires and A. M. Burden. Numerical Analysis. Boston, MA : Cengage Learning, Tenth edition (2016).

[2] J.M. Carnicer, Y. Khiar and J.M. Peña. Optimal interval length for the collocation of the Newton interpolation basis. Numer Algor 82, 895–908 (2019). https://arxiv.org/pdf/1709.06787.pdf

3. (7 puntos) Halle la solución general del siguiente sistema EDO:

$$\begin{aligned} x_1'(t) &= 5x_1(t) - x_2(t) + 2x_3(t) + x_4(t) \\ x_2'(t) &= -x_1(t) + 4x_2(t) + 2x_4(t) \\ x_3'(t) &= 2x_1(t) + 4x_3(t) + x_4(t) \\ x_4'(t) &= x_1(t) + 2x_2(t) + x_3(t) + 5x_4(t) \end{aligned}$$

(Sugerencia, implemente el método QR, y el método de Householder.

Solución: Escribimos el sistema en forma matricial: $x'(t) = Ax(t)$, donde

$$x(t) = \begin{bmatrix} x_1(t) \\ x_2(t) \\ x_3(t) \\ x_4(t) \end{bmatrix} \quad \wedge \quad A = \begin{bmatrix} 5 & -1 & 2 & 1 \\ -1 & 4 & 0 & 2 \\ 2 & 0 & 4 & 1 \\ 1 & 2 & 1 & 5 \end{bmatrix}$$

la solución general del sistema de ecuaciones diferenciales es

$$x(t) = c_1 e^{\lambda_1 t} v_1 + c_2 e^{\lambda_2 t} v_2 + c_3 e^{\lambda_3 t} v_3 + c_4 e^{\lambda_4 t} v_4,$$

donde $\lambda_1, \lambda_2, \lambda_3\ \lambda_4$ son los valores propios de A, v_i, $i = 1, 2, 3, 4$ los respectivos vectores propios y los c_i constantes reales.

a) Usamos el método de Householder para encontrar una matriz tridiagonal simétrica similar a A, para ello utilizamos las transformaciones simétricas y ortogonales $P = I - 2ww^t$, donde I representa la matriz identidad y $w \in \mathbb{R}^4$ con $\|w\|_2 = 1$.

b) Utilizamos la factorización QR para encontrar una secuencia de matrices

$$A = A^{(1)},\ A^{(2)},\ A^{(3)}, \cdots,$$

de la siguiente manera

$$A^{(i)} = Q^{(i)} R^{(i)} \quad \wedge \quad A^{(i+1)} = R^{(i)} Q^{(i)},$$

donde $Q^{(i)}$ es una matriz ortogonal y $R^{(i)}$ una matriz triangular superior. La sucesión $A^{(n)}$ converge a una matriz diagonal D cuya diagonal contiene los valores propios de A. Las columnas del producto de los $Q^{(i)}$, es decir $V = \prod_i Q^{(i)}$, son los vectores propios correspondientes a los valores propios de A.
Teóricamente los valores propios son

$$\begin{aligned} \lambda_1 &= -\sqrt{5} + 4 \approx 1{,}7639320, & \lambda_2 &= \sqrt{5} + 4 \approx 6{,}2360680 \\ \lambda_3 &= -\sqrt{6} + 5 \approx 2{,}5505103, & \lambda_4 &= \sqrt{6} + 5 \approx 7{,}4494897. \end{aligned}$$

Y sus respectivos vectores propios:

$$v_1 = \begin{bmatrix} -1 \\ -\frac{3\sqrt{5}}{5} \\ \frac{\sqrt{5}}{5} \\ 1 \end{bmatrix} \approx \begin{bmatrix} -1 \\ -1{,}3416408 \\ 0{,}4472136 \\ 1 \end{bmatrix}, \quad v_2 = \begin{bmatrix} -1 \\ \frac{3\sqrt{5}}{5} \\ -\frac{\sqrt{5}}{5} \\ 1 \end{bmatrix} \approx \begin{bmatrix} -1 \\ 1{,}3416408 \\ -0{,}4472136 \\ 1 \end{bmatrix},$$

$$v_3 = \begin{bmatrix} 1 \\ \frac{-\sqrt{6}-1}{5} \\ \frac{-3\sqrt{6}-3}{5} \\ 1 \end{bmatrix} \approx \begin{bmatrix} 1 \\ -0{,}6898979 \\ -2{,}0696938 \\ 1 \end{bmatrix}, \quad v_4 = \begin{bmatrix} 1 \\ \frac{\sqrt{6}-1}{5} \\ \frac{3\sqrt{6}-3}{5} \\ 1 \end{bmatrix} \approx \begin{bmatrix} 1 \\ 0{,}2898979 \\ 0{,}8696938 \\ 1 \end{bmatrix}.$$

Los resultados obtenidos después de aplicar 80 iteraciones la descomposición QR a las secuencias $A^{(i)}$ son

```
Matriz diagonal semejante a A:
[[ 7.44948948e+00  6.79857401e-07 -1.46686217e-16 -2.11404323e-16]
 [ 6.79857401e-07  6.23606784e+00  1.73666663e-16 -1.12239002e-16]
 [ 1.55575106e-37 -1.47370531e-31  2.55051025e+00 -9.33501111e-14]
 [ 2.99327870e-50  5.59971218e-44 -9.33670101e-14  1.76393203e+00]]
---------------------
Producto de los Q:
[[ 0.59334834 -0.49999973 -0.38462701 -0.50000001]
 [ 0.17201015  0.67082047  0.26535338 -0.67082039]
 [ 0.51603138 -0.2236066   0.79606012  0.22360679]
 [ 0.59334799  0.50000024 -0.384627    0.5       ]]
---------------------
Identificando los valores y vectores propios:
autovalor  1: 7.4494895
autovector 1: [0.59335   0.17201  0.51603  0.59335]
autovalor  2: 6.2360678
autovector 2: [-0.5      0.67082 -0.22361  0.5    ]
autovalor  3: 2.5505103
autovector 3: [-0.38463  0.26535  0.79606 -0.38463]
autovalor  4: 1.7639320
autovector 4: [-0.5     -0.67082  0.22361  0.5    ]
```

```python
# Solucion 3
# Metodo QR con Householder

import numpy as np
import numpy.linalg as npl

def householder(A):

    m = A.shape[1]
    n = A.shape[0]
    R = np.copy(A)
    Q = np.identity(m)
    for i in range(n-1):
        vk = np.copy(R[:,i])
        q = 0
        while q<i:
            vk[q] = 0
            q = q+1
        vk[i] = vk[i] + npl.norm(vk)*np.where(vk[i]>=0,1,-1)
        vk = np.transpose(np.array([vk]))
        Qk = np.identity(m) - 2*vk*np.transpose(vk)/npl.norm(vk)**2
        R = np.dot(Qk,R)
        Q = np.dot(Qk,Q)
    Q = np.transpose(Q)
    return(Q,R)

A = np.array([[5,-1,2,1],[-1,4,0,2],[2,0,4,1],[1,2,1,5]],dtype='f4')
N = len(A)

maxiter = 80
i=0
Q=np.identity(N)
V=np.identity(N)
while i<maxiter :
    (Q, R) = householder(A)
```

```python
    A = np.matmul(R,Q)
    V = np.matmul(V,Q)
    i+=1

print(f'Matriz diagonal semejante a A:\n{A}')
print(f'-------------------------------')
print(f'Producto de los Q:\n{V}')
print(f'-------------------------------')
print(f'Identificando los valores y vectores propios:')
for i in range(N):
    print(f'autovalor  {i+1}: {A[i,i]:5.7f}')
    print(f'autovector {i+1}: {V[:,i].round(5)}')
```