Universidad Nacional de Ingeniería
Facultad de Ciencias
Escuela Profesional de Matemática

Ciclo 2023-1

CM 4F1A

CURSO: ANÁLISIS Y MODELAMIENTO NUMERICO I

Los Profesores

SOLUCIONARIO EXAMEN SUSTITUTORIO

Pregunta 1 (3 puntos)

Para medir la altura de un árbol, L, se mide la longitud de su sombra, L_1, la altura de un objeto de referencia, L_2, y la longitud de sombra del objeto, L_3. Entonces, por semejanza, se obtiene que:

$$L = L_1 \frac{L_2}{L_3}$$

Si las medidas realizadas son:

$L_1 = 200 \pm 2$ cm, $L_2 = 100.0 \pm 0.4$ cm, $L_3 = 10.3 \pm 0.2$ cm

Determine la altura, L, así como el error propagado y como expresaría la altura total del árbol.

RESPUESTA

$L_1 = 200 \pm 2$ cm, $L_2 = 100.0 \pm 0.4$ cm, $L_3 = 10.3 \pm 0.2$ cm $\quad L = L_1 \frac{L_2}{L_3} =$

$$L = 200 \times \frac{100}{10} = 2000 \text{ cm}$$

Su error será

$$\frac{\delta L}{|L|} \approx \frac{\delta L_1}{|L_1|} + \frac{\delta L_2}{|L_2|} + \frac{\delta L_3}{|L_3|} = \frac{2}{200} + \frac{0.4}{100} + \frac{0.2}{10.3} =$$

$$= (1 + 0.4 + 2)\% = 3.4\% \rightarrow \delta L = \frac{3.4}{100} \times 2000 = 68$$

$$\boxed{L = 2000 \pm 70 \text{ cm}}$$

<u>Pregunta 2</u> (3 puntos)

Sean:

$x \pm \delta(x)$, $y \pm \delta(y)$ y $q(x,y) = kx^{\alpha} y^{\beta}$

Determine el error propagado en el cálculo de la función *q* para $\alpha = \beta = 2$

RESPUESTA

$q = kx^{\alpha} y^{\beta}$. $x \pm \delta(x)$ $y \pm \delta(y)$

$$\frac{\delta q}{|z|} \approx \alpha \frac{\delta x}{|x|} + \beta \frac{\delta y}{|y|} \Rightarrow \alpha = \beta = 2 \Rightarrow 2\frac{\delta x}{|x|} + 2\frac{\delta y}{|y|}$$

$$\frac{\delta q}{|q|} = \sqrt{\left[\alpha \frac{\delta x}{|x|}\right]^2 + \left[\beta \frac{\delta y}{|y|}\right]^2} \Rightarrow \alpha = \beta = 2$$

$$\frac{\delta q}{|q|} = \sqrt{\left[2 \frac{\delta x}{|x|}\right]^2 + \left[2 \frac{\delta y}{|y|}\right]^2}$$

,

<u>Pregunta 3</u> (4 puntos)

Calcular aproximadamente el punto fijo de la función $f(x) = \sqrt{\frac{10}{4+x}}$ con 10 iteraciones, y calcular el error cometido

RESPUESTA

Si *x* es punto fijo de *f*, entonces *f(x)*=*x*

Sea g(x)=f(x)-x=0

$$x_{n+1} = g(x_n)$$

$$\lim_{n\to\infty} x_{n+1} = x^*, tal\ \ que\ \ x^* = f(x^*)$$

$$x = \sqrt{\frac{10}{4+x}} \Rightarrow x^2 = \frac{10}{4+x} \Rightarrow x^3 + 4x^2 - 10 = 0$$

$$x_{n+1} = \sqrt{\frac{10}{4+x_n}} \Rightarrow n = 0,\ x_0 = 0 \Rightarrow x_1 = \sqrt{\frac{10}{4}},\ x_2 =$$

Para $x = 1$

$$g(1) = \sqrt{\frac{10}{5}} - 1 = 0.41$$

Para $x = 2$

$$g(2) = \sqrt{\frac{10}{6}} - 2 = -0.71$$

Entonces para resolver el problema con el método del punto fijo tomaremos que $x \in [1, 2]$

Utilizando el programa presentado, con 10 iteraciones se tendrá:

$$\underline{X = 1.36523}$$

y un error muy pequeño de

$x \in [1, 2]$

$$\varepsilon = 4.794 \times 10^{-10}$$

```python
#punto fijo

from numpy import *

def puntofijo(f,x0):

    eps=1E-8

    maxiter=100

    f0=f(x0)

    i=0

    while i<maxiter and abs(f0-x0)>=eps:

        x0=f0

        f0=f(x0)

        i=i+1

        print(f'x0={x0:5.5f} f(x0)={f0:5.5f}')

    if abs(f0-x0)>=eps:

        print(f'no converge')

    else:

        print(f'solucion c={x0}')

        print(f'iteraciones={i}')

```

```python
puntofijo(f=lambda x: (10/(x+4))**0.5 , x0=0)
```

```
x0=1.58114 f(x0)=1.33856
x0=1.33856 f(x0)=1.36864
x0=1.36864 f(x0)=1.36480
x0=1.36480 f(x0)=1.36529
x0=1.36529 f(x0)=1.36522
x0=1.36522 f(x0)=1.36523
x0=1.36523 f(x0)=1.36523
x0=1.36523 f(x0)=1.36523
x0=1.36523 f(x0)=1.36523
x0=1.36523 f(x0)=1.36523
solucion c=1.3652300115770295
iteraciones=10
```

Esto es con un error de : 1E-8

<u>Pregunta 4</u> (10 puntos)

a) **Mediante el método de Newton y el de Jacobi, resolver e implementar los algoritmos de resolución del siguiente problema:**

$$xy = 108$$
$$x^2 + y^2 = 225$$

Condición inicial

$\boldsymbol{x}_0 = (3, 4)$

b) Grafique la solución y las curvas de convergencia con ambos métodos numéricos

c) Compare y comente respecto a sus resultados obtenidos.

d) Adjuntar a sus respuestas los códigos de resolución y gráficos implementados computacionalmente.

RESPUESTA

Definimos

$f_1 = xy - 108$

$f_2 = x^2 + y^2 - 225$

Se tomara el valor inicial:

$x_0 = (3, 4)$

$$\begin{cases} DF(x_k)s = -F(x_k) \\ x_{k+1} = x_k + s. \end{cases}$$

$$J = \begin{bmatrix} \frac{\partial f_1}{\partial x} & \frac{\partial f_1}{\partial y} \\ \frac{\partial f_2}{\partial x} & \frac{\partial f_2}{\partial y} \end{bmatrix} = \begin{bmatrix} y & x \\ 2x & 2y \end{bmatrix}$$

y su inversa:

$$J^{-1} = \frac{1}{2y^2 - 2x^2} \begin{bmatrix} 2y & -x \\ -2x & y \end{bmatrix}$$

$$x_{i+1} = x_i - A_i^{-1} F(x_i).$$

Para Newton se tomará J^{-1}, pero para Jacobi solo la diagonal que será:

$$\frac{1}{2y^2 - 2x^2} \begin{bmatrix} 2y & 0 \\ 0 & y \end{bmatrix}$$ presentado en el programa por:

a) Valores de (x, y) por los métodos de Newton y Jacobi:

- Newton:

```python
import numpy as np
import sympy as sym

def matrizJacobiano(variables, funciones):
    n = len(funciones)
    m = len(variables)
    # matriz Jacobiano inicia con ceros
    Jcb = sym.zeros(n,m)
    for i in range(0,n,1):
        unafi = sym.sympify(funciones[i])
        for j in range(0,m,1):
            unavariable = variables[j]
            Jcb[i,j] = sym.diff(unafi, unavariable)
    return Jcb

#Datos
x = sym.Symbol('x')
y = sym.Symbol('y')

f1 = x*y-108
f2 = x**2+y**2-225

x0 = 3
y0 = 4

tolera = 0.0001

#Algoritmo
funciones = [f1,f2]
variables = [x,y]
n = len(funciones)
m = len(variables)

Jxy = matrizJacobiano(variables, funciones)

xi = x0
yi = y0

itera = 0
tramo = tolera*2
while (tramo>tolera):
    J = Jxy.subs([(x,xi),(y,yi)])

    # determinante de J
    Jn = np.array(J,dtype=float)
    determinante =  np.linalg.det(Jn)

    # iteraciones
    f1i = f1.subs([(x,xi),(y,yi)])
    f2i = f2.subs([(x,xi),(y,yi)])

    numerador1 = f1i*Jn[n-1,m-1]-f2i*Jn[0,m-1]
    xi1 = xi - numerador1/determinante
    numerador2 = f2i*Jn[0,0]-f1i*Jn[n-1,0]
    yi1 = yi -numerador2/determinante
    tramo = np.max(np.abs([xi1-xi,yi1-yi]))
    xi = xi1
    yi = yi1

    itera = itera +1
    print("* Iteración:",itera , "x =",xi,"y =", yi, "\n")
    print("Jacobiano con puntos iniciales:")
    print(J,"\n")
    print("error:",tramo, "\n")
```

Salida:

```
* Iteración: 1 x = 15.0000000000000 y = 20.0000000000000

Jacobiano con puntos iniciales:
Matrix([[4, 3], [6, 8]])

error: 16.0000000000000

* Iteración: 2 x = 10.2000000000000 y = 13.6000000000000

Jacobiano con puntos iniciales:
Matrix([[20.0000000000000, 15.0000000000000], [30.0000000000000, 40.0000000000000]])

error: 6.40000000000001

* Iteración: 3 x = 9.07058823529412 y = 12.0941176470588

Jacobiano con puntos iniciales:
Matrix([[13.6000000000000, 10.2000000000000], [20.4000000000000, 27.2000000000000]])

error: 1.50588235294117

* Iteración: 4 x = 9.00027466239414 y = 12.0003662165255

Jacobiano con puntos iniciales:
Matrix([[12.0941176470588, 9.07058823529412], [18.1411764705882, 24.1882352941176]])

error: 0.0937514305333025

* Iteración: 5 x = 9.00000000419095 y = 12.0000000055879

Jacobiano con puntos iniciales:
Matrix([[12.0003662165255, 9.00027466239414], [18.0005493247883, 24.0007324330510]])

error: 0.000366210937587041

* Iteración: 6 x = 9.00000000000000 y = 12.0000000000000

Jacobiano con puntos iniciales:
Matrix([[12.0000000055879, 9.00000000419095], [18.0000000083819, 24.0000000111759]])

error: 5.58793544769287e-9
```

Codigo de la solución con el Método de NEWTON

```
from math import *

import numpy as np



#Metodo de Newton

#introducir las funciones

def Fs(x1,x2):

    f1=x1*x2-108

    f2=x1**2+x2**2-225

    return np.matrix([[f1],[f2]])


def Jinv(x1,x2):

    #introducir derivadas de f1 f2

    J=np.matrix([ [ x2, x1 ] , [2*x1, 2*x2 ]  ])

    JV=np.linalg.inv(J)

    return [J,JV]

#introducir el vector inicial

P0=[3,4]

k=0

x1,x2=P0

TOL=1E-5

print("k \t x1 \t x2 \t //x(k)-x(k-1)//")

print("{0:1d} \t {1:1.4f} \t {2:1.4f}  \t".format(k,x1,x2))
```

- **Jacobi:**

```
import numpy as np
import sympy as sym

def matrizJacobiano(variables, funciones):
    n = len(funciones)
    m = len(variables)
    # matriz Jacobiano inicia con ceros
    Jcb = sym.zeros(n,m)
    for i in range(0,n,1):
        unafi = sym.sympify(funciones[i])
        for j in range(0,m,1):
            unavariable = variables[j]
            Jcb[i,j] = sym.diff(unafi, unavariable)
    return Jcb

#Datos
x = sym.Symbol('x')
y = sym.Symbol('y')

f1 = x*y-108
f2 = x**2+y**2-225

x0 = 3
y0 = 4

tolera = 0.0001

#Algoritmo
funciones = [f1,f2]
variables = [x,y]
n = len(funciones)
m = len(variables)

Jxy = matrizJacobiano(variables, funciones)

xi = x0
yi = y0

itera = 0
tramo = tolera*2
while (tramo>tolera):
    J = Jxy.subs([(x,xi),(y,yi)])

    # determinante de J
    Jn = np.array(J,dtype=float)
    determinante =  np.linalg.det(Jn)

    # iteraciones
    f1i = f1.subs([(x,xi),(y,yi)])
    f2i = f2.subs([(x,xi),(y,yi)])

    numerador1 = f1i*Jn[1,1]
    xi1 = xi - numerador1/determinante
    numerador2 = f2i*Jn[0,0]
    yi1 = yi -numerador2/determinante
    tramo = np.max(np.abs([xi1-xi,yi1-yi]))
    xi = xi1
    yi = yi1

    itera = itera +1
    print("* Iteración:",itera , "x =",xi,"y =", yi, "\n")
    print("Jacobiano con puntos iniciales:")
    print(J,"\n")
    print("error:",tramo, "\n")
```

```python
while k<10:

    J, JI=Jinv(x1, x2)

    F=Fs(x1,x2)

    Y=-JI*F

    X=np.matrix(P0).T+Y

    x1,x2=float(X[0][0]),float(X[1][0])

    magnitud=sqrt( (x1-P0[0])**2+(x2-P0[1])**2)

    P0=[x1,x2]

    k=k+1

    print("{0:1d} \t {1:1.4f} \t {2:1.4f} ".format(k,x1,x2,magnitud))

    if sqrt(Y[0][0]**2+Y[1][0]**2)<TOL:

        print("exitoso")

        break
```

```
k         x1       x2       //x(k)-x(k-1)//
0         3.0000            4.0000
1         15.0000           20.0000
2         10.2000           13.6000
3         9.0706            12.0941
4         9.0003            12.0004
5         9.0000            12.0000
6         9.0000            12.0000
exitoso
```

UNI, 19 de Julio del 2023