



Cálculo Numérico

Atividade #4

Instruções:

- Entrega individual, via “Tarefas” do Teams e arquivo único em .pdf;
- Use este arquivo .docx para fazer sua atividade, e ao finalizar, gere o .pdf.
- Além de incluir os algoritmos no .pdf, eles devem ser upados em anexo, cada um individualmente e um arquivo txt;

- **Discente:** Daniel Marques da Silva

- 1) Desenvolva uma **função** genérica no Python para resolução do método de Gauss-Jacobi. Use como critério de parada o erro absoluto aproximado. Faça o pré-processamento com base no critério das linhas.

Resposta: Como uma das necessidades era realizar com antecedência o teste de convergência, no Projeto-Código foi realizado ele inicialmente antes da entrada dos valores iniciais para a resolução. Após a entrada dos dados, a cada interação temos uma saída estimada dos valores para X_1 , X_2 , X_3 . A última saída antes do tempo de processamento são os valores estimados mais próximos para a solução do sistema. Por fim é apresentado o número de interações necessárias para que o sistema tenha convergência conforme o erro indicado pelo o programador. Algumas considerações que podem ser feitas para esse é referente ao cálculo da matriz interna do processo que segue o modelo proposto por Rugierro&Lopes onde executamos o processo segundo $X_n = C * x_{n-1} + b$, onde foi mantida o sistema de calculo da matriz C dentro do processo, ou seja, a cada nova interação ela é recalculada. Pode ocorrer desse processo ser um desperdício de poder computacional, mas que por opção foi deixada desse modo, ao qual se necessária uma substituição da matriz fora dessa interação uma parte do processo já se encontra integrado ao código, além de nos testes realizados pouco houve de diferença de tempos.

```
bbi.py + C:\Program Files (x86)\Microsoft Visual Studio
Temos garantia de convergencia
x=
reDignite x0:.7
maDignite x0:-1.6
maDignite x0:.6
fo
[ 0.96 -1.86 0.94]
[ 0.978 -1.98 0.966]
[ 0.9994 -1.9888 0.9984]
5.853099999999995 ms
Numero de interações: 3
Press any key to continue . . .
```

Fig.1 – Saída de dados-resposta

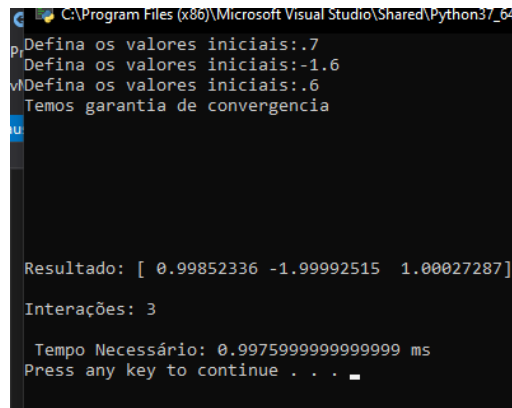
```
C=np.zeros((np.size(b),np.size(b)))
for i in range(np.size(b)):
    for j in range(np.size(b)):
        c[i,j]=-mtx[i,j]/mtx[i,i]
    c[i,i]=0
sol=np.zeros(np.size(b))
for i in range(np.size(b)):
```

Fig.2 – Cálculo referido para considerações

Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento.

- 2) Desenvolva uma **função** genérica no Python para resolução do método de Gauss-Seidel. Use como critério de parada o erro absoluto aproximado. Faça o pré-processamento com base no critério das linhas.

Resposta: Gauss-Seidel segue o mesmo princípio de cálculo que Gauss-Jacobi, todavia o seu diferencial é que a cada interação ele deve atualizar a matriz solução X, ou seja, da primeira para segunda interação, atualizamos a Matriz X com o valor calculado anteriormente, com isso adquirimos uma convergência mais ágil. Outra consideração feita aqui é adição do Pivotamento como forma de garantir uma Convergência, não que sem isso a solução já não pudesse convergir é apenas uma garantia de Teorema.



```
C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python37_64\
> Defina os valores iniciais:0.7
> Defina os valores iniciais:-1.6
> Defina os valores iniciais:0.6
> Temos garantia de convergencia

Resultado: [ 0.99852336 -1.99992515  1.00027287]

Interações: 3

Tempo Necessário: 0.9975999999999999 ms
Press any key to continue . . .
```

Fig.3 – Resultado por Gauss-Seidel

Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento.

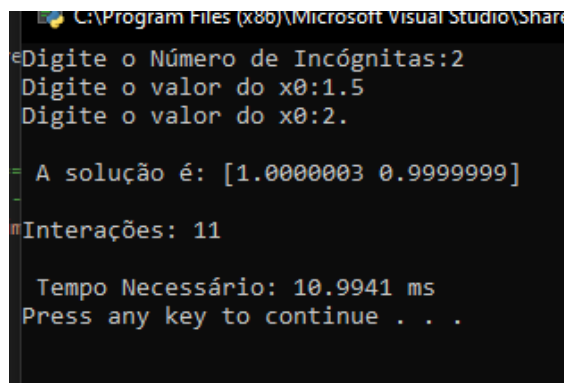
- 3) Elabore uma **função** genérica no Python para resolução do método de Newton-Raphson. Use como critério de parada o erro absoluto aproximado. No método iterativo, resolva como um sistema linear ao invés de inverter J.

Resposta: Conforme o Código descrito ao final deste relatório de atividade, foi elaborado uma solução pelo método de Newton para sistemas não lineares. Também conforme indicado, foi utilizado uma solução $Ax = B$ para solucionar a Jacobiana J, não sendo feita sua inversa (pois é necessário muito tempo computacional para tal). Como regra geral para esse, a solução por Newton é dada por $X_{i+1} = X_i - J * f_i$, onde a Jacobiana carrega as derivadas parciais de

F_1 e F_2 em relação a x_1 e x_2 para cada função, sendo um exemplo dado da seguinte maneira:

$$\begin{bmatrix} \frac{\partial F_1}{\partial x_1} & \frac{\partial F_1}{\partial x_2} \\ \frac{\partial F_2}{\partial x_1} & \frac{\partial F_2}{\partial x_2} \end{bmatrix}$$

Uma solução para um sistema proposto por Ruggiero (página 205, ex. 2.a) é o seguinte:



```
C:\Program Files (x86)\Microsoft Visual Studio\Share
> Digite o Número de Incógnitas:2
> Digite o valor do x0:1.5
> Digite o valor do x0:2.

A solução é: [1.0000003 0.99999999]

Interações: 11

Tempo Necessário: 10.9941 ms
Press any key to continue . . .
```

Fig.4 – Solução pelo método de Newton

Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento.

- 4) Elabore uma **função** genérica no Python para resolução do método de Newton Modificado (Ruggiero, página 200), que é uma modificação da matriz J do método anterior. O método modificado consiste se em tomar a cada iteração k a matriz $J(x^0)$ em vez de $J(x^k)$, isto é, a matriz J é mantida fixa.

Resposta: Usando a mesma função proposta para o exercício anterior, a implementação de Newton Modificado tomou seguinte resposta:

```

Digite o Número de Incógnitas:2
Digite o valor do x0:1.5
Digite o valor do x0:2.

A solução é: [1.0000008  1.00068783]

Interações: 21

Tempo Necessário: 2.012 ms
Press any key to continue . . .

```

Fig.5 – Newton Modificado

É interessante notar que o tempo necessário foi muito inferior ao método tradicional, contudo o número de interações foi maior. Isso demonstra que as atualizações dos valores estimados na Jacobiana tornam o sistema mais demorado mas menos cálculos são necessários, ao contrario da modificação que carrega mais cálculos para compensar essa fixação da Jacobiana.

Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento.

- 5) Resolva os seguintes sistemas lineares $[A]_{n \times n} [X]_{n \times 1} = [B]_{n \times 1}$ usando o método de Gauss-Jacobi e Gauss-Seidel. Use o erro absoluto aproximado de 10^{-6} (obs: não está em porcentagem) e condições iniciais igual a zero.

Resposta: Na solução de Jacobi para a matriz B temos uma condição não satisfeita na construção da solução trata-se de um zero na diagonal principal. Esse fator causa um erro que impossibilita a solução por tal método. Seidel não sofre do mesmo e conseguimos uma solução. As imagens a seguir mostram os resultados alcançados por cada Código.

| | Matriz A | Matriz B |
|---------------|---|--|
| Gauss- Jacobi | <pre> Temos garantia de convergencia Digite x0:0 Digite x0:0 Digite x0:0 [2.7 10.25 -4.3] [0.22 7.46666667 -6.89] [0.51766667 7.84333333 -5.83733333] [0.5476 8.04538889 -5.9722] [0.49370222 7.98546667 -6.01859778] [0.50104689 7.99694963 -5.99583378] [0.5010267 8.0008653 -5.9995993] [0.49986701 7.99962022 -6.0003784] [0.50003812 7.99994036 -5.99989745] [0.50002218 8.00001513 -5.9999957] [0.49999741 7.99999034 -6.00000746] [0.50000119 7.99999881 -5.99999755] 15.5854 ms Numero de interações: 12 Press any key to continue . . . </pre> | Não Possível, Zero na Diagonal Principal |

| | | |
|--------------|---|--|
| Gauss-Seidel | <pre> Defina os valores iniciais:0 Defina os valores iniciais:0 Defina os valores iniciais:0 Temos garantia de convergencia Resultado: [0.49999967 7.9999999 -5.99999991] Interações: 8 Tempo Necessário: 0.0 ms Press any key to continue . . . </pre> | <pre> Defina os valores iniciais:0 Defina os valores iniciais:0 Defina os valores iniciais:0 Temos garantia de convergencia Resultado: [0.35588236 3.21470588 0.48529412] Interações: 7 Tempo Necessário: 0.0 ms Press any key to continue . . . </pre> |
|--------------|---|--|

Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento.

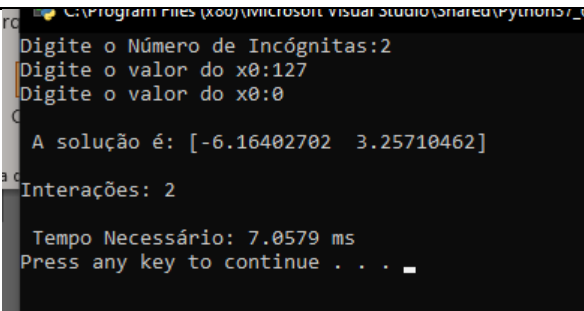
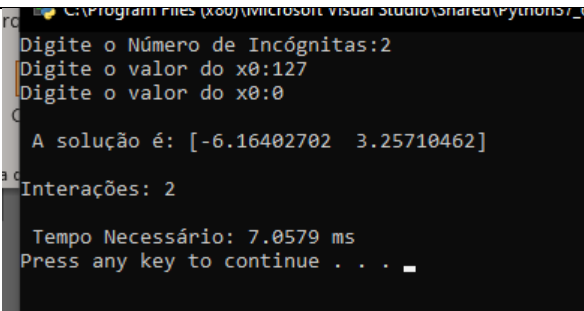
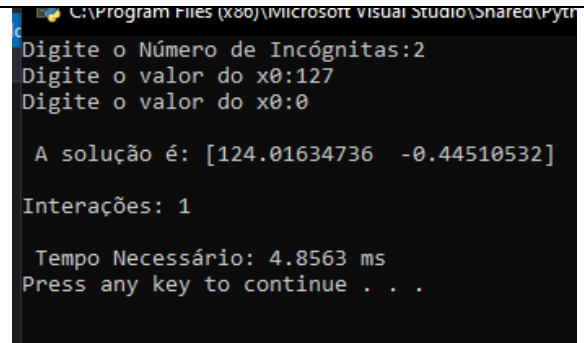
6) Resolva e compare (se convergiu, tempo e iterações) os seguintes sistemas usando o método de Gauss-Jacobi; Gauss-Seidel e Newton-Raphson e Newton-Raphson modificado. Use o erro absoluto aproximado de 10^{-6} .

| Resposta: | | |
|-------------------|---|---|
| | Matriz A | Matriz B |
| Gauss Jacobi | Não convergiu | Não convergiu |
| Gauss Seidel | Não convergiu | Não convergiu |
| Newton | <pre> Digite o Número de Incógnitas:2 Digite o valor do x0:1.2 Digite o valor do x0:1.2 A solução é: [1.4684644 -0.05794877] Interações: 5 Tempo Necessário: 8.3394 ms Press any key to continue . . . </pre> | <pre> Digite o Número de Incógnitas:2 Digite o valor do x0:1.5 Digite o valor do x0:1.5 A solução é: [1.60048518 1.56155281] Interações: 3 Tempo Necessário: 3.9943999999999997 ms Press any key to continue . . . </pre> |
| Newton Modificado | <pre> Digite o Número de Incógnitas:2 Digite o valor do x0:1.2 Digite o valor do x0:1.2 A solução é: [-0.08329747 0.09602468] Interações: 11 Tempo Necessário: 9.993599999999999 ms O thread 'MainThread' (0x1) foi fechado com o código 0 (0x0). O programa "python.exe" foi fechado com o código 0 (0x0). </pre> | <pre> Digite o Número de Incógnitas:2 Digite o valor do x0:1.5 Digite o valor do x0:1.5 A solução é: [1.60048398 1.56155278] Interações: 4 Tempo Necessário: 1.9987 ms Press any key to continue . . . </pre> |

Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento.

7) Considere o circuito abaixo. Calcule a tensão fatorial sobre a carga (nó 2). Observe que a impedância da carga é desconhecida e o circuito é monofásico. Compare os resultados utilizando Gauss-Jacobi, Gauss-Seidel e Newton-Raphson original e modificado. Considere a condição inicial $V = 127\text{ V}$ e $\theta = 0\text{ rad}$ e critério de parada $\epsilon = 0,0001\%$.

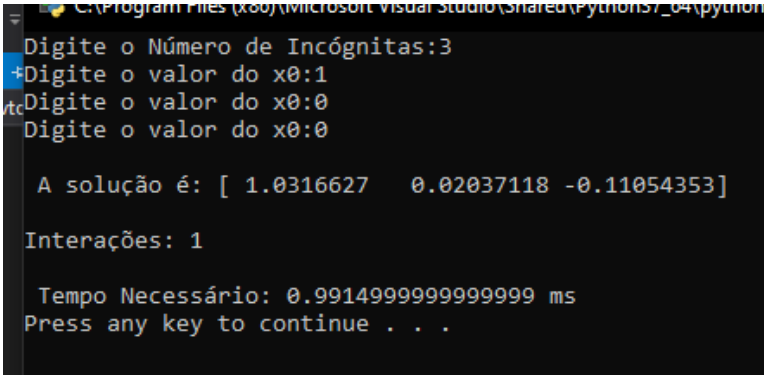
| Resposta: | |
|-----------|----------------------|
| | Resultado Adquiridos |

| | |
|-------------------|---|
| Gauss-Jacobi |  <pre> C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python7_0\python Digite o Número de Incógnitas:2 Digite o valor do x0:127 Digite o valor do x0:0 A solução é: [-6.16402702 3.25710462] Interações: 2 Tempo Necessário: 7.0579 ms Press any key to continue . . . </pre> |
| Gauss-Seidel | *Provável erro de implementação |
| Newton |  <pre> C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python7_0\python Digite o Número de Incógnitas:2 Digite o valor do x0:127 Digite o valor do x0:0 A solução é: [-6.16402702 3.25710462] Interações: 2 Tempo Necessário: 7.0579 ms Press any key to continue . . . </pre> |
| Newton Modificado |  <pre> C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python7_0\python Digite o Número de Incógnitas:2 Digite o valor do x0:127 Digite o valor do x0:0 A solução é: [124.01634736 -0.44510532] Interações: 1 Tempo Necessário: 4.8563 ms Press any key to continue . . . </pre> |

Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento.

- 8) Considere uma rede elétrica de três barras (1, 2 e 3), e linhas de transmissão conectado barra 1 com 2, 2 com 3, e 1 com 3. São conhecidas: a potência ativa é nas barras 2 e 3, a potência reativa na 3, o módulo da tensão na 1 e 3, e ângulo na 1. O...

Resposta:



```

C:\Program Files (x86)\Microsoft Visual Studio\Shared\Python7_0\python
Digite o Número de Incógnitas:3
Digite o valor do x0:1
Digite o valor do x0:0
Digite o valor do x0:0

A solução é: [ 1.0316627  0.02037118 -0.11054353]

Interações: 1

Tempo Necessário: 0.9914999999999999 ms
Press any key to continue . . .

```

Figura 6 – Solução por Newton

```

Digite o Número de Incógnitas:3
Digite o valor do x0:1
Digite o valor do x0:0
Digite o valor do x0:0

A solução é: [ 1.0316627  0.02037118 -0.11054353]

Interações: 1

Tempo Necessário: 1.0241 ms
Press any key to continue . . .

```

Figura 7 – Solução por Newton Modificado

Como é de se observar nas figuras acima, os dois métodos apresentaram o mesmo resultado de saída, sendo apenas a diferença entre eles o tempo necessário para solução, sendo aproximadamente 0.02 ms de diferença, algo pouco significativo para o sistema solucionado.

Obs. Os Projetos-Códigos encontram-se .txt anexo a esse documento.

Lista de Códigos

```

#####
### EXERCÍCIO 1

#=====
# ===== Gauss-Jacobi =====
#####
# Transform  $Ax=B$  on  $x=Cx+g$ 
# Autor: Daniel Marques
# Electrical Engeneering - 2021
#####
#=====

import numpy as np
import time
# ===== Space for Functions =====
def GaussJacobino(mtx,x,b,mak):
    c=np.zeros([np.size(b),np.size(b)])
    for i in range(np.size(b)):
        for j in range(np.size(b)):
            c[i,j]=-mtx[i,j]/mtx[i,i]
        c[i,i]=0
    sol=np.zeros(np.size(b))
    for i in range(np.size(b)):
        sol=(c@x)
    for i in range(np.size(b)):
        sol[i]=sol[i]+(b[i]/mtx[i,i])
        if abs(sol[i])>mak:
            mak=abs(sol[i])
    return sol,mak
# ===== Space for Input =====

matrix=np.array([[10, 2, 1],[1, 5, 1],[2, 3, 10]]) # Linear System
b=np.array([7, -8, 6]) # B matrix
x=np.zeros(np.size(b))
resolut=[]
max=k=0
mak=0

```

```

for i in range(np.size(b)):
    alp=0
    for j in range(np.size(b)):
        if j!=i:
            alp=alp+matrix[i,j]
        if (alp/matrix[i,i])>max:
            max=alp/matrix[i,i]
if max<1:
    print('Temos garantia de convergencia')
for i in range(np.size(b)):
    x[i]=float(input('Digite x0:'))
# ===== Error Definition =====
error = .05
e=2*error
# ===== Main Loop/Output =====
start=time.time_ns()
while (error<e):
    resolut,mak=GaussJacobino(matrix,x,b,mak)
    max=0
    print(resolut)                                #Solution of Iteration
    for i in range(np.size(b)):
        a=abs(resolut[i]-x[i])
        if a>max:
            max=a

    for i in range(np.size(x)):
        x[i]=resolut[i]                            #Update the Resolution X
    e=max/mak                                        #Error atualization
    k+=1                                             #Number of iterations
end=time.time_ns()
print((end-start)*10**-6,'ms')
print('Numero de interações:',k)

#####
### EXERCÍCIO 2

#=====
# ===== Gauss-Seidel =====
#####
# Similar of Gauss-Jacobi, but more faster and use the before Xn
# Autor : Daniel Marques
# Electrical Engineering - 2021
#####
#=====

import numpy as np
import time
# ===== Space for Functions =====

def pivot (mtx,l,k):
    i=0
    while (i+1)<k:
        if np.abs(mtx[l+i,l])>np.abs(mtx[l,l]):
            a=mtx[l].copy()
            mtx[l]=mtx[l+i]
            mtx[l+i]=a
        i+=1
    return mtx
def GaussSeidel(mtx,x,b,mak):
    c=np.zeros([np.size(b),np.size(b)])
    sol=np.zeros(np.size(b))

    for i in range(np.size(b)):
        for j in range(np.size(b)):
            c[i,j]=-mtx[i,j]/mtx[i,i]
            c[i,i]=0
        # Construction of Matrix C
        #
        #
    for i in range(np.size(b)):

```

```

a=0
for j in range(np.size(b)):
    a+=c[i,j]*x[j]                                #Calc for Xn

sol[i]=a+(b[i]/mtx[i,i])
x[i]=sol[i]                                        #Update the matrix X for interactions

if abs(sol[i])>mak:
    mak=abs(sol[i])                                # Take the major valor absolute
return sol,mak                                     #

# ===== Space for Input =====
matrix=np.array([[10, 2, 1],[1, 5, 1],[2, 3, 10]]) #Linear System
b=np.array([7, -8, 6])                             #B matrix
x=np.zeros(np.size(b))
aux=np.zeros(np.size(b))
resolut=[]
mak=max=0
alp=n=0
# ===== Error Definition =====
Error = .05
e=2*Error
# ===== Main Loop/Output =====
for i in range(np.size(b)):
    x[i]=float(input('Defina os valores iniciais:'))
    aux[i]=x[i]
start=time.time_ns()
i=1
k=np.size(b)
while(i<k):
    matrix=pivot(matrix,i,k)
    i+=1
for i in range(k):
    alp=0
    for j in range(k):
        if j!=i:
            alp=alp+matrix[i,j]
        if (alp/matrix[i,i])>max:
            max=alp/matrix[i,i]
if max<1:
    print('Temos garantia de convergencia')
while(e>Error):
    resolut,mak=GaussSeidel(matrix,x,b,mak)
    max=0
    print("\n")
    for i in range(k):
        a=abs(resolut[i]-aux[i])
        if a>max:
            max=a
    e=max/mak
    for i in range(np.size(x)):
        aux[i]=resolut[i]
    n+=1
end=time.time_ns()
print('\nResultado:',x)
print('\nInterações:',n)
print('\n Tempo Necessário:',(end-start)*10**-6,'ms')

# ===== Space for Plots =====
#####

### EXERCÍCIO 3

#=====
# ===== Método de Newton =====
#####
# Considerations of project

```



```

# Autor : Daniel Marques
# Electrical Engineering - 2021
#####
#=====

import numpy as np
import time
import math as mt
# ===== Space for Functions =====
def modNewton(x,delta,n):
    Jac = np.zeros([n,n])
    Fi = np.zeros(n)

    Jac[0,0] = 2*x[0]
    Jac[0,1] = 2*x[1]
    Jac[1,0] = mt.exp(x[0]-1)
    Jac[1,1] = 3*x[1]**2

    Fi[0] = x[0]**2+x[1]-2
    Fi[1] = mt.exp(x[0]-1)+x[1]**3-2

    delta = np.linalg.solve(Jac,Fi)

    return delta
# ===== Space for Input =====
#----- User Input -----
n = int(input('Digite o Número de Incógnitas:'))
sol = []
aux = np.zeros(n)
x = np.zeros(n)
mak = 0
for i in range(n):
    x[i]= float(input('Digite o valor do x0:'))
    aux[i] = x[i]
# Definitions of the equation
# F1 = x^2+y-2=0;
# F2 = exp^(x-1)+y^3-2=0
# ===== Error Definition =====
Error = 10**(-6)
e = 100
# ===== Main Loop/Output =====
start = time.time_ns()
while (Error<e):

    sol = modNewton(x,sol,n)

    for i in range(n):
        aux[i] = x[i]-sol[i]
        if(abs(aux[i]-x[i])/aux[i])<e :
            e = (abs(aux[i]-x[i])/aux[i])
        x[i] = aux[i]

    mak+=1
stop = time.time_ns()
print('\n A solução é:',x)
print('\nInterações:',mak)
print('\n Tempo Necessário:',(stop-start)*10**-6, 'ms')

#####
### EXERCÍCIO 4

#=====
# ===== Método de Newton Modificado =====

```

```
#####
# Considerations of project
# Autor : Daniel Marques
# Electrical Engineering - 2021
#####
#=====

import numpy as np
import time
import math as mt
# ===== Space for Functions =====
def modNewton(x,delta,Jac,n):
    Fi = np.zeros(n)

    Fi[0] = x[0]**2+x[1]-2
    Fi[1] = mt.exp(x[0]-1)+x[1]**3-2

    delta = np.linalg.solve(Jac,Fi)

    return delta
# ===== Space for Input =====
#----- User Input -----
n = int(input('Digite o Número de Incógnitas:'))
sol = []
aux = np.zeros(n)
x = np.zeros(n)
mak = 0
Jac = np.zeros([n,n])
for i in range(n):
    x[i]= float(input('Digite o valor do x0:'))
    aux[i] = x[i]
# Definitions of the equation
# F1 = x^2+y-2=0;
# F2 = exp^(x-1)+y^3-2=0
# ===== Error Definition =====
Error = 10**(-6)
e = 2*Error
# ===== Main Loop/Output =====
start = time.time_ns()
Jac[0,0] = 2*x[0]
Jac[0,1] = 2*x[1]
Jac[1,0] = mt.exp(x[0]-1)
Jac[1,1] = 3*x[1]**2

while (Error<e):

    sol = modNewton(x,sol,Jac,n)

    for i in range(n):
        aux[i] = x[i]-sol[i]
        if(abs(aux[i]-x[i])/aux[i])<e :
            e = (abs(aux[i]-x[i])/aux[i])
            x[i] = aux[i]

    mak+=1
stop = time.time_ns()
print('\n A solução é:',x)
print('\nInterações:',mak)
print('\n Tempo Necessário:',(stop-start)*10**-6, 'ms')
```