TRABALHO COMPUTACIONAL 2

Nomes:

- Daiana Santos 120.357
- Isadora Muniz 120.431
- Luciana Bello 120.506
- Maria Victória Siqueira 120.529

PARTE 1

1. Temperatura no interior de materiais

Para resolver a equação, o primeiro método a ser testado foi o método de Newton, porém não foi possível obter o resultado, o segundo método a ser testado foi o método de Heron que também não foi possível chegar no resultado, o terceiro método foi o método da secante, na qual obteve o resultado da temperatura igual a 0,514965..

Código em Python:

```
#METODO DA SECANTE
import numpy as np
def CalculoFuncao(t1):
    f0 = ((np.exp((-0.5)*t1)) * (np.arccosh(np.exp((0.5)*t1)))) - np.sqrt(0.67/2)
    return(f0)

t0 = 1 # valor inicial
t1 = 2 # valor inicial
cont = 1 #variável contadora de interações

while(True):# laço para iterações de aproximação da raiz
    F0 = CalculoFuncao(t0)
    F1 = CalculoFuncao(t1)
    T = t1 - F1 * ((t1 - t0)/(F1 - F0))

    if(abs(CalculoFuncao(T)) < 10**-6): #condição de parada da iteração -> valor da
função x mais proximo de zero
        print(T)
        break
    t0 = t1 #atualização do valor de t0
    t1 = T #atualização do valor de t1
    cont += 1 #incremento do numero de iterações

print(cont)
```

2. Método de Newton em Sistemas Não-Lineares:

A partir da implementação do Método de Newton para Sistemas Não-Lineares, foi feita nas primeiras etapas do programa a linearização das equações possibilitando assim a adaptação do problema para os moldes de um sistema linear. Como foi feito do Trabalho Computacional 1, o Método de Newton usou a função original e a sua primeira derivada para encontrar soluções aproximadas em algumas iterações. Logo, o programa em Python soluciona o problema desta forma:

```
import numpy as np #importação da biblioteca que contem as funções de resolução do sistema

def F(x):   #função que define F(x), do qual contém as equações do sistema

    y = np.zeros(2)
    y[0] = x[0]**3 + 3*x[1]**2 - 21
    y[1] = x[0]**2 + 2*x[1] - 2
    return y


def JF(x):   #função que define JF(x), do qual contém a derivada das equações do sistema que estão em F(x)

    y = np.zeros((2,2))
    y[0,0] = 3*x[0]**2
    y[0,1] = 6*x[1]
    y[1,0] = 2*x[0]
    y[1,1] = 2
    return y

def MetodoNewton(F,JF,x0,TOL,N):
    x = np.copy(x0).astype('double') #inicialização do vetor x e contador k
    k=0
    while (k < N):    #laço
       k += 1
       delta = -np.linalg.inv(JF(x)).dot(F(x))  #resolução do metodo
       x = x + delta
       if (np.linalg.norm(delta,np.inf) < TOL): #condição de parada
           return x

    raise NameError('num. max. iter. excedido.') #condição de erro

x0 = np.array([1.0,-1.0])  #valores iniciais
print(MetodoNewton(F, JF, x0, 10**-6, 10)) # impressão dos valores da raizes do sistema (iguais a [ 2.31829366 -1.68724275])
```

3. Busca por Duas Soluções em Sistemas Não-Lineares:

Segue programa em Python para encontrar ao menos duas soluções que sejam próximas à origem para sistemas não-lineares:

```
import numpy as np #importação da biblioteca que contem as funções de resolução do
sistema
import math

def F(x):   #função que define F(x), do qual contém as equações do sistema

    y = np.zeros(2)
    y[0] = x[0]**2 + x[0] - x[1]**2 - 1
    y[1] = -math.sin(x[0]**2) + x[1]
    return y


def JF(x):   #função que define JF(x), do qual contém a derivada das equações do
sistema que estão em F(x)

    y = np.zeros((2,2))
    y[0,0] = 2*x[0] + 1
    y[0,1] = -2*x[1]
    y[1,0] = -math.cos(x[0]**2)*2*x[0]
    y[1,1] = 1
    return y

def MetodoNewton(F,JF,x0,TOL,N):
    x = np.copy(x0).astype('double') #inicialização do vetor x e contador k
    k=0
    while (k < N):    #laço
       k += 1
       delta = -np.linalg.inv(JF(x)).dot(F(x))  #resolução do metodo
       x = x + delta
       if (np.linalg.norm(delta,np.inf) < TOL): #condição de parada
           return x

    raise NameError('num. max. iter. excedido.') #condição de erro

x0 = np.array([1.0,-1.0])  #valores iniciais
print(MetodoNewton(F, JF, x0, 10**-6, 10)) # impressão dos valores da raizes do
sistema (iguais a [0.72595073 0.5029465 ] )
```

4. Sistema LORAN

A partir da equação dada em enunciado, é calculado a posição de barcos de acordo com os pontos de localização gerando um mapa 2D. Sabendo disso, foi feito com o Método de Newton apresentado em exercícios anteriores a linearização das equações seguida das soluções. Dessa forma, o programa em Python resolve o problema abaixo:

```
import numpy as np
import math

def Funcao(x,y): #função que define F(x), do qual contém as equações do sistema
    f0 = x**2/(186)**2 + y**2/(300**2 - 186**2) - 1
    f1 = (y - 500)**2/279**2 - (x - 300)**2/(500**2 - 279**2) - 1
    return f0,f1

s1 = 1
s2 = 2

x = -500  #valor inicial
y = 100 #valor inicial

while(min(abs(s1), abs(s2)) > (10**-6) and min(abs(Funcao(x,y) [0]), abs(Funcao(x,y)
[1])) > (10**-6)):  #atribuições das funções iniciais derivadas
    derivF0dx = 2/(186)**2 * x
    derivF0dy = -1*(2/(300**2 - 186**2))*y
    derivF1dx = -2*y + 600/(500**2 - 279**2)
    derivF1dy = 2*x-1000/279**2
    jx = np.array([[derivF0dx, derivF0dy], [derivF1dx, derivF1dy]])
    fx = np.array([Funcao(x,y) [0], Funcao(x,y) [1]])
    mt = np.linalg.solve(jx,np.negative(fx))
    s1 = mt[0]
    s2 = mt[1]
    x = x + s1
    y = y + s2
    x0 = np.array([x,y])
print(x0) # impressão dos valores da raizes do sistema (iguais a [-178.31859115
66.94550288] )
```

PARTE 2

1. Método de Eliminação Gaussiana

```
import numpy as np

n = 4 # O NUMERO DE EQUACOES ###### ENTRADA #######
matriz = []
x = []
for i in range(n):
    matriz.append([0]*n)

b = np.zeros(n) #INICIALIZA COM ZERO

matriz = [1, -1, 0, 5], [3, -2, 1, -1], [1, 1, 9, 4], [1, -7, 2, 3] #A MATRIZ ######
ENTRADA #######

b = [18, 8, 47, 32] # A MATRIZ AUMENTADA ###### ENTRADA #######

########METODO DA ELIMINACAO############
for i in range(0, n-1):
    for j in range(i+1, n):
        m = matriz[j][i]/matriz[i][i]
        matriz[j][i] = 0;
        for k in range(i+1, n):
            matriz[j][k] = matriz[j][k] - (m * matriz[i][k])
        b[j] = b[j] - m*b[i]


print("A Matriz final ficou:")
print(matriz) #MATRIZ TRIANGULAR SUPERIOR ###### SAIDA ####### U=([1, -1, 0, 5], [0,
1.0, 1.0, -16.0], [0, 0, 7.0, 31.0], [0, 0, 0, -133.42857142857142])

x = np.linalg.solve(matriz, b)
print("O vetor solucao eh:")
print(x) #VETOR SOLUCAO ###### SAIDA ####### x = [ 1. -2.  4.  3.]
```

2. Problema da Fábrica

Sabendo que uma fábrica procura gastar todo o seu estoque fabricando o máximo de mesas, cadeiras e armários, para solucionar o problema foi montado duas equações lineares. Dessa maneira, o programa em Python abaixo representa a solução:

```
import numpy as np

a = np.array([[1,1,1], [1, 1, 2], [2, 3, 5]]) #primeira equacao correspondente a cadeira, segunda equacao correspondente a mesa e terceira equacao correspondente ao armario
b = np.array([400, 600, 1500]) #valores disponiveis dos materiais
x = np.linalg.solve(a, b)#função de resolução do sistema
print(x) #impressao do resultado correspondente a 100 cadeiras, 100 mesas e 200 armarios
```