
# UNIVERSIDADE FEDERAL DA PARAÍBA
# DEPARTAMENTO DE INFORMÁTICA - CI
# CÁLCULO NUMÉRICO
# TRABALHO

Gabriel Lucena Camboim

6 de fevereiro de 2020

# Sumário

**Sumário** . . . . . **2**
1 Introdução . . . . . 3
2 Questão 1(3) . . . . . 3
3 Questão 2(3) . . . . . 6
4 Questão 4(2) . . . . . 8
5 Questão 6 . . . . . 9
6 Questão 8(3) . . . . . 11
7 Questão 9(2) . . . . . 12
8 Questão 10(3) . . . . . 16

## 1 Introdução

Foi resolvida as questões do Grupo 3 como designado, todas utilizando a linguagem Python de programação e a biblioteca MatPlotLib para plotagem de gráficos das funções. Os métodos vão sendo explicados em resumo no decorrer de cada questão.

| Grupo 3 | 1(3) | 2(3) | 4(2) | 6 | 8(3) | 9(2) | 10(3) |
|---|---|---|---|---|---|---|---|

Figura 1 – Grupo de questões

## 2 Questão 1(3)

$$1 - \left( \left( \frac{20^2}{9.81\left( \left( 3y + \left( \frac{y^2}{2} \right) \right)^3 \right)} \right) (3 + y) \right)$$

Figura 2 – Função utilizada

A profundidade crítica é a única incógnita não fornecida na questão, logo, para encontrar, é necessário substituir os valores que ele fornece na questão.

```python
from matplotlib import pyplot as plt

Q = 20.0
g = 9.81

def A(y): return (3.0 * y) + ((y ** 2)/2.0)

def B(y): return 3.0 + y

def f(y): return 1 - (((Q ** 2.0) / (g * A(y) ** 3.0)) * B(y))

pontos = 100

a, b = 1, 5
incremento = (abs(a)+abs(b)) / pontos

valores, posicoes = [], []
n = 1
i = 0
while i < pontos:
    posicoes.append(n)
    valores.append(f(n))
    n += incremento
    i += 1

plt.axhline(y=0, ls='--', color='r')
plt.plot(posicoes, valores, color='b')
plt.show()
```

Figura 3 – Código do matlab

Na questão 1(3) letra (a), foi feita a plotagem do gráfico da função fornecida, substituindo os valores, deixando apenas o 'Y' de incógnita. O resultado é possível ver na figura abaixo (Note que a raiz está próxima do valor x = 1.8).

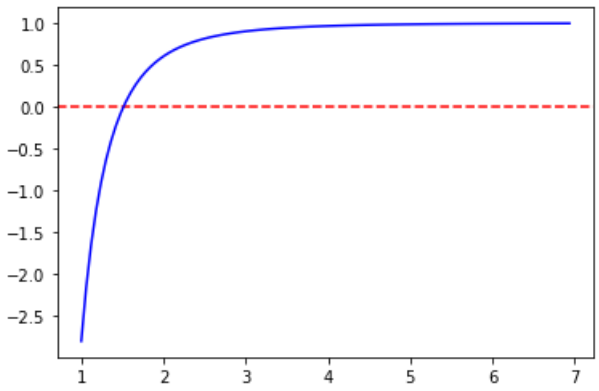


Figura 4 – Gráfico gerado

Para letra (b), é pedido a implementação do Método da bisseção, onde é requerido um intervalo inicial (Já dado na questão), a função (feita acima), além também do erro. O método é iterativo e baseado na média dos intervalos, visando diminuir a distância entre o valor baseado no erro.

```
def bissecao(f, x0, x1, erro):

    f = eval('lambda x:' + f)
    fx0 = f(x0)
    fx1 = f(x1)

    done = 0

    xm = (x1+x0)/2
    it = 1

    while abs(x0-x1) > erro and (not done):
        fxm = f(xm)
        print("(it = {0:d}) f(xm)={1:f}  | f(x0)={2:f}  | f(x1)={3:f}".format(
            it, fxm, fx0, fx1))

        if fx0*fxm < 0:          # Raiz está à esquerda
            x1 = xm
            fx1 = fxm
            xm = (x0+x1)/2
        elif fxm*fx1 < 0:       # Raiz está à direita
            x0 = xm
            fx0 = fxm
            xm = (x0+x1)/2
        else:                   # Raiz encontrada
            done = 1
        it += 1

    return xm


f = '1-(((20**2)/(9.81*((3*x + ((x**2)/2))**3)))*(3+x))'

bissecao(f, 0.6, 2.6, 0.001)
```

Figura 5 – Método da bisseção

O resultado pode ser visto na figura abaixo, onde após 11 iterações foi possível definir um resultado.

```
(it = 1) f(xm)=0.165477  | f(x0)=-17.910278  | f(x1)=0.836599
(it = 2) f(xm)=-1.807448  | f(x0)=-17.910278  | f(x1)=0.165477
(it = 3) f(xm)=-0.452469  | f(x0)=-1.807448  | f(x1)=0.165477
(it = 4) f(xm)=-0.089091  | f(x0)=-0.452469  | f(x1)=0.165477
(it = 5) f(xm)=0.049022  | f(x0)=-0.089091  | f(x1)=0.165477
(it = 6) f(xm)=-0.017035  | f(x0)=-0.089091  | f(x1)=0.049022
(it = 7) f(xm)=0.016704  | f(x0)=-0.017035  | f(x1)=0.049022
(it = 8) f(xm)=0.000017  | f(x0)=-0.017035  | f(x1)=0.016704
(it = 9) f(xm)=-0.008463  | f(x0)=-0.017035  | f(x1)=0.000017
(it = 10) f(xm)=-0.004212  | f(x0)=-0.008463  | f(x1)=0.000017
(it = 11) f(xm)=-0.002095  | f(x0)=-0.004212  | f(x1)=0.000017
1.51357421875
```

Figura 6 – Resposta da letra b

Já na letra (c), é pedido com os mesmos intervalos, porém com uma precisão diferente, para ser implementado, a mesma estrutura, porém utilizado outro método.

```python
def falsa_posicao(f, x0, x1, erro):

    f = eval('lambda x:' + f)
    x2 = x1
    it = 0

    while (min(abs(x1-x0), abs(x2)) > erro):
        it += 1

        x2 = ((f(x1) * x0)-(f(x0) * x1)) / (f(x1) - f(x0))
        print("(it = {0:d}) f(x2)={1:f}  | x2={2:f}  | erro={1:f}".format(
            it, f(x2), x2, abs(x2-x1)))

        if f(x2) == 0:
            print(f"Resultado: {x2} // iterações: {it}  // precisão: {erro}")
            return x2
        elif abs(x2-x1) < erro:
            x0 = x2
        else:
            x1 = x2
    else:
        print(f"Resultado: {x2} // iterações: {it}  // precisão: {erro}")
        return x2


f = '1-(((20**2)/(9.81*((3*x + ((x**2)/2))**3)))*(3+x))'

falsa_posicao(f, 0.6, 2.6, 0.0001)
```

Figura 7 – Método da falsa posição

O método da falsa posição implementado requer os mesmos parâmetros do método da bisseção, porém trabalha de forma diferente pois parte do pressuposto que há uma raiz no intervalo mencionado.

```
(it = 45) f(x2)=0.018583  | x2=1.522765  | erro=0.018583
(it = 46) f(x2)=0.016564  | x2=1.521809  | erro=0.016564
(it = 47) f(x2)=0.014762  | x2=1.520957  | erro=0.014762
(it = 48) f(x2)=0.013154  | x2=1.520199  | erro=0.013154
(it = 49) f(x2)=0.011719  | x2=1.519523  | erro=0.011719
(it = 50) f(x2)=0.010439  | x2=1.518922  | erro=0.010439
(it = 51) f(x2)=0.009298  | x2=1.518387  | erro=0.009298
(it = 52) f(x2)=0.008280  | x2=1.517910  | erro=0.008280
(it = 53) f(x2)=0.007374  | x2=1.517486  | erro=0.007374
(it = 54) f(x2)=0.006566  | x2=1.517109  | erro=0.006566
(it = 55) f(x2)=0.005846  | x2=1.516772  | erro=0.005846
(it = 56) f(x2)=0.005205  | x2=1.516473  | erro=0.005205
(it = 57) f(x2)=0.004633  | x2=1.516207  | erro=0.004633
(it = 58) f(x2)=0.004125  | x2=1.515970  | erro=0.004125
(it = 59) f(x2)=0.003672  | x2=1.515759  | erro=0.003672
(it = 60) f(x2)=0.003268  | x2=1.515572  | erro=0.003268
(it = 61) f(x2)=0.002909  | x2=1.515404  | erro=0.002909
(it = 62) f(x2)=0.002589  | x2=1.515256  | erro=0.002589
(it = 63) f(x2)=0.002304  | x2=1.515124  | erro=0.002304
(it = 64) f(x2)=0.002051  | x2=1.515006  | erro=0.002051
(it = 65) f(x2)=0.001825  | x2=1.514901  | erro=0.001825
(it = 66) f(x2)=0.001624  | x2=1.514808  | erro=0.001624
Resultado: 1.5148078259634798 // iterações: 66  // precisão: 0.0001
1.5148078259634798
```

Figura 8 – Resposta da letra c

Note que na resposta acima, foi omitida as primeiras iterações para não ocupar muito espaço na página.

## 3 Questão 2(3)

Para a segunda questão, é questionado o tempo caso os suburbanos ultrapassassem em quarenta e cinco por cento a cidade, ou seja, Ps(t)=1.45*Pu(t). Substituindo os valores mantendo apenas o ’t’ como incógnita, ficamos com a função da figura abaixo:

$$\left(\frac{320000}{\left(1+\left(31e^{(-0.09x)}\right)\right)}\right)-1.45\left(\left(80000\left(e^{(-0.05x)}\right)\right)+110000\right)$$

Figura 9 – Função da questão

Para letra (a), novamente foi feita a plotagem utilizando o MatPlotLib

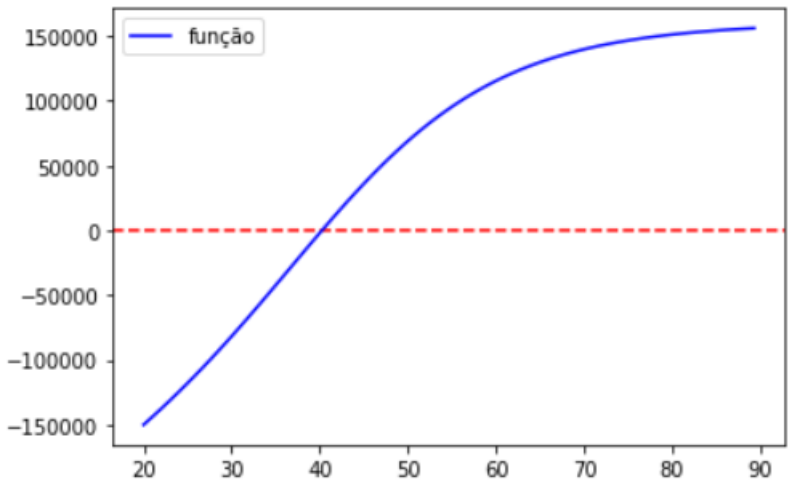


Figura 10 – Gráfico da função

Para a letra (b), foi aproveitado o método da falsa posição anteriormente implementado, onde foi jogado os valores fornecidos do problema como parâmetro da função. O intervalo escolhido foi selecionado a partir de onde a função converge, onde f(a)*f(b)<0.

```
(it = 1) f(x2)=14193.025490  | x2=42.063664  | erro=14193.025490
(it = 2) f(x2)=-1947.774337  | x2=40.000065  | erro=-1947.774337
(it = 3) f(x2)=317.085551  | x2=40.285961  | erro=317.085551
(it = 4) f(x2)=-50.513328  | x2=40.239490  | erro=-50.513328
(it = 5) f(x2)=8.075733  | x2=40.246895  | erro=8.075733
(it = 6) f(x2)=-1.290363  | x2=40.245711  | erro=-1.290363
(it = 7) f(x2)=0.206196  | x2=40.245900  | erro=0.206196
Resultado: 40.245900276138364 // iterações: 7  // precisão: 0.001
40.245900276138364
```

Figura 11 – Resposta da letra b

Já na letra (c), foi implementado o método de newton, o qual faz uso da derivada da função em sua fórmula iterativa para definir a raiz.

```
def newton(x0, f, df, erro):

    f = eval('lambda x:' + f)
    df = eval('lambda x:' + df)

    it = 0

    print('Estimativa inicial: x0 = {0}\n'.format(x0))

    while(1):
        it += 1

        x = x0 - f(x0)/df(x0)  # Função de iteração

        e = abs(x-x0)/abs(x)  # Erro

        print('{0:d}  {1:f}  {2:f}  {3:f}  {4:e}'.format(it, x, f(x), df(x), e))

        if(e <= erro):
            break

        x0 = x

    print('Solução obtida: x = {0:.10f}'.format(x))

    return x


df = '(5800*np.exp(-0.05*x)) + (((892800*np.exp(-0.09*x))) / (((31*np.exp(-0.09*x)) + 1))**2)'

f = '(320000/(1+(31*np.exp(-0.09*x)))) - 1.45*((80000*(np.exp(-0.05*x)))+110000)'

newton(5, f, df, 0.001)
```

Figura 12 – Método de newton

O resultado pode ser verificado na figura abaixo.

```
Estimativa inicial: x0 = 5

1  45.162229  37213.593637  7135.467363  8.892880e-01
2  39.946930  -2369.555478  7940.437272  1.305557e-01
3  40.245347  -4.175058  7912.065104  7.414925e-03
4  40.245874  -0.000014  7912.012861  1.311147e-05
Solução obtida: x = 40.2458742132
40.24587421320161
```

Figura 13 – Resposta letra c

## 4 Questão 4(2)

Para a letra (a), foi plotada a função fornecida no enunciado da questão, onde resultou na figura abaixo.

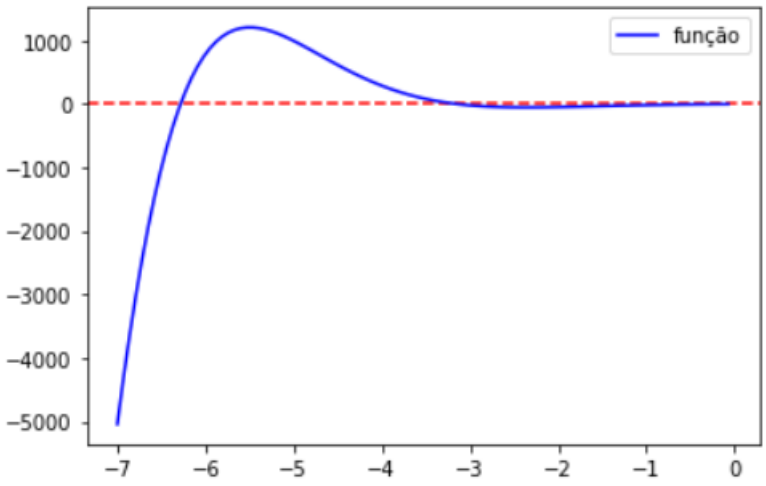


Figura 14 – Resposta letra a

Já na letra (b), foi re-aproveitado a implementação do método de Newton.

```
Estimativa inicial: x0 = -4

1  -3.465286  70.215553  -283.487001  1.543060e-01
2  -3.217600  12.271585  -187.543888  7.697826e-02
3  -3.152167  0.731078  -165.428763  2.075814e-02
4  -3.147748  0.003192  -163.985071  1.403954e-03
5  -3.147728  0.000000  -163.978725  6.184619e-06
Solução obtida: x = -3.1477283471
-3.1477283471346684
```

Figura 15 – Resposta letra b

E na letra (c) foi implementado o método das secantes que usa uma sequência de raízes de linhas secantes para aproximar cada vez melhor a raiz da função. Foi utilizado os parâmetros dados na questão.

```python
from math import sin, cos, e

#Método da Secante

def secante(x0,x1,f,erro):

    f = eval('lambda x:' + f)

    it = 0

    print('Estimativas iniciais: x0 = {0}; x1 = {1} \n'.format(x0,x1))

    while(1):
        it+=1
        x = (x0*f(x1) - x1*f(x0))/(f(x1) - f(x0))

        e = abs(x-x1)/abs(x)  # erro

        print('it={0:d}  x={1:f} f(x)={2:f}  erro={3:e}'.format(it,x,f(x),e))

        x0 = x1
        x1 = x

        if(e <= erro):
            break

    print('Solução obtida: x = {0:.10f}'.format(x))

    return x

f = '(7 * sin(x) * (e ** (x * -1))) - 1'

secante(0, 0.1, f, 0.0001)
```

Figura 16 – Método da secante

A resposta logo abaixo, confere com o valor da raiz vista na plotagem.

```
Estimativas iniciais: x0 = 0; x1 = 0.1

it=1  x=0.158145 f(x)=-0.058847  erro=3.676689e-01
it=2  x=0.169225 f(x)=-0.004609  erro=6.547340e-02
it=3  x=0.170166 f(x)=-0.000066  erro=5.533044e-03
it=4  x=0.170180 f(x)=-0.000000  erro=8.062321e-05
Solução obtida: x = 0.1701799779
0.1701799779216659
```

Figura 17 – Resposta letra c

## 5 Questão 6

Para a questão seis, foi implementado o Método do ponto fixo, onde usa uma função (chamada de função de iteração) a partir da função dada ( g(x)= x, tendo obrigatoriamente que convergir )

$$g(x) = sen\left(\sqrt{x}\right)$$

Figura 18 – Função de iteração da questão

O método implementado recebe como parâmetro todos os dados fornecidos na questão.

```
from numpy import linspace
import numpy as np

# Sendo x0 a aproximação inicial, foi variado pra 0, 0.5 e 1000 (letras a, b e c)

x0 = 0

f = eval('lambda x:' + 'np.sin(x**(1/2)) - x')
g = eval('lambda x:' + 'np.sin(x**(1/2))')

it = 0
x, xn = x0, x0 + 1

e = abs(x-xn)/abs(x)

print('i\t x\t\t f(x)\t\t ER')
print('{0:d}\t {1:f}\t {2:f}\t {3:e}'.format(it, x, f(x), e))

while e >= 0.0001:
    it += 1
    xn = x
    x = g(xn)
    e = abs(x-xn)/abs(x)
    print('{0:d}\t {1:f}\t {2:f}\t {3:e}'.format(it, x, f(x), e))

print('Raiz: ', x)
```

Figura 19 – Método do ponto fixo

Para as letras (a), (b) e (c), foi utilizado o mesmo código, variando apenas a aproximação inicial.

Para letra (a) em específico, ocorre um erro em utilizar o zero como aproximação inicial, pois, na própria fórmula de iteração para o cálculo do erro, ocorre uma divisão com a aproximação inicial.

```
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-13-4b11dd10a314> in <module>()
     12 x, xn = x0, x0 + 1
     13
---> 14 e = abs(x-xn)/abs(x)
     15
     16 print('i\t x\t\t f(x)\t\t ER')

ZeroDivisionError: division by zero
```

Figura 20 – Resposta letra a

| i | x | f(x) | ER |
|---|---|---|---|
| 0 | 0.500000 | 0.149637 | 2.000000e+00 |
| 1 | 0.649637 | 0.071887 | 2.303393e-01 |
| 2 | 0.721524 | 0.029377 | 9.963200e-02 |
| 3 | 0.750901 | 0.011196 | 3.912282e-02 |
| 4 | 0.762097 | 0.004151 | 1.469063e-02 |
| 5 | 0.766248 | 0.001524 | 5.417686e-03 |
| 6 | 0.767772 | 0.000557 | 1.984329e-03 |
| 7 | 0.768329 | 0.000203 | 7.249574e-04 |
| 8 | 0.768532 | 0.000074 | 2.646108e-04 |
| 9 | 0.768606 | 0.000027 | 9.655062e-05 |

Raiz: 0.7686062313130103

Figura 21 – Resposta letra b

| i | x | f(x) | ER |
|---|---|---|---|
| 0 | 1000.000000 | -999.794622 | 1.000000e-03 |
| 1 | 0.205378 | 0.232455 | 4.868067e+03 |
| 2 | 0.437833 | 0.176618 | 5.309210e-01 |
| 3 | 0.614451 | 0.091574 | 2.874402e-01 |
| 4 | 0.706025 | 0.038787 | 1.297039e-01 |
| 5 | 0.744812 | 0.015001 | 5.207644e-02 |
| 6 | 0.759813 | 0.005594 | 1.974276e-02 |
| 7 | 0.765406 | 0.002057 | 7.308051e-03 |
| 8 | 0.767463 | 0.000753 | 2.680449e-03 |
| 9 | 0.768216 | 0.000275 | 9.797823e-04 |
| 10 | 0.768491 | 0.000100 | 3.576896e-04 |
| 11 | 0.768591 | 0.000037 | 1.305220e-04 |
| 12 | 0.768628 | 0.000013 | 4.761987e-05 |

Raiz: 0.7686278341212526

Figura 22 – Resposta letra c

## 6 Questão 8(3)

Para a letra (a), é necessário multiplicar as matrizes (A e X) para saber se o resultado é maior ou menor que o b.

$$\begin{pmatrix} 1 & 2 & 0 & 3 \\ 1 & 0 & 2 & 2 \\ 0 & 0 & 1 & 1 \end{pmatrix} \times \begin{pmatrix} 1200 \\ 700 \\ 450 \\ 200 \end{pmatrix} = \begin{pmatrix} 3200 \\ 2500 \\ 650 \end{pmatrix}$$

Figura 23 – Resposta da letra a

Nesse caso, podemos observar que o resultado é menor que a matriz b, logo, haverá comida suficiente para alimentar cada espécie!

Na letra (b), para descobrir quantos animais pode existir de cada espécie, basta multiplicar Ax = b e resolver o sistema.

$$X = \begin{pmatrix} 900 \\ 1300 - \frac{3}{2} \times x_4 \\ 900 - x_4 \\ x_4 \end{pmatrix}$$

Figura 24 – Resposta da letra b

Note que há uma dependência do valor de X4, para x4 menor ou igual à 866, o consumo se manterá o mesmo, mas caso ultrapasse 866, X2 deixará de existir pois X2 seria igual a zero.

Na letra (c), caso a terceira espécie for extinta, isso implica em X4 = 900 e X2 extinto.

## 7 Questão 9(2)

Na letra (a), Utilizando a lei de Kirchhoff para equacionar os nós do circuito (Observando as correntes que entram e saem de cada nó), foi possível chegar no sistema da análise nodal, considerando V1 = 200 e V6=0 (Fornecidos no circuito).

$$\begin{aligned} &\text{Nó 2: } \frac{V_2-V_1}{5} + \frac{V_2-V_3}{10} + \frac{V_2-V_5}{10} = 0 \\ &\text{Nó 3: } \frac{V_3-V_2}{10} + \frac{V_3-V_4}{5} = 0 \\ &\text{Nó 4: } \frac{V_4-V_3}{5} + \frac{V_4-V_5}{15} = 0 \\ &\text{Nó 5: } \frac{V_5-V_2}{10} + \frac{V_5-V_4}{15} + \frac{V_5-V_6}{20} = 0 \end{aligned} \quad ]$$

Figura 25 – Nós do circuito

A partir do sistema desenvolvido, é possível extrair sua matriz proposta na figura abaixo.

$$A = \begin{bmatrix} 0.4 & -0.1 & 0 & -0.1 \\ -0.1 & 0.3 & -0.2 & 0 \\ 0 & -0.2 & 0.26666 & -0.06666 \\ -0.1 & 0 & -0.06666 & 0.21666 \end{bmatrix}$$

$$b = \begin{bmatrix} 40 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

]

Figura 26 – Matriz e vetor resultante do sistema proposto

Para letra (b), foi necessário implementar o Método de pivotamento parcial (Com decomposição LU), onde foca em transformar uma matriz em uma matriz triangular superior (Preenchida com zeros), encaixando os valores restantes em outra matriz triangular superior (respectivamente L e U).

```python
import numpy as np

def pivotamentoLU(matrix):
    n, m = matrix.shape
    P    = np.identity(n)
    L    = np.identity(n)
    U    = matrix.copy()
    PF   = np.identity(n)
    LF   = np.zeros((n,n))
    for k in range(0, n - 1):
        index = np.argmax(abs(U[k:, k]))
        index = index + k
        if index != k:
            P = np.identity(n)
            P[[index, k], k:n] = P[[k, index], k:n]
            U[[index, k], k:n] = U[[k, index], k:n]
            PF = np.dot(P, PF)
            LF = np.dot(P, LF)
        L = np.identity(n)
        for j in range(k+1,n):
            L[j, k]  = -(U[j, k] / U[k, k])
            LF[j, k] =  (U[j, k] / U[k, k])
        U = np.dot(L,U)
    np.fill_diagonal(LF, 1)
    return PF, LF, U

A = [[0.4, -0.1, 0, -0.1], [-0.1, 0.3, -0.2, 0], [0, -0.2, 0.26666, -0.06666], [-0.1, 0, -0.06666, 0.21666]]
A = np.array(A)
P1, L1, U1 = pivotamentoLU(A)
```

]

Figura 27 – Método de pivotamento parcial com decomposição LU

```
P:

[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 [0. 0. 0. 1.]]

L:

[[ 1.          0.          0.          0.        ]
 [-0.25        1.          0.          0.        ]
 [ 0.         -0.72727273  1.          0.        ]
 [-0.25       -0.09090909 -0.6999835   1.        ]]

U:

[[ 0.4        -0.1         0.         -0.1       ]
 [ 0.          0.275      -0.2        -0.025     ]
 [ 0.          0.          0.12120545 -0.08484182]
 [ 0.          0.          0.          0.1299994 ]]
]
```

Figura 28 – Letra b após a aplicação do método

A partir dessas informações, são encontrados os valores dos nós: V2=169.23, V3=153.84, V4=146.14, V5=123.07.

Já para a letra (c), foi implementado o Método de pivotamento total, que além de ter a matriz permutação de linha, também possui a matriz de permutação da coluna.

```python
import numpy as np
def pivotamentoTotal(A):
    m = len(A)
    U = A.copy().astype(float)
    L = np.eye(m)
    P = np.eye(m)
    Q = np.eye(m)
    for k in range(m):
        pivo = abs(U[k, k])
        maxindex_i = k
        maxindex_j = k

        for i in range(k, m):  #pivotamento
            for j in range(k, m):
                if abs(U[i, j]) > pivo:
                    # U[[k, i]] = U[[i, k]]
                    pivo = abs(U[i, j])
                    maxindex_i = i
                    maxindex_j = j
        Q[:, [k, maxindex_j]] = Q[:, [maxindex_j, k]]
        P[[k, maxindex_i]] = P[[maxindex_i, k]]

        #troca U - cada elemento da linha/coluna
        for i in range(m):
            swap = U[maxindex_i, i]
            U[maxindex_i, i] = U[k, i]
            U[k, i] = swap
        for j in range(m):
            swap1 = U[j, maxindex_j]
            U[j, maxindex_j] = U[j, k]
            U[j, k] = swap1

        # troca L
        for i in range(k):
            swap0 = L[maxindex_i, i]
            L[maxindex_i, i] = L[k, i]
            L[k, i] = swap0
        for j in range(k):
```

Figura 29 – Parte do código de pivotamento total

```
P:

[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]]
Q:

[[1. 0. 0. 0.]
 [0. 1. 0. 0.]
 [0. 0. 0. 1.]
 [0. 0. 1. 0.]]

L:

[[ 1.          0.          0.          0.        ]
 [-0.25        1.          0.          0.        ]
 [-0.25       -0.09090909  1.          0.        ]
 [ 0.         -0.72727273 -0.44798057  1.        ]]

U:

[[ 0.4        -0.1        -0.1         0.        ]
 [ 0.          0.275      -0.025      -0.2       ]
 [ 0.          0.          0.18938727 -0.08484182]
 [ 0.          0.          0.          0.08319797]]
                                                 ]
```

Figura 30 – Resultado do pivotamento total

A partir do resultado, os valores obtidos foram: V2=169.26, V3=153.93, V4=146.26, V5=123.12.

## 8 Questão 10(3)

Para resolver a letra (a), foi necessário decompor as forças horizontais e verticais que atuam na treliça, para isso, foi denominada essas forças.

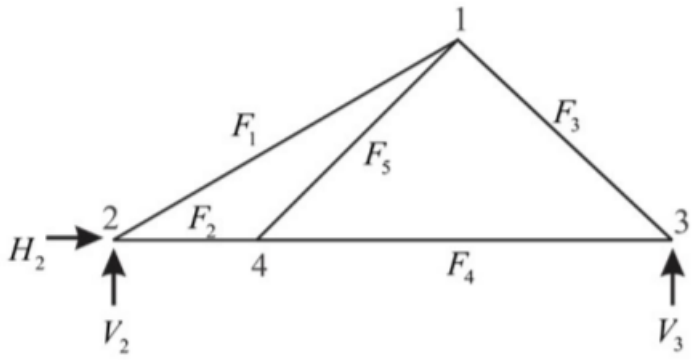


Figura 31 – Forças atuando na treliça

Foram decompostas as forças no eixo Y e no eixo X de cada uma delas e considerando os ângulos definidos na questão Cos(30º)=0.8666, Sen(30º)=0.5, Cos(45º)=0.707, Sen(45º)=0.707. A partir disso, podemos montar o sistema abaixo.

$$\begin{cases} -0.866F_1 + 0.707F_3 = -700 \\ -0.5F_1 - 0.070f_3 = 1100 \\ 0.0866F_1 + F_2 + H_2 = 0 \\ 0.5F_1 + v2 = 0 \\ -0.707F_3 - f_4 = 0 \\ 0.707F_3 + v_3 = 0 \\ -F_2 + f_4 = -500 \end{cases}$$

Figura 32 – Sistema proposto

E a partir dele, é possível obter a matriz abaixo

$$A = \begin{bmatrix} -0.866 & 0 & 0.707 & 0 & 0 & 0 & 0 \\ -0.5 & 0 & -0.707 & 0 & 0 & 0 & 0 \\ 0.866 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0.5 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & -0.707 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0.707 & 0 & 0 & 1 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$x = \begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \\ V_2 \\ V_3 \\ H_2 \end{bmatrix}$$

$$b = \begin{bmatrix} -700 \\ 1100 \\ 0 \\ 0 \\ 0 \\ 0 \\ -500 \end{bmatrix}$$

Figura 33 – Matriz e vetores propostos

Para letra (b), foi possível tornar a matriz diagonal dominante, porém, devido à segunda linha, não foi possível torná-la estritamente diagonal dominante, pois a soma dos outros elementos da linha era igual ao módulo do elemento da diagonal. As modificações feitas para tornar a matriz diagonal dominante estão dispostas abaixo.

L7 -> L2 // L7 -> L3 // L4 -> L5

$$A' = \begin{bmatrix} -0.866 & 0 & 0.707 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 0 & 0 \\ -0.5 & 0 & -0.707 & 0 & 0 & 0 & 0 \\ 0 & 0 & -0.707 & -1 & 0 & 0 & 0 \\ 0.5 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0.707 & 0 & 0 & 1 & 0 \\ 0.866 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Figura 34 – Matriz diagonal dominante

Para as letras (c) e (d), foi implementado o Método iterativo de Gauss-Seidel.

```
def gaussSeidel(A, b, x, N, tol):
    maxIterations = 1000000
    xprev = [0.0 for i in range(N)]
    for i in range(maxIterations):
        for j in range(N):
            xprev[j] = x[j]
        for j in range(N):
            summ = 0.0
            for k in range(N):
                if (k != j):
                    summ = summ + A[j][k] * x[k]
            x[j] = (b[j] - summ) / A[j][j]
        diff1norm = 0.0
        oldnorm = 0.0
        for j in range(N):
            diff1norm = diff1norm + abs(x[j] - xprev[j])
            oldnorm = oldnorm + abs(xprev[j])
        if oldnorm == 0.0:
            oldnorm = 1.0
        norm = diff1norm / oldnorm
        if (norm < tol) and i != 0:
            print("Converge para:  [", end="")
            for j in range(N - 1):
                print(x[j], ",", end="")
            print(x[N - 1], "]. Levou", i + 1, "iteracoes")
            return
    print("Nao converge")
```

Figura 35 – Implementação do Método de Gauss-Seidel

Iniciando com os parâmetros fornecidos (vetor nulo), foi possível chegar ao resultado com 20 iterações

```
Converge para:  [-292.85808578508755 ,
1453.6151022898857 ,
-1348.7566578606172 ,
953.5709571074564 ,
146.42904289254378 ,
953.5709571074564 ,
253.6151022898858 ]. Levou 20 iteracoes
```

Figura 36 – Resposta letras C e D

Analisando A'*(resultado) vemos que ele não converge para o vetor b (solução exata).

$$A' * x^* = \begin{bmatrix} -703.87 \\ -500 \\ 1104.05 \\ 4.04 \\ 0.01 \\ -4.04 \\ -1.17 \end{bmatrix} \quad b = \begin{bmatrix} -700 \\ -500 \\ 1100 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Figura 37 – Resposta letras C e D