Atividade Computacional 1

Grupo:

- Guilherme Ferreira Lourenço
- Harrison Caetano Cândido
- Thiago Cardoso Carvalho

Importação de todas as bibliotecas

```
import math
import numpy as np
import matplotlib.pyplot as plt
from IPython.display import display
```

Método da Bissecção

```
def f(x):
  return math.pow(x, 3) + x - 4

def bisseccao(intervalo, erro, it_max):
    ak = intervalo[0]
    bk = intervalo[1]
    xk = (ak + bk)/2
    sk = bk - ak
    it = 0
    # lista das k linhas
    line_data = []
    # colunas necessarias para plotar
    column_labels = ['k', 'xk', 'f(xk)', 'stepk']

    while(np.abs(bk - ak) > erro and it < it_max):
      xk = (ak + bk) / 2
      sk = np.abs(bk - ak) # como nao tem do-while em python vai ter que fazer a diferença
      it += 1

      # se chegou aqui eh pq o teorema de bolzano nao pode provar que existe ao menos um z
      if (f(ak) > 0 and f(bk) > 0 and f(xk) > 0) or (f(ak) < 0 and f(bk) < 0 and f(xk) < 0
        print("Pelo Teorema de Bolzano, temos que f(a)f(b) >= 0, logo nao podemos afirmar
        return

      # saida: x*, f(x*), num iteracoes. vamos fazendo isso inserindo cada tupla de linha
      line_data.append([it, xk, f(xk), sk])

      # f(ak) < 0 and f(bk) > 0 and f(xk) > 0, then we gonna use ak as ak and bk as xk
      if(f(ak) < 0 and f(bk) > 0 and f(xk) > 0):
        bk = xk
        continue
```

```
      # f(ak) > 0 and f(bk) < 0 and f(xk) > 0, then we gonna use ak as xk and bk as bk
      if(f(ak) > 0 and f(bk) < 0 and f(xk) > 0):
        ak = xk
        continue

      # f(ak) > 0 and f(bk) > 0 and f(xk) < 0 and f(ak) > f(bk), then we gonna use ak as b
      if(f(ak) > 0 and f(bk) > 0 and f(xk) < 0 and f(ak) > f(bk)):
        ak = bk
        bk = xk
        continue

      # f(ak) > 0 and f(bk) > 0 and f(xk) < 0 and f(ak) < f(bk), then we gonna use ak as a
      if(f(ak) > 0 and f(bk) > 0 and f(xk) < 0 and f(ak) < f(bk)):
        bk = xk
        continue

      # f(ak) < 0 and f(bk) < 0 and f(xk) > 0 and f(ak) > f(bk), then we gonna use ak as a
      if(f(ak) < 0 and f(bk) < 0 and f(xk) > 0 and f(ak) > f(bk)):
        bk = xk
        continue

      # f(ak) < 0 and f(bk) < 0 and f(xk) > 0 and f(ak) < f(bk), then we gonna use ak as x
      if(f(ak) < 0 and f(bk) < 0 and f(xk) > 0 and f(ak) < f(bk)):
        ak = xk
        continue

      # f(ak) > 0 and f(bk) < 0 and f(xk) < 0, then we gonna use ak as ak and bk as xk
      if(f(ak) > 0 and f(bk) < 0 and f(xk) < 0):
        bk = xk
        continue

      # f(ak) < 0 and f(bk) > 0 and f(xk) < 0, then we gonna use ak as xk and bk as bk
      if(f(ak) < 0 and f(bk) > 0 and f(xk) < 0):
        ak = xk
        continue

    # configura a quantidade de graficos
    fig, ax = plt.subplots(figsize=(8,6))
    ax.set_axis_off()
    table = ax.table(cellText=line_data,
                     cellLoc='center',
                     colLabels=column_labels,
                     loc='center')
    plt.savefig('Tabela_Bisseccao.pdf', dpi=200)
    plt.show()


erro = 10**-10
it_max = 200
intervalo = (1, 4)

bisseccao(intervalo, erro, it_max)
```

k	xk	f(xk)	stepk
1	2.5	14.125	3
2	1.75	3.109375	1.5
3	1.375	-0.025390625	0.75
4	1.5625	1.377197265625	0.375
5	1.46875	0.637176513671875	0.1875
6	1.421875	0.2965202331542969	0.09375
7	1.3984375	0.13326025009155273	0.046875
8	1.38671875	0.05336350202560425	0.0234375
9	1.380859375	0.013844214379787445	0.01171875
10	1.3779296875	-0.005808655906231403	0.005859375
11	1.37939453125	0.004008884658105671	0.0029296875
12	1.378662109375	-0.0009021193400258198	0.00146484375
13	1.3790283203125	0.0015528278327110456	0.000732421875
14	1.37884521484375	0.00032521555817766057	0.0003662109375
15	1.378753662109375	-0.00028848656066315925	0.00018310546875
16	1.3787994384765625	1.8355831034710945e-05	9.1552734375e-05
17	1.3787765502929688	-0.00013506753170888786	4.57763671875e-05
18	1.3787879943847656	-5.8356392065306295e-05	2.288818359375e-05
19	1.378793716430664	-2.0000415947851735e-05	1.1444091796875e-05
20	1.3787965774536133	-8.223263145978876e-07	5.7220458984375e-06
21	1.378798007965088	8.7667438952721e-06	2.86102294921875e-06
22	1.3787972927093506	3.972206673807932e-06	1.430511474609375e-06
23	1.378796935081482	1.5749396506947733e-06	7.152557373046875e-07
24	1.3787967562675476	3.763065361539475e-07	3.5762786865234375e-07
25	1.3787966668605804	-2.230099225286608e-07	1.7881393432617188e-07
26	1.378796711564064	7.664829537494835e-08	8.940696716308594e-08
27	1.3787966892123222	-7.318081429730228e-08	4.470348358154297e-08
28	1.3787967003881931	1.7337420388230385e-09	2.2351741790771484e-08
29	1.3787966948002577	-3.572353657332883e-08	1.1175870895385742e-08
30	1.3787966975942254	-1.69948974892975e-08	5.587935447692871e-09
31	1.3787966989912093	-7.630577947281836e-09	2.7939677238464355e-09
32	1.3787966996897012	-2.9484183983186085e-09	1.3969838619232178e-09
33	1.3787967000389472	-6.073386238369949e-10	6.984919309616089e-10
34	1.3787967002135701	5.63201041359207e-10	3.4924596548080444e-10

Método Newton

```
def f(x):
    return x**2 - 2

def df(x):
    return 2*x

# lista das k linhas
line_data = []
# colunas necessarias para plotar
column_labels = ['it', 'xk', 'f(xk)', 'df(xk)', 'stepk']

def newton(x0, erro, it_max):
    x = x0
    it = 0
    # evitar erro na primeira iteracao
    sk = erro + 1

    line_data.append([it, x, f(x), df(x),'-'])

    while(sk >= erro and it < it_max):
        xold = x
        x = x - f(x)/df(x)
        sk = np.abs(x - xold)
        it += 1

        line_data.append([it, x, f(x), df(x), sk])

    # configura a tabela
```

```
    fig, ax = plt.subplots(figsize=(8,6))
    ax.set_axis_off()
    table = ax.table(cellText=line_data,
                     cellLoc='center',
                     colLabels=column_labels,
                     loc='center')
    plt.savefig('Pt1_Ex2.1.pdf', dpi=200)
    plt.show()

x0 = 5
erro = 10**-12
it_max = 6

newton(x0, erro, it_max)
```

k	xk	f(xk)	df(xk)	stepk
0	5	23	10	-
1	2.7	5.290000000000001	5.4	2.3
2	1.7203703703703703	0.9596742112482852	3.4407407407407407	0.9796296296296299
3	1.441455368617765	0.07779357844481647	2.8829107363553	0.27891500219272025
4	1.414470981367771	0.0007281571315052027	2.828941962735542	0.026984386809879135
5	1.4142135857968836	6.625247950253765e-08	2.828427171593767	0.000257395570887331
6	1.4142135623730951	4.440892098500626e-16	2.8284271247461903	2.3423788464427275e-08

Método Secante

```
def f(x):
    return 31 - ((9.8*x)/13)*(1. - np.exp(-6.0*(13.0/x)))

def secante(x0, x1, erro, it_max):
    xk1 = x0
    xk2 = x1
    it = 0
    sk = np.abs(xk2 - xk1)
    # lista das k linhas
    line_data = []
    # colunas necessarias para plotar
    column_labels = ['k', 'xk', 'f(xk)', 'stepk']

    """
    xk1 = xk-1
    xk2 = xk
    x = xk+1
    """
```

```
    while(sk >= erro and abs(f(xk2)) >= erro and abs(f(xk1)) >= erro and it < it_max):
      x = xk2 - f(xk2)*(xk2 - xk1)/(f(xk2) - f(xk1))
      xk1 = xk2
      xk2 = x
      it += 1
      sk = np.abs(xk2 - xk1)
      line_data.append([it, x, f(x), sk])

    # configura o tamanho do grafico
    fig, ax = plt.subplots(figsize=(6,2))
    ax.set_axis_off()
    table = ax.table(cellText=line_data,
                     cellLoc='center',
                     colLabels=column_labels,
                     loc='center')
    plt.savefig('Tabela_Secante.pdf', dpi=200)
    plt.show()

x0 = 52
x1 = 58
erro = 10**-6
it_max = 200

secante(x0, x1, erro, it_max)
```

k	xk	f(xk)	dsepk
1	58.7483122616446	-0.02512695707897281	42516877383554
2	58.66841027231532	0.0011804747097485802	0.08190198932928183
3	58.67008540211362	-9.963049798500379e-07	0.0016751298208841376

Parte 1

• Faça o seu gráfico utilizando recursos computacionais.

• Aplique o código implementado do Método de Newton adotando os pontos iniciais indicados em cada caso, tomando ε = 10−8 e maxit = 20 .

• Imprima a tabela referente aos resultados de cada iteração, conforme as instruções.

• A partir dos gráficos, de argumentos matemáticos e/ou dos resultados exibidos nas tabelas, explique o comportamento do Método de Newton para cada ponto inicial dado.

```
#Função de plotar gráfico:

# configura o grafico
def graf(y):
    ticks_frequency = 1

    fig, ax = plt.subplots(figsize = (10, 10))
```

```
    fig.patch.set_facecolor('#ffffff')

    ax.set(xlim=(-6,6), ylim=(-6,6), aspect='equal')
    ax.spines['bottom'].set_position('zero')
    ax.spines['left'].set_position('zero')

    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)

    ax.set_xlabel('$x$', size=14, labelpad=-24, x=1.02)
    ax.set_ylabel('$y$', size=14, labelpad=-21, y=1.02, rotation=0)

    plt.text(0.49, 0.49, r"$O$", ha='right', va='top',
      transform=ax.transAxes,
        horizontalalignment='center', fontsize=14)

    x_ticks = np.arange(-5, 6, ticks_frequency)
    y_ticks = np.arange(-5, 6, ticks_frequency)
    ax.set_xticks(x_ticks[x_ticks != 0])
    ax.set_yticks(y_ticks[y_ticks != 0])
    ax.set_xticks(np.arange(-5, 6), minor=True)
    ax.set_yticks(np.arange(-5, 6), minor=True)

    ax.grid(which='both', color='grey', linewidth=1, linestyle='-', alpha=0.2)

    # Create the plot

    x = np.linspace(-5, 10, 100)

    plt.plot(x, y, 'b', linewidth=2)

    # Show the plot
    plt.show()

#Exercício 1.1

def f(x):
    return x*np.e**(-x)

def df(x):
    return np.e**(-x)-np.e**(-x)*x

# lista das k linhas
line_data = []
# colunas necessarias para plotar
column_labels = ['it', 'xk', 'f(xk)', 'df(xk)', 'stepk']

def newton(x0, erro, it_max):
    x = x0
    it = 0
    # evitar erro na primeira iteracao
    sk = erro + 1

    line_data.append([it, x, f(x), df(x),'-'])
```

```
    while(sk >= erro and it < it_max):
        xold = x
        x = x - f(x)/df(x)
        sk = np.abs(x - xold)
        it += 1

        line_data.append([it, x, f(x), df(x), sk])

    # configura a tabela
    fig, ax = plt.subplots(figsize=(8,6))
    ax.set_axis_off()
    table = ax.table(cellText=line_data,
                     cellLoc='center',
                     colLabels=column_labels,
                     loc='center')
    plt.savefig('Pt1_Ex2.1.pdf', dpi=200)
    plt.show()

    # configura o grafico
    ticks_frequency = 1

    fig, ax = plt.subplots(figsize = (10, 10))
    fig.patch.set_facecolor('#ffffff')

    ax.set(xlim=(-6,6), ylim=(-6,6), aspect='equal')
    ax.spines['bottom'].set_position('zero')
    ax.spines['left'].set_position('zero')

    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)

    ax.set_xlabel('$x$', size=14, labelpad=-24, x=1.02)
    ax.set_ylabel('$y$', size=14, labelpad=-21, y=1.02, rotation=0)

    plt.text(0.49, 0.49, r"$O$", ha='right', va='top',
      transform=ax.transAxes,
        horizontalalignment='center', fontsize=14)

    x_ticks = np.arange(-5, 6, ticks_frequency)
    y_ticks = np.arange(-5, 6, ticks_frequency)
    ax.set_xticks(x_ticks[x_ticks != 0])
    ax.set_yticks(y_ticks[y_ticks != 0])
    ax.set_xticks(np.arange(-5, 6), minor=True)
    ax.set_yticks(np.arange(-5, 6), minor=True)

    ax.grid(which='both', color='grey', linewidth=1, linestyle='-', alpha=0.2)

    # Create the plot

    x = np.linspace(-5, 10, 100)
    y = x*np.e**(-x)


    plt.plot(x, y, 'b', linewidth=2)
```

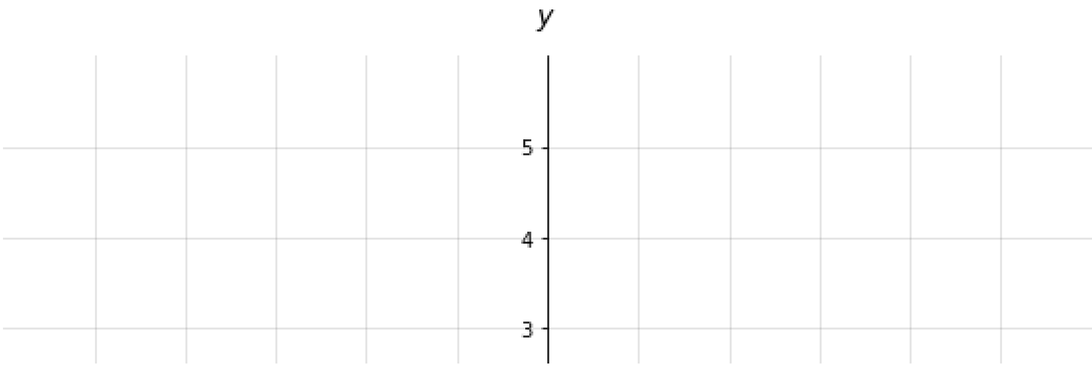
```
    # Show the plot
    plt.show()

x0 = 2
erro = 10**-8
it_max = 20
newton(x0, erro, it_max)
```

it	xk	f(xk)	df(xk)	stepk

```
#Exercício 1.2

def f(x):
    return x*np.e**(-x)

def df(x):
    return np.e**(-x)-np.e**(-x)*x

# lista das k linhas
line_data = []
# colunas necessarias para plotar
column_labels = ['it', 'xk', 'f(xk)', 'df(xk)', 'stepk']

def newton(x0, erro, it_max):
    x = x0
    it = 0
    # evitar erro na primeira iteracao
    sk = erro + 1

    line_data.append([it, x, f(x), df(x),'-'])

    while(sk >= erro and it < it_max):
        xold = x
        x = x - f(x)/df(x)
        sk = np.abs(x - xold)
        it += 1

        line_data.append([it, x, f(x), df(x), sk])

    # configura a tabela
    fig, ax = plt.subplots(figsize=(8,6))
    ax.set_axis_off()
    table = ax.table(cellText=line_data,
                     cellLoc='center',
                     colLabels=column_labels,
                     loc='center')
    plt.savefig('Pt1_Ex2.1.pdf', dpi=200)
    plt.show()

    # configura o grafico
    ticks_frequency = 1

    fig, ax = plt.subplots(figsize = (10, 10))
    fig.patch.set_facecolor('#ffffff')

    ax.set(xlim=(-6,6), ylim=(-6,6), aspect='equal')
    ax.spines['bottom'].set_position('zero')
    ax.spines['left'].set_position('zero')

    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)
```

```
    ax.set_xlabel('$x$', size=14, labelpad=-24, x=1.02)
    ax.set_ylabel('$y$', size=14, labelpad=-21, y=1.02, rotation=0)

    plt.text(0.49, 0.49, r"$O$", ha='right', va='top',
      transform=ax.transAxes,
        horizontalalignment='center', fontsize=14)

    x_ticks = np.arange(-5, 6, ticks_frequency)
    y_ticks = np.arange(-5, 6, ticks_frequency)
    ax.set_xticks(x_ticks[x_ticks != 0])
    ax.set_yticks(y_ticks[y_ticks != 0])
    ax.set_xticks(np.arange(-5, 6), minor=True)
    ax.set_yticks(np.arange(-5, 6), minor=True)

    ax.grid(which='both', color='grey', linewidth=1, linestyle='-', alpha=0.2)

    # Create the plot

    x = np.linspace(-5, 10, 100)
    y = x*np.e**(-x)


    plt.plot(x, y, 'b', linewidth=2)

    # Show the plot
    plt.show()

x0 = 0.5
erro = 10**-8
it_max = 20
newton(x0, erro, it_max)
```

it	xk	f(xk)	df(xk)	stepk
0	0.5	0.3032653298563167	0.3032653298563167	-
1	-0.5	-0.8243606353500641	2.4730819060501923	1.0
2	-0.16666666666666669	-0.19689340214427437	1.3782538150099204	0.3333333333333333
3	-0.023809523809523808	-0.024383219846499493	1.0484784533994782	0.14285714285714288
4	-0.0005537098560354364	-0.0005540165355380033	1.0011078797171737	0.023255813953488372
5	-3.0642493416461764e-07	-3.0642502806087233e-07	1.0000006128500092	0.0005534034311012718
6	-9.389621148813321e-14	-9.389621148814203e-14	1.0000000000001878	3.0642484026840615e-07
7	-8.80999858950826e-27	-8.80999858950826e-27	1.0	9.389621148881244e-14

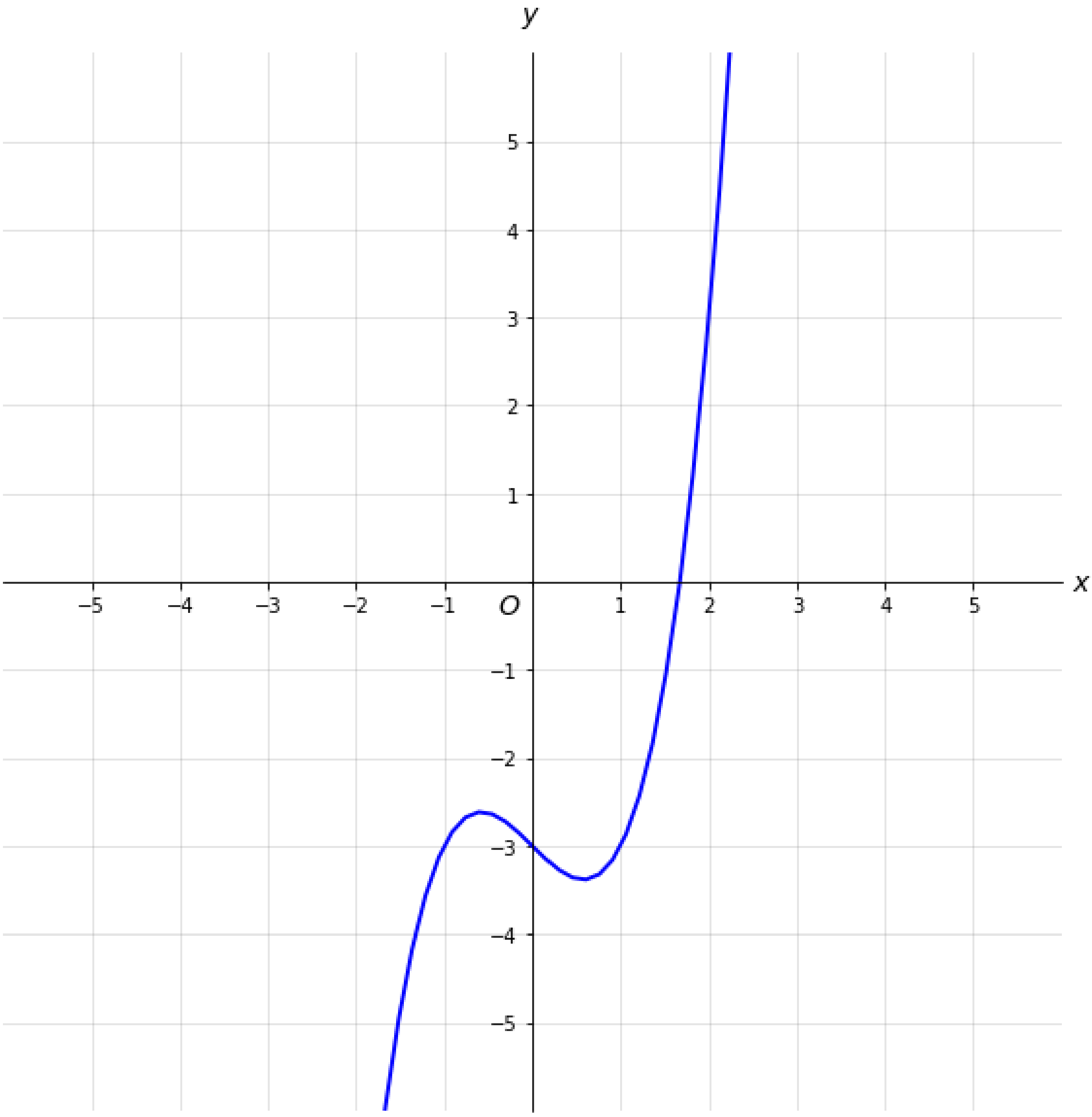



```
#Exercício 2.1
#f(x) = x**3 - x - 3, x0 = 1, x0 = 2

def f(x):
    return x**3 - x - 3

def df(x):
    return 3*(x**2) - 1

# lista das k linhas
line_data = []
# colunas necessarias para plotar
column_labels = ['it', 'xk', 'f(xk)', 'df(xk)', 'stepk']

def Newton(x0, Erro, itMax):
    x = x0
    Er = 1
    it = 0

    line_data.append([it, x, f(x), df(x),'-'])

    while(Er >= Erro and it < itMax):
        xold = x
        x = x - f(x)/df(x)
        Er = np.abs((x-xold)/x)
        sk = np.abs(x - xold)
```

```
        it += 1

        line_data.append([it, x, f(x), df(x), sk])

# configura a tabela
    fig, ax = plt.subplots(figsize=(8,6))
    ax.set_axis_off()
    table = ax.table(cellText=line_data,
                     cellLoc='center',
                     colLabels=column_labels,
                     loc='center')
    plt.savefig('Pt1_Ex2.1.pdf', dpi=200)
    plt.show()

# configura o grafico
    ticks_frequency = 1

    fig, ax = plt.subplots(figsize = (10, 10))
    fig.patch.set_facecolor('#ffffff')

    ax.set(xlim=(-6,6), ylim=(-6,6), aspect='equal')
    ax.spines['bottom'].set_position('zero')
    ax.spines['left'].set_position('zero')

    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)

    ax.set_xlabel('$x$', size=14, labelpad=-24, x=1.02)
    ax.set_ylabel('$y$', size=14, labelpad=-21, y=1.02, rotation=0)

    plt.text(0.49, 0.49, r"$O$", ha='right', va='top',
      transform=ax.transAxes,
        horizontalalignment='center', fontsize=14)

    x_ticks = np.arange(-5, 6, ticks_frequency)
    y_ticks = np.arange(-5, 6, ticks_frequency)
    ax.set_xticks(x_ticks[x_ticks != 0])
    ax.set_yticks(y_ticks[y_ticks != 0])
    ax.set_xticks(np.arange(-5, 6), minor=True)
    ax.set_yticks(np.arange(-5, 6), minor=True)

    ax.grid(which='both', color='grey', linewidth=1, linestyle='-', alpha=0.2)

    # Create the plot

    x = np.linspace(-5, 10, 100)
    y = x**3 - x - 3


    plt.plot(x, y, 'b', linewidth=2)

    # Show the plot
    plt.show()

x0 = 1
```

```
Erro = 10**-8
itMax = 20

Newton(x0, Erro, itMax)
```

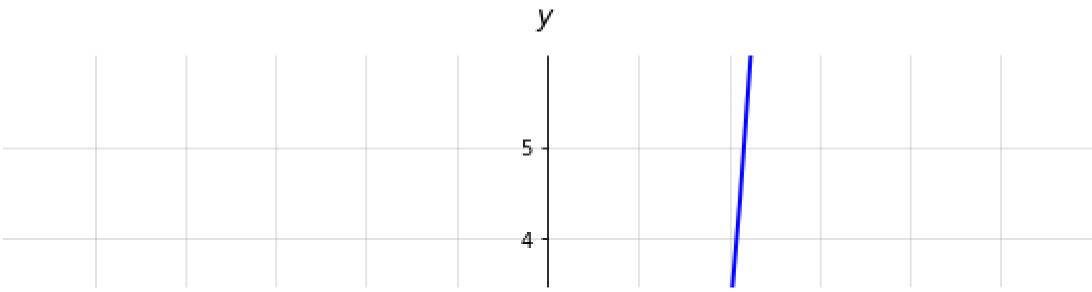
it	xk	f(xk)	df(xk)	stepk
0	1	-3	2	-
1	2.5	10.125	17.75	1.5
2	1.9295774647887325	2.2547588646339456	10.169807577861537	0.5704225352112675
3	1.7078664002114352	0.2736513656831989	7.750422922913499	0.22171106457729728
4	1.6725584733531251	0.008343316785037523	7.392355540356011	0.03530792685831008
5	1.6717003819436402	3.693987285391387e-06	7.383746500971538	0.0008580914094848957
6	1.671699881657331	1.255440196246127e-12	7.383741482999403	5.002863092684606e-07
7	1.6716998816571609	-8.881784197001252e-16	7.3837414829976975	1.7008616737257398e-13



```
#Exercício 2.2
#f(x) = x**3 - x - 3, x0 = 1, x0 = 2

def f(x):
    return x**3 - x - 3

def df(x):
    return 3*(x**2) - 1

# lista das k linhas
line_data = []
# colunas necessarias para plotar
column_labels = ['it', 'xk', 'f(xk)', 'df(xk)', 'stepk']

def Newton(x0, Erro, itMax):
    x = x0
    Er = 1
    it = 0

    line_data.append([it, x, f(x), df(x),'-'])

    while(Er >= Erro and it < itMax):
        xold = x
        x = x - f(x)/df(x)
        Er = np.abs((x-xold)/x)
        sk = np.abs(x - xold)
        it += 1

        line_data.append([it, x, f(x), df(x), sk])

# configura a tabela
    fig, ax = plt.subplots(figsize=(8,6))
    ax.set_axis_off()
    table = ax.table(cellText=line_data,
                     cellLoc='center',
                     colLabels=column_labels,
                     loc='center')
    plt.savefig('Pt1_Ex2.2.pdf', dpi=200)
    plt.show()

# configura o grafico
    ticks_frequency = 1

    fig, ax = plt.subplots(figsize = (10, 10))
    fig.patch.set_facecolor('#ffffff')

    ax.set(xlim=(-6,6), ylim=(-6,6), aspect='equal')
    ax.spines['bottom'].set_position('zero')
    ax.spines['left'].set_position('zero')

    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)

    ax.set_xlabel('$x$', size=14, labelpad=-24, x=1.02)
```

```
    ax.set_ylabel('$y$', size=14, labelpad=-21, y=1.02, rotation=0)

    plt.text(0.49, 0.49, r"$O$", ha='right', va='top',
      transform=ax.transAxes,
        horizontalalignment='center', fontsize=14)

    x_ticks = np.arange(-5, 6, ticks_frequency)
    y_ticks = np.arange(-5, 6, ticks_frequency)
    ax.set_xticks(x_ticks[x_ticks != 0])
    ax.set_yticks(y_ticks[y_ticks != 0])
    ax.set_xticks(np.arange(-5, 6), minor=True)
    ax.set_yticks(np.arange(-5, 6), minor=True)

    ax.grid(which='both', color='grey', linewidth=1, linestyle='-', alpha=0.2)

    # Create the plot

    x = np.linspace(-5, 10, 100)
    y = x**3 - x - 3

    plt.plot(x, y, 'b', linewidth=2)

    # Show the plot
    plt.show()

x0 = 2
Erro = 10**-8
itMax = 20

Newton(x0, Erro, itMax)
```

it	xk	f(xk)	df(xk)	stepk
0	2	3	11	-
1	1.7272727272727273	0.4259954921111948	7.950413223140496	0.2727272727272727
2	1.6736911736911737	0.014723079585861054	7.403726434675216	0.05358155358155359
3	1.671702569747502	1.9848200399685823e-05	7.383768445101206	0.0019886039436716185
4	1.671699881662069	3.623945588060451e-11	7.383741483046926	2.6880854331334803e-06
5	1.6716998816571609	-8.881784197001252e-16	7.3837414829976975	4.908073947262892e-12

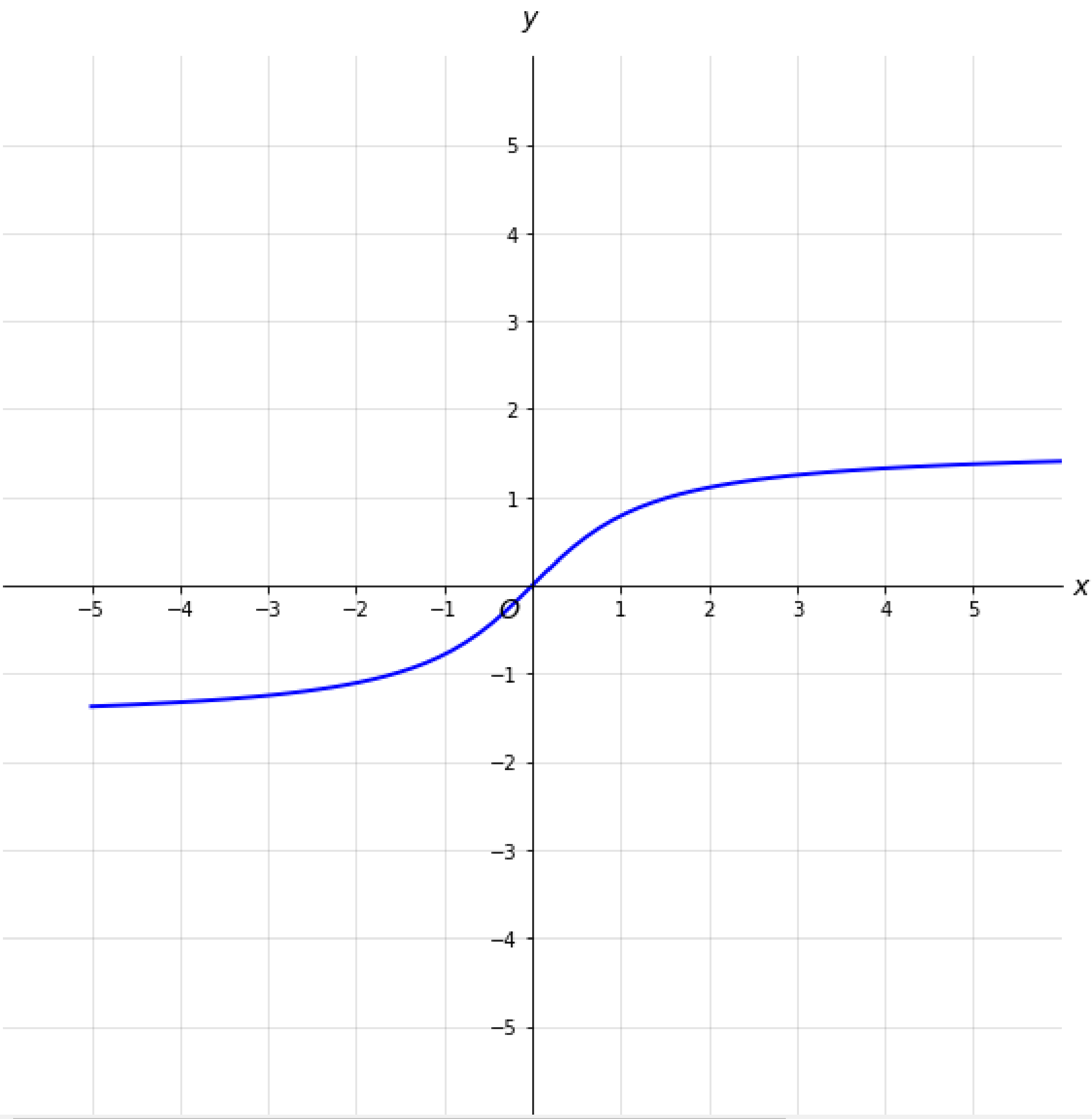



```
#Exercício 3.1
def f(x):
    return np.arctan(x)

def df(x):
    return 1/(x**2 + 1)

# lista das k linhas
line_data = []
# colunas necessarias para plotar
column_labels = ['it', 'xk', 'f(xk)', 'df(xk)', 'stepk']

def Newton(x0, Erro, itMax):
    x = x0
    Er = 1
    it = 0

    line_data.append([it, x, f(x), df(x),'-'])

    while(Er >= Erro and it < itMax):
        xold = x
        x = x - f(x)/df(x)
        Er = np.abs((x-xold)/x)
        sk = np.abs(x - xold)
        it += 1

        line_data.append([it, x, f(x), df(x), sk])
```

```
# configura a tabela
    fig, ax = plt.subplots(figsize=(8,6))
    ax.set_axis_off()
    table = ax.table(cellText=line_data,
                     cellLoc='center',
                     colLabels=column_labels,
                     loc='center')
    plt.savefig('Pt1_Ex3.1.pdf', dpi=200)
    plt.show()

# configura o grafico
    ticks_frequency = 1

    fig, ax = plt.subplots(figsize = (10, 10))
    fig.patch.set_facecolor('#ffffff')

    ax.set(xlim=(-6,6), ylim=(-6,6), aspect='equal')
    ax.spines['bottom'].set_position('zero')
    ax.spines['left'].set_position('zero')

    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)

    ax.set_xlabel('$x$', size=14, labelpad=-24, x=1.02)
    ax.set_ylabel('$y$', size=14, labelpad=-21, y=1.02, rotation=0)

    plt.text(0.49, 0.49, r"$O$", ha='right', va='top',
      transform=ax.transAxes,
        horizontalalignment='center', fontsize=14)

    x_ticks = np.arange(-5, 6, ticks_frequency)
    y_ticks = np.arange(-5, 6, ticks_frequency)
    ax.set_xticks(x_ticks[x_ticks != 0])
    ax.set_yticks(y_ticks[y_ticks != 0])
    ax.set_xticks(np.arange(-5, 6), minor=True)
    ax.set_yticks(np.arange(-5, 6), minor=True)

    ax.grid(which='both', color='grey', linewidth=1, linestyle='-', alpha=0.2)

    # Create the plot

    x = np.linspace(-5, 10, 100)
    y = np.arctan(x)

    plt.plot(x, y, 'b', linewidth=2)

    # Show the plot
    plt.show()

#Configura e calcula por Newton
x0 = 1.45
Erro = 10**-8
itMax = 20

Newton(x0, Erro, itMax)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:6: RuntimeWarning: ov

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:22: RuntimeWarning: d
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:23: RuntimeWarning: i
```

it	xk	f(xk)	df(xk)	stepk
0	1.45	0.9670469933974603	0.32232070910556004	-
1	-1.5502632970156205	-0.9979075580246077	0.29383105029545525	3.0002632970156204
2	1.8459317511972355	1.0743231874315433	0.22688784143798943	3.396195048212856
3	-2.889109054086138	-1.237575582047004	0.10698675819276336	4.735040605283371
4	8.678449426536321	1.456074323932289	0.013103500648927308	11.567558480622457
5	-102.44256799803165	-1.5610350697449524	9.527911300484988e-05	111.12101742456797
6	16281.36937686645	1.5707349068998253	3.7724035058131336e-09	16383.811944864483
7	-416358823.91280764	-1.5707963243931222	5.768520678200044e-18	416375105.2821845
8	2.7230487842881142e+17	1.5707963267948966	1.3488186341842e-35	2.7230487884517024e+17
9	-1.1647446409081358e+35	-1.5707963267948966	7.3712061661483e-71	1.1647446409081358e+35
10	2.1309895441653207e+70	1.5707963267948966	2.202101650107294e-141	2.1309895441653207e+70
11	-7.133169019324616e+140	-1.5707963267948966	1.9653276789371744e-282	7.133169019324616e+140
12	7.99254161852727e+281	1.5707963267948966	0.0	7.99254161852727e+281
13	-inf	-1.5707963267948966	0.0	inf

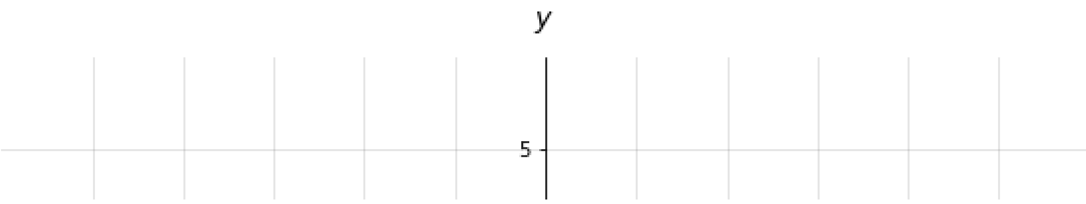


```
#Exercício 3.2
def f(x):
    return np.arctan(x)

def df(x):
    return 1/(x**2 + 1)

# lista das k linhas
line_data = []
# colunas necessarias para plotar
column_labels = ['it', 'xk', 'f(xk)', 'df(xk)', 'stepk']

def Newton(x0, Erro, itMax):
    x = x0
    Er = 1
    it = 0

    line_data.append([it, x, f(x), df(x),'-'])

    while(Er >= Erro and it < itMax):
        xold = x
        x = x - f(x)/df(x)
        Er = np.abs((x-xold)/x)
        sk = np.abs(x - xold)
        it += 1

        line_data.append([it, x, f(x), df(x), sk])

# configura a tabela
    fig, ax = plt.subplots(figsize=(8,6))
    ax.set_axis_off()
    table = ax.table(cellText=line_data,
                     cellLoc='center',
                     colLabels=column_labels,
                     loc='center')
    plt.savefig('Pt1_Ex3.2.pdf', dpi=200)
    plt.show()

# configura o grafico
    ticks_frequency = 1

    fig, ax = plt.subplots(figsize = (10, 10))
    fig.patch.set_facecolor('#ffffff')

    ax.set(xlim=(-6,6), ylim=(-6,6), aspect='equal')
    ax.spines['bottom'].set_position('zero')
    ax.spines['left'].set_position('zero')

    ax.spines['top'].set_visible(False)
    ax.spines['right'].set_visible(False)

    ax.set_xlabel('$x$', size=14, labelpad=-24, x=1.02)
    ax.set_ylabel('$y$', size=14, labelpad=-21, y=1.02, rotation=0)

    plt.text(0.49, 0.49, r"$O$", ha='right', va='top',
```

```
      transform=ax.transAxes,
        horizontalalignment='center', fontsize=14)

    x_ticks = np.arange(-5, 6, ticks_frequency)
    y_ticks = np.arange(-5, 6, ticks_frequency)
    ax.set_xticks(x_ticks[x_ticks != 0])
    ax.set_yticks(y_ticks[y_ticks != 0])
    ax.set_xticks(np.arange(-5, 6), minor=True)
    ax.set_yticks(np.arange(-5, 6), minor=True)

    ax.grid(which='both', color='grey', linewidth=1, linestyle='-', alpha=0.2)

    # Create the plot

    x = np.linspace(-5, 10, 100)
    y = np.arctan(x)

    plt.plot(x, y, 'b', linewidth=2)

    # Show the plot
    plt.show()

#Configura e calcula por Newton
x0 = 1
Erro = 10**-8
itMax = 20

Newton(x0, Erro, itMax)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:23: RuntimeWarning: divi
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:23: RuntimeWarning: inva
```

k	xk	f(xk)	df(xk)	stepk
0	1	0.7853981633974483	0.5	-
1	-0.5707963267948966	-0.5186693692550166	0.7542567725392094	1.5707963267948966
2	0.1168599039989131	0.11633226511389591	0.9865277431717276	0.6876562307938097
3	-0.0010610221170447 16	-0.0010610217188900932	0.9999988742333344	0.11792092611595782
4	7.963096044106416e-10	7.963096044106416e-10	1.0	0.0010610229133543204
5	0.0	0.0	1.0	7.963096044106416e-10
6	0.0	0.0	1.0	0.0



Parte 2

2 Prove que a equação (2) possui pelo menos uma raiz no intervalo [0, 1] na variável x.

```
def f(x):
  F = 1.5
  p = 2*10**(-5)
  q = 5*10**(-5)
  r = 1
  e0 = 8.85*10**(-12)
  return x / math.sqrt((x**2 + r**2)**3) - 4 * math.pi * e0 * F / p * q

def bisseccao(intervalo, erro, it_max):
    ak = intervalo[0]
    bk = intervalo[1]
    xk = (ak + bk)/2
    sk = bk - ak
    it = 0
    # lista das k linhas
    line_data = []
    # colunas necessarias para plotar
    column_labels = ['k', 'xk', 'f(xk)', 'stepk']

    while(np.abs(bk - ak) > erro and it < it_max):
      xk = (ak + bk) / 2
```

```
    sk = np.abs(bk - ak) # como nao tem do-while em python vai ter que fazer a diferença
    it += 1

    # se chegou aqui eh pq o teorema de bolzano nao pode provar que existe ao menos um z
    if (f(ak) > 0 and f(bk) > 0 and f(xk) > 0) or (f(ak) < 0 and f(bk) < 0 and f(xk) < 0
      print("Pelo Teorema de Bolzano, temos que f(a)f(b) >= 0, logo não podemos afirmar
      return

    # saida: x*, f(x*), num iteracoes. vamos fazendo isso inserindo cada tupla de linha
    line_data.append([it, xk, f(xk), sk])

    # f(ak) < 0 and f(bk) > 0 and f(xk) > 0, then we gonna use ak as ak and bk as xk
    if(f(ak) < 0 and f(bk) > 0 and f(xk) > 0):
      bk = xk
      continue

    # f(ak) > 0 and f(bk) < 0 and f(xk) > 0, then we gonna use ak as xk and bk as bk
    if(f(ak) > 0 and f(bk) < 0 and f(xk) > 0):
      ak = xk
      continue

    # f(ak) > 0 and f(bk) > 0 and f(xk) < 0 and f(ak) > f(bk), then we gonna use ak as b
    if(f(ak) > 0 and f(bk) > 0 and f(xk) < 0 and f(ak) > f(bk)):
      ak = bk
      bk = xk
      continue

    # f(ak) > 0 and f(bk) > 0 and f(xk) < 0 and f(ak) < f(bk), then we gonna use ak as a
    if(f(ak) > 0 and f(bk) > 0 and f(xk) < 0 and f(ak) < f(bk)):
      bk = xk
      continue

    # f(ak) < 0 and f(bk) < 0 and f(xk) > 0 and f(ak) > f(bk), then we gonna use ak as a
    if(f(ak) < 0 and f(bk) < 0 and f(xk) > 0 and f(ak) > f(bk)):
      bk = xk
      continue

    # f(ak) < 0 and f(bk) < 0 and f(xk) > 0 and f(ak) < f(bk), then we gonna use ak as x
    if(f(ak) < 0 and f(bk) < 0 and f(xk) > 0 and f(ak) < f(bk)):
      ak = xk
      continue

    # f(ak) > 0 and f(bk) < 0 and f(xk) < 0, then we gonna use ak as ak and bk as xk
    if(f(ak) > 0 and f(bk) < 0 and f(xk) < 0):
      bk = xk
      continue

    # f(ak) < 0 and f(bk) > 0 and f(xk) < 0, then we gonna use ak as xk and bk as bk
    if(f(ak) < 0 and f(bk) > 0 and f(xk) < 0):
      ak = xk
      continue

  # configura a quantidade de graficos
  fig, ax = plt.subplots(figsize=(8,6))
  ax.set_axis_off()
```

```
    table = ax.table(cellText=line_data,
                     cellLoc='center',
                     colLabels=column_labels,
                     loc='center')
    plt.savefig('Tabela_Bisseccao.pdf', dpi=200)
    plt.show()


erro = 10**-8
it_max = 1
intervalo = (0, 1)

bisseccao(intervalo, erro, it_max)
```

k	xk	f(xk)	stepk
1	0.5	0.35777087598291996	1

3. Utilize o código implementado do Método da Bissecção para encontrar uma raiz aproximada da equação (2) no intervalo [0, 1] , adotando ε = 10−8 e maxit = 50.

```
def f(x):
  F = 1.5
  p = 2*10**(-5)
  q = 5*10**(-5)
  r = 1
  e0 = 8.85*10**(-12)
  return (x / math.sqrt((x**2 + r**2)**3)) - 4 * math.pi * e0 * F / p * q

def bisseccao(intervalo, erro, it_max):
    ak = intervalo[0]
    bk = intervalo[1]
    xk = (ak + bk)/2
    sk = bk - ak
    it = 0
    # lista das k linhas
    line_data = []
```

```
    # colunas necessarias para plotar
    column_labels = ['k', 'xk', 'f(xk)', 'stepk']

    while(np.abs(bk - ak) > erro and it < it_max):
      xk = (ak + bk) / 2
      sk = np.abs(bk - ak) # como nao tem do-while em python vai ter que fazer a diferença
      it += 1

      # se chegou aqui eh pq o teorema de bolzano nao pode provar que existe ao menos um z
      if (f(ak) > 0 and f(bk) > 0 and f(xk) > 0) or (f(ak) < 0 and f(bk) < 0 and f(xk) < 0
        print("Pelo Teorema de Bolzano, temos que f(a)f(b) >= 0, logo não podemos afirmar
        return

      # saida: x*, f(x*), num iteracoes. vamos fazendo isso inserindo cada tupla de linha
      line_data.append([it, xk, f(xk), sk])

      # f(ak) < 0 and f(bk) > 0 and f(xk) > 0, then we gonna use ak as ak and bk as xk
      if(f(ak) < 0 and f(bk) > 0 and f(xk) > 0):
        bk = xk
        continue

      # f(ak) > 0 and f(bk) < 0 and f(xk) > 0, then we gonna use ak as xk and bk as bk
      if(f(ak) > 0 and f(bk) < 0 and f(xk) > 0):
        ak = xk
        continue

      # f(ak) > 0 and f(bk) > 0 and f(xk) < 0 and f(ak) > f(bk), then we gonna use ak as b
      if(f(ak) > 0 and f(bk) > 0 and f(xk) < 0 and f(ak) > f(bk)):
        ak = bk
        bk = xk
        continue

      # f(ak) > 0 and f(bk) > 0 and f(xk) < 0 and f(ak) < f(bk), then we gonna use ak as a
      if(f(ak) > 0 and f(bk) > 0 and f(xk) < 0 and f(ak) < f(bk)):
        bk = xk
        continue

      # f(ak) < 0 and f(bk) < 0 and f(xk) > 0 and f(ak) > f(bk), then we gonna use ak as a
      if(f(ak) < 0 and f(bk) < 0 and f(xk) > 0 and f(ak) > f(bk)):
        bk = xk
        continue

      # f(ak) < 0 and f(bk) < 0 and f(xk) > 0 and f(ak) < f(bk), then we gonna use ak as x
      if(f(ak) < 0 and f(bk) < 0 and f(xk) > 0 and f(ak) < f(bk)):
        ak = xk
        continue

      # f(ak) > 0 and f(bk) < 0 and f(xk) < 0, then we gonna use ak as ak and bk as xk
      if(f(ak) > 0 and f(bk) < 0 and f(xk) < 0):
        bk = xk
        continue

      # f(ak) < 0 and f(bk) > 0 and f(xk) < 0, then we gonna use ak as xk and bk as bk
      if(f(ak) < 0 and f(bk) > 0 and f(xk) < 0):
        ak = xk
```

```
        continue

    # configura a quantidade de graficos
    fig, ax = plt.subplots(figsize=(8,6))
    ax.set_axis_off()
    table = ax.table(cellText=line_data,
                     cellLoc='center',
                     colLabels=column_labels,
                     loc='center')
    plt.savefig('Tabela_Bisseccao.pdf', dpi=200)
    plt.show()


erro = 10**-8
it_max = 50
intervalo = (0, 1)

bisseccao(intervalo, erro, it_max)
```

k	xk	f(xk)	stepk
1	0.5	0.35777087598291996	1
2	0.25	0.2252688231465611	0.5
3	0.125	0.1221265074861742	0.25
4	0.0625	0.06213556867138244	0.125
5	0.03125	0.03120427903152579	0.0625
6	0.015625	0.015619279282787722	0.03125
7	0.0078125	0.007811784381 7820665	0.015625
8	0.00390625	0.0039061601776916544	0.0078125
9	0.001953125	0.0019531134071359704	0.00390625
10	0.0009765625	0.0009765606859713787	0.001953125
11	0.00048828125	0.00048828065833064456	0.0009765625
12	0.000244140625	0.00024414018612570402	0.00048828125
13	0.0001220703125	0.00012206989272509118	0.000244140625
14	6.103515625e-05	6.103473886251472e-05	0.0001220703125
15	3.0517578125e-05	3.051716103594267e-05	6.103515625e-05
16	1.52587890625e-05	1.5258372010746166e-05	3.0517578125e-05
17	7.62939453125e-06	7.628977484159102e-06	1.52587890625e-05
18	3.814697265625e-06	3.8142802191169693e-06	7.62939453125e-06
19	1.9073486328125e-06	1.9069315563773277e-06	3.814697265625e-06
20	9.5367431640625e-07	9.532572699801849e-07	1.9073486328125e-06
21	4.76837158203125e-07	4.7642011177819834e-07	9.5367431640625e-07
22	2.384185791015625e-07	2.3800153267677812e-07	4.76837158203125e-07
23	1.1920928955078125e-07	1.1879224312601467e-07	2.384185791015625e-07
24	5.960464477539063e-08	5.918759835062626e-08	1.1920928955078125e-07
25	2.9802322387695312e-08	2.9385275962931228e-08	5.960464477539063e-08
26	1.4901161193847656e-08	1.4484114769083608e-08	2.9802322387695312e-08
27	7.450580596923828e-09	7.033534172159783e-09	1.4901161193847656e-08

4. Tomando o ponto inicial x0 = 0.3, utilize o código implementado do Método de Newton para encontrar uma raiz aproximada da equação (2), adotando ε = 10−8 e maxit = 20.

```
def f(x):
  F = 1.5
  p = 2*10**(-5)
  q = 5*10**(-5)
  r = 1
  e0 = 8.85*10**(-12)
  return (x / math.sqrt((x**2 + r**2)**3)) - 4 * math.pi * e0 * F / p * q

def df(x):
  F = 1.5
  p = 2*10**(-5)
```

```
  q = 5*10**(-5)
  r = 1
  e0 = 8.85*10**(-12)
  return ((r**2) - 2* (x**2)) / (math.sqrt(x**2 + r**2) * (x**2 + r**2)**2)

# lista das k linhas
line_data = []
# colunas necessarias para plotar
column_labels = ['it', 'xk', 'f(xk)', 'df(xk)', 'stepk']

def newton(x0, erro, it_max):
    x = x0
    it = 0
    # evitar erro na primeira iteracao
    sk = erro + 1

    line_data.append([it, x, f(x), df(x),'-'])

    while(sk >= erro and it < it_max):
        xold = x
        x = x - f(x)/df(x)
        sk = np.abs(x - xold)
        it += 1

        line_data.append([it, x, f(x), df(x), sk])

    # configura a tabela
    fig, ax = plt.subplots(figsize=(8,6))
    ax.set_axis_off()
    table = ax.table(cellText=line_data,
                     cellLoc='center',
                     colLabels=column_labels,
                     loc='center')
    plt.savefig('Pt1_Ex2.1.pdf', dpi=200)
    plt.show()

x0 = 0.3
erro = 10**-8
it_max = 20

newton(x0, erro, it_max)
```

5. Tomando os pontos iniciais x0 = 0.3 e x1 = 0.6 , utilize o código implementado do Método da Secante para encontrar uma raiz aproximada da equação (2), adotando ε = 10−8 e maxit = 20.

```
def f(x):
  F = 1.5
  p = 2*10**(-5)
  q = 5*10**(-5)
  r = 1
  e0 = 8.85*10**(-12)
  return (x / math.sqrt((x**2 + r**2)**3)) - 4 * math.pi * e0 * F / p * q

def secante(x0, x1, erro, it_max):
    xk1 = x0
    xk2 = x1
    it = 0
    sk = np.abs(xk2 - xk1)
    # lista das k linhas
    line_data = []
    # colunas necessarias para plotar
    column_labels = ['k', 'xk', 'f(xk)', 'stepk']

    """
    xk1 = xk-1
    xk2 = xk
    x = xk+1
    """

    while(sk >= erro and abs(f(xk2)) >= erro and abs(f(xk1)) >= erro and it < it_max):
      x = xk2 - f(xk2)*(xk2 - xk1)/(f(xk2) - f(xk1))
      xk1 = xk2
      xk2 = x
      it += 1
      sk = np.abs(xk2 - xk1)
      line_data.append([it, x, f(x), sk])

    # configura o tamanho do grafico
    fig, ax = plt.subplots(figsize=(6,2))
    ax.set_axis_off()
    table = ax.table(cellText=line_data,
                     cellLoc='center',
                     colLabels=column_labels,
                     loc='center')
    plt.savefig('Tabela_Secante.pdf', dpi=200)
    plt.show()
```

```
x0 = 0.3
x1 = 0.6
erro = 10**-8
it_max = 20

secante(x0, x1, erro, it_max)
```

k	xk	f(xk)	stepk
1	-0.38980551849194325	-0.31518370293373316	0.9898055184919432
2	0.060159870130452764	0.05983474422495841	0.449765388622396
3	-0.011600855842398087	-0.011598517371093893	0.07176072855443383
4	5.0829292696895e-05	5.082887545348851e-05	0.011651687716677765
5	-9.799149091159888e-09	-1.0216195515923933e-08	5.083909184598616e-05
6	4.170484643485185 4e-10	3.958447364889162 8e-17	1.0216195555508408e-08

6. Aplique novamente o Método de Newton para encontrar uma raiz aproximada da equação (2), tomando o ponto inicial x0 = 0.7, ε = 10−4 e maxit = 10 . Explique e justifique o comportamento observado.

```
def f(x):
  F = 1.5
  p = 2*10**(-5)
  q = 5*10**(-5)
  r = 1
  e0 = 8.85*10**(-12)
  return (x / math.sqrt((x**2 + r**2)**3)) - 4 * math.pi * e0 * F / p * q

def df(x):
  F = 1.5
  p = 2*10**(-5)
  q = 5*10**(-5)
  r = 1
  e0 = 8.85*10**(-12)
  return ((r**2) - 2* (x**2)) / (math.sqrt(x**2 + r**2) * (x**2 + r**2)**2)

# lista das k linhas
line_data = []
# colunas necessarias para plotar
column_labels = ['it', 'xk', 'f(xk)', 'df(xk)', 'stepk']

def newton(x0, erro, it_max):
    x = x0
    it = 0
    # evitar erro na primeira iteracao
    sk = erro + 1

    line_data.append([it, x, f(x), df(x),'-'])

    while(sk >= erro and it < it_max):
        xold = x
        x = x - f(x)/df(x)
        sk = np.abs(x - xold)
```

```
        it += 1

        line_data.append([it, x, f(x), df(x), sk])

    # configura a tabela
    fig, ax = plt.subplots(figsize=(8,6))
    ax.set_axis_off()
    table = ax.table(cellText=line_data,
                     cellLoc='center',
                     colLabels=column_labels,
                     loc='center')
    plt.savefig('Pt1_Ex2.1.pdf', dpi=200)
    plt.show()

x0 = 0.7
erro = 10**-4
it_max = 10

newton(x0, erro, it_max)
```

it	xk	f(xk)	df(xk)	stepk
0	0.7	0.38487405620263704	0.007380135313898183	-
1	-51.44999994349034	-0.0003775579864538156	-1.4668365980508539e-05	52.14999994349034
2	-77.18980836037319	-0.00016779309276927472	-4.346451398556673e-06	25.739808416882845
3	-115.79422564080426	-7.457272962260963e-05	-1.2878702769121848e-06	38.60461728043107
4	-173.69813953608121	-3.314313150398834e-05	-3.8159380243781925e-07	57.90391389527696
5	-260.55262011792195	-1.4730307418972722e-05	-1.1306403220185631e-07	86.85448058184073
6	-390.83549728149507	-6.5468953494884194e-06	-3.349958774891418e-08	130.28287716357312
7	-586.2676141946539	-2.9098362187884843e-06	-9.925182305224117e-09	195.43211691315884
8	-879.4447195921933	-1.2933680466059846e-06	-2.940374679140079e-09	293.17710539753944
9	-1319.309766849473	-5.749382621508745e-07	-8.70941359091252e-10	439.86504705727964
10	-1979.4440640386454	-2.556362600320027e-07	-2.5786949026810433e-10	660.1342973891724