

Atividade 2

Exercícios referentes aos slides da Aula 6 parte 3.

Aluno: Camilo Henrique Martins dos Santos - 202004940027

Linguagem utilizada: python

Bibliotecas de python utilizadas neste relatório:

In [149]:

```
import math
import pandas as pd
```

Etapa 1- Definindo funções

Abaixo estão as 5 funções que serão utilizadas nesta atividade, as quais seguem as seguintes regras do comando da questão

- Teste esses algoritmos em 5 funções que misturam:
 - Polinômios de graus > 5
 - Logaritmos
 - Exponenciais
 - Senos e cossenos
 - Divisões e raízes

Funções utilizadas neste trabalho:

Função 1

$$6x^6 - 5x - 4x + 3x^3 - 2x^2 + x + 2$$

Função 2

$$\log_{10}(25 + x^2) + 2x$$

Função 3

$$2^x - 5x + 3$$

Função 4

$$\frac{\sin(x) + 2x}{\cos(x)} - 5$$

Função 5

$$\frac{2x}{\sqrt{x}} - 9$$

In [150]:

```
def f1(x,boole=True): #(0.240882, 0.966543)
    if boole:
        return 6*(x**6) - x**5 - x**4 + 3*(x**3) - 2*(x**2) + x + 2
    return 36*(x**5) + 9*(x**2) - 4*x - 8 # derivada

def f2(x,boole=True): # -0,70322344
    if boole:
        a = 25 + x**2
        try:
            return math.log10(a) + 2*x
        except:
            return 'undefined'
    return 2*x/(math.log(10) * (25+x**2)) + 2 # derivada

def f3(x,boole=True): # (1 , 4.14957)
    if boole:
        return 2**x - 5*x + 3
    return (2**x) * math.log(2) - 5 # derivada

def f4(x,boole=True):# 1.042494987
    if boole:
        return math.sin(x) + (2*x / math.cos(x)) - 5
    return (1/math.cos(x)) * ((math.cos(x))**2 + (2*x) * math.tan(x) + 2) # derivada

def f5(x,boole=True): # 20.25
    if boole:
        try:
            return (2*x / math.sqrt(x)) - 9
        except:
            return 'undefined'
    return 1/math.sqrt(x) # derivada
```

In [151]:

```
def mostrarTabela(f,intervalo,step):
    x = []
    if len(intervalo) > 2:
        x = intervalo
    else:
        length = math.floor((intervalo[1]-intervalo[0])/step)
        #x = len
        val = intervalo[0]
        for i in range(length):
            x.append(val)
            val += step
    for i in x:
        print(f'x: {i} y: {f(i)}')
```

Etapa 2 - Definindo funções de refinamento

Abaixo estão as três funções de refinamento desenvolvidas para esta atividade:

- Bissecção

- Secante
- Newton-Raphson

In [152]:

```
#bissecção
def medium_point(a, b):
    pi = (a + b) / 2
    return pi

def bisseccao(a, b, e, func):
    ai = a
    bi = b

    pi = medium_point(a, b)
    iter = 0
    while abs(func(pi)) > e:
        pi = medium_point(ai, bi)

        if func(ai) * func(pi) < 0:
            bi = pi
        else:
            ai = pi
        iter += 1
    root = pi
    return (root,str(func(root)),iter)
```

In [153]:

```
# método de secante
def secante(x0,x1,e,f):
    k = 0

    while abs(f(x1)) > e and abs(x1 - x0) > e:
        x2 = x1 - f(x1) * (x1-x0)/(f(x1) - f(x0))
        x0 = x1
        x1 = x2
        k += 1
    return (x1,str(f(x1)),k)
```

In [154]:

```
# Método de Newton-Raphson
def newton(x0,e,f):
    k = 1
    if (abs(f(x0)) < e):
        return (x0,f(x0),k)
    x1 = x0 - (f(x0) / f(x0,False))

    while True:
        x1 = x0 - (f(x0) / f(x0,False))

        if abs(f(x1)) < e or abs(x1 - x0) < e:
            return (x1, str(f(x1)),k)
        x0 = x1
        k += 1
```

Etapa 3 - Definindo o intervalo das raízes

Abaixo foi desenvolvido uma função capaz de determinar o Y de uma função dado um intervalo de valores de X. Dessa forma pode-se determinar o intervalo da raiz desta função, pela mudança do Y, de positivo para negativo, ou de negativo para positivo.

In [155]:

```
def mostrarTabela(f,intervalo,step):
    x = []
    if len(intervalo) > 2:
        x = intervalo
    else:
        length = math.floor((intervalo[1]-intervalo[0])/step)
        val = intervalo[0]
        for i in range(length):
            x.append(val)
            val += step
    for i in x:
        print(f'x: {i} y: {f(i)}')
```

Etapa 3.1 tabelas do intervalo das raízes

In [156]:

```
print('Função 1')
mostrarTabela(f1,[-3,3],0.5)
```

```
Função 1
x: -3 y: 4301
x: -2.5 y: 1427.46875
x: -2.0 y: 370.0
x: -1.5 y: 67.71875
x: -1.0 y: 11.0
x: -0.5 y: 5.21875
x: 0.0 y: 2.0
x: 0.5 y: -2.03125
x: 1.0 y: 1.0
x: 1.5 y: 63.96875
x: 2.0 y: 386.0
x: 2.5 y: 1481.21875
```

Na função 1, x: 0 e x: 0.5 apresentaram mudança de sinal, significando que entre esses valores de x há uma raiz. Por isso, x0 e x1 para função 1 serão -1 e 1 respectivamente.

In [157]:

```
print('Função 2')
mostrarTabela(f2, [-5,5],1)
```

Função 2

x: -5 y: -8.301029995663981
x: -4 y: -6.3872161432802645
x: -3 y: -4.468521082957745
x: -2 y: -2.537602002101044
x: -1 y: -0.585026652029182
x: 0 y: 1.3979400086720377
x: 1 y: 3.414973347970818
x: 2 y: 5.462397997898956
x: 3 y: 7.531478917042255
x: 4 y: 9.612783856719735

Na função 2, x: -1 e x: 0 apresentaram mudança de sinal, significando que entre esses valores de x há uma raiz. Por isso, x0 e x1 para função 1 serão -1 e 1 respectivamente.

In [158]:

```
print('Função 3')
mostrarTabela(f3, [-5,5],1)
```

Função 3

x: -5 y: 28.03125
x: -4 y: 23.0625
x: -3 y: 18.125
x: -2 y: 13.25
x: -1 y: 8.5
x: 0 y: 4
x: 1 y: 0
x: 2 y: -3
x: 3 y: -4
x: 4 y: -1

Na função 3, x: 0 e x: -3 apresentaram mudança de sinal, significando que entre esses valores de x há uma raiz. Por isso, x0 e x1 para função 1 serão 0.5 e 2 respectivamente.

In [159]:

```
print('Função 4')
mostrarTabela(f4, [-2,3],0.5)
```

Função 4

x: -2 y: 3.7026944200638425
x: -1.5 y: -48.40799369551377
x: -1.0 y: -9.543102420169747
x: -0.5 y: -6.618919465928752
x: 0.0 y: -5.0
x: 0.5 y: -3.381080534071248
x: 1.0 y: -0.4568975798302528
x: 1.5 y: 38.40799369551377
x: 2.0 y: -13.702694420063843
x: 2.5 y: -10.642606113240133

Na função 4, x: 1 e x: 1.5 apresentaram mudança de sinal, significando que entre esses valores de x há uma

raiz. Por isso, x0 e x1 para função 1 serão 1 e 2 respectivamente.

In [160]:

```
print('Função 5')
mostrarTabela(f5,[16,26],1)
```

```
Função 5
x: 16 y: -1.0
x: 17 y: -0.7537887487646788
x: 18 y: -0.5147186257614287
x: 19 y: -0.2822021129186538
x: 20 y: -0.05572809000084078
x: 21 y: 0.16515138991167966
x: 22 y: 0.38083151964685946
x: 23 y: 0.59166304662544
x: 24 y: 0.7979589711327133
x: 25 y: 1.0
```

Na função 5, x: 20 e x: 21 apresentaram mudança de sinal, significando que entre esses valores de x há uma raiz. Por isso, x0 e x1 para função 1 serão 20 e 21 respectivamente

In [161]:

```
x0_f1,x1_f1 = -1,1
x0_f2,x1_f2 = -1,1
x0_f3,x1_f3 = 0.5,2
x0_f4,x1_f4 = 1,2
x0_f5,x1_f5 = 20,21
```

Etapa 4 - Execução dos refinamentos

Determinado as funções (Etapa 1) e o intervalo de suas raízes (Etapa 2), é possível utilizar estes intervalos nas funções de refinamento desenvolvidas (Etapa 3) para achar uma das raízes de cada função. Vale ressaltar que a precisão utilizada é igual a 10^{-7} .

In [162]:

```
e = 0.0000001 #precisão

f1_bisseccao = bisseccao(x0_f1,x1_f1,e,f1)
f2_bisseccao = bisseccao(x0_f2,x1_f2,e,f2)
f3_bisseccao = bisseccao(x0_f3,x1_f3,e,f3)
f4_bisseccao = bisseccao(x0_f4,x1_f4,e,f4)
f5_bisseccao = bisseccao(x0_f5,x1_f5,e,f5)

f1_secante = secante(x0_f1,x1_f1,e,f1)
f2_secante = secante(x0_f2,x1_f2,e,f2)
f3_secante = secante(x0_f3,x1_f3,e,f3)
f4_secante = secante(x0_f4,x1_f4,e,f4)
f5_secante = secante(x0_f5,x1_f5,e,f5)

f1_newton = newton(x0_f1,e,f1)
f2_newton = newton(x0_f2,e,f2)
f3_newton = newton(x0_f3,e,f3)
f4_newton = newton(x0_f4,e,f4)
f5_newton = newton(x0_f5,e,f5)
```

Etapa 5 - visualização dos dados

Nesta última etapa, é possível visualizar cada uma das saídas das funções de refinamento em suas respectivas tabelas, inclusive, a raiz encontrada em cada função. As tabelas foram desenvolvidas por meio da biblioteca Pandas.

In [163]:

```
def tabela(entradas,Lbissec,LSec,LNew):
    df = pd.DataFrame({'Entrada e saída':['Dados Iniciais','raiz retornada','f(x)','Número
        'Bisseção':[[entradas[0],entradas[1]],Lbissec[0],Lbissec[1],Lbissec[2]],
        'Secante':[[entradas[0],entradas[1]],LSec[0],LSec[1],LSec[2]]],
        'Newton-Raphson':[entradas[0],LNew[0],LNew[1],LNew[2]]})}

    return df
```

Tabela da função 1:

$$6x^6 - 5x - 4x + 3x^3 - 2x^2 + x + 2$$

In [164]:

```
df1 = tabela([x0_f1,x1_f1],f1_bisseccao,f1_secante,f1_newton)
df1
```

Out[164]:

	Entrada e saída	Bissecção	Secante	Newton-Raphson
0	Dados Iniciais	[-1, 1]	[-1, 1]	-1
1	raiz retornada	0.240882	0.966543	0.240882
2	f(x)	-5.492491439440528e-08	1.0435652342266621e-11	4.7273296388539165e-12
3	Número de Iterações	25	7	16

Tabela da função 2:

$$\log_{10}(25 + x^2) + 2x$$

In [165]:

```
df2 = tabela([x0_f2,x1_f2],f2_bisseccao,f2_secante,f2_newton)
df2
```

Out[165]:

	Entrada e saída	Bissecção	Secante	Newton-Raphson
0	Dados Iniciais	[-1, 1]	[-1, 1]	-1
1	raiz retornada	-0.703223	-0.703223	-0.703223
2	f(x)	-3.881092336399661e-08	4.298019939952269e-09	8.157519992835205e-09
3	Número de Iterações	23	3	2

Tabela da função 3:

$$2^x - 5x + 3$$

In [166]:

```
df3 = tabela([x0_f3,x1_f3],f3_bisseccao,f3_secante,f3_newton)
df3
```

Out[166]:

	Entrada e saída	Bissecção	Secante	Newton-Raphson
0	Dados Iniciais	[0.5, 2]	[0.5, 2]	0.5
1	raiz retornada	1.0	1.0	1.0
2	f(x)	-5.3848410175305617e-08	4.7830184257691144e-11	2.6310309486632377e-09
3	Número de Iterações	25	5	4

Tabela da função 4:

$$\sin(x) + \frac{2x}{\cos(x)} - 5$$

In [167]:

```
df4 = tabela([x0_f4,x1_f4],f4_bisseccao,f4_secante,f4_newton)
df4
```

Out[167]:

	Entrada e saída	Bissecção	Secante	Newton-Raphson
0	Dados Iniciais	[1, 2]	[1, 2]	1
1	raiz retornada	1.042495	1.042495	1.042495
2	f(x)	4.6763713790198835e-08	-3.448024976648867e-09	6.544624397974985e-09
3	Número de Iterações	26	7	4

Tabela da função 5:

$$\frac{2x}{\sqrt{x}} - 9$$

In [168]:

```
df5 = tabela([x0_f5,x1_f5],f5_bisseccao,f5_secante,f5_newton)
df5
```

Out[168]:

	Entrada e saída	Bissecção	Secante	Newton-Raphson
0	Dados Iniciais	[20, 21]	[20, 21]	20
1	raiz retornada	20.25	20.25	20.25
2	f(x)	0.0	1.332534083076098e-10	-1.653818415547903e-09
3	Número de Iterações	2	3	2