Universidade de São Paulo – USP

Instituto de Física

Cálculo Numérico com Aplicações em Física - EP1

Aluno: Raphael Rolim

Agosto/2024

Conteúdo

Preâmbulo

```python
import numpy as np
import matplotlib.pyplot as plt
```

Funções

```python
def root_with_bissection_method(function, x1, x2, error_acceptance, save_data=False, data_name='bm_data'):
    import pandas as pd
    import numpy as np

    def f(x):
        f = eval(function)
        return f

    data = {'x_1':[], 'x_2':[], 'x_m': [], 'f(x_2)': [], 'f(x_m)': [], 'e_n': []}

    while np.abs(x1 - x2) > error_acceptance:
        xm = (x1 + x2) / 2
        data['x_1'].append(round(x1, 4))
        data['x_2'].append(round(x2, 4))
        data['x_m'].append(round(xm, 4))
        data['f(x_2)'].append(round(f(x2), 4))
        data['f(x_m)'].append(round(f(xm), 4))
        data['e_n'].append(round(np.abs(x1 - x2), 4))

        if f(xm) / f(x1) > 0:
            x1 = xm
        else:
            x2 = xm

    xm = (x1 + x2) / 2
    data['x_1'].append(round(x1, 4))
    data['x_2'].append(round(x2, 4))
    data['x_m'].append(round(xm, 4))
    data['f(x_2)'].append(round(f(x2), 4))
    data['f(x_m)'].append(round(f(xm), 4))
    data['e_n'].append(round(np.abs(x1 - x2), 4))

    df = pd.DataFrame(data=data)
    if save_data:
        df.to_csv(f'{data_name}.csv', index=False)

    return xm, df


def root_with_NR(function, dvfunction, x0, error_acceptance, iterations, save_data=False,
    data_name='nr_data'):
    import pandas as pd
    import numpy as np

    def f(x):
        f = eval(function)
        return f

```

```python
    def dvf(x):
        dvf = eval(dvfunction)
        return dvf

    data = {'x_n': [], 'f(x_n)': [], 'f\'(x_n)': [], 'e_n': []}

    xn = x0
    for i in range(iterations):
        a = xn - f(xn) / dvf(xn)

        data['x_n'].append(round(xn, 4))
        data['f(x_n)'].append(round(f(xn), 4))
        data['f\'(x_n)'].append(round(dvf(xn), 4))
        data['e_n'].append(round(np.abs(a - xn) / np.abs(xn), 4))

        if np.abs(a - xn) / np.abs(xn) <= error_acceptance:
            break

        xn = a

    df = pd.DataFrame(data=data)
    if save_data:
        df.to_csv(f'{data_name}.csv', index=False)

    return a, df


def root_with_secant(function, x0, x1, error_acceptance, iterations, save_data=False, data_name='sec_data'):
    import pandas as pd
    import numpy as np

    def f(x):
        return eval(function)

    data = {'x_n-1': [], 'x_n': [], 'f(x_n)': [], 'f(x_n-1)': [], 'e_n': []}

    xn_minus_1 = x0
    xn = x1

    for i in range(iterations):
        fxn = f(xn)
        fxn_minus_1 = f(xn_minus_1)

        a = xn - fxn * (xn - xn_minus_1) / (fxn - fxn_minus_1)

        data['x_n-1'].append(round(xn_minus_1, 4))
        data['x_n'].append(round(xn, 4))
        data['f(x_n)'].append(round(fxn, 4))
        data['f(x_n-1)'].append(round(fxn_minus_1, 4))
        data['e_n'].append(round(np.abs(a - xn) / np.abs(xn), 4))

        if np.abs(a - xn) / np.abs(xn) <= error_acceptance:
            break

        xn_minus_1 = xn
        xn = a

    df = pd.DataFrame(data=data)
    if save_data:
        df.to_csv(f'{data_name}.csv', index=False)

    return a, df
```

Item a)

```
bm_root, bm_data = root_with_bissection_method('x**(3/4)-np.cos(x**2)', -1, 1, 1e-4)
print('x =', np.float32(bm_root))
```

A partir do método de bissecção, em $x^{3/4} - \cos(x^2) = 0$ obtemos que $x \approx 0.774353$ para um $\epsilon = 10^{-4}$. A partir dessa condição, obtemos os seguintes valores na Tabela 1:

Tabela 1: Tabela com os valores do item a). $x_m = \frac{x_1+x_2}{2}$ e $e_n = |x_2 - x_1|$.

x_1	x_2	x_m	$f(x_2)$	$f(x_m)$	e_n
-1.0000	1.0000	0.0000	0.4597	-1.0000	2.0000
0.0000	1.0000	0.5000	0.4597	-0.3743	1.0000
0.5000	1.0000	0.7500	0.4597	-0.0400	0.5000
0.7500	1.0000	0.8750	0.4597	0.1838	0.2500
0.7500	0.8750	0.8125	0.1838	0.0659	0.1250
0.7500	0.8125	0.7812	0.0659	0.0115	0.0625
0.7500	0.7812	0.7656	0.0115	-0.0146	0.0312
0.7656	0.7812	0.7734	0.0115	-0.0016	0.0156
0.7734	0.7812	0.7773	0.0115	0.0049	0.0078
0.7734	0.7773	0.7754	0.0049	0.0017	0.0039
0.7734	0.7754	0.7744	0.0017	0.0000	0.0020
0.7734	0.7744	0.7739	0.0000	-0.0008	0.0010
0.7739	0.7744	0.7742	0.0000	-0.0004	0.0005
0.7742	0.7744	0.7743	0.0000	-0.0002	0.0002
0.7743	0.7744	0.7744	0.0000	-0.0001	0.0001
0.7744	0.7744	0.7744	0.0000	-0.0000	0.0001

Além disso, a partir do Gráfico 1, podemos ver que a função passa apenas uma vez no 0 e continua crescendo sem retornar, ou seja, x é a única raiz.

```
x = np.linspace(0, 100)

plt.plot(x, x**(3/4)-np.cos(x**2), c='indigo')
plt.axhline(0, c='black')
plt.title('Plot da função')
plt.xlabel(r'$x$')
plt.ylabel(r'$f(x)$')
plt.grid()
plt.show()
```

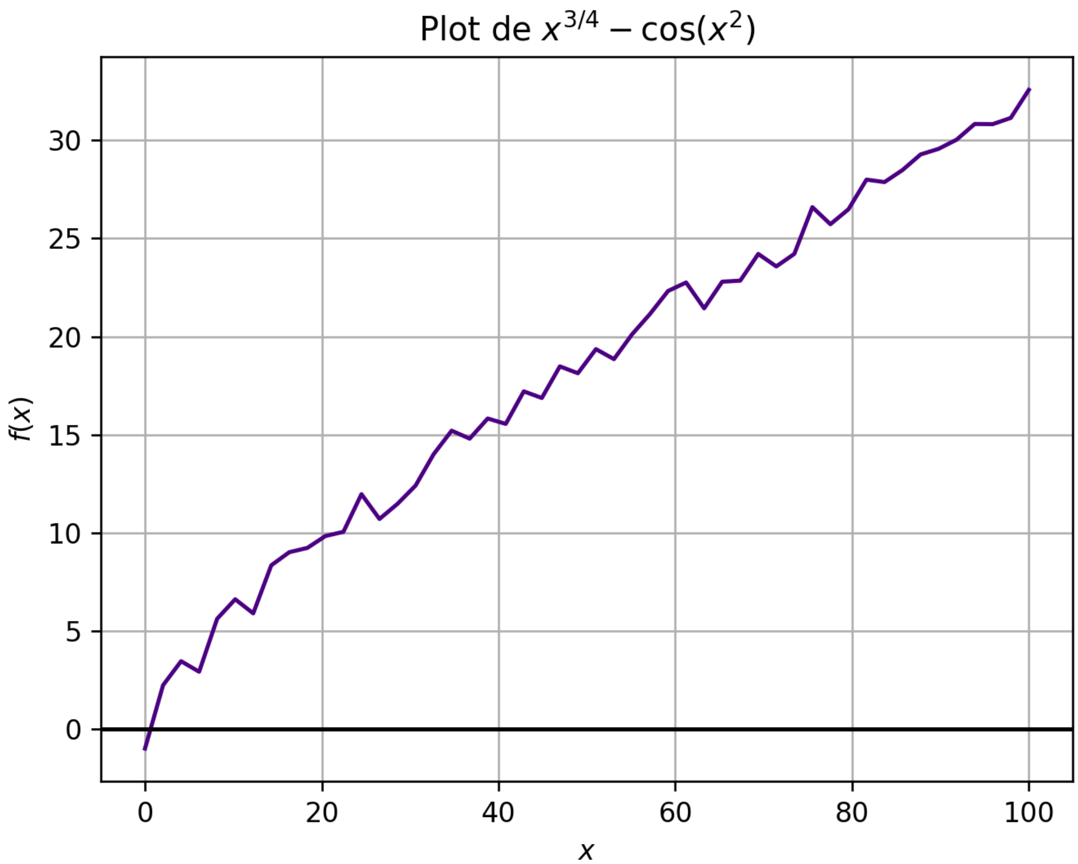

Figura 1: Gráfico da função $x^{3/4} - \cos(x^2)$.

Item b)

```
nr_root, nr_data = root_with_NR('x**(3/4)-np.cos(x**2)', '(3/4)*x**(-1/4)+2*x*np.sin(x**2)', 0.5,
    1e-4, 20)
print('x =', np.float32(nr_root))
```

Para a mesma função $x^{3/4} - \cos(x^2)$ mas com o método de Newton-Raphson, obtemos $x \approx 0.77439666$ para $\epsilon = 10^{-4}$, com a Tabela 2.

Tabela 2: Tabela com os valores do item b).

x_n	$f(x_n)$	$f'(x_n)$	e_n
0.5000	-0.3743	1.1393	0.6571
0.8285	0.0949	1.8364	0.0624
0.7768	0.0041	1.6806	0.0031
0.7744	0.0000	1.6736	0.0000

Item c)

```
# Config
V_0 = 1.38e3 #eV
r_0 = 0.328 #A
r = np.linspace(0.01,2.5,100)

def V(r):
    V = -14.4/r + V_0*np.exp(-r/r_0)
    return V

def F(r):
    F = -14.4/r**2 + (V_0/r_0)*np.exp(-r/r_0)
    return F
```

i)

```
# V(r) x r
plt.plot(r, V(r), c='indigo')
plt.axhline(0, c='black')
plt.title(r'$V(r) \times r$')
plt.xlabel(r'$r$ ($\AA$)')
plt.ylabel(r'$V(r)$ (eV)')
plt.grid()
plt.show()
```

Considerando os valores disponibilizados pelo enunciado, onde $V_0 = 1.38 \cdot 10^3$ eV, $r_0 = 0.328$ e $e^2/4\pi\epsilon_0 = 1.14$ eV , temos que para a Função 1, obtemos o Gráfico 2,

$$V(r) = -\frac{e^2}{4\pi\epsilon_0 r} + V_0 \exp\left(-\frac{r}{r_0}\right). \tag{1}$$

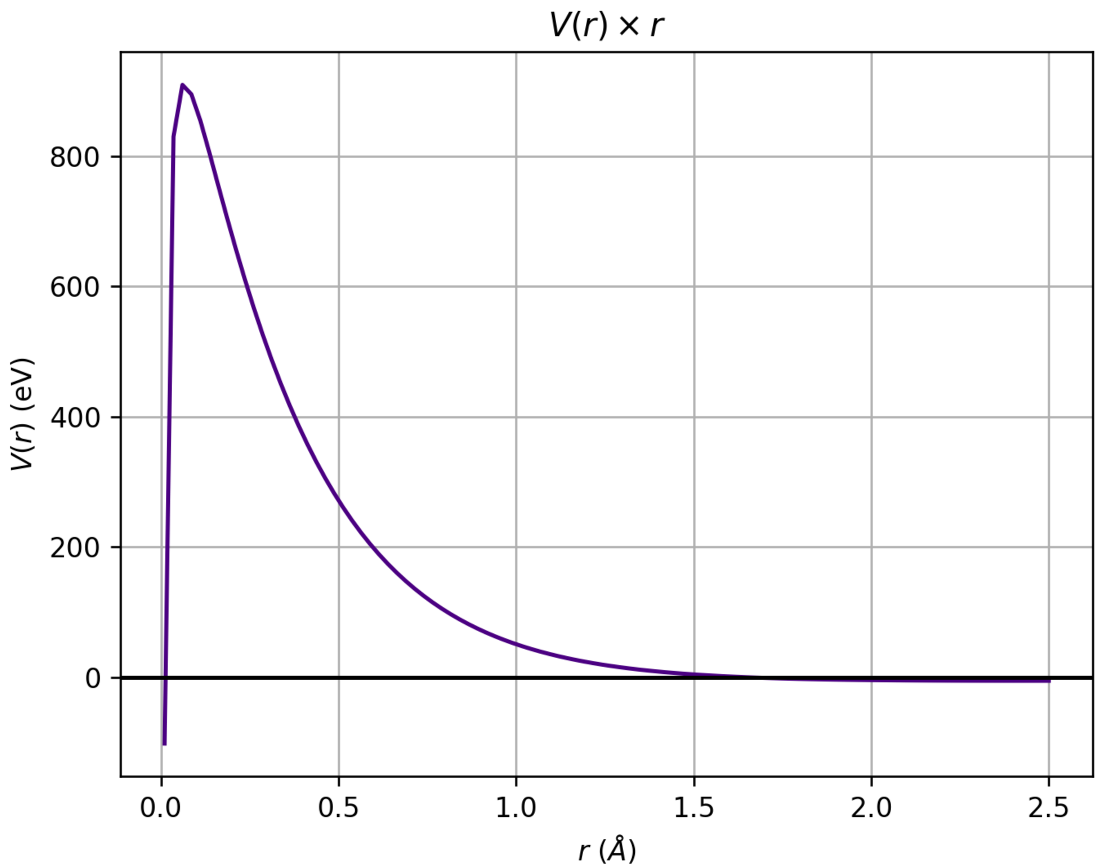

Figura 2: Gráfico de $V(r) \times r$.

```
# F(r) x r
plt.plot(r, F(r), c='indigo')
plt.axhline(0, c='black')
plt.title(r'$F(r) \times r$')
plt.xlabel(r'$r$ ($\AA$)')
plt.ylabel(r'$F(r)$ (eV/$\AA$)')
plt.ylim(-5*1e2, 2.2*1e3)
plt.grid()
plt.show()
```

Da mesma maneira, para a Função 2, obtemos o Gráfico 3,

$$F(r) = -\frac{\mathrm{d}V(r)}{\mathrm{d}r} = -\frac{e^2}{4\pi\epsilon_0 r^2} + \frac{V_0}{r_0}\exp\left(-\frac{r}{r_0}\right). \tag{2}$$

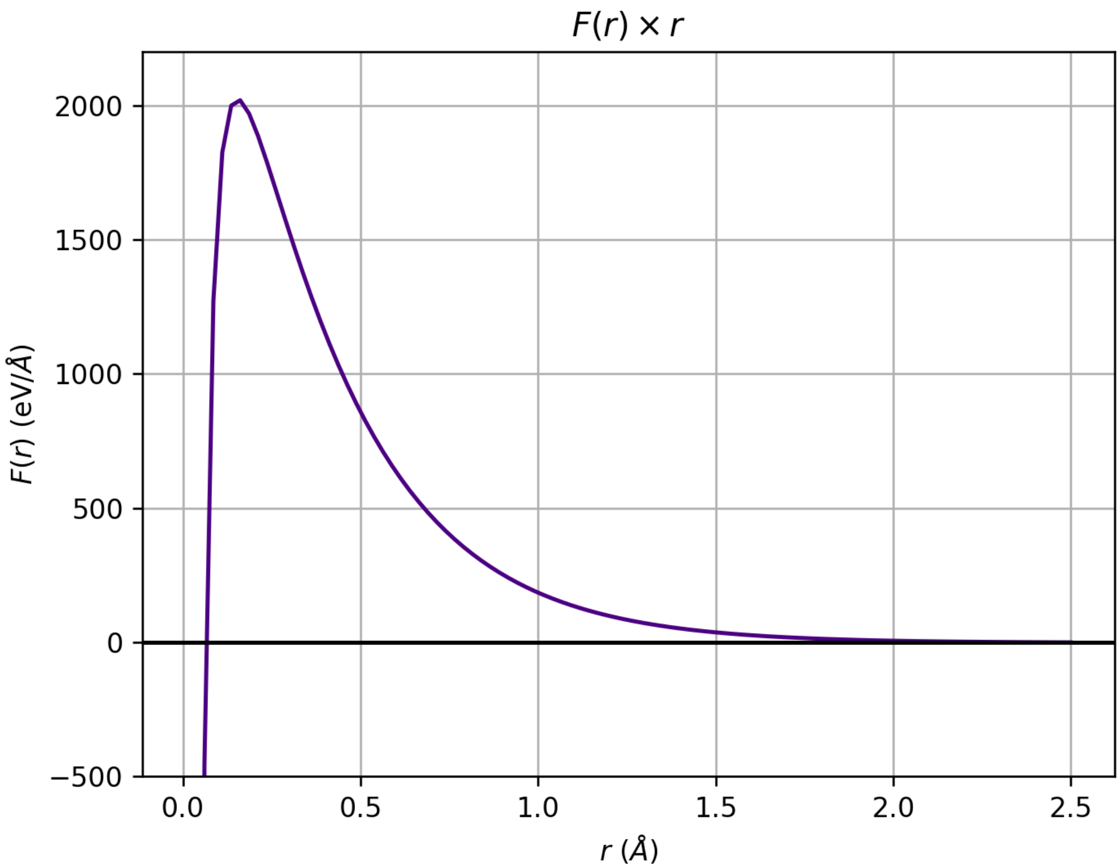

Figura 3: Gráfico de $F(r) \times r$.

ii)

```
sec_root, sec_data = root_with_secant('-14.4/x**2 + (V_0/r_0)*np.exp(-x/r_0)', 2.3, 2.7, 1e-4, 30)
print('x =', np.float32(sec_root), '(Método da Secante)')
```

Como sugerido na apostila do curso, podemos encontrar os limites em x para o método da secante a partir do Gráfico 3 onde $F(r) \approx 0$, escolhendo o limite de $x_0 = 2.3$ e $x_1 = 2.7$ para $\epsilon = 10^{-4}$, obtemos que $r_{\text{eq}} \approx 2.4500072$, onde $F(r_{\text{eq}}) = 5.39 \cdot 10^{-8} \approx 0$ e $V(r_{\text{eq}})$ é mínimo. Os valores estão disponíveis na Tabela 3

Tabela 3: Tabela com os valores do item c).

x_{n-1}	x_n	$f(x_n)$	$f(x_{n-1})$	e_n
2.3000	2.7000	-0.8558	1.0680	0.0659
2.7000	2.5221	-0.3380	-0.8558	0.0461
2.5221	2.4059	0.2565	-0.3380	0.0208
2.4059	2.4560	-0.0318	0.2565	0.0023
2.4560	2.4505	-0.0026	-0.0318	0.0002
2.4505	2.4500	0.0000	-0.0026	0.0000