MAP0214 - Cálculo Numérico
EP1

Guilherme Pacheco Paredes 12693754

28/08/2024

1

Resolva numericamente a equação

$$x^{3/4} - \cos(x^2) = 0 \tag{1.1}$$

usando o método de bisseção. Use um critério de parada escolhendo ϵ adequado. Existem outras raízes?

Solução
Primeira, plotemos o gráfico da equação (1.1) para determinar um intervalo seguro a e b. A seguir, o código em Python utilizado:

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

x1 = np.linspace(0, 6, 400)
y1 = x1 ** (3 / 4)
x2 = np.linspace(0, 6, 400)
y2 = np.cos(x2 ** 2)

plt.figure(figsize=(10, 6))
plt.plot(x1, y1, label=r"$x^{3/4}$",
color='blue')
plt.plot(x2, y2, label=r"$\cos(x^2)$",
color='red')
plt.axhline(0, color="black",linewidth=0.5)
plt.title("Grafico das Funcoes $x^{3/4} e \cos(x^2)$")
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.grid(True)
plt.xlim(0.1,6)
plt.show()
```

Listing 1: Código escrito em Python p/ o plot da função.

E o gráfico das funções:

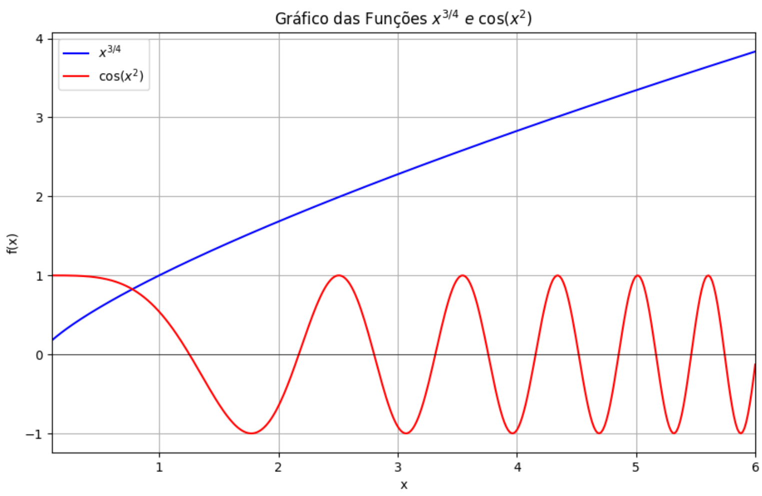

Figura 1: Gráfico das funções.

Pelo método da bisseção, definindo $\epsilon = 10^{-5}$ e o intervalo de confiança $a = 0.5$ e $b = 1.0$:

```python
import numpy as np

def bissecao(f, a, b, e=10 ** -5):

    #Verificar se a funcao muda de sinal no intervalo [a, b]
    if f(a) * f(b) >= 0:
        raise ValueError("A funcao deve ter sinais opostos em a e b.")

    iter_count = 0  #conta o numero de iteracoes
    print("-"*108)
    print(f"|{'Iteracao':^10}|{'x1':^15}|{'x2':^15}|{'xm':^15}|{'f(x1)':^15}
    |{'f(xm)':^15}|{'Precisao':^15}|")
    print("-"*108)
    while (b - a) / 2.0 > e:
        m = (a + b) / 2.0  # Ponto medio
        f_a = f(a)
        f_m = f(m)

        #Impressao dos valores atuais
        print(f"|{iter_count + 1:^10}|{a:^15.8f}|{b:^15.8f}|{m:^15.8f}|{f_a:^15.8
        f}|{f_m:^15.8f}|{abs(b - a):^15.8f}|")

        if f_m == 0.0:
            return m  # A raiz exata foi encontrada
        elif f_a * f_m < 0:
            b = m  # A raiz esta no intervalo [a, c]
        else:
            a = m  # A raiz esta no intervalo [c, b]
        iter_count += 1

    #Impressao final dos valores quando a precisao desejada e alcancada
    print(
        f"|{iter_count + 1:^10}|{a:^15.8f}|{b:^15.8f}|{(a + b) / 2.0:^15.8f}|{f(a
)
        :^15.8f}|{f((a + b)/2.0):^15.8f}|{abs(b - a):^15.8f}|")
    print("-"*108)

    return (a + b) / 2.0  # Retorna a raiz aproximada


#Definindo a funcao x^(3/4) - cos(x^2)
def minha_funcao(x):
    return x ** (3 / 4) - np.cos(x ** 2)


#Definindo intervalo
a = 0.5
b = 1.0

#Calculando a raiz
raiz = bissecao(minha_funcao, a, b)
print(f"\nA raiz aproximada e: {raiz:.8f}")
```

Listing 2: Código escrito em Python p/ o método da bisseção.

A seguir, a saída do código:

Iteracao	x1	x2	xm	f(x1)	f(xm)	Precisao
1	0.50000000	1.00000000	0.75000000	-0.37430886	-0.03999705	0.50000000
2	0.75000000	1.00000000	0.87500000	-0.03999705	0.18375364	0.25000000
3	0.75000000	0.87500000	0.81250000	-0.03999705	0.06589421	0.12500000
4	0.75000000	0.81250000	0.78125000	-0.03999705	0.01153717	0.06250000
5	0.75000000	0.78125000	0.76562500	-0.03999705	-0.01457145	0.03125000
6	0.76562500	0.78125000	0.77343750	-0.01457145	-0.00160397	0.01562500
7	0.77343750	0.78125000	0.77734375	-0.00160397	0.00494472	0.00781250
8	0.77343750	0.77734375	0.77539062	-0.00160397	0.00166493	0.00390625
9	0.77343750	0.77539062	0.77441406	-0.00160397	0.00002912	0.00195312
10	0.77343750	0.77441406	0.77392578	-0.00160397	-0.00078776	0.00097656
11	0.77392578	0.77441406	0.77416992	-0.00078776	-0.00037941	0.00048828
12	0.77416992	0.77441406	0.77429199	-0.00037941	-0.00017516	0.00024414
13	0.77429199	0.77441406	0.77435303	-0.00017516	-0.00007303	0.00012207
14	0.77435303	0.77441406	0.77438354	-0.00007303	-0.00002195	0.00006104
15	0.77438354	0.77441406	0.77439880	-0.00002195	0.00000358	0.00003052
16	0.77438354	0.77439880	0.77439117	-0.00002195	-0.00000919	0.00001526

Figura 2: Listagem do método de bisseção.

Pelo método da bisseção, o valor encontrado para a raiz é 0.77439117 e ela é única. Como podemos ver na Figura 1, $x^{3/4}$ é uma função crescente e $\cos(x^2)$ só pode variar entre $[1, -1]$. Logo, $x = 0.77439117$ é o único ponto onde as funções se intersectam.

2

Repita o item **1** com método de Newton-Raphson.

Solução

Pelo método de Newtonn-Raphson, definindo $x_0 = 1.0$ como ponto de partida $\epsilon = 10^{-3}$ como tolerância e o valor absoluto da função aplicada ao ponto x_n sendo maior que ϵ como parada do loop, temos o seguinte código:

```
import numpy as np

def newton_raphson(f, df, x0, e=10 ** -3):

    iter_count = 0
    print("-"*76)
    print(f"|{'n':^10}|{'xn':^15}|{'f(xn)':^15}|{'df(xn)':^15}|{'Precisao':^15}|"
    )
    print("-"*76)

    x = x0
    while abs(f(x)) > e: # Sendo abs o valor absoluto
        f_x = f(x)
        df_x = df(x)

        if df_x == 0:
            raise ZeroDivisionError("A derivada e zero. O metodo de Newton-
            Raphson nao pode continuar.")

        # Impressao dos valores atuais
        print(f"|{iter_count + 1:^10}|{x:^15.8f}|{f_x:^15.8f}|{df_x:^15.8f}
        |{abs(f_x):^15.8f}|")

        x = x - f_x / df_x
        iter_count += 1

    # Impressao final dos valores quando a precisao desejada e alcancada
    print(
        f"|{iter_count + 1:^10}|{x:^15.8f}|{f(x):^15.8f}|{df(x):^15.8f}|
        {abs(f(x)):^15.8f}|")
    print("-"*76)

    return x

# Definindo a funcao x^(3/4) - cos(x^2) e sua derivada
def minha_funcao(x):
    return x ** (3 / 4) - np.cos(x ** 2)

# Definindo derivada da funcao
def derivada_funcao(x):
    return (3 / 4) * x ** (-1 / 4) + 2 * x * np.sin(x ** 2)

# Valor inicial
x0 = 1

# Calculando a raiz
raiz = newton_raphson(minha_funcao, derivada_funcao, x0)
print(f"\nA raiz aproximada e: {raiz:.8f}")
```

Listing 3: Código escrito em Python p/ o método de Newton-Raphson.

E abaixo, a saída do código:

n	xn	f(xn)	df(xn)	Precisao
1	1.00000000	0.45969769	2.43294197	0.45969769
2	0.81105275	0.06331194	1.78203980	0.06331194
3	0.77552496	0.00189015	1.67684241	0.00189015
4	0.77439775	0.00000181	1.67362507	0.00000181

Figura 3: Listagem de iterações do método de Newton-Raphson.

Pelo método de Newton-Raphson, o valor encontrado para a raiz é 0.77439775 e ela é única, como podemos ver na Figura 2.

3

Vamos calcular a distância de ligação da molécula diatômica de NaBr a partir do potencial de interação dos íons Na^+ e Br^-. Assumindo que o potencial de interação é $V(r)$ quando os dois íons estão separados pela distância r, a distância de ligação r_{eq} é a de equilíbrio quando o potencial $V(r)$ é mínimo. Pode-se modelar o potencial entre os íons Na^+ e Br^- como

$$V(r) = -\frac{e^2}{4\pi\epsilon_0 r} + V_0 \exp\left(-\frac{r}{r_0}\right), \tag{3.1}$$

onde e é a carga do elétron, ϵ_0 é a permissividade do vácuo, e V_0 e r_0 são parâmetros da ação efetiva. O primeiro termo vem da atração Coulombiana de longo alcance entre os dois íons e o segundo termo é resultado da repulsão eletrônica de curto alcance do sistema. Vamos usar $V_0 = 1.38 \times 10^3$ eV e $r_0 = 0.328$ Å, que são os parâmetros cristalinos [?]. No equilíbrio, a força entre os dois íons,

$$F(r) = -\frac{dV(r)}{dr} = -\frac{e^2}{4\pi\epsilon_0 r^2} + \frac{V_0}{r_0} \exp\left(-\frac{r}{r_0}\right), \tag{3.2}$$

é zero.

a) Faça os gráficos de $V(r) \times r$ e $F(r) \times r$.

b) Use o método de secantes e encontre o ponto de equilíbrio $r(= r_{eq})$ em Å, que é a solução de $F(r) = 0$ na região onde $V(r)$ é mínimo. Em unidades convenientes, $\frac{e^2}{4\pi\epsilon_0} = 14.4$ eV Å.

Solução

a) Fazendo as substituições convenientes, temos abaixo código para o plot dos gráficos:

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

# Gerar dados para o grafico
x1 = np.linspace(0,10,400)  # Criar um intervalo que inclui a raiz
y1 =  -(14.4 / x1) + (1.38e3) * np.exp(-(x1 / 0.328)) # Funcao V(r)

x2 = np.linspace(0,10,400)
y2= -(14.4 / x2**2) + (1.38e3 / 0.328) * np.exp(-(x2 / 0.328)) # Funcao F(r)

# Plotar a funcao
plt.figure(figsize=(10, 6))
plt.plot(x1, y1, label='$V(r) = - e^2/4\pi\epsilon_0r +V_0 exp(-r/r_0)$',
color='blue')
plt.plot(x2, y2, label='$F(r) = -e^2/4\pi\epsilon_0r^2 + (V_0/r_0)exp(-r/r_0)$'
, color='red')
plt.axhline(0, color='black', linewidth=0.5)  # Linha horizontal no y = 0

# Adicionar titulo e rotulos
plt.title('Grafico da Funcao $V(r) x r e F(r) x r$')
plt.xlabel('r (  )')
plt.ylabel('V(r) e F(r)')
plt.legend()
plt.grid(True)

plt.ylim(-6, 5)
plt.xlim(0,8)

plt.show()
```

Listing 4: Código escrito em Python p/ o gráfico das funções.

Abaixo está o gráfico, contendo as funções $V(r) \times r$ e $F(r) \times r$:

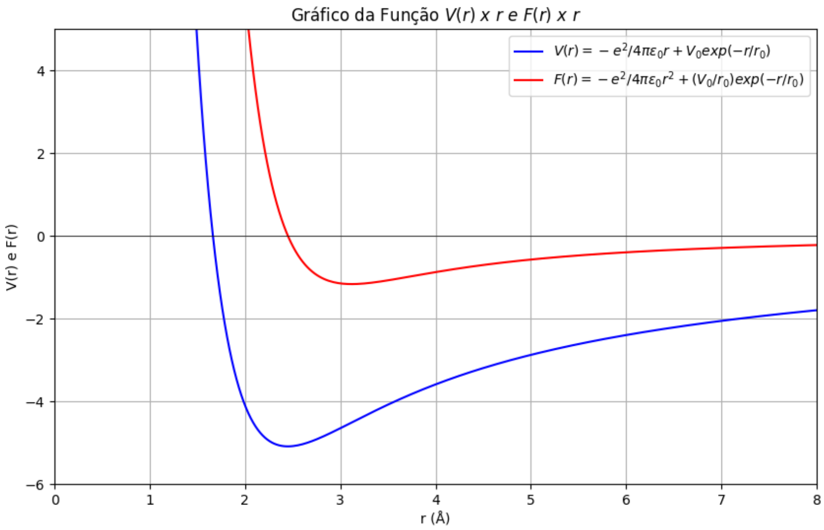

Figura 4: $V(r) \times r$ e $F(r) \times r$

b) O método consiste em uma variação do método de Newton-Raphson, com a diferença na aproximação utilizada para a derivada da função $f(x_n)$, dada por

$$f'(x_n) = x_n - \frac{f(x_n)(x_n - x_{n-1})}{f(x_n) - f(x_{n-1})}. \tag{3.3}$$

Abaixo está o código utilizado para a solução de $F(r) = 0$, com o parâmetro de tolerância $\epsilon = 10^{-5}$ e os valores iniciais $x_0 = 1$ e $x_1 = 2$.

```
import numpy as np

def secante(f, x0, x1, e=10 ** -5):
    iter_count = 0
    print("-"*76)
    print(f"|{'n':^10}|{'xn':^15}|{'f(xn)':^15}|{'Precisao':^15}|")
    print("-"*76)

    while abs(f(x1)) > e:
        f_x0 = f(x0)
        f_x1 = f(x1)

        if f_x1 == f_x0:
            raise ZeroDivisionError("A diferenca f(x_n) - f(x_{n-1}) e zero. O
metodo da secante nao pode continuar.")

        # Impressao dos valores atuais
        print(f"|{iter_count + 1:^10}|{x1:^15.8f}|{f_x1:^15.8f}|{abs(f_x1):^15.8f
}|")

        # Atualizacao do ponto x2
        x2 = x1 - f_x1 * (x1 - x0) / (f_x1 - f_x0)

        # Atualizando os pontos
        x0, x1 = x1, x2
```

```
        iter_count += 1

    # Impressao final dos valores quando a precisao desejada e alcancada
    print(f"|{iter_count + 1:^10}|{x1:^15.8f}|{f(x1):^15.8f}|{abs(f(x1)):^15.8f}|
")
    print("-"*76)

    return x1

# Definindo a funcao F(r)
def minha_funcao(x):
    return -(14.4 / x**2) + (1.38e3 / 0.328) * np.exp(-(x / 0.328))

# Valores iniciais
x0 = 1
x1 = 2

# Calculando a raiz
raiz = secante(minha_funcao, x0, x1)

print(f"\nA raiz aproximada   : {raiz:.8f}")
```

Listing 5: Código escrito em Python p/ o método de secantes.

Abaixo, está a listagem de cada iteração tal que $F(r) = 0$.

n	xn	f(xn)	Precisao
1	1.50000000	37.04126158	37.04126158
2	1.54508880	31.82992391	31.82992391
3	1.82048318	12.00662408	12.00662408
4	1.98728472	6.18719279	6.18719279
5	2.16462735	2.65327506	2.65327506
6	2.29777668	1.08846357	1.08846357
7	2.39039371	0.35700034	0.35700034
8	2.43559668	0.07927613	0.07927613
9	2.44849983	0.00809562	0.00809562
10	2.44996735	0.00021348	0.00021348
11	2.45000710	0.00000060	0.00000060

Figura 5: Listagem de iterações do método das secantes.

Portanto, foi encontrada a raiz 2.45000710, ponto tal que $r_e q = 2.45000710$ Å e $V(r_{eq})$ é mínimo.