FIS0610 - Física Computacional I

Uma Breve Introdução ao Python[1]

Paulo Douglas Santos de Lima
paulo.douglas.lima@fisica.ufrn.br

Sumário

[1] Python 2.7.12

1 Problema 1

2 Problema 2

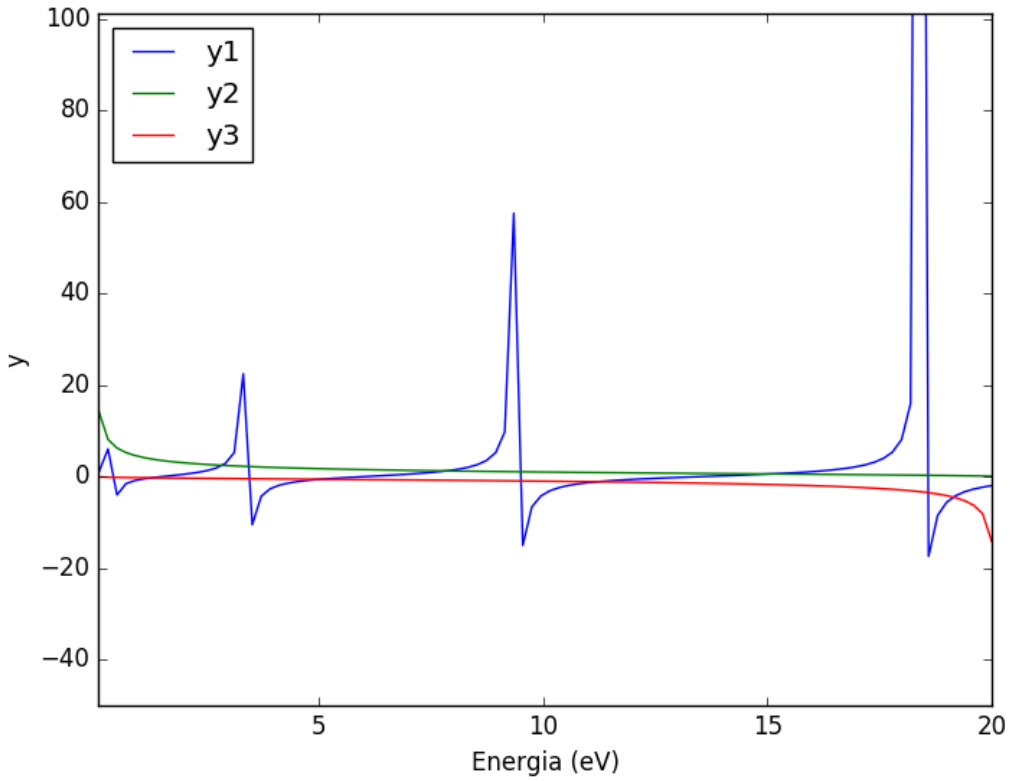
3 Problema 3

O gráfico com as quantidades y_1, y_2 e y_3 está apresentado na figura 1 com seu código abaixo. Note que para evitar divisões por 0, os valores de E foram iniciados em 0.1.

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

# Declaracao das constantes
m = 9.1094e-31
w = 1.0000e-9
h = 6.582119514e-16
V = 20.1
E = np.linspace(0.1, 20, 100)

# Impressao dos graficos
plt.plot(E, np.tan(np.sqrt((w*w*m*E)/(2*h*h*1.6e-19))))
plt.plot(E, np.sqrt((V - E)/E))
plt.plot(E, -np.sqrt(E/(V - E)))
plt.xlabel('Energia (eV)')
plt.ylabel('y')
plt.legend(["y1", "y2", "y3"], loc="upper left")
plt.savefig('ex3a.png')
plt.show()
```



Analisando cuidadosamente o gráfico e evitando os pontos de singularidades da função tangente, obtemos uma estimativa dos primeiros seis níveis de energia:

Table 1: Valores estimados para os seis primeiros níveis de energia.

Níveis Pares (eV)	Níveis Ímpares (eV)
0,318	1,255
2,850	5,125
7,875	11,45

Na luz desses valores, o algoritmo abaixo foi desenvolvido para o cálculo das raízes usando o método da bisecção com intervalos adequados. O valor máximo usado de erro para a raiz foi de 0.001 e o número de interações foi calculado com base na inequação:

$$n_{max} \geq \frac{\log(b-a) - \log(2\epsilon)}{\log(2)}\,, \tag{1}$$

que garante a convergência do método após $n_{max} + 1$ iterações.

```
from math import sqrt, tan, log10

# Declaracao das constantes
m = 9.1094e-31        # Massa do eletron
w = 1.0000e-9
h = 6.582119514e-16   # Constant de Plank/2pi
V = 20.1
#Tolerancia
epsilon = 0.001
# Funcoes
```

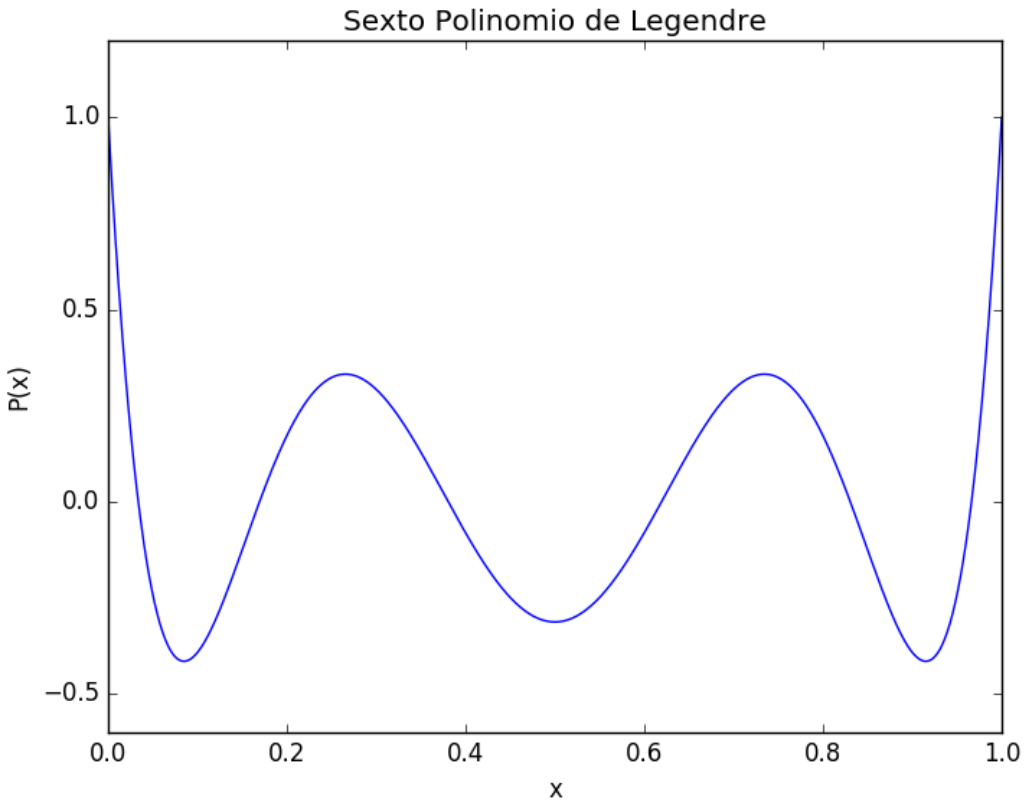
```
def y_1(x):
        # Niveis pares
        return tan(sqrt((w*w*m*x/(2*h*h*1.6e-19)))) - sqrt((V - x)/x)
def y_2(x):
        # Niveis impares
        return tan(sqrt((w*w*m*x/(2*h*h*1.6e-19)))) + sqrt(x/(V - x))
# Procedimento da biseccao
def bisec(f, a, b, e):
        nmax = int(round((log10(b-a) - log10(2*e))/log10(2))+1)
        fa = f(a)
        fb = f(b)
        if fa*fb > 0:
                print "Nao ha raizes nesse intervalo!"
        error = b - a
        for i in range (nmax):
                error = error/2
                c = a + error
                fc = f(c)
                if abs(error)<e:
                        return c
                if fa*fc < 0:
                        b = c
                        fb = fc
                else:
                        a = c
# Exemplo para o primeiro nivel de energia
print bisec(y_1, 0.3, 0.4, epsilon)
```

4 Problema 4

O algoritmo para a construção do polinômio de Legendre e o gráfico estão abaixo:

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

x = np.linspace(0, 1, 1000)
p_x = 924*x**6 - 2772*x**5 + 3150*x**4 - 1680*x**3 + 420*x**2 - 42*x + 1
print np.diff(p_x)
# Impressao dos graficos
plt.plot(x, p_x)
plt.title('Sexto Polinomio de Legendre')
plt.xlabel('x')
plt.ylabel('P(x)')
plt.savefig('ex4.png')
plt.show()
```


Sexto Polinomio de Legendre
1.0
0.5
0.0
−0.5
P(x)
0.0
0.2
0.4
0.6
0.8
1.0
x

Para determinar as raízes utilizando o método de Newton, foi utilizado a biblioteca *scipy.optmize*[2], que já implementa internamente este método. O ponto inicial usado foi o estimado pela análise gráfica, enquanto que a tolerância para os valores das raízes de 10^{-10}, que levou ao número de 800 iterações.

```
import numpy as np
import scipy.optimize

def P(x):
        return 924*x**6 - 2772*x**5 + 3150*x**4 - 1680*x**3 + 420*x**2 - 42*x + 1
def dP(x):
        return 4620*x**5 - 13860*x**4 + 12600*x**3 - 5040*x**2 + 840*x - 42

x_0 = float(input())
print scipy.optimize.newton(P, x_0, dP, tol=1e-10, maxiter=800)
```

Table 2: Comparação entre o valor das raízes estimadas e numéricas.

Raíz Estimada	Raiz Calculada Numericamente
0.035	0.0337652428984
0.168	0.169395306767
0.382	0.38069040712
0.618	0.380690407087
0.832	0.83060469899
0.967	0.0337652428984

5 Problema 5

[2]https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.newton.html