Relatório F 625 - Lista 2

Grupo 4

Pedro V. Pinho N. - 185886
Ian D. Resende Barbosa - 174822

18 de outubro de 2019

Exercício 5.17: A função Γ

5.17 a)

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt

###################################
#--Programa que gráfica a função--#
#--f(x) = x**(a-1)*np.exp(-x)-----#
#--para diferentes valores de a---#
###################################

def f(a,x):
    return x**(a-1)*np.exp(-x)

# a = array com dif. valores de a
# c = array com cores para cada curva

a = [2, 3, 4]
```

```
c = ['-r', '-k', '-b']
x = np.linspace(0,5,100)
y = [f(a,x) for a in a]

for i in range(0,3):
    plt.plot(x, y[i], c[i], label = 'a = %d' %a[i])

plt.legend(loc = 'best')
plt.xlabel('x', size = 14)
plt.ylabel('f(x)', size = 14)
plt.grid()
plt.show()
```

Na figura 1 temos as curvas geradas pelo programa acima. Podemos ver que todas elas possuem um comportamento parecido: crescendo até um valor máximo e decaindo até zero.

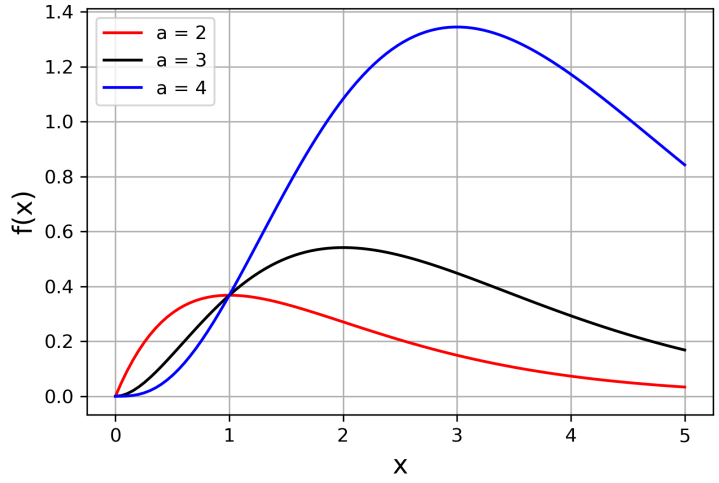

Figura 1: Curvas do integrando da função Γ para diferentes valores de a.

5.17 b)

O integrando da função Γ é $\gamma(x) = x^{a-1} \cdot e^{-x}$. Para encontrar o ponto de máximo basta encontrar o valor x_0 onde $\frac{d\gamma}{dx} = 0$. Usando a regra do produto vemos que:

$$\frac{d\gamma}{dx} = e^{-x}x^{a-1}((a-1)x^{-1} - 1)$$

Igualando a equação com zero temos que:

$$(a-1)x_0^{-1} = 1$$

$$\implies x_0 = (a-1)$$

5.17 c)

Dada a transformação $z = x/(x + c)$, encontrar o valor de c que mapeie o ponto $x = a - 1$ em $z = 1/2$ é uma questão de substituição:

$$\begin{aligned} \frac{1}{2} &= \frac{a-1}{c+a-1} \\ \implies c &= a + 3 \end{aligned} \tag{1}$$

Portanto o mapeamento $x \to z$ é $z = x/(x + a + 3)$.

5.17 d)

Para evitar overflow/underflow no integrando de Γ, é interessante fazer a seguinte manipulação:

$$x^{a-1} = e^{(a-1)ln(x)}$$

O integrando portanto toma a forma:

$$e^{(a-1)ln(x)-x}$$

Realizar uma operação com tal integrando é, computacionalmente, mais estável uma vez que o argumento da exponencial toma uma forma linear a medida que x cresce. Essa correção não nos livra de todos os problemas pois ainda há um certo empecilho para valores muito próximos de zero, quando o integrando assume valores grandes demais. Entretanto, com a manipulação feita, menos erros computacionais são gerados, devolvendo para o usuário um valor mais fidedigno. Assim, o novo integrando da função Γ é:

$$\gamma(x) = e^{(a-1)ln(x)-x} \tag{2}$$

5.17 e)

```
# -*- coding: utf-8 -*-
import numpy as np
from gaussxw import gaussxwab

###############################################
#--Função que calcula a função Gamma para----#
#--um deteminado valor de 'a'. Para evitar---#
#--erro de precisão, a troca de variavel-----#
#--z = x/(x + a - 1) foi feita, alterando o--#
#--intervalo de integração de [0, \inf] para-#
#--[0, 1].-----------------------------------#
###############################################

def gamma(a):
    c = a - 1
```

```
    # Integrando da função Gamma na nova variável z
    def f(c,z):
        return c* np.exp(c*(np.log(c*z/(1-z)))) * np.exp(-(c*z)/(1-z))/(1 - z)**2

    # Integração feita por método de Quadratura Gaussiana
    N = 50
    zp, wp = gaussxwab(N, 0, 1)

    I = 0.0
    for k in range(len(zp)):
        I+= wp[k]*f(c,zp[k])

    # O valor de Gamma(a) é entregue arredondado
    return round(I,4)

# Testando a função para calcular um valor conhecido e alguns fatoriais
a = [3/2, 3, 6, 10]
print([gamma(a_i) for a_i in a])
```

5.17 f)

O programa apresentado acima é capaz de calcular o valor da função Γ com boa precisão. A tabela 1 apresenta o valor de a e $\Gamma(a)$ calculado pelo programa.

a	$\Gamma(a)$
$3/2$	0.886226
3	2
6	120
10	362880

Tabela 1: Valor da função Γ para alguns valores de a.

Exercício 5.19: Grade de Difração

5.19 a)

A função $q(u)$ pode ser entendido como o fator de forma da estrutura que esta difratando a onda. Para uma grade de difração, $q(u)$ está relacionada diretamente com a posição das fendas, assim, visto que temos $q(u) = sin^2(\alpha u)$ podemos interpretar os picos da função como o centro das fendas. A separação entre os máximos de $q(u)$ é igual a separação entre as fendas:

$$\frac{dq(u)}{du} = \alpha sin(2\alpha u)$$

$$\implies u_{max} = \frac{2\pi m}{2\alpha}$$

$$\implies \Delta u_{max} = \frac{\pi}{\alpha}$$

Portanto, sendo d a separação entre as fendas, temos que $\alpha = \pi/d$. É importante notar que a separação entre as fendas é igual a separação entre **MÁXIMOS** da função $q(u)$ e não, simplesmente, a separação dos pontos onde $q'(u) = 0$.

5.19 b)

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import numpy as np

########################################
#--Esse programa retorna a função de----#
#--transmissão (q(u)) de uma grade de---#
#--difração com distância entre fendas--#
#--igual a d.---------------------------#
########################################

d = 20e-6     #[m]

x = np.linspace(-2*d, 2*d, 500)
q = np.sin(np.pi*x/d)**2

plt.plot(x*1e6, q, "-k")
plt.xlabel('Posição na grade [um]', size = 14)
plt.ylabel('q(u)', size = 14)
plt.grid(alpha = 0.5)
plt.show()
```

A figura 2 apresenta a função $q(u)$ ao longo da grade de difração. Como discutido anteriormente, os máximos da função correspondem ao centro das fendas da grade.

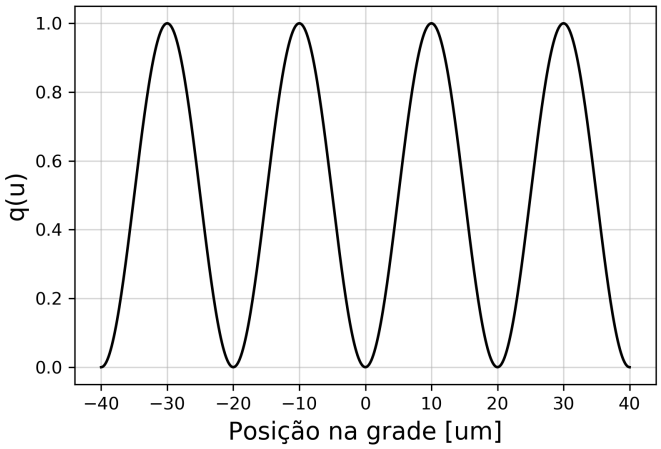

Figura 2: Função de transmissão da grade de difração

5.19 c)

```
# -*- coding: utf-8 -*-
import matplotlib.pyplot as plt
import numpy as np
from gaussxw import gaussxwab

########################################
#--Este programa, dado os parâmetros---#
#--abaixo, entrega uma visualização----#
#--unidimensional e bidimensional------#
#--do padrão de interferência de uma---#
#--grade de difração.------------------#
########################################

# N:      Número de fendas
# lambd:  Comprimento de onda
# f:      Distância focal
# L:      Tamanho da tela
# d:      Espaçamento entre fendas
# w:      Tamanho total da grade de difração

N     = 10
lambd = 500e-9      #[m]
f     = 1           #[m]
L     = 10e-2       #[m]
d     = 20e-6       #[m]
w     = N*d         #[m]

# Função de transmissão da grade de difração
def q(u,d):
    return np.sin(u*(np.pi/d))**2
```

```
################################################
#--O padrão de interferência é a Transformada---#
#--de Fourier da raiz da função de transmissão--#
#-----------------------------------------------#
#--Para otimizar a função, os parâmetros para---#
#--a integração por quadratura gaussiana foram--#
#--calculados fora da função e entregues como --#
#--argumentos.---------------------------------#
################################################

def I_patt(x,lambd,f,d,w,u,p):
    i = complex(0,1)
    s = 0.0
    for k in range(n):
        s += p[k]*np.sqrt(q(u[k],d))*np.exp(i*2*np.pi*x*u[k]/(lambd*f))
    return abs(s)**2

n   = 500
u,p = gaussxwab(n, -w/2, w/2)

x = np.linspace(-L/2, L/2, 500)
I = I_patt(x,lambd,f,d,w,u,p)/max(I_patt(x,lambd,f,d,w,u,p))

plt.figure()
plt.plot(x*1e2, I, '-k')
plt.xlabel('Posição no Anteparo [cm]', size = 14)
plt.ylabel('Intensidade Relativa', size = 14)
plt.show()

plt.figure()
plt.imshow((I,), aspect = 4*len(x)/20, cmap='gray', vmax = 0.15, interpolation = 'gaussi
plt.show()
```

A figura 3 apresenta o padrão de interferência da grade de difração para os parâmetros utilizados. Da teoria ondulatória sabemos que para uma grade de difração com N fendas é de se esperar que entre máximos consecutivos existam $N-2$ picos. Podemos ver que é o caso com o padrão de interferência apresentado: é possível ver claramente 8 picos secundários, uma vez que $N = 10$.

A separação entre máximos no padrão de interferência segue a relação $\Delta x_{max} = \lambda z/d$, onde z é a distância entre a grade e o anteparo. Para os parâmetros utilizados é de se esperar que $\Delta x_{max} = 2.5cm$. Observando a figura 3 podemos ver que esse valor é confirmado.

5.19 d)

O código apresentado acima é capaz de gerar uma representação do que um observador veria em um anteparo se realizasse tal experimento na vida real. Em comparação com a figura 3, a figura 4 não traz novas informações mas é igualmente interessante.

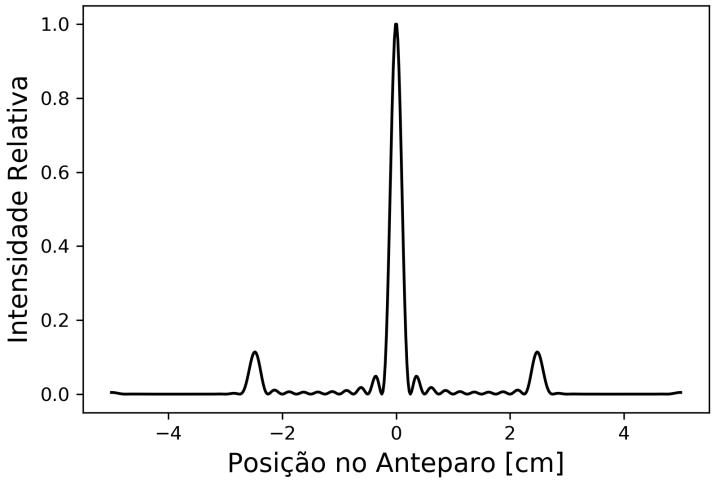

Figura 3: Padrão de interferência no anteparo

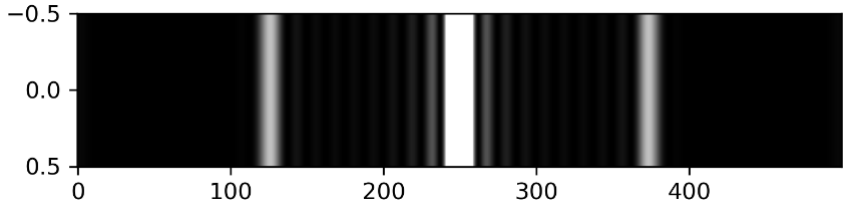

Figura 4: Padrão de interferência visto por um observador

5.19 e)

As figuras 5 e 6 apresentam, respectivamente, os padrões de interferência para as seguintes funções de transmissão:

$$q(u) = (sin(\alpha u) sin(\alpha u/2))^2 \tag{3}$$

$$q(u) = \begin{cases} 1, & 20\mu m \leq x \leq 40\mu m \\ 1, & -35\mu m \leq x \leq -25\mu m \\ 0, & c.c \end{cases} \tag{4}$$

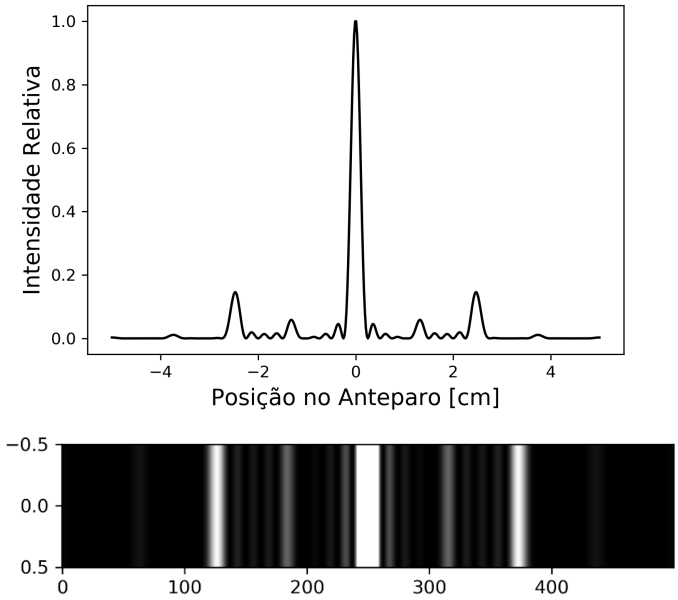

Figura 5: Interferência para $q(u) = (sin(\alpha u)sin(\alpha u/2))^2$

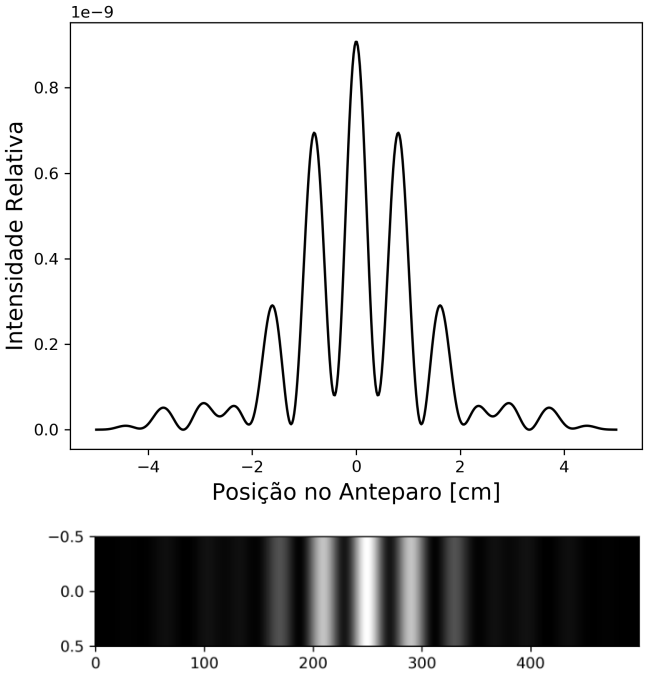

Figura 6: Interferência para 2 fendas separadas por $60\mu m$, uma com $20\mu m$ e a outra com $10\mu m$.

Exercício 5.21: $\vec{E}$ de uma distribuição de carga

5.21 a)

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt

################################################
#--Este programa demonstra o potencial ao redor-#
#--de duas cargas puntiformes (q1 e q2)---------#
#--separadas por uma distancia D.---------------#
################################################

def Pot(D,q1,q2):
    # qi:     Valor das cargas em Coulomb
    # D:      Distância entre cargas
    # d:      Espaçamento entre pontos
    # l:      Tamanho do plano
    # eps_0:  Constante elétrica do vácuo
    # N:      Escala do Grid [NxN]

    d     = 1e-2        #[m]
    L     = 1           #[m]
    eps_0 = 8.8542e-12 #[C^2 N^-1 m^-2]
    N     = int(L/d)

    # Um grid é formado para calcular o potencial em cada ponto
    x = y = np.linspace(-L/2, L/2, N)
    X,Y   = np.meshgrid(x,y, sparse = True)

    # R1 e R2 são as distâncias do ponto até as cargas
    R1  = ((X + D/2)**2 + Y**2)**0.5
    R2  = ((X - D/2)**2 + Y**2)**0.5

    pot = (q1/R1 + q2/R2)/(4*np.pi*eps_0)

    plt.figure()
    P_color = plt.imshow(pot, extent = [-L/2, L/2, -L/2, L/2])
    plt.xlabel('Posição X [m]', size = 14)
    plt.ylabel('Posição Y [m]', size = 14)
    plt.colorbar(P_color)
    plt.show()

# As cargas são espaçadas por 10cm
D = 10e-2   #[m]

#Carga do elétron
```

```
q_e = 1.6e-12  #[C]

# Função recebe a informação da distância das
# cargas, além do seus valores em Coulomb
Pot(D, q_e, q_e)
Pot(D, q_e, -q_e)
```

As figuras 7 e 8 apresentam o potencial gerado por duas cargas puntiformes de sinais iguais e opostos, respectivamente. Os gráficos de densidade dos potenciais seguem o esperado pela teoria Eletromagnética. Podemos ver que o potencial decaí muito rápido e temos, em grande parte do plano, um potencial nulo.

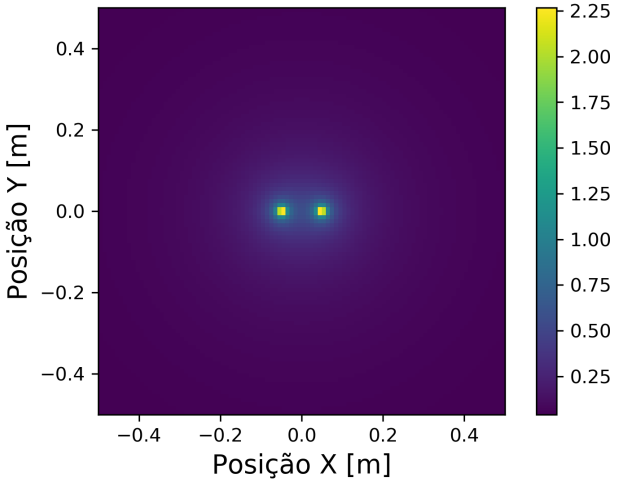

Figura 7: Potencial de duas cargas puntiformes de cargas $q_1 = 1\mu C$ e $q_2 = -1\mu C$ separadas por $D = 10cm$.

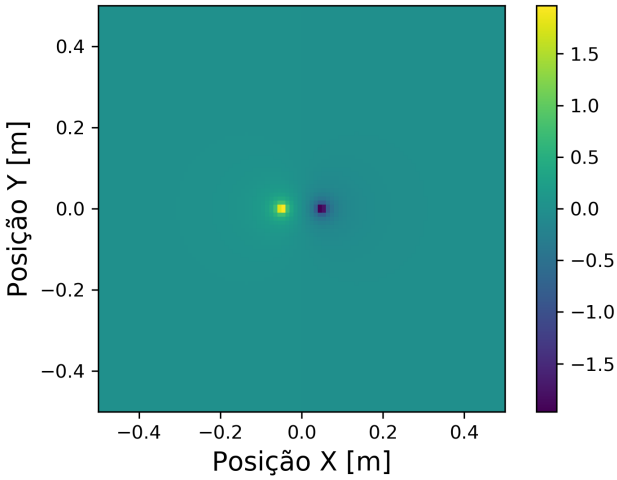

Figura 8: Potencial de duas cargas puntiformes de cargas $q_1 = 1\mu C$ e $q_2 = 1\mu C$ separadas por $D = 10cm$.

5.21 b)

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt

#####################################################
#--Este programa calcula o potencial entre duas----#
#--cargas puntiformes e calcula o gradiente--------#
#--para obter o campo elétrico. As cargas são------#
#--separadas por uma distância d e possuem cargas--#
#--q1 e q2. A direção do campo é dita pela direção-#
#--dos vetores, enquanto que a intensidade é-------#
#--traduzida para cores.--------------------------#
#####################################################

def E_field(D,q1,q2):
    # qi:    Valor das cargas em Coulomb
    # d:     Espaçamento entre pontos
    # D:     Distância entre cargas
    # N:     Escala do grid [N X N]
    # eps_0: Constante elétrica do vácuo
    # l:     Tamanho do plano

    d     = 1e-2       #[m]
    L     = 1          #[m]
    eps_0 = 8.8542e-12 #[C^2 N^-1 m^-2]
    N     = int(L/d)

    # Um grid é formado para calcular o potencial em cada ponto
    x = y = np.linspace(-L/2, L/2, N)
    X,Y   = np.meshgrid(x,y, sparse = True)

    # R1 e R2 são as distâncias do ponto até as cargas
    R1  = ((X + D/2)**2 + Y**2)**0.5
    R2  = ((X - D/2)**2 + Y**2)**0.5

    # O potencial é calculado a menos de constantes 1/(4[U+03F5]_0)
    pot    = (q1/R1 + q2/R2)/(4*np.pi*eps_0)
    Ey, Ex = np.gradient(-pot)

    plt.figure()
    E_stream = plt.streamplot(x, y, Ex, Ey, linewidth=0.5, color = 'w',
                              density=1.5, arrowstyle='->', arrowsize=1.3)
    E_strgth = plt.imshow(np.hypot(Ex,Ey), extent = [-L/2, L/2, -L/2, L/2])
    plt.xlabel('Posição X [m]', size = 14)
    plt.ylabel('Posição Y [m]', size = 14)
    plt.colorbar(E_strgth)
    plt.show()
```

```
# As cargas são espaçadas por 10cm
D = 10e-2   #[m]

#Carga do elétron
q_e = 1.6e-12  #[C]

E_field(D, q_e, q_e)
E_field(D, q_e, -q_e)
```

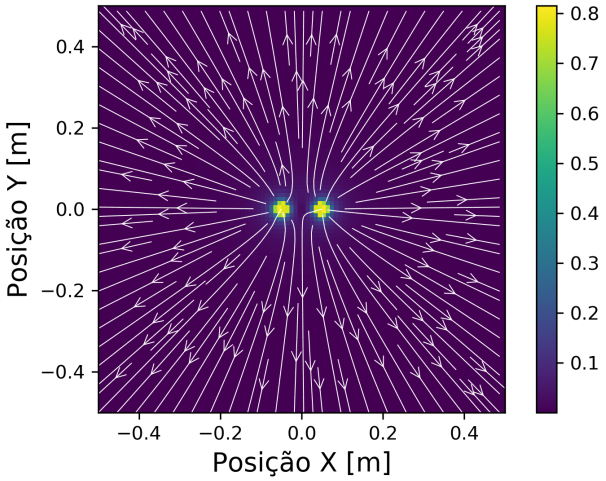

Figura 9: Campo elétrico de duas cargas puntiformes de cargas $q_1 = 1\mu C$ e $q_2 = 1\mu C$ separadas por $D = 10cm$.

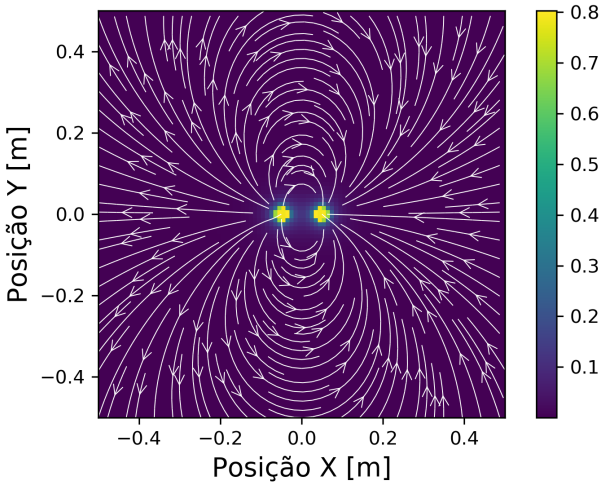

Figura 10: Campo elétrico de duas cargas puntiformes de cargas $q_1 = 1\mu C$ e $q_2 = -1\mu C$ separadas por $D = 10cm$.

Os resultados apresentados nas figuras 9 e 10 são bastante familiares para aqueles que realizaram um curso básico de Eletromagnetismo. Podemos ver que quando as cargas são de mesmo sinal

existe uma repulsão das linhas de campo. Enquanto que para cargas de sinais opostos as linhas de campo *nascem* na carga positiva e *morrem* na carga negativa. A respeito do modulo, seguindo a legenda nas figuras, podemos ver que a medida que nos afastamos das cargas, o modulo do campo decai.

5.21 c)

O potencial elétrico de uma distribuição de carga é dado por:

$$V(\vec{r}) = \frac{1}{4\pi\epsilon_0}\int_{\Sigma}\frac{dq}{|\vec{r}|} \tag{5}$$

Onde Σ é a distribuição de carga no espaço. Como estamos trabalhando com uma placa plana temos que $dq = \sigma(x', y')dA = \sigma(x', y')dx'dy'$. O vetor $\vec{r}$ liga o elemento de carga na posição (x', y') e o ponto de interesse (x, y). Portanto pode ser escrito como $(x' - x, y' - y)$. Para uma placa com dimensões L x L podemos então escrever:

$$V(x, y) = \frac{1}{4\pi\epsilon_0}\int_{-L/2}^{L/2}\int_{-L/2}^{L/2}\frac{\sigma(x', y')dx'dy'}{\sqrt{(x' - x)^2 + (y' - y)^2)}} \tag{6}$$

Abaixo temos a implementação de um programa que calcula o potencial e o campo elétrico de uma placa carregada com a seguinte distribuição de carga, em Cm^{-2}:

$$\sigma(x, y) = q_0 sin\left(\frac{2\pi x}{L}\right) sin\left(\frac{2\pi y}{L}\right)$$

As figuras 11 e 12 apresentam o calculo do potencial e do campo elétrico de tal distribuição de cargas. Como pode ser observado temos o que parece um quadrupolo com simetria ao longo de $y = x$ e $y = -x$. Isso decorre da forma da distribuição de carga $\sigma(x, y)$. Na região quadrada centrada na origem de dimensão L x L temos quatro picos nos pontos (L/4, L/4), (-L/4, L/4), (L/4, -L/4) e (-L/4, -L/4) com sinais de carga alternados.

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
from gaussxw import gaussxwab

######################################################################
#--Este programa calcula o potencial e o campo gerado por uma---------#
#--placa carregada de tamanho LxL, onde L = 10cm. A placa possui------#
#--uma distribuição de carga definida dada pela função----------------#
#--"charge_dist". O potencial e o campo são todos calculados em-------#
#--um plano de 1m^2 com pontos espaçados de 1cm. O potencial é--------#
#--calculado por meio de uma Quadratura Gaussiana Dupla e o campo-----#
#--é o gradiente.-----------------------------------------------------#
######################################################################
```

```
def charge_dist(x, y, L):
    q_0 = 100    #[Cm^-2]
    L  = 10e-2  #[m]
    return q_0*np.sin(2*np.pi*x/L)*np.sin(2*np.pi*y/L)

def Potencial(xp, wp, x, y, L):
    eps_0 = 8.8541e-12  #[C^2 N^-1 m^-2]
    N     = len(xp)

    pot = 0
    for i in range(N):
        for j in range(N):
            R    = ((x - xp[i])**2 + (y - xp[j])**2)**0.5
            pot += wp[i]*wp[j]*charge_dist(xp[i],xp[j], L)/R

    return pot/(4*np.pi*eps_0)

# d: Espaçamento dos pontos
# D: Tamanho do plano
# L: Tamanho da placa
# N: Número de pontos na quadratura

d = 1e-2  #[m]
D = 1     #[m]
L = 10e-2 #[m]
N = 100

# Pontos e pesos usados na quadratura Gaussiana
xp,wp = gaussxwab(N, -L/2, L/2)

# Criação do Meshgrid para calculo do potencial
x     = np.linspace(-D/2, D/2, int(D/d))
y     = x.copy()
X, Y  = np.meshgrid(x,y)

# Potencial e Campo Elétrico
Pot     = Potencial(xp,wp,X,Y,L)
Ey,Ex   = np.gradient(-Pot)

plt.figure()
P_color = plt.imshow(Pot, extent = [-D/2, D/2, -D/2, D/2])
plt.colorbar(P_color)
plt.xlabel('Posição X [m]', size = 14)
plt.ylabel('Posição Y [m]', size = 14)
plt.show()


plt.figure()
```

```
E_stream = plt.streamplot(x, y, Ex, Ey, color='w', linewidth=0.5,
                          density=1.5, arrowstyle='->', arrowsize=1.3)
E_strght = plt.imshow(np.hypot(Ex,Ey), extent = [-D/2, D/2, -D/2, D/2])
plt.xlabel('Posição X [m]', size = 14)
plt.ylabel('Posição Y [m]', size = 14)
plt.colorbar(E_strght)
plt.show()
```

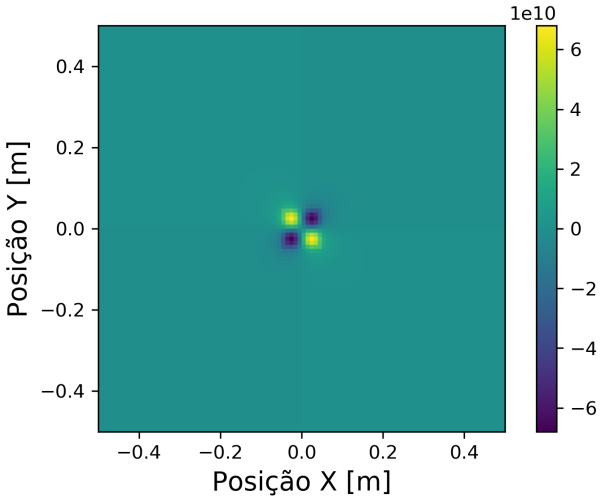

Figura 11: Potencial elétrico de uma placa carregada com $\sigma(x, y)$ de dimensões 10cm x 10cm.

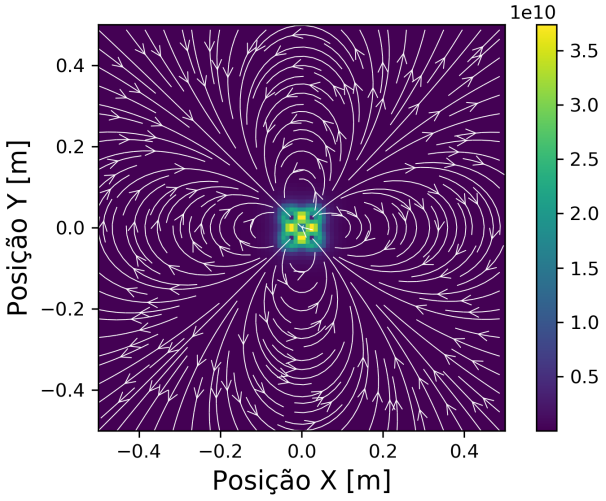

Figura 12: Campo elétrico de uma placa carregada com $\sigma(x, y)$ de dimensões 10cm x 10cm.

Exercício 5.22: Diferenciação por Integração

```
# -*- coding: utf-8 -*-
import numpy as np
from math import factorial as fact
```

```
################################################
#--Programa que calcula a m-ésima derivada de----#
#--uma função f(z) em z=0 utilizando a Integral--#
#--de Cauchy no plano complexo. A integral é-----#
#--aproximada pelo método de trapézios onde o----#
#--caminho da integral é o circulo unitário------#
#--centrado na origem.--------------------------#
################################################

# Função a ser derivada
def f(z):
    return np.exp(2*z)

def deriv_f(m):
    N = 10000
    i = complex(0,1)

    Sum = 0
    for k in range(N):
        Sum += f(np.exp(i*2*np.pi*k/N))*np.exp(-i*2*np.pi*k*m/N)

    return fact(m)*Sum/N

# As 21 primeiras derivadas de f(z = 0) são calculadas
m = np.array(range(1,21))
deriv = np.array([deriv_f(m) for m in m])

# Os elementos de 'deriv' são essencialmente complexo, vamos imprimir
# seus modulos arredondados em 2 casas decimais
print([round(abs(deriv[i]), 2) for i in range(len(deriv))])
```

O módulo das derivadas de $f(z) = e^{2z}$ em z = 0 são potências de 2, isto é, $\frac{d^m f}{dz^m} = 2^m$. Os resultados entregues pelo programa, para as vinte primeiras derivadas de f são apresentados na tabela 2:

m	$\frac{d^m f}{dz^m}\big\|_{z=0}$
1	2
2	4
3	8
4	16
5	32
6	64
7	128
8	256
9	512
10	1024
11	2048
12	4096
13	8192
14	16384
15	32768
16	65536
17	131072.05
18	262145.79
19	524330.67
20	1049119.23

Tabela 2: Resultados das primeiras m-ésimas derivadas de $f(z)$.

Podemos ver que esse método é bastante preciso, uma vez que apenas na 17° derivada é que temos divergências do valor real.

Exercício 5.23: Processamento de Imagens e STM

5.23 a)

O gradiente de uma função escalar $f(x, y)$ é simplesmente $\nabla f(x, y) = (\partial f/\partial x, \partial f/\partial y)$. Para o problema dado temos um objeto semelhante a uma matriz, com dimensões [Nx, Ny]. Portanto, no final, teremos duas matrizes de dimensões iguais, correspondente a ($\partial f/\partial x$ e $\partial f/\partial y$).

Realizar a derivada numérica não é um problema até que estejamos nas bordas. Ao chegar nas bordas devemos passar a utilizar *forward difference* ou *backward difference* dependendo de qual borda estamos lidamos para que não tenhamos problemas de *"out of bounds"*. Para pontos, interiores, fora das bordas podemos utilizar *central difference*.

Abaixo temos implementado um programa que calcula o gradiente de um array de dimensões [Nx, Ny]. Dado um arquivo f e especificando o espaçamento entre os pontos, devolve-se as derivadas parciais de f.

```
# -*- coding: utf-8 -*-
import numpy as np

#####################################################################
#--Essa função calcula o gradiente de um campo escalar------------#
```

```
#--f(x,y), discreto com pontos equidistantes. As entradas são-----#
#--um grid com informações dos valores de f(x,y) para cada--------#
#--ponto e o espaçamento entre eles. O código faz as derivadas----#
#--direcionais levando em conta a posição do ponto no array.------#
#--Pontos nas bordas são calculados por "forward differences"-----#
#--ou "backwards differences" enquanto que os centrais são--------#
#--calculados por "central differences".-------------------------#
#####################################################################

def Grad(file, h):
    f = np.asanyarray(file)
    Nx, Ny = np.shape(f)[0], np.shape(f)[1]

    dx = np.zeros([Nx,Ny], float)
    dy = dx.copy()

    # Cálculo de df/dy
    for i in range(Nx):
        for j in range(Ny):
            # 'Forward diff.' para pontos na borda superior
            if i == 0:
                dy[i,j] = (f[i+1,j] - f[i,j])/h
            # 'Backward diff.' para pontos na borda inferior
            if i == Nx - 1:
                dy[i,j] = (f[i,j] - f[i-1,j])/h
            # 'Central diff.' para pontos interiores
            else:
                dy[i,j] = (f[i+1,j] - f[i-1,j])/(2*h)

    # Cálculo de df/dy
    for i in range(Nx):
        for j in range(Ny):
            # 'Forward diff.' para pontos na borda esquerda
            if j == 0:
                dx[i,j] = (f[i,j+1] - f[i,j])/h
            # 'Backward diff.' para pontos na borda direita
            if j == Ny - 1:
                dx[i,j] = (f[i,j] - f[i,j-1])/h
            # 'Central diff.' para pontos interiores
            else:
                dx[i,j] = (f[i,j+1] - f[i,j-1])/(2*h)

    return dx,dy

H     = np.loadtxt('altitude.txt')
h     = 3000
dx,dy = Grad(H, h)
```

```
print(dx, '\n', dy)
```

5.23 b)

Se *f(x,y)* é um campo escalar que parametriza uma superfície qualquer e temos uma incidência de luz com um ângulo ϕ, então, para um ponto (x,y), a intensidade de iluminação é dada por:

$$I(x,y) = \frac{cos\phi(\partial f/\partial x) + sin\phi(\partial f/\partial y)}{\sqrt{(\partial f/\partial x)^2 + (\partial f/\partial y)^2 + 1}} \tag{7}$$

O arquivo *"altitude.txt"* contém a altitude de pontos na superfície da terra espaçados por $h = 30km$. A partir da equação 7 e do programa de gradiente é possível implementar um código capaz de gerar uma imagem 3D do mapa-mundi. Abaixo apresentamos tal código:

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt

#########################################################################
#--Programa que calcula a intensidade de irradiação solar sobre um------#
#--terreno dado ângulo de incidência. Em especial, com esse programa----#
#--podemos simular uma imagem 3D de um mapa mundi iluminado com---------#
#--ângulo de 45 graus. O arquivo de texto utilizado contém a altura-----#
#--para cada ponto do mapa mundi, separados por uma distância de 30km---#
#########################################################################

def Grad(file, h):
    f = np.asanyarray(file)
    Nx, Ny = np.shape(f)[0], np.shape(f)[1]

    dx = np.zeros([Nx,Ny], float)
    dy = dx.copy()

    for i in range(Nx):
        for j in range(Ny):
            if i == 0:
                dy[i,j] = (f[i+1,j] - f[i,j])/h
            if i == Nx - 1:
                dy[i,j] = (f[i,j] - f[i-1,j])/h
            else:
                dy[i,j] = (f[i+1,j] - f[i-1,j])/(2*h)

    for i in range(Nx):
        for j in range(Ny):
            if j == 0:
                dx[i,j] = (f[i,j+1] - f[i,j])/h
            if j == Ny - 1:
```

```
                dx[i,j] = (f[i,j] - f[i,j-1])/h
            else:
                dx[i,j] = (f[i,j+1] - f[i,j-1])/(2*h)

    return dx,dy

H     = np.loadtxt('altitude.txt')
h     = 3000
theta = (45/180)*np.pi
dx,dy = Grad(H, h)

I = (np.cos(theta)*dx + np.sin(theta)*dy)/(dx**2 + dy**2 + 1)**0.5

plt.figure()
plt.imshow(I, cmap = 'hot')
plt.show()
```

A figura 13 apresenta o produto final deste código. Podemos ver uma simulação de sombras criadas por regiões mais altas, o que cria um efeito 3D.

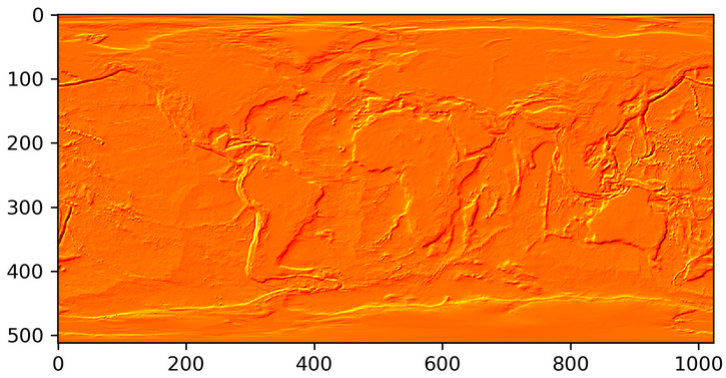

Figura 13: Perspectiva 3D do mapa-mundi iluminado com luz por um ângulo de 45°

.

5.23 c)

Outro arquivo é *"stm.txt"*, este apresenta os dados da varredura por *STM* da superfície (111) de Silício. O código utilizado é o mesmo, fazendo pequenas alterações, como a mudança do espaçamento entre pontos e a leitura do novo arquivo. É possível ver na figura 14 uma perspectiva 3D da varredura.

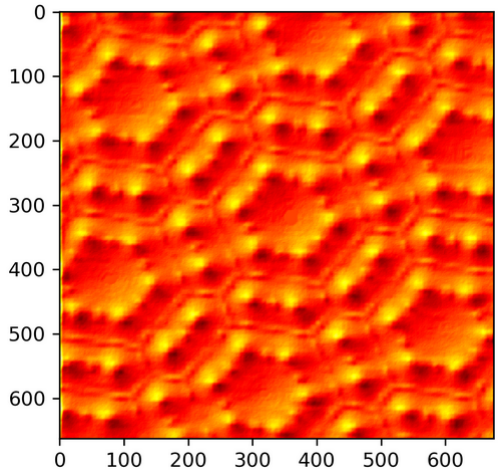

Figura 14: Perspectiva 3D da superfície (111) do Silício iluminado com luz por um ângulo de 45°

.

Exercício 6.9: Poço Quântico Assimétrico

6.9 a)

Assumindo que nas fronteiras, $x = 0$ e $x = L$, temos $V(x) = 0$ e que a função de onda deve se anular em tais pontos, podemos assumir que a solução da Eq. de Schrödinger é uma série de senos de Fourier:

$$\Psi = \sum_{n=1}^{\infty} \psi_n sin\left(\frac{n\pi x}{L}\right) \tag{8}$$

Utilizando a equação 8 na Eq. de Schrödinger independente do tempo, temos que:

$$-\frac{\hbar^2}{2m}\sum_{n=1}^{\infty}\psi_n\frac{d^2}{dx^2}\left(sin\left(\frac{n\pi x}{L}\right)\right) + V(x)\sum_{n=1}^{\infty}\psi_n sin\left(\frac{n\pi x}{L}\right) = E\sum_{n=1}^{\infty}\psi_n sin\left(\frac{n\pi x}{L}\right) \tag{9}$$

Podemos reescrever a equação 9 como:

$$\sum_{n=1}^{\infty}\psi_n\left[-\frac{\hbar^2}{2m}\frac{d^2}{dx^2} + V(x)\right] sin\left(\frac{n\pi x}{L}\right) = E\sum_{n=1}^{\infty}\psi_n sin\left(\frac{n\pi x}{L}\right) \tag{10}$$

O termo entre colchetes é o operador Hamiltoniano ($\hat{H}$). Utilizando a ortogonalidade entre senos, podemos finalmente escrever:

$$\sum_{n=1}^{\infty}\psi_n\int_0^L sin\left(\frac{m\pi x}{L}\right)\hat{H}sin\left(\frac{n\pi x}{L}\right)dx = \frac{1}{2}LE\psi_m \tag{11}$$

Dado que a função de onda Ψ é escrita na base das funções seno $u_n = sin(n\pi x/L)$, a integral da equação 11 - a menos de um fator $2/L$ - é o termo $\langle u_m | \hat{H} | u_n \rangle$, ou seja, o elemento H_{mn} da matriz do Hamiltoniano:

$$\sum_{n=1}^{\infty} \psi_n H_{mn} = E\psi_m$$

$$\implies \boldsymbol{H\psi} = E\boldsymbol{\psi}$$

6.9 b)

Para um poço de potencial tal que $V(x) = ax/L$ temos:

$$H_{mn} = \frac{2}{L}\int_0^L sin\left(\frac{m\pi x}{L}\right)\left[-\frac{\hbar^2}{2m}\frac{d^2}{dx^2}sin\left(\frac{n\pi x}{L}\right) + \frac{a}{L}xsin\left(\frac{n\pi x}{L}\right)\right]dx$$

$$\implies H_{mn} = \frac{\hbar^2 n^2\pi^2}{mL^3}\int_0^L sin\left(\frac{m\pi x}{L}\right)sin\left(\frac{n\pi x}{L}\right)dx + \frac{2a}{L^2}\int_0^L xsin\left(\frac{m\pi x}{L}\right)sin\left(\frac{n\pi x}{L}\right)dx$$

Utilizando a relação de ortogonalidade da segunda integral podemos escrever:

$$H_{mn} = \begin{cases} \frac{a}{2} + \frac{(\hbar n\pi L)^2}{2M}, & m = n \\ \frac{-8a}{\pi^2}\frac{mn}{(m^2-n^2)^2}, & m \neq n, \textit{um par e outro impar} \\ 0, & m \neq n, \textit{ambos par ou impar} \end{cases} \tag{12}$$

Independentemente dos valores de m e n, já que varrem todos os números naturais, H_{mn} assume apenas valores reais. Devido ao fator $(m^2 - n^2)^2$, os elementos H_{mn} e H_{nm} são iguais, portanto esta matriz é real e simétrica. Assim, neste problema temos uma simetria de reversão no tempo $(t \to -t)$.

Abaixo apresenta-se um exemplo de código capaz de calcular os elementos H_{mn} deste Hamiltoniano.

```
# -*- coding: utf-8 -*-
import numpy as np

a = 10          #[eV]
L = 5e-10       #[m]

# Esta função calcula os elementos H_mn da matrix do Hamiltoniano
def H_mn(m,n,a,L):
    h_bar = 1.0546e-34 #[Js]
    M     = 9.1094e-31 #[kg]
    q     = 1.6022e-19 #[C]
```

```
    # A carga do elétron é utilizada para converter eV em J
    if m == n:
        Hmn = (a*q)/2 + (h_bar*n*np.pi/L)**2/(2*M)
        return Hmn/q

    if m!=n and (m+n)%2 != 0:
        Hmn = (-8*a*q*m*n)/(np.pi*(m**2 - n**2))**2
        return Hmn/q

    else:
        return 0

m,n = list(map(int, input("Digite m e n separados por espaço: ").split()))
print(H_mn(m,n,a,L))
```

6.9 c) d) e)

A matriz abaixo foi gerada a partir do código apresentado abaixo. Aqui apresenta-se valores truncados para facilitar a visualização. Podemos ver que como dito anteriormente, a matriz **H** é simétrica e real. Os elementos tem unidade de eV.

$$H = \begin{bmatrix} 6.504 & -1.801 & 0 & -0.144 & 0 & -0.04 & 0 & -0.016 & 0. & -0.008 \\ -1.801 & 11.017 & -1.945 & 0 & -0.184 & 0 & -0.056 & 0 & -0.025 & 0 \\ 0 & -1.945 & 18.538 & -1.985 & 0 & -0.2 & 0 & -0.064 & 0 & -0.029 \\ -0.144 & 0 & -1.985 & 29.067 & -2.001 & 0 & -0.208 & 0 & -0.069 & 0 \\ 0 & -0.184 & 0 & -2.001 & 42.604 & -2.01 & 0 & -0.213 & 0 & -0.072 \\ -0.04 & 0 & -0.2 & 0 & -2.01 & 59.15 & -2.014 & 0 & -0.216 & 0 \\ 0 & -0.056 & 0 & -0.208 & 0 & -2.014 & 78.705 & -2.017 & 0 & -0.218 \\ -0.016 & 0 & -0.064 & 0 & -0.213 & 0 & -2.017 & 101.267 & -2.019 & 0 \\ 0 & -0.025 & 0 & -0.069 & 0 & -0.216 & 0 & -2.019 & 126.838 & -2.021 \\ -0.008 & 0 & -0.029 & 0 & -0.072 & 0 & -0.218 & 0 & -2.021 & 155.418 \end{bmatrix}$$

Utilizando o pacote de álgebra linear do *numpy* foi calculado os autovalores desta matriz. A tabela 3 apresenta os 10 primeiros autovalores do sistema, calculados utilizando a aproximação 10x10 de **H** e 100x100.

Podemos ver que para as 10 primeiras energias o cálculo considerando uma matriz 10x10 é suficiente. A maior diferença ocorreu na energia mais alta, em torno de 0.13 eV. A depender do problema físico, do experimento conduzido e de outros fatores, utilizar mais termos pode ser de interesse, porém, para o proposito deste relatório, podemos calcular os autovalores[1] com uma matriz 10x10.

Sabendo os autovetores associados às 3 primeiras energias podemos graficar as funções de onda dos estados. A figura 15 apresenta as densidades de probabilidade ($|\Psi|^2$) para o estado fundamental (n = 1) e os dois primeiros estados excitados (n = 2 e n = 3).

[1]Para os autovetores é necessário utilizar a maior matriz possível.

N = 10	**N = 100**
5.83645	5.83645
11.18131	11.18131
18.66339	18.66339
29.14509	29.14508
42.65647	42.65646
59.18726	59.18721
78.73209	78.73204
101.28905	101.28842
126.8559	126.85507
155.5609	155.43128

Tabela 3: Valores dos autovalores para diferentes dimensões NxN da matriz **H**

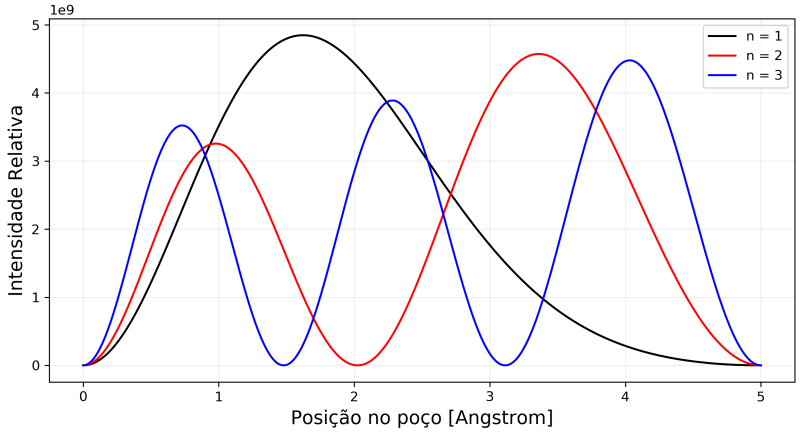

Figura 15: Densidade de probabilidade para os 3 primeiros estados do sistema

Algumas conclusões podem ser tiradas analisando a figura 15. Podemos ver uma assimetria com respeito ao centro do poço de potencial, isso se dá ao formato do potencial, a forma das funções de onda refletem diretamente isso. Mesmo assim, podemos fazer algumas comparações com os resultados do poço de potencial convencional apresentado em cursos de Mecânica Quântica. Podemos ver que o modo $n = 1$ possui apenas 1 máximo, $n = 2$ possui 2 máximos e $n = 3$ possui 3 máximos. Em um poço com paredes de potencial infinito o Hamiltoniano é compacto, ou seja, seu espectro é contável. A energia de tal sistema cresce com n^2 (portanto é crescente monotônica), como as soluções são oscilatórias e devem obedecer condições de contorno que cancelam as funções de onda nas bordas, a única opção é que a função de onda desenvolva um novo máximo para cada valor crescente de n.

As funções de onda são normalizadas para que a probabilidade de se encontrar uma partícula em $x \in [0, L]$ seja 100%. O código utilizado para gerar tais resultados pode ser encontrado abaixo:

```
# -*- coding: utf-8 -*-
import numpy as np
import matplotlib.pyplot as plt
import scipy.integrate as integ
```

```
############################################################
#--Este programa calcula os autovalores e autovetores------#
#--de um Hamiltoniano para uma particula presa em um-------#
#--poço de potencial V(x) = ax/L com barreiras infinitas.--#
#--Uma solução de superposição de ondas planas é-----------#
#--assumido para o problemas e o Hamiltoniano toma a forma-#
#--de uma matriz. A solução do problema é normalizada.-----#
############################################################

# a: Parâmetro de energia para o poço
# L: Largura do poço

a = 10          #[eV]
L = 5e-10       #[m]

# Esta função calcula os elementos H_mn da matrix do Hamiltoniano
def H_mn(m,n,a,L):
    h_bar = 1.0546e-34 #[Js]
    M     = 9.1094e-31 #[kg]
    q     = 1.6022e-19 #[C]

    if m == n:
        Hmn = (a*q)/2 + (h_bar*n*np.pi/L)**2/(2*M)
        return Hmn/q

    if m!=n and (m+n)%2 != 0:
        Hmn = (-8*a*q*m*n)/(np.pi*(m**2 - n**2))**2
        return Hmn/q

    else:
        return 0

# N: Dimensão do Hamiltoniano [N x N]
N = 10
H = [[H_mn(m,n,a,L) for m in range(1, N+1)] for n in range(1, N+1)]

# w: Autovalores
# v: autovetores
w,v = np.linalg.eigh(H)

# Imprimindo a matriz H e seus 10 primeiros autovalores;
# Os valores são impressos com 5 casas decimais em notação científica;
np.set_printoptions(suppress = True)
np.set_printoptions(precision = 5)
print("A matrix do Hamiltoniano: ", '\n', np.matrix(H),'\n')
print("Os 10 primeiros autovalores são: ", '\n', w[:10])

# Autovetor associado para cada nivel energetico:
```

```
# 0: Estado Fundamental
# 1 e 2: Dois primeiros estados excitados
phi_1 = v[:,0]
phi_2 = v[:,1]
phi_3 = v[:,2]

x = np.linspace(0, L, 500)

# Função que entrega a densidade de probabilidade do estado;
# Ela recebe um array com valores de x e outro com valores de phi_n;
# A função de onda é dada por Psi = sum( phi_n*sin(nx/L) );
# A função de onda é normalizada;
def Psi_sq_norm(x, phi, L):
    Psi = np.zeros(len(x), dtype = float)
    for n in range(len(phi)):
        Psi_n = phi[n]*np.sin((n+1)*np.pi*x/L)
        Psi  += Psi_n

    norm     = integ.trapz(Psi**2, x)**0.5
    Psi_norm = Psi/norm
    return Psi_norm**2

Psi_1 = Psi_sq_norm(x, phi_1, L)
Psi_2 = Psi_sq_norm(x, phi_2, L)
Psi_3 = Psi_sq_norm(x, phi_3, L)

# Gráfico das densidades de probabilidade
plt.figure()
plt.plot(x*1e10, Psi_1, '-k', label = 'n = 1')
plt.plot(x*1e10, Psi_2, '-r', label = 'n = 2')
plt.plot(x*1e10, Psi_3, '-b', label = 'n = 3')

# Embelezamento das figuras
plt.xlabel('Posição no poço [Angstrom]', size = 14)
plt.ylabel('Intensidade Relativa', size = 14)
plt.legend(loc = 'best')
plt.grid(alpha = 0.2)
plt.show()

# Verificação da norma de cada estado
norm    = [0,0,0]
norm[0] = integ.trapz(Psi_1, x)
norm[1] = integ.trapz(Psi_2, x)
norm[2] = integ.trapz(Psi_3, x)

for i in range(3):
    print('A norma do estado n = %d é igual a: %.4f' %(i+1, norm[i]))
```

Exercício 6.12:

6.12 a)

O biomecanismo de glicólise pode ser modelado pelas equações:

$$\frac{dx}{dt} = -x + ay + x^2y$$
$$\frac{dy}{dt} = b - ay - x^2y$$

O ponto estacionário da glicólise ocorre quando ambas derivadas se anulam. Ou seja, no ponto que é solução do sistema de equações:

$$\begin{cases} -x + ay + x^2y = 0 \\ b - ay - x^2y = 0 \end{cases} \tag{13}$$

Somando ambas equações encontramos que $x = b$. Substituindo em qualquer uma das equações encontramos, por final, que $y = b/(a + b^2)$.

6.12 b)

A partir da equação 13 podemos reescrever as equações a fim de utilizar o método de relaxação para resolver o sistema. Primeiro, começando com a equação superior podemos afastar o termo linear de x do termo quadrático. Na segunda equação podemos isolar y. Realizando essas operações chegamos em:

$$\begin{cases} x = y(a + x^2) \\ y = \frac{b}{a+x^2} \end{cases} \tag{14}$$

Abaixo apresenta-se um código responsável por resolver o sistema de equação 14 pelo método de relaxação. O código é incapaz de convergir em uma solução. 100 iterações do método de relaxação foram realizadas e o resultado oscila entre valores que não tem relação com o resultado correto. A tabela 4 apresenta os dez últimos resultados das cem iterações. Podemos ver que os valores oscilam e esse comportamento é independente do número de iterações.

$\#N$	a	b
91	1.4369	0.6526
92	-0.6948	1.3488
93	0.6977	1.3451
94	0.6903	1.3545
94	0.7090	1.3309
94	0.6617	1.3908
94	0.7817	1.2413
94	0.4827	1.6219
94	1.2439	0.7850
94	-0.4298	1.6881

Tabela 4: Não-convergência do método de relaxação.

```
# -*- coding: utf-8 -*-
# p0 : Chute inicial
# a,b: Parametros do modelo
p0 = [1,1]
a  = 1
b  = 2

# Inicializando soluções
x,y  = p0[0], p0[1]

# Número de iterações
N    = 100
for i in range(N):
    # Atualizando soluções
    x = y*(a - x**2)
    y = b/(a + x**2)

    print(x,y)
```

6.12 c)

Um método para solucionar esse problema é trabalhar com a função inversa de cada equação. Assim, partindo da equação 13 e isolando y na equação superior e x na equação inferior temos:

$$\begin{cases} y = \frac{x}{a+x^2} \\ x = \left(\frac{b-ay}{y}\right)^{1/2} \end{cases} \tag{15}$$

O código abaixo trabalha com esse novo sistema de equações para buscar uma solução. Funcionando melhor que o anterior, ele é capaz de encontrar uma solução.

Dado valores $a = 1$ e $b = 2$, calculou-se $x = 2.0$ e $y = 0.4$. Esses valores concordam com a solução analítica.

```
# -*- coding: utf-8 -*-
# p0 : Chute inicial
# a,b: Parametros do modelo
p0 = [1,1]
a  = 1
b  = 2

# Inicializando soluções
x,y  = p0[0], p0[1]

# Número de iterações
N    = 100
for i in range(N):
    # Atualizando soluções
```

```
    x = ((b - a*y)/y)**0.5
    y = x/(a + x**2)

    print(x,y)
```

Exercício 6.18: A temperatura de uma lâmpada

6.18 a) b)

A eficiência de emissão de um corpo negro pra uma determinada temperatura T é a porcentagem em que a emissão ocorre no espectro visível ($\lambda_1 = 390$nm até $\lambda_1 = 750$nm). A equação 16 explicita a equação capaz de calcular tal eficiência dada uma temperatura T:

$$\eta = \frac{15}{\pi^4} \int_{hc/\lambda_2 k_B T}^{hc/\lambda_1 k_B T} \frac{x^3}{e^x - 1} dx \tag{16}$$

Abaixo apresenta-se um código capaz de resolver tal equação de forma numérica por meio de integração Gaussiana utilizando-se 100 pontos. Além disso, o código conta com uma implementação de *Golden Ratio Search* para encontrar a temperatura onde a eficiência é máxima:

```
# -*- coding: cp1252 -*-
import numpy as np
import matplotlib.pyplot as plt
from gaussxw import gaussxwab

def f(x):
    return x**3/(np.exp(x) - 1)

############################################################
#--Esse c[U+FFFD]o calcula a efici[U+FFFD]ia (percentual da---------#
#--intensidade emitida na faixa do v[U+FFFD]vel) para uma-------#
#--l[U+FFFD]ada (aproximada para um corpo negro). A efici[U+FFFD]ia--#
#--[U+FFFD]alculada para uma temperatura T do corpo.------------#
############################################################

def Eff(T):
    # k: Constante de Boltzmann
    # c: Velocidade da luz
    # h: Constante de Planck
    # lamb_i: Limites da faixa de radia[U+FFFD] v[U+FFFD]ivel

    k      = 1.38064852e-23  # [m^2 kg s^-2 k^-1]
    c      = 299792458       # [m/s]
    h      = 6.62607015e-34  # [m^2 kg/s]
    lamb_1 = 390e-9          # [m]
    lamb_2 = 750e-9          # [m]
```

```
    a = (h*c)/(k*T*lamb_2)
    b = (h*c)/(k*T*lamb_1)

    # Integração Gaussiana
    N     = 100
    xp,wp = gaussxwab(N, a, b)
    I = 0
    for k in range(N):
        I += wp[k]*f(xp[k])

    return 15*I/np.pi**4

T   = np.linspace(300, 10000, 100)
eff = np.array([Eff(T) for T in T])

#######################################################################
#--Tendo o perfil da efici[U+FFFD]ia pela temperatura podemos-----------#
#--calcular a temperatura de maior efici[U+FFFD]ia pelo m[U+FFFD]do-----------#
#--de busca da raz[U+FFFD]aurea. A fun[U+FFFD] necessita do array-------------#
#--onde ser[U+FFFD]eita a busca, pontos in[U+FFFD]ais e uma acur[U+FFFD]a limite.--#
#######################################################################
def gld_srch(f, p0, acc):
    x0, x3 = p0[0], p0[1]
    phi    = (1 + np.sqrt(5))/2
    d      = (x3 - x0)/phi

    x1 = x0 + d
    x2 = x3 - d

    while (x3 - x0) > acc:
        if f(x1) > f(x2):
            x0 = x2
            x2 = x1
            x1 = x0 + (x3 - x0)/phi

        else:
            x3 = x1
            x1 = x2
            x2 = x3 - (x3 - x0)/phi

    return (x3 + x0)/2

# A temperatura de m[U+FFFD]ma efici[U+FFFD]ia [U+FFFD]alcula com precis[U+FFFD]de 1K
Max_T = gld_srch(Eff, [6e3,8e3], 1)
print("A temperatura de maior efici[U+FFFD]ia [U+FFFD]%d K" %Max_T)

plt.plot(T/1000, eff, '-k')
plt.grid(alpha = 0.5)
```

```
plt.xlabel('T[k]/1000', size = 14)
plt.ylabel('Efici[U+FFFD]ia', size = 14)
plt.axvline(x = Max_T/1000, linestyle = '--', color = 'r', ymax = 0.96)
plt.text(7, 0.35, "6928 K", fontsize = 12)
plt.show()
```

A figura 16 apresenta o perfil da eficiência em função da temperatura do corpo. Podemos ver claramente que existe um máximo de eficiência. Este máximo se encontra em $T = 6928K$.

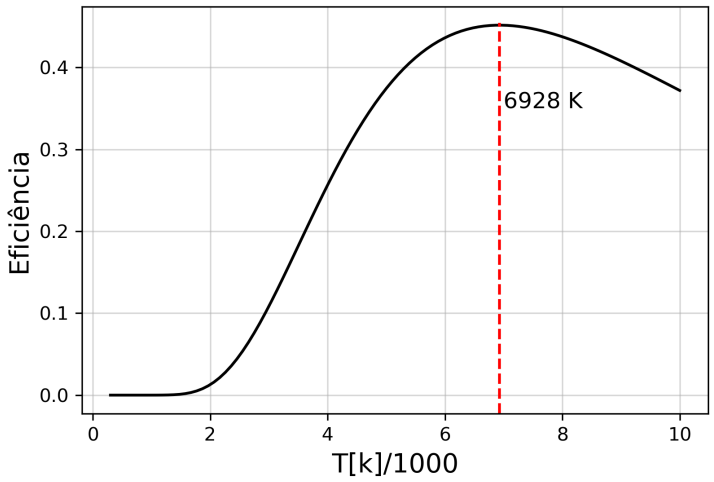

Figura 16: Eficiência de emissão para um corpo negro em função da sua temperatura. O máximo da curva é explicitado

6.18 c)

Como visto, o máximo de eficiência ocorre em $T = 6928K$. O filamento de lâmpadas convencionais é feito de Tungstênio, cuja temperatura de fusão é de 3422°, ou seja, $3695K$, logo, entra em fusão em uma temperatura muito menor que a de máxima eficiência. Portanto, além de não ser prático, é impossível ter uma lâmpada incandescente em seu máximo de eficiência.

Assumindo uma temperatura de funcionamento de $3000K$ temos que a eficiência gira em torno de 10%. Assim, pode-se notar que lâmpadas incandescentes são ótimas fontes de calor e não de luz.