# Simulação do Modelo de Ising 2D - Metropolis - Guilherme de Abreu Lima Buitrago Miranda - 2018054788

July 25, 2021

## 1 Introdução à Física Estatística Computacional

### 1.1 Simulação do Modelo de Ising 2D - Metropolis

Aluno: Guilherme de Abreu Lima Buitrago Miranda

Matrícula: 2018054788

#### 1.1.1 Imports

[1]:
```
import matplotlib.pyplot as plt
from numba import jit
import numpy as np

plt.style.use('seaborn-colorblind')
plt.ion()
```

#### 1.1.2 Funções

As funções abaixo foram extraídas dos arquivos fornecidos no enunciado do exercício. No fim, há ainda outras funções criadas por mim.

[2]:
```
@jit(nopython=True)
def estado_ini(N):
    #Gera um estadon inicial aleatório para rede
    s = np.zeros(N,dtype=np.int8)
    for i in range(N):
        s[i] = np.sign(2*np.random.random()-1)
    return s

@jit(nopython=True)
def vizinhos(L,N):
    #Define a tabela de vizinhos
    viz = np.zeros((N,4),dtype=np.int16)
    for k in range(N):
        viz[k,0]=k+1
        if (k+1) % L == 0: viz[k,0] = k+1-L
```

```
        viz[k,1] = k+L
        if k > (N-L-1): viz[k,1] = k+L-N
        viz[k,2] = k-1
        if k % L == 0: viz[k,2] = k+L-1
        viz[k,3] = k-L
        if k < L: viz[k,3] = k+N-L
    return viz

@jit(nopython=True)
def energia(s,viz, N):
    #Calcula a energia da configuração s
    ener = 0
    for i in range(N):
        h = s[viz[i,0]]+s[viz[i,1]]
        ener -= s[i]*h
    ener = int((ener+2*N)/4)
    return ener
```

[3]:
```
@jit(nopython=True)
def expos(beta):
    ex = np.zeros(5,dtype=np.float32)
    ex[0]=np.exp(8.0*beta)
    ex[1]=np.exp(4.0*beta)
    ex[2]=0.0
    ex[3]=np.exp(-4.0*beta)
    ex[4]=np.exp(-8.0*beta)
    return ex
```

[4]:
```
@jit(nopython=True)
def mcstep(beta,s,viz,ener,mag):
    N=len(s)
    ex=expos(beta)
    for i in range(N):
        h = s[viz[i,0]]+s[viz[i,1]]+s[viz[i,2]]+s[viz[i,3]] # soma dos vizinhos
        de = int(s[i]*h*0.5+2)
        if np.random.random() < ex[de]:
            ener=ener+2*s[i]*h
            mag -= 2*s[i]
            s[i]=-s[i]
    return ener,mag,s
```

[5]:
```
def execute_all(l, n, Niter, boxes = 10):
    s = estado_ini(n)
    viz = vizinhos(l, n)
    ener = energia(s, viz, n)
    x_order = np.linspace(1, 5, 17)
    x = x_order[::-1]
```

```
    med_e = []
    med_cvs = []
    med_mag_list = []
    err_cvs = []
    chi = []
    mag = np.absolute(np.sum(s))
    for x_ in x:

        ener_list = []
        mag_list = []
        # mag = 0

        for i in range(Niter):
            ener, mag, s = mcstep(1/x_, s, viz, ener, mag)
            if i > 1e5:
                ener_list.append(ener)
                mag_list.append(mag)

        m = len(ener_list)//boxes

        ener_list = np.array(ener_list)
        mag_list = np.array(mag_list)

        e_var = []
        e_var_2 = []
        mag_var = []
        mag_var_2 = []

        for i in range(boxes):
            e_var.append(1/m * np.sum(ener_list[i*m: (i+1)*(m-1)]))
            e_var_2.append(1/m * np.sum(ener_list[i*m: (i+1)*(m-1)] ** 2))

            mag_var.append(1/m * np.sum(np.absolute(mag_list[i*m:␣
↪(i+1)*(m-1)])))
            mag_var_2.append(1/m * np.sum(np.absolute(mag_list[i*m:␣
↪(i+1)*(m-1)]) ** 2))

        e_var = np.array(e_var)
        e_var_2 = np.array(e_var_2)

        med_e.append(np.mean(e_var))
        med_mag_list.append(np.mean(mag_var))

        chi.append(((1/x_)/n) * (np.mean(mag_var_2) - np.mean(mag_var) ** 2))

        cv = ((1/x_) ** 2)/n * (e_var_2 - (e_var ** 2))
```

```
        cv_var = np.mean(cv)

        med_cvs.append(cv_var)
        err_cvs.append(np.sqrt(((cv_var - cv) ** 2).sum() / (n * (n - 1))))

    return np.array(med_e[::-1]) / n, med_cvs[::-1], np.array(med_mag_list[::
↪-1]) / n, err_cvs[::-1], chi[::-1], x_order
```

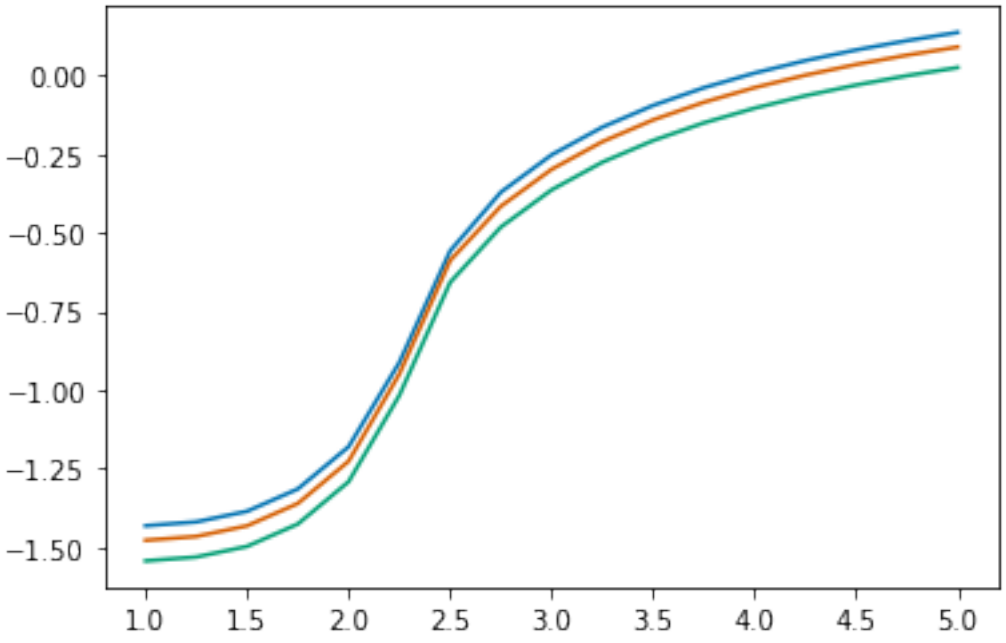
[6]:
```
def plot(y_list, x):
    for y in y_list:
        plt.plot(x, y)
```

[7]:
```
ls = [18, 24, 36]
ns = [l * l for l in ls]

plot1 = []
plot2 = []
plot3 = []
plot4 = []
plot5 = []

for l in ls:
    med_e, med_cv, med_mag_list, err_cvs, chi, x = execute_all(l, (l * l),␣
↪int(1.1e6))
    plot1.append(med_e)
    plot2.append(med_cv)
    plot3.append(med_mag_list)
    plot4.append(err_cvs)
    plot5.append(chi)
```

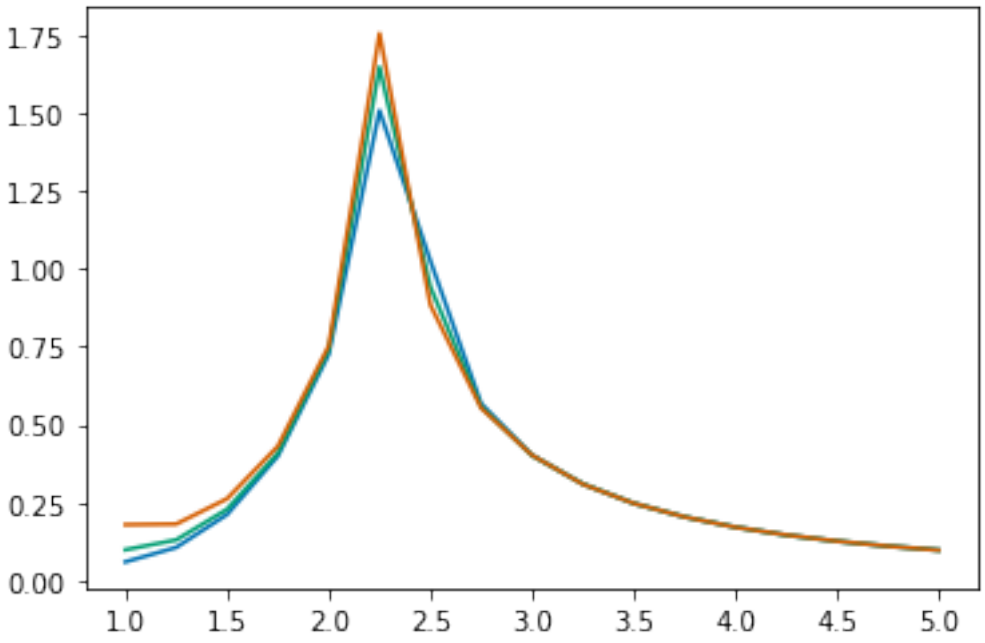
### 1.1.3 Gráficos

Abaixo, encontram-se os gráficos do exercício na seguinte ordem: Média de Energia por Spin, Calor Específico por Temperatura, Magnetização Média por Spin, Erros Estatísticos Associados e Susceptibilidade Magnética.

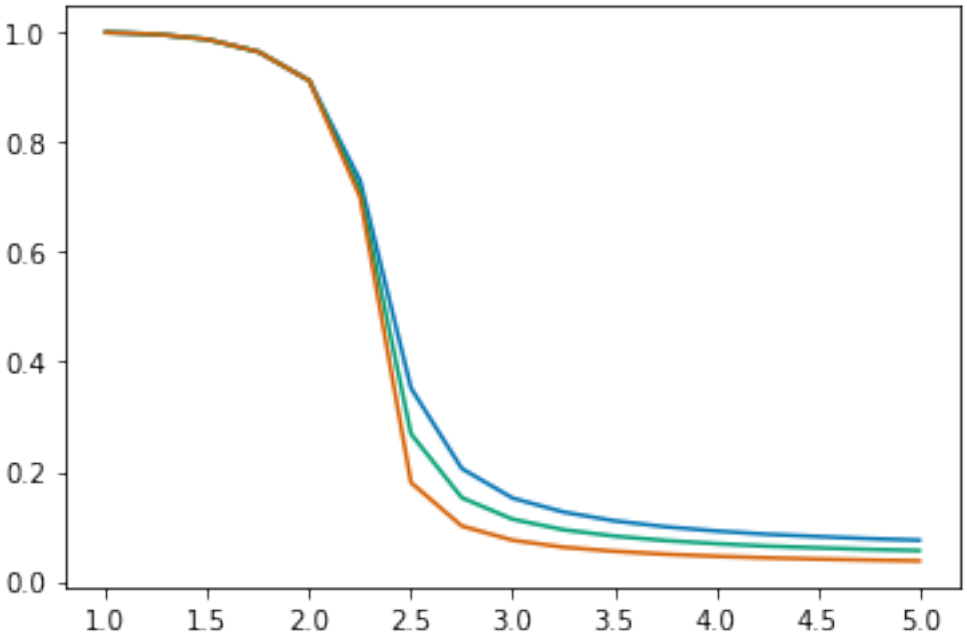
[8]:
```
plot(plot1, x)
```



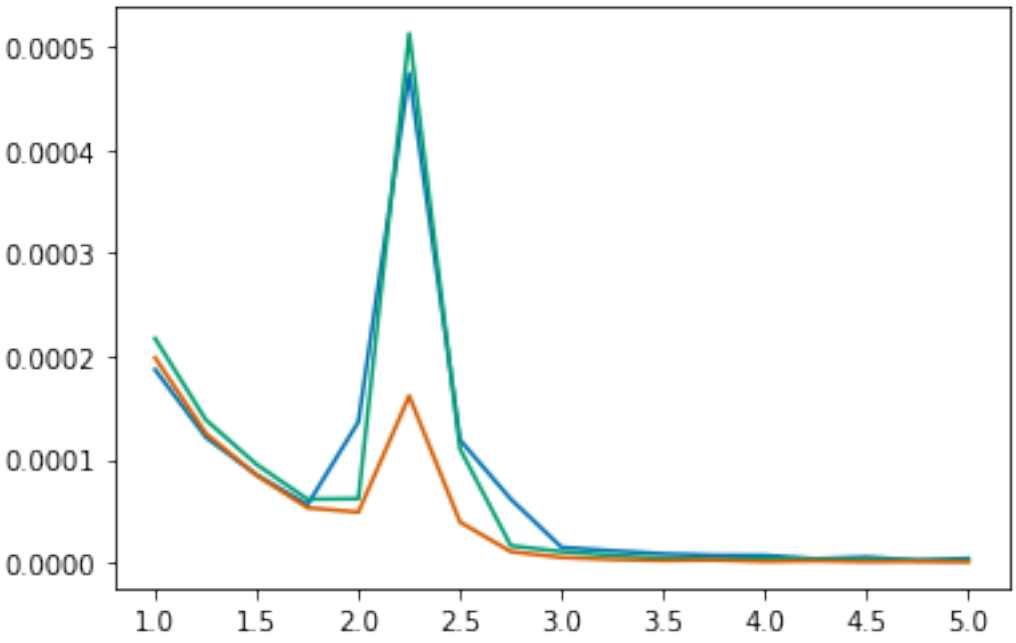

[9]:
```
plot(plot2, x)
```



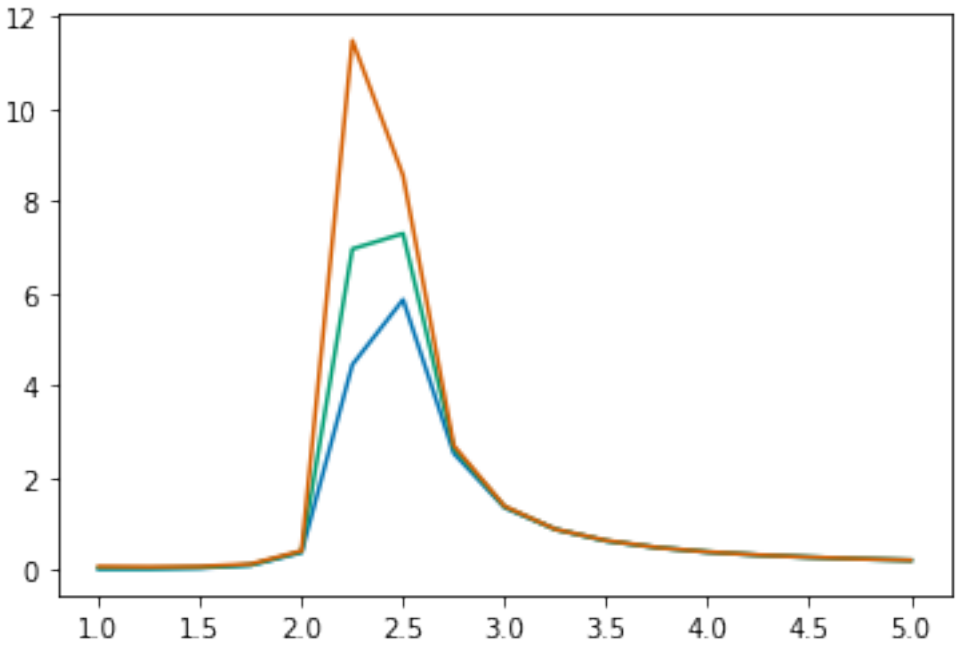

[10]:
```
plot(plot3, x)
```




[11]:
```
plot(plot4, x)
```




[12]:
```
plot(plot5, x)
```




### 1.1.4 Análise dos Resultados Obtidos

Analisando os resultados obtidos, pode-se observar, a partir do primeiro gráfico, que em todos os modelos a média de energia por spin cresce juntamente com a temperatura, em decorrência a uma maior desordem no sistema quanto mais alta a temperatura.

Já para o segundo gráfico, o calor específico cresce rapidamente entre temperaturas de 1.5 e 3, decaindo após esse intervalo.

O terceiro gráfico tem um comportamento análogo ao primeiro, porém de forma descendente, ou seja, o módulo da magnetização cai a medida que a temperatura aumenta.

O quarto e o quinto gráfico tem padrões parecidos (um rápido crescimento entre a temperatura 2 e 3, seguido por um decrescimento), variando apenas os valores do eixo y (em razão da natureza das grandezas observadas) e a curva da simulação com maior crescimento.

Comparativamente ao experimento de Wang-Landau anterior, nota-se que os resultados obtidos apresentam em conformidade com a teoria vista durante as aulas expositivas, e, portanto, parece-me correto. Além disso, o uso das técnicas de Monte Carlo possibilitou a execução de uma rede muito maior e com mais rapidez, o que é uma vantagem bastante significativa em vista da atividade previamente desenvolvida.