O Método de Monte Carlo - Algoritmo de Metropolis - Guilherme de Abreu Lima Buitrago Miranda - 2018054788

July 18, 2021

1 Introdução à Física Estatística Computacional

1.1 O Método de Monte Carlo - Algoritmo de 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
```

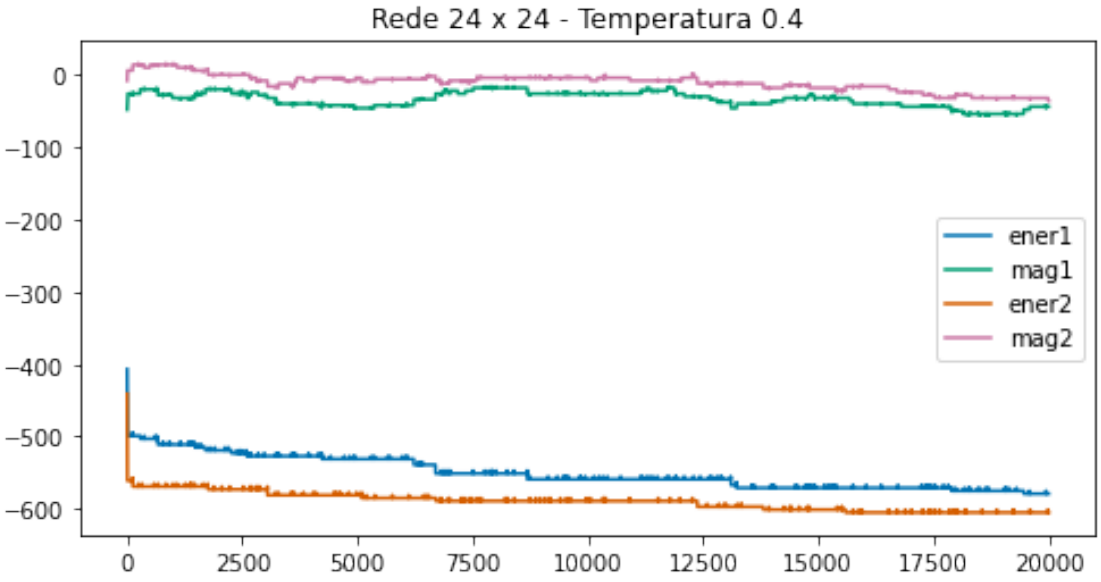
[3]:
```
@jit(nopython=True)
def expos(beta):
    ex = np.zeros(5,dtype=np.float32)
    ex[0]=np.exp(8.0*beta)
    ex[1]=np.exp(4.0*beta)
    ex[2]=0.0
    ex[3]=np.exp(-4.0*beta)
    ex[4]=np.exp(-8.0*beta)
    return ex
```

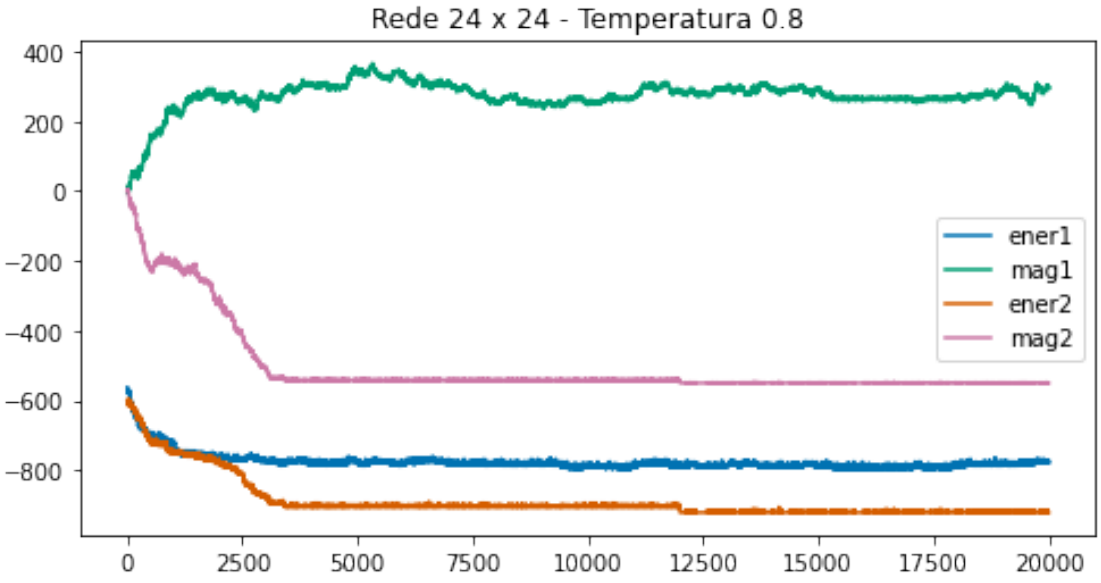
[4]:
```
@jit(nopython=True)
def mcstep(beta,s,viz,ener,mag):
    N=len(s)
    ex=expos(beta)
    for i in range(N):
        h = s[viz[i,0]]+s[viz[i,1]]+s[viz[i,2]]+s[viz[i,3]] # soma dos vizinhos
        de = int(s[i]*h*0.5+2)
        if np.random.random() < ex[de]:
            ener=ener+2*s[i]*h
 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

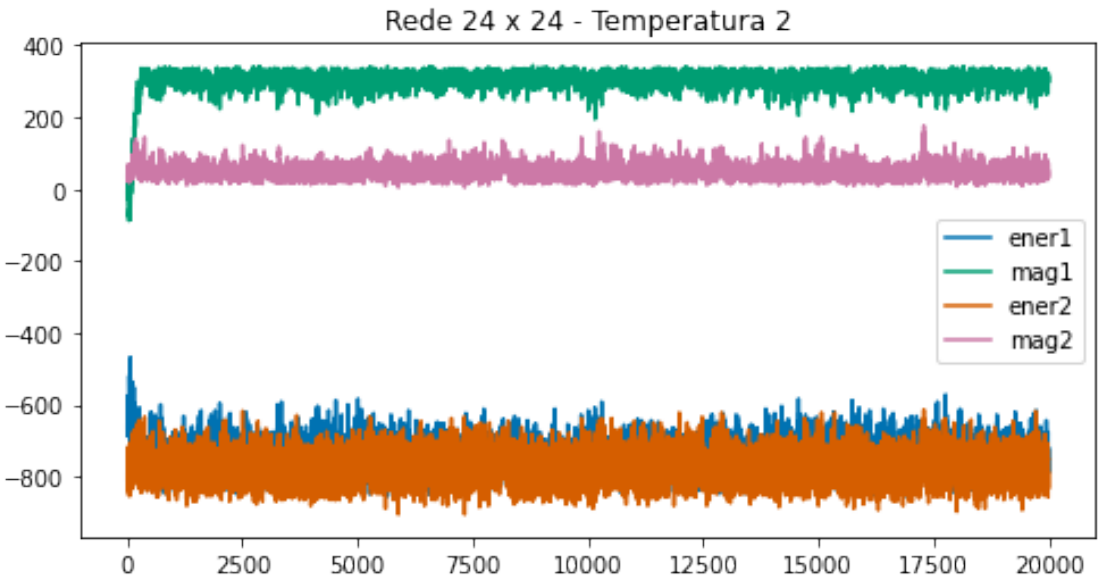
[7]:
```
def execute_all(l, n, Niter):
    s = estado_ini(n)
    viz = vizinhos(l, n)
    ener = energia(s, viz, n)

    s_2 = estado_ini(n)
    ener_2 = energia(s_2, viz, n)
    x = [0.4, 0.8, 2, 3]
```

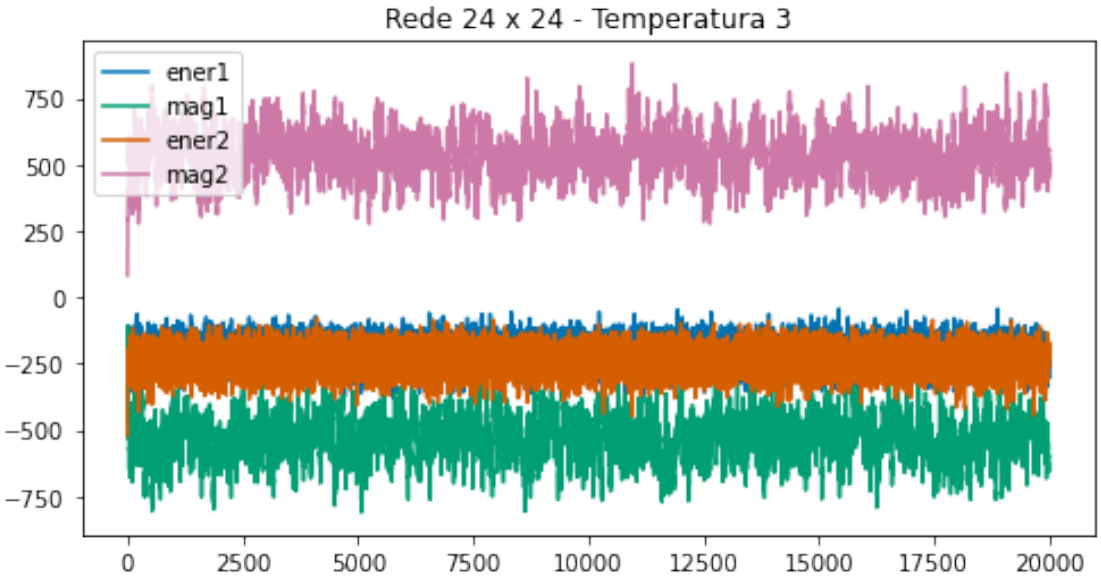
```
    #x = [3]
    #x = [1.5]

    for x_ in x:

        ener_list = []
        mag_list = []
        ener_list_2 = []
        mag_list_2 = []
        mag = 0
        mag_2 = 0

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

            ener_2, mag_2, s_2 = mcstep(1/x_, s_2, viz, ener_2, mag_2)
            ener_list_2.append(ener_2)
            mag_list_2.append(mag_2)

        plt.figure(figsize=(8, 4))
        plt.plot(range(Niter), ener_list)
        plt.plot(range(Niter), mag_list)

        plt.plot(range(Niter), ener_list_2)
        plt.plot(range(Niter), mag_list_2)
        plt.title("Rede " + str(l) + " x " + str(l) + " - Temperatura " +␣
↪str(x_))
        plt.legend(['ener1', 'mag1', 'ener2', 'mag2'], loc='best')
        plt.show()

    #return ener_list, mag_list, s_list
```

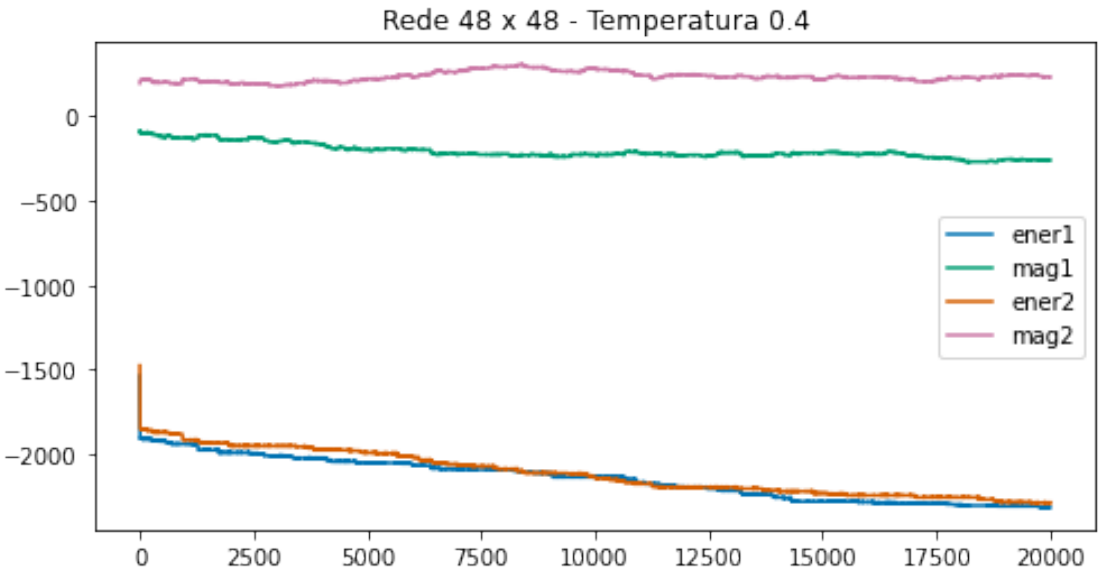
[8]:
```
ls = [24, 48, 100]
#ls = [32]
ns = [l * l for l in ls]



for l in ls:
    execute_all(l, (l * l), 20000)
```

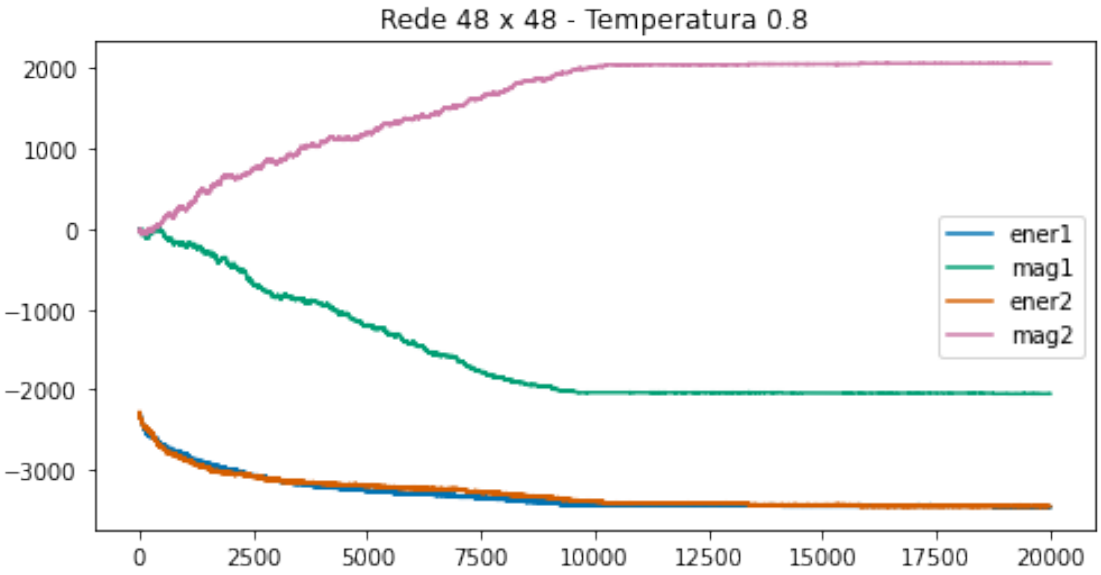

Rede 24 x 24 - Temperatura 0.4
0
−100
−200
−300
−400
−500
−600
0
2500
5000
7500
10000
12500
15000
17500
20000
ener1
mag1
ener2
mag2

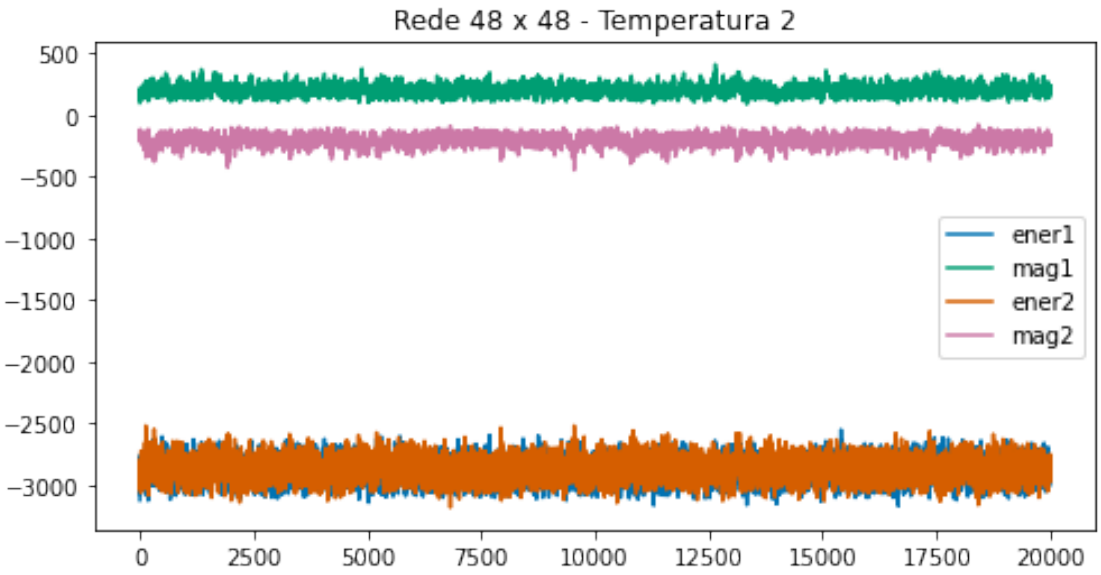

Rede 24 x 24 - Temperatura 0.8
400
200
0
−200
−400
−600
−800
0
2500
5000
7500
10000
12500
15000
17500
20000
ener1
mag1
ener2
mag2

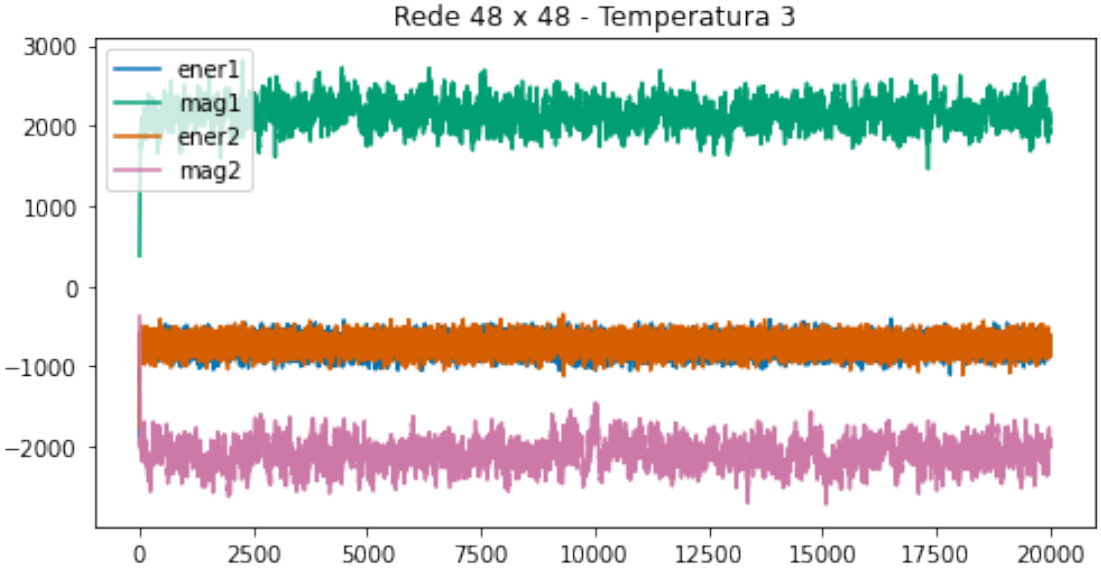

Rede 24 x 24 - Temperatura 2
400
200
0
−200
−400
−600
−800
0
2500
5000
7500
10000
12500
15000
17500
20000
ener1
mag1
ener2
mag2

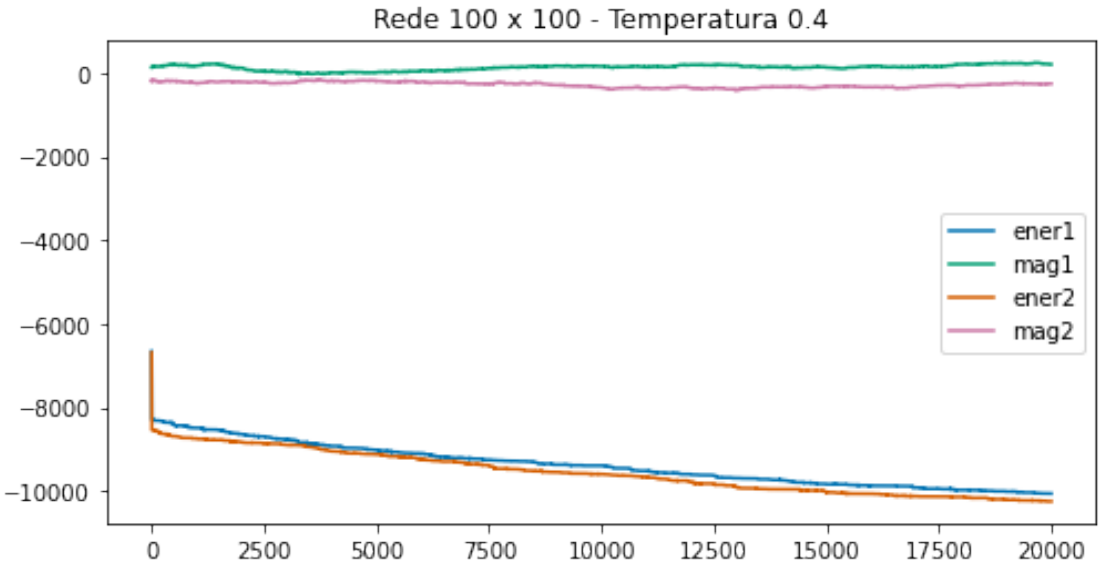

Rede 24 x 24 - Temperatura 3
750
500
250
0
−250
−500
−750
0
2500
5000
7500
10000
12500
15000
17500
20000
ener1
mag1
ener2
mag2

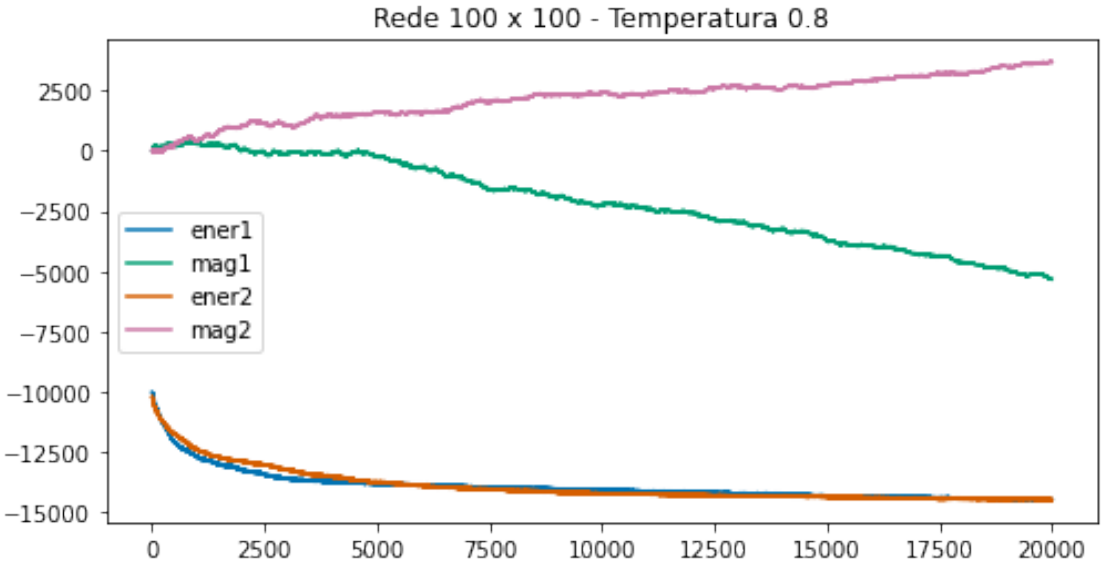

Rede 48 x 48 - Temperatura 0.4
0
−500
−1000
−1500
−2000
0
2500
5000
7500
10000
12500
15000
17500
20000
ener1
mag1
ener2
mag2

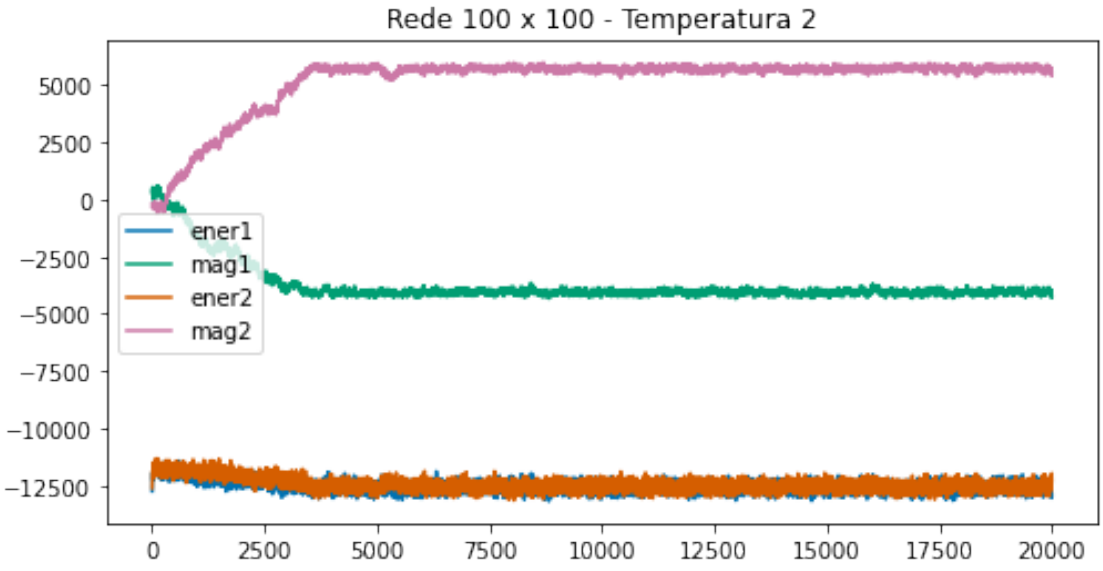

Rede 48 x 48 - Temperatura 0.8
2000
1000
0
−1000
−2000
−3000
0
2500
5000
7500
10000
12500
15000
17500
20000
ener1
mag1
ener2
mag2

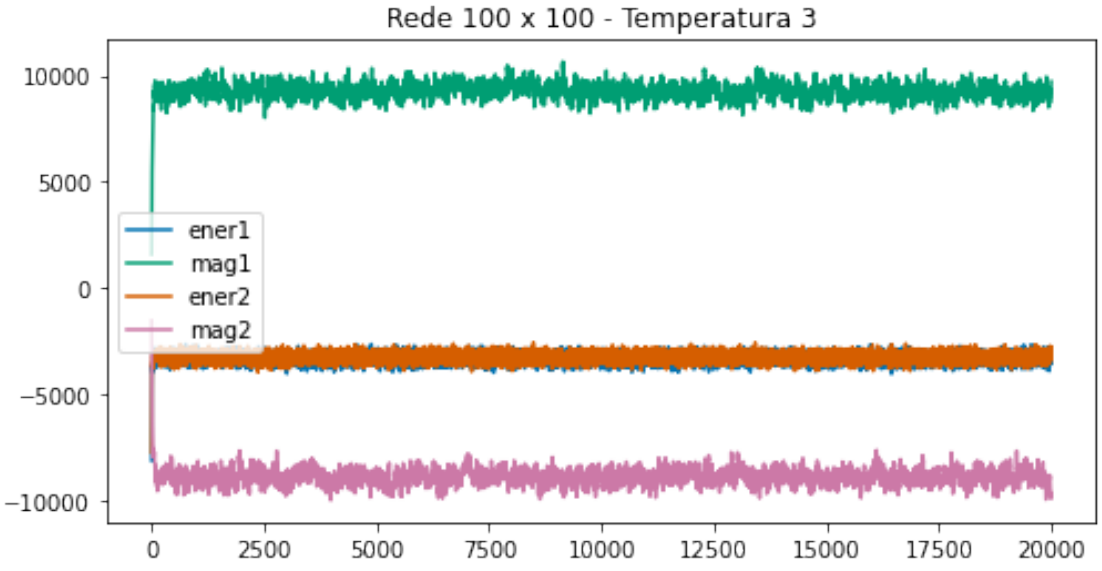

Rede 48 x 48 - Temperatura 2
500
0
−500
−1000
−1500
−2000
−2500
−3000
0
2500
5000
7500
10000
12500
15000
17500
20000
ener1
mag1
ener2
mag2


Rede 48 x 48 - Temperatura 3
3000
2000
1000
0
−1000
−2000
0
2500
5000
7500
10000
12500
15000
17500
20000
ener1
mag1
ener2
mag2


Rede 100 x 100 - Temperatura 0.4
0
−2000
−4000
−6000
−8000
−10000
0
2500
5000
7500
10000
12500
15000
17500
20000
ener1
mag1
ener2
mag2


Rede 100 x 100 - Temperatura 0.8
2500
0
−2500
−5000
−7500
−10000
−12500
−15000
0
2500
5000
7500
10000
12500
15000
17500
20000
ener1
mag1
ener2
mag2





1.1.3 Estimativas de passos

Tamanho/Temperatura	0.4	0.8	2	3
24 x 24	20	3000	750	750
48 x 48	20	8500	1500	750
100 x 100	20	20000+	3000	500

1.1.4 Análise dos Resultados Obtidos

Analisando os resultados obtidos, os gráficos gerados e a tabela de estimativas de passos acima, deve-se destacar, em primeiro lugar, que, dado a natureza aleatória da aplicação, nem sempre a mesma instância do problema tem o mesmo comportamento. Assim, a tabela acima e as análises a seguir foram pensadas observando tendências após várias execuções.

Um padrão interessante de se observar é que, quanto maior a temperatura do sistema, maior o ruído contido nas curvas plotadas, o que significa que a desordem do sistema é proporcional à temperatura. Quanto menor a temperatura, portanto, mais estável é o sistema e mais rápido ocorre sua convergência, conforme observado para as instâncias de temperatura 0.4.

Por fim, é também possível observar uma tendência ao agrupamento das curvas em boa parte das instâncias, evidenciando um padrão, ainda que a implementação tenha sido feita usando a aleatoriedade pelo método de Monte Carlo (o que resulta em execuções mais velozes).

[ ]: