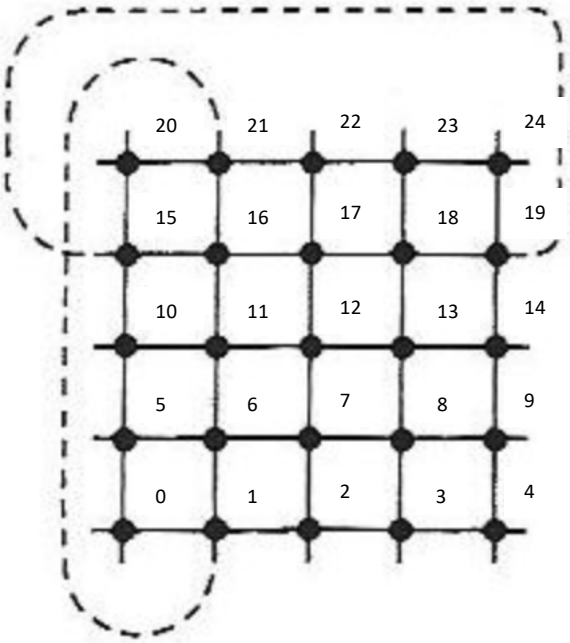
Implementação computacional do modelo de Ising.

Visando uma implementação um pouco mais otimizada do modelo de Ising do que um programador sem experiência na área faria, abaixo você será guiado em formas mais eficientes e gerais de representar modelos em redes.

Iremos considerar o modelo de Ising numa rede quadrada bidimensional com $L \times L$ sítios. A posição geométrica de cada sítio, ou seja, suas coordenadas x e y, pelo menos em nossa abordagem inicial, não será necessária. Assim, é mais conveniente e eficiente que cada sítio seja indexado por apenas um número inteiro. A convenção a ser adotada aqui é a representada na figura abaixo para uma rede 5x5.



De posse dessa convenção, o estado do sistema pode ser representado por um array unidimensional com $N = L^2$ elementos. Cada elemento recebe o valor ± 1, indicando o valor da variável de spin do sítio correspondente. Uma sugestão é nomear esse array de S.

Uma vez definido o estado do sistema, devemos ser capazes de calcular a energia deste estado. Por definição, a energia é dada por

$$E = -\sum_{(i,j)} s_i s_j - B \sum_i s_i.$$

Por simplicidade vamos considerar apenas o caso $B = 0$. Nesta definição, o símbolo (i, j) representa os primeiros vizinhos na rede quadrada. Assim, por exemplo, o sítio 13 tem como primeiros vizinhos os sítios 14, 18, 12 e 8. Uma pergunta que cabe neste ponto é como trataremos os spins nas bordas. Quais seriam os vizinhos do sítio 4, por exemplo. Uma vez que estamos interessados, em geral, no comportamento do sistema no limite termodinâmico, ou seja, de um sistema muito grande, suas bordas devem ser irrelevantes. Assim, para minimizarmos os efeitos das fronteiras (bordas) da rede sobre

as propriedades do sistema, é comum usar condições de contorno periódicas. Neste tipo de tratamento das fronteiras, consideraremos que o sítio 4 tem como vizinho à esquerda o sítio 0 e como vizinho abaixo, o sítio 24. É como se transformássemos essa rede quadrada em um toro, unindo suas extremidades. Uma forma de guardarmos de forma adequada essas informações é utilizar uma tabela de vizinhos. Há diversas formas de fazermos isso. Aqui, iremos usar um array bidimensional onde o primeiro elemento representará o sitio da rede enquanto o segundo representará qual dos vizinhos estamos considerando. Utilizaremos o índice 0 para o vizinho à direita, o 1 para o vizinho acima, o 2 para o vizinho à esquerda e o 3 para o vizinho abaixo. Os sítios da linha inferior têm índice, k, menor que L. Já os da lateral esquerda são aqueles para os quais o resto da divisão inteira de k por L é zero. Os da lateral direita são aqueles para os quais o resto da divisão inteira de $k+1$ por L é zero e os da linha superior são aquelas para os quais $k > N-1-L$. Podemos, então, usar o seguinte pseudocódigo:

Para $k = 0, \ldots, N-1$ ***faça***
$viz(k,0) \leftarrow k+1$
se $(k+1$ ***mod*** $L = 0)$ $viz(k,0) \leftarrow k+1-L$
$viz(k,1) \leftarrow k+L$
se $(k > N-1-L)$ $viz(k,1) \leftarrow k+L-N$
$viz(k,2) \leftarrow k-1$
se $(k$ ***mod*** $L = 0)$ $viz(k,2) \leftarrow k+L-1$
$viz(k,3) \leftarrow k-L$
se $(k < L)$ $viz(k,3) \leftarrow k+N-L$

Uma vez que definimos a rede, os vizinhos e as variáveis de spin, podemos calcular a energia de qualquer configuração. Perceba, no entanto, que para contar cada par de spins apenas uma vez podemos considerar apenas os vizinhos à direita (0) e acima (1) de cada sítio. Então, um pseudocódigo do cálculo da energia fica:

$E \leftarrow 0$
Para $i = 0, \ldots, N-1$ ***faça***
$h \leftarrow s(viz(i,0)) + s(viz(i,1))$
$E \leftarrow E - s(i) * h$

Neste ponto você deve estar se perguntando: "se precisamos de apenas de 2 dos 4 vizinhos, por que nos preocupamos em definir os 4 vizinhos anteriormente?" A definição dos 4 vizinhos se justifica pelo fato de o cálculo da energia total envolver um loop de tamanho N. Mas, ao flipar (mudar a direção) apenas um spin, podemos calcular a diferença de energia considerando apenas os 4 vizinhos. Assim, se fliparmos o spin do sítio i, ou seja, se fizermos $s_i' = -s_i$, a energia total do sistema após o flip, E_f, pode ser obtida da energia antes de fliparmos o spin, E_i, fazendo

$$E_f = E_i + 2 * s_i * \left(\sum_j s_j \right) = E_i - 2 * s_i' * \left(\sum_j s_j \right).$$

Essa última expressão mostra que as energias permitidas para o modelo de Ising estão no intervalo de $-2N$ a $2N$ em passos de 4, sendo que os estados de energia $-2N+4$ e $2N-4$ não são permitidos. Verifique essas afirmações.

Aplicação do Algoritmo de Metropolis ao Modelo de Ising 2D

O algoritmo de Metropolis pode ser escrito como:

1) Gere uma configuração inicial para o sistema (aleatória, por exemplo).
2) Escolha um dos spins da rede (S_i).
3) Determine a diferença de energia caso o spin S_i fosse flipado, ΔE.
4) Calcule $P = e^{-\beta \Delta E}$ e compare com um número aleatório, r, uniformemente distribuído no intervalo (0,1).
    a. Se $r \leq P$, aceite a nova configuração, ou seja, flipe o spin fazendo $S_i = -S_i$.
    b. Se $r > P$, mantenha o sistema na configuração em que ele se encontrava.
5) Volte ao passo 2.

Um detalhe que pode parecer estranho à primeira vista, mas que é o procedimento correto, é que se o estado proposto for rejeitado, o estado no qual o sistema se encontrava deve ser contado novamente na sequência de estados. Se não fizéssemos isso, a distribuição gerada não seria a correta. Nesse algoritmo, se considerarmos cada iteração descrita acima, ou seja, cada passo sendo a tentativa de flipar um único spin da rede, a dinâmica do sistema será lenta, no sentido que muitos passos serão necessários para que a configuração mude apreciavelmente. Com isso, um dos pressupostos do processo markoviano vai ser violado, já que a correlação entre dois passos consecutivos será enorme. Uma forma que encontramos de diminuir um pouco esse efeito é considerar como um passo de tempo a tentativa de flipar todos os N spins da rede. Assim, nas simulações consideraremos o número de passos de Monte Carlo como uma medida do tempo, sendo um passo de Monte Carlo a tentativa de se flipar N spins.

Em termos práticos de implementação, alguns detalhes devem ser levados em conta. Um ponto importante é que como o número de iterações (passos de Monte Carlo) é muito grande, qualquer economia de tempo computacional nos passos de 2 a 4 do algoritmo acima é muito bem vinda. Nesse sentido, podemos notar que as diferenças de energia no modelo de Ising 2D podem assumir apenas os valores $-8, -4, 0, 4, 8$, de forma que se tabelarmos os valores das exponenciais de $-\beta \Delta E$ para cada temperatura ($\beta = 1/T$), economizaremos um tempo enorme no cálculo de exponenciais. Sugiro fazer isso através de uma função como a delineada abaixo

```
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]=1.0
    ex[3]=np.exp(-4.0*beta)
    ex[4]=np.exp(-8.0*beta)
    return ex
```

Assim, ao tentar flipar o spin i, podemos calcular a diferença de energia e reescalá-la para usá-la como índice do array ex como delineado abaixo,

```
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)
```

de forma que *ex[de]* fornece os valores das exponenciais. Perceba que fazendo dessa forma, mesmo nos casos onde a energia aumenta, podemos usar a exponencial calculada para aceitar ou rejeitar a configuração proposta. Isso se deve ao fato de nestes casos a exponencial assumir valores maiores que 1 de forma que ao compará-la a um número aleatório escolhido no intervalo [0,1) a configuração proposta sempre será aceita.

Aconselho, fortemente, o uso da biblioteca Numba (https://numba.pydata.org/) para quem for programar em Python. Essencialmente a biblioteca pré-compila uma rotina transformando-a num objeto que é executado ao ser chamado, aumentando muito a performance do programa. Para usá-la basta importa-la no início do programa:

```
from numba import jit
```

e utilizar o comando @jit(nopython=True) logo acima da definição da rotina como no exemplo abaixo:

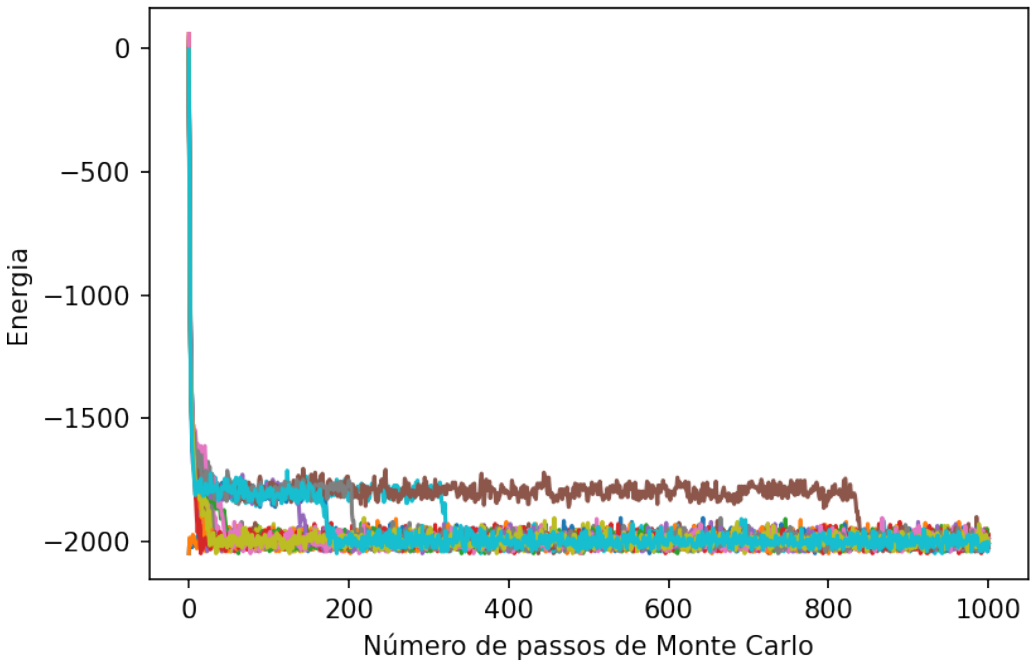
```
@jit(nopython=True)
def vizinhos(N):
    #Define a tabela de vizinhos
    L=int(np.sqrt(N))
    viz = np.zeros((N,4),dtype=np.int16)
    for k in range(N):
        viz[k,0]=k+1
        if (k+1) % L == 0: viz[k,0] = k+1-L
        viz[k,1] = k+L
        if k > (N-L-1): viz[k,1] = k+L-N
        viz[k,2] = 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
```

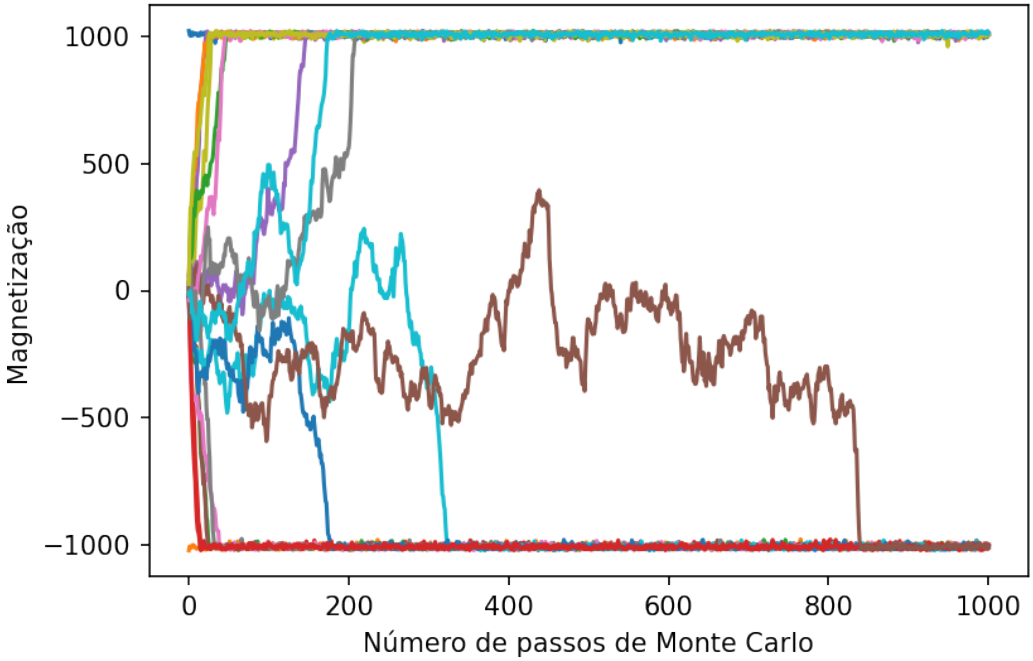
Usando o Numba na rotina que executa um passo de Monte Carlo diminui enormemente o tempo de computação.

Atividade:

A atividade que iremos realizar é uma exploração do **processo de termalização.** Iremos tentar estimar quantos passos de Monte Carlo devem ser descartados no início de uma simulação para atingirmos um estado estacionário. Para tanto, vocês terão que implementar um programa que realiza uma simulação do modelo de Ising 2D usando o algoritmo de Metropolis. As rotinas delineadas anteriormente poderão ser aproveitadas.

Para estimar o número de passos necessários para termalizar o sistema, façam gráficos da energia e da magnetização em função do número de passos de Monte Carlo considerando que o sistema parte de diferentes configurações iniciais. Para isso, para cada tamanho de rede e temperatura escolhidos, uma nova configuração aleatória deve ser gerada e, partindo dessa, um determinado número de passos de Monte Carlo deve ser executado. Não se esqueçam de redefinir os valores de energia e magnetização da configuração inicial. No exemplo das figuras abaixo, mostro a evolução temporal da energia total e da magnetização total partindo de 20 configurações iniciais diferentes feitas para uma rede com $L = 32$ e na temperatura $1{,}5$. Como podem perceber, apesar da maioria das simulações passarem a oscilar em torno de um mesmo valor comum para cerca de 400 passos, uma especificamente demorm um tempo maior que 800 passos para convergir. A cada execução do programa que gerou esses resultados, os tempo obtidos são diferente, podendo ocorrer, inclusive, de alguma simulação não convergir dentro do tempo máximo estudado. Por esse motivo, em geral, para termos confidência do tempo de termalização necessário pegamos um número consideravelmente superior ao estimado desta forma, uma vez que o resultado pode depender muito da configuração inicial e da sequência de números aleatórios usada. Note que os valores estacionários da magnetização podem ser tanto positivos quanto negativos.





Vocês devem, então, verificar de forma **qualitativa** como o número de passos de termalização varia de acordo com o tamanho do sistema e com a temperatura da simulação. Sugiro tamanhos de rede entre 24 e 100 e temperaturas entre 0.4 e 3, mas sintam-se livres para explorar mais tamanhos ou temperaturas. Esse exercício de "brincar" com diferentes parâmetros, valores iniciais dos spins, etc, ensina muito sobre os métodos de Monte Carlo e incentivo que explorem o máximo possível. Comportamentos estranhos podem surgir, principalmente em baixas temperaturas, não se assustem! Discutiremos isso posteriormente. Para os tamanhos de rede e temperaturas escolhidos, façam estimativas do número de passos necessários para termalizar o sistema e apresente-as numa tabela.

Novamente vocês devem fazer um breve relatório apresentando os resultados encontrados e sua análise.

Referências:

- M. Newman, G. Barkema, "Monte Carlo Methods in Statistical Physics", Claredon Press (1999)
- D. Landau, K. Binder, "A Guide to Monte Carlo Simulations in Statistical Physics", Cambridge University Press (2014)
- W. Krauth, "Statistical Mechanics: Algorithms and Computations" Oxford University Press (2006)