O Método de Monte Carlo – Algoritmo de Metropolis

Conforme vimos, as propriedades termodinâmicas de sistemas físicos como o modelo de Ising podem ser obtidas analiticamente como no caso unidimensional, cuja solução é desinteressante por não apresentar transições de fase, ou o caso bidimensional onde o modelo apresenta transição de fase, mas a dedução original de Onsager beira o incompreensível. Para "driblar" as dificuldades encontradas fazendo as contas analíticas podemos por um computador para calcular a função de partição "na marra", o que é conhecido por enumeração exata. No entanto, como para uma rede de N spins o número de estados é 2^N, o tempo computacional necessário aumenta muito rapidamente com o tamanho do sistema, limitando nossa análise a sistemas extremamente pequenos. Se quisermos resultados relevantes e úteis para o modelo de Ising (ou similares), o melhor é abdicar de ficar procurando a solução exata e partir para métodos aproximados. Um método que permite um controle maior da relação entre o número de iterações e a acurácia dos resultados é o método de Metropolis. Este foi o primeiro método de Monte Carlo a ser usado, sendo simples e altamente versátil, podendo ser aplicado nas mais diversas situações, desde cálculos de integrais a simulações de reações nucleares, por exemplo. No que segue esse método será brevemente exposto e sua aplicação ao modelo de Ising delineada.

O algoritmo de Metropolis

O algoritmo de Metropolis foi proposto em 1953 por Nicolas Metropolis e colaboradores. O trabalho foi considerado como a primeira simulação de Monte Carlo e apresenta, além do algoritmo, a simulação de um sistema de discos rígidos em duas dimensões. Muitas vezes o algoritmo de Metropolis é confundido com o método de Monte Carlo em si, talvez pelo fato de ser o método mais utilizado.

Antes de apresentarmos os fundamentos do método, vamos lembrar que os valores esperados de grandezas físicas podem ser calculados como uma soma sobre todas as configurações acessíveis ao sistema, i.e.,

$$\langle A \rangle = \frac{\sum_{\{\sigma\}} A e^{-\beta E_\sigma}}{\sum_{\{\sigma\}} e^{-\beta E_\sigma}} = \sum_{\{\sigma\}} A p_\sigma(\beta),$$

onde $p_\sigma(\beta) = e^{-\beta E_\sigma}/Z$ é a probabilidade de encontrar o sistema no estado σ na temperatura $T = 1/\beta$, a chamada probabilidade de Boltzmann. A ideia básica do método é chamada de **amostragem por importância** (importance sampling) e reside no fato que ao invés de calcularmos o valor esperado como uma soma sobre todas as configurações acessíveis, onde cada valor da grandeza seria ainda multiplicado pela probabilidade de encontrar o sistema naquele estado, podemos calculá-lo como uma média aritmética simples sobre estados que sejam escolhidos seguindo a distribuição de probabilidades desejada. Assim, se escolhermos M estados seguindo a distribuição de probabilidades $p_\sigma(\beta) = e^{-\beta E_\sigma}/Z$, o valor esperado pode ser obtido através de uma média aritmética simples,

$$\langle A \rangle = \frac{1}{M} \sum_{i=1}^{M} A_i,$$

onde A_i é o valor da grandeza na i-ésima configuração gerada. A questão agora é como gerar configurações seguindo uma distribuição de probabilidades que não conhecemos, já que Z não foi calculado.

Uma forma de gerar estados seguindo uma dada distribuição de probabilidades é através de um **Processo Markoviano.** Poderíamos dizer que um processo Markoviano é um mecanismo que, dado um estado μ ele gera um novo estado ν. Isso é feito de forma aleatória pois para um estado μ específico nem sempre o mesmo estado ν será gerado. Assim, a probabilidade de gerar o estado ν sendo dado o estado μ, será $P(\mu \to \nu)$, que é chamada de **probabilidade de transição** ou **taxa de transição.** Um processo Markoviano será, então, definido por essas taxas, que irão determinar a dinâmica do processo. Para podermos chamar de processo Markoviano, devemos fazer duas exigências: 1) As taxas $P(\mu \to \nu)$ devem ser fixas, ou seja, não podem variar à medida que o tempo passa ou que o processo ocorre; 2) As taxas devem depender apenas das propriedades dos estados μ e ν e não podem depender de nenhum outro estado pelo qual o sistema tenha passado ou venha a passar. Isso indica que um processo Markoviano não tem nenhum tipo de memória e que a probabilidade de gerar o estado ν partindo do estado μ depende apenas dos estados e de nada mais. Obviamente, as taxas também devem satisfazer a relação

$$\sum_{\nu} P(\mu \to \nu) = 1,$$

uma vez que um estado qualquer **deve** ser gerado. Note também que podemos ter $P(\mu \to \mu) \neq 0$ desde que a equação acima seja satisfeita.

Numa simulação de Monte Carlo convencional o que fazemos é usar um procedimento de Markov repetidamente para gerar uma sequência de estados que chamamos de uma **cadeia de Markov**. No algoritmo de Metropolis o que faremos é gerar, através de um processo Markoviano, uma sequência de estados que siga a distribuição de probabilidades desejada, a distribuição de Boltzmann. Para tanto, iremos impor mais duas condições, **ergodicidade** e **balanço detalhado**.

A **condição de Ergodicidade** estabelece que através do processo Markoviano temos que ser capazes de, partindo de uma dada configuração qualquer μ, chegar a qualquer outro estado acessível ao sistema após executar o procedimento um número suficientemente grande de vezes, ou, em outras palavras, por tempo suficiente. Isso implica que mesmo que a probabilidade de sair do estado μ e chegar no estado ν seja extremamente pequena, temos que ser capazes de gerar todas as configurações acessíveis ao sistema usando o processo Markoviano. Abrir mão dessa condição seria impor a existência de algumas configurações terem probabilidade nula de ocorrer, o que não é o caso. Perceba que mesmo que $P(\mu \to \alpha) = 0$, podemos satisfazer a condição de ergodicidade se $P(\mu \to \nu) \neq 0$ e $P(\nu \to \alpha) \neq 0$, de forma que da configuração μ podemos atingir a configuração α ao passarmos antes pela configuração ν.

Antes de tratarmos da condição de balanço detalhado, vamos considerar uma sequência muito grande de estados gerados por um processo markoviano. Nesse caso, poderíamos escrever a **equação mestra,** que descreve o comportamento da probabilidade de o sistema estar num estado μ, $p_\mu(t)$, como função do tempo t,

$$\frac{dp_\mu}{dt} = \sum_{\nu} p_\nu P(\nu \to \mu) - \sum_{\nu} p_\mu P(\mu \to \nu).$$

O primeiro termo do somatório representa a probabilidade de o sistema estar no estado ν e ir para o estado μ no próximo passo de tempo, enquanto o segundo representa a probabilidade de o sistema

estar no estado μ e ir para o estado ν no próximo passo de tempo, diminuindo assim $p_\mu(t)$. O que queremos no final das contas é gerar uma sequência de estados que siga a distribuição de Boltzmann e que a distribuição de estados gerados não varie com o tempo. Temos, então, que atingir um **estado estacionário** no processo. Isto quer dizer que devemos exigir que a distribuição de probabilidades dos estados gerados seja independente do tempo. Perceba que isso é diferente de exigir que as taxas de transição sejam independentes do tempo. Vamos exigir, então, que $\frac{dp_\mu}{dt} = 0$, o que nos leva à **condição de balanço global**,

$$\sum_\nu p_\nu P(\nu \to \mu) = \sum_\nu p_\mu P(\mu \to \nu),$$

que pode ter várias soluções, umas mais simples e outras bem complicadas. A solução mais simples que podemos pensar é a **condição de balanço detalhado**:

$$p_\nu P(\nu \to \mu) = p_\mu P(\mu \to \nu).$$

Assim, ao impor a condição de balanço detalhado, conseguimos garantir que a distribuição de probabilidades dos estados gerados seja estacionária, ou seja, não dependa do tempo. Além disso, um fato que não será demonstrado aqui, mas que pode ser encontrado nas referências ao final do texto, é que ao exigir que a condição de balanço detalhado seja satisfeita, conseguimos também garantir que após um tempo suficientemente longo o processo markoviano gerará uma sequência de estados que siga a distribuição de probabilidades desejada.

Voltemos à questão inicial de gerar um conjunto de estados que sigam a distribuição de Boltzmann $p_\mu(\beta) = e^{-\beta E_\mu}/Z$. Para tanto, utilizaremos um processo markoviano que será definido pelas taxas $P(\mu \to \nu)$ e que satisfará as condições de ergodicidade e balanço detalhado. Assim, garantiremos que após um tempo suficientemente longo o conjunto de configurações gerado seguirá a distribuição de Boltzmann. Em geral é difícil demonstrar rigorosamente que a condição de ergodicidade é satisfeita, e não faremos isso aqui. Iremos apenas supor, por construção, que esse é o caso. Para satisfazer a condição de balanço detalhado iremos exigir:

$$\frac{P(\mu \to \nu)}{P(\nu \to \mu)} = \frac{p_\nu}{p_\mu} = \frac{e^{-\beta E_\nu}}{e^{-\beta E_\mu}} = e^{-\beta(E_\nu - E_\mu)}.$$

Esta condição pode ser satisfeita por

$$P(\mu \to \nu) = \begin{cases} e^{-\beta(E_\nu - E_\mu)}, & se\ E_\nu - E_\mu > 0, \\ 1 \quad caso\ contrário. \end{cases}$$

Ou seja, se a energia for reduzida, o estado proposto será sempre aceito. Se a energia do novo estado for maior que a energia do estado onde o sistema estava, o novo estado será aceito com probabilidade proporcional a $e^{-\beta(E_\nu - E_\mu)}$. A taxa de transição acima que define o algoritmo de Metropolis.

Alguns detalhes muito importantes: Temos garantia que obteremos uma distribuição de probabilidades estacionária e que esta é a desejada, porém isso ocorre apenas após um número suficientemente grande de passos no procedimento markoviano. Mas o que é um número suficientemente grande? Infelizmente não há uma resposta simples para isso. O que devemos fazer é acompanhar a evolução do sistema para garantir que ele atingirá um estado estacionário, ou seja, que os valores das grandezas obtidas não variem mais com o tempo. Assim, assumiremos que após atingir o estado estacionário a distribuição gerada seja a desejada. Como veremos, em alguns casos esse tempo que devemos esperar é relativamente longo e em outros extremamente curtos. Dito isto, fica claro que os valores esperados das grandezas só podem ser obtidos se considerarmos que temos a distribuição de probabilidades estacionária desejada. Antes de atingirmos esse estado estacionário

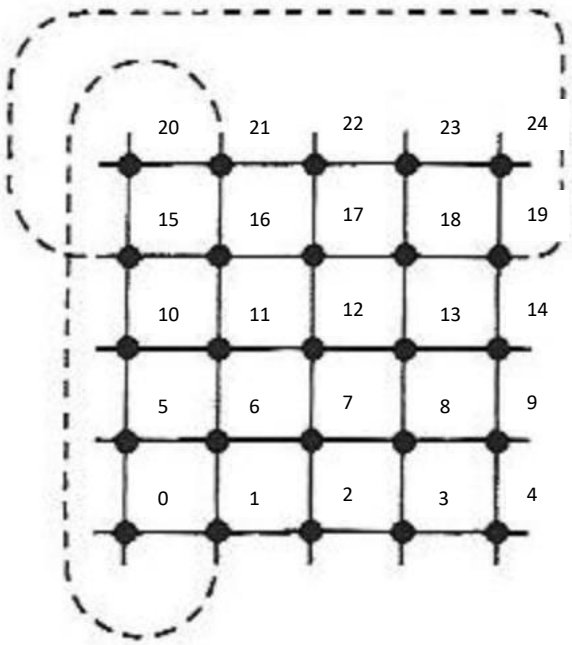
as propriedades obtidas das configurações geradas não corresponderão às desejadas. Devemos então descartar o conjunto inicial de configurações que é gerado antes de atingir a distribuição estacionária. Esse processo é chamado de **processo de termalização** e é um passo extremamente importante nas simulações de Monte Carlo.

No que segue, iremos aplicar o algoritmo acima ao modelo de Ising 2D. Antes de apresentar o algoritmo propriamente dito, ou seja, de apresentar uma lista de instruções para implementar a taxa de transição apresentada acima ao modelo de Ising, vamos abordar como podemos implementar computacionalmente de forma minimamente eficiente o modelo em si em um programa computacional.

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 é necessária. O que importa, de fato, são as conexões entre os diferentes sítios da rede (a rede quadrada é um grafo regular). Assim, é mais conveniente e eficiente que cada sítio seja indexado por apenas um número inteiro e que uma tabela de adjacências seja construída, indicando o vizinho de cada sítio. Aqueles que já tiveram algum contato com grafos devem ter percebido que a proposta apresentada aqui é a mesma utilizada para representar grafos. 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 sua energia e o valor de sua magnetização. Por definição, a energia é dada por

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

e a magnetização é dada por

$$M = \sum_i s_i.$$

Na definição da energia, 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. Para entender esse ponto melhor, preceba que na rede 5x5 representada acima 16 dos 25 spins estão na borda, o que representa 64% do total de spins. Já numa rede 500x500, apenas 1996 spins estariam na borda, o que representa aproximadamente 0,8% do total de spins! 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 bordas. Uma forma de guardarmos de forma adequada essas informações é utilizar uma tabela de vizinhos, que pode ser vista como uma forma mais adequada aos objetivos propostos de uma matriz de adjacências. 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 seus 4 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 código em python para construir essa tabela:

```
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
```

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 (cada aresta do grafo) apenas uma vez podemos considerar apenas os vizinhos à direita (0) e acima (1) de cada sítio. Então, um código python para o cálculo da energia fica:

```
def energia(s,viz):
    #Calcula a energia da configuração representada no array s
    N=len(s)
    ener = 0
    for i in range(N):
        h = s[viz[i,0]]+s[viz[i,1]]  # soma do valor dos spins a direita e acima
        ener -= s[i]*h
    return ener
```

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 primeiros vizinhos do sítio que teve o spin flipado. Note que de todas as ligações (arestas) existentes, apenas as 4 que incluem o sítio flipado se modificaram. As demais permanecem inalteradas. Assim, se fliparmos o spin do sítio i fazendo $s_i' = -s_i$, a diferença de energia será:

$$\Delta E = 2 * s_i * \left(\sum_j s_j \right) = -2 * s_i' * \left(\sum_j s_j \right),$$

onde o somatório em j envolve apenas os 4 sítios primeiros vizinhos do sítio i. A energia após o flip será dada por $E_f = E_i + \Delta E$, já que $\Delta E = E_f - E_i$. A expressão acima mostra que existem apenas 5 valores possíveis para diferença de energia ao flipar um spin, $-8, -4, 0, 4$ e 8 (sugiro que gastem alguns minutos mostrando esse fato). Com isto, 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

Como vimos, a taxa de transição dada por

$$P(\mu \to \nu) = \begin{cases} e^{-\beta(E_\nu - E_\mu)}, & se\ E_\nu - E_\mu > 0, \\ 1 & caso\ contrário. \end{cases}$$

permite gerar uma cadeia de Markov onde após o descarte de um número suficientemente grande de passos seremos levados a uma sequência de estados que segue a distribuição de Boltzmann. O algoritmo abaixo, conhecido como algoritmo de Metropolis, mostra como utilizar a prescrição acima para esse fim:

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 que está sendo gerada. 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 e utilizar uma medida de tempo mais adequada é considerar como um passo de tempo a tentativa de flipar N spins da rede, seja de forma sequencial, seja de forma aleatória. 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, e lembrando que as diferenças de energia no modelo de Ising 2D podem assumir apenas os valores $-8, -4, 0, 4, 8$, a construção de uma tabela com os valores das exponenciais de $-\beta\Delta E$ para cada temperatura ($\beta = 1/T$) economiza um tempo enorme no cálculo de exponenciais. Isso pode ser feito de várias formas diferentes. Uma sugestão é 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
```

que gera um array com os valores das exponenciais em termos de um índice, de, relacionado à diferença de energia. No mapeamento proposto, o índice de assumirá o valor 0 para $\Delta E = -8$, de=1 para $\Delta E = -4$, de=2 para $\Delta E = 0$, de=3 para $\Delta E = 4$ e de=4 para $\Delta E = 8$. Neste mapeamento o índice de pode ser calculado como

```
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).
```

Fazendo desta forma, *ex[de]* fornecerá os valores das exponenciais e a diferença de energia será dada por 2*s[i]*h. 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
```

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

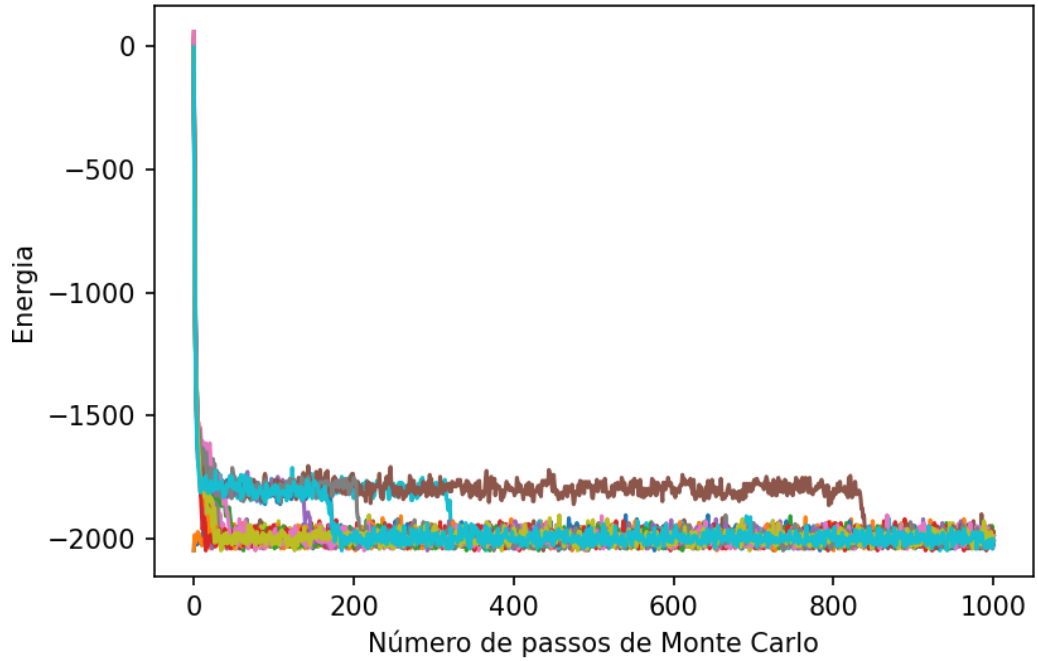
Outro fator que pode diminuir o tempo de simulação é notar que a energia total e a magnetização total não precisam ser calculadas a cada passo de Monte Carlo. Podemos calculá-las no início das simulações e ir atualizando seus valores a cada vez que um spin é flipado. Isso é computacionalmente mais barato, já que a diferença de energia já é determinada no algoritmo de Metropolis e que a variação no valor da magnetização é $-2 * s[i]$, onde $s[i]$ é o valor do spin antes de ser flipado (verifique!).

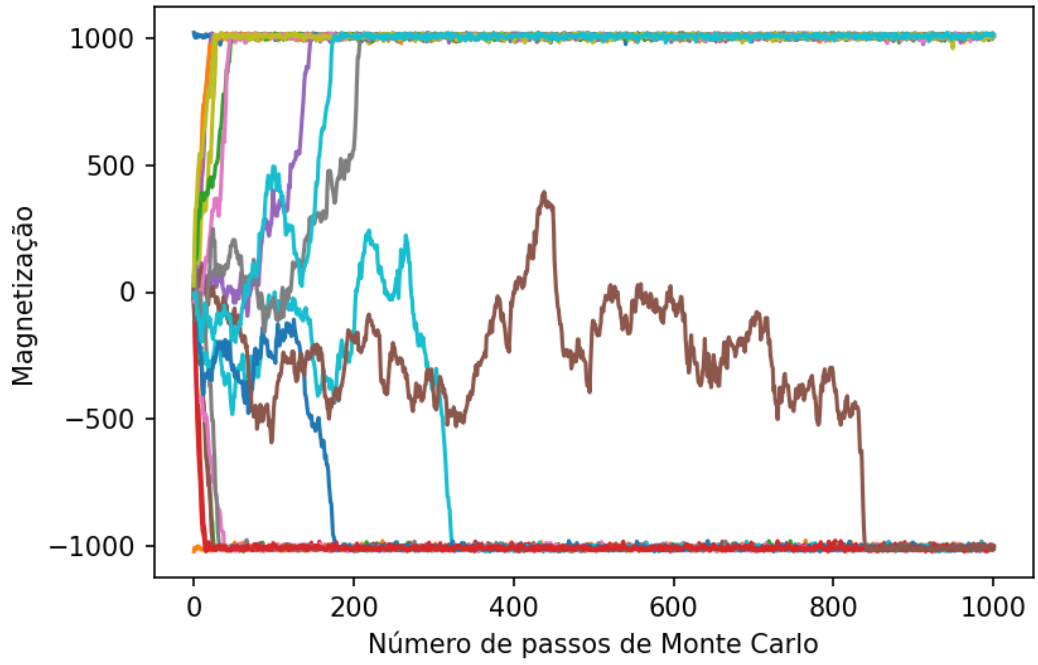
Atividade (parte 1):

A primeira parte da 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 demoram um tempo maior que 800 passos para convergir. A cada execução do programa que gerou esses resultados, os tempos obtidos são diferentes, 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. Em suma, consideramos que o sistema está termalizado quando todas as propriedades termodinâmicas do sistema passam a oscilar em torno de um mesmo valor médio constante para aquela temperatura.





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.

Simulação do Modelo de Ising 2D

Agora que já fizemos uma exploração inicial do algoritmo de Metropolis aplicado ao modelo de Ising 2D, explorando os tempos de termalização necessários, estamos em condições de fazer uma simulação estimando grandezas termodinâmicas em um certo intervalo de temperaturas. Conforme deve ter ficado evidente, os dados gerados pelo método de Metropolis variam consideravelmente nos passos iniciais do algoritmo e, após certo tempo, convergem para um estado estacionário, passando a flutuar em torno de um valor médio bem definido. Assim, para estimarmos as grandezas termodinâmicas, devemos descartar os passos necessários para o algoritmo termalizar. O número exato de passos que devemos descartar não pode ser determinado. O que fazemos na prática é utilizar um número de passos consideravelmente maior que os estimados na tarefa anterior. De fato, há grande arbitrariedade nessas escolhas e deve-se sempre optar por estimativas mais conservadoras, no sentido de garantirem de forma mais consistente que a termalização será atingida.

Outro ponto que deve ser considerado com bastante cuidado são as fontes de erros nas simulações.

Podemos dividir os erros em duas classes: Erros Estatísticos e Erros Sistemáticos. Os erros sistemáticos são aqueles advindos do método em si e, portanto, não temos meios muito eficazes de controlá-los e estimá-los. Talvez, a maior fonte de erros sistemáticos em métodos de Monte Carlo são os intervalos de termalização e as correlações existentes entre configurações consecutivas. Em parte, o que fizemos na tarefa anterior, serve basicamente para evitarmos a primeira fonte de erro mencionada. Apesar de tanto para simulações do modelo de Ising como de outros modelos termos meios de explorar com cuidado e atenção as fontes de erros sistemáticos, neste curso não entraremos nesses detalhes, ficando assim para pesquisas futuras dos interessados. Aqui, trataremos basicamente dos erros estatísticos, que são aqueles advindos das mudanças aleatórias em medidas feitas em execuções diferentes da simulação, advindas basicamente da natureza estatística (aleatória) do método de Monte Carlo.

Erros estatísticos

Existem diversas formas para se estimar os erros estatísticos presentes numa simulação. Aqui, trataremos a forma mais simples e direta para estimar tais erros, o método de caixas. Ressalto, no entanto, que tratamentos mais elaborados devem ser usados em situações mais críticas. Para os mais interessados, recomendo pesquisarem pelos métodos de bootstrap e jackknife.

Suponha que numa simulação do modelo de Ising queiramos determinar os valores da energia, magnetização, calor específico e susceptibilidade magnética. O calor específico é definido como

$$c_v = \frac{\beta^2}{N}(\langle E^2\rangle - \langle E\rangle^2),$$

e a susceptibilidade é dada por

$$\chi = \frac{\beta}{N}(\langle M^2\rangle - \langle M\rangle^2),$$

onde $M = |\sum S_i|$ é a magnetização e $E = -\sum_{<i,j>} S_i S_j$ é a energia. Os valores de energia por spin e magnetização por spin são obtidos ao dividir o valor médio calculado para energia e magnetização pelo número de spins da rede. Perceba que utilizamos o módulo da magnetização uma vez que estados com todos spins em -1, o que equivale a magnetização $M = -N$, ou com todos spins em $+1$, o que equivale a magnetização $M = +N$, são completamente equivalentes. Note, ainda, que tais grandezas dependem dos valores esperados da magnetização e da energia. Como vimos, os valores de energia e magnetização obtidos após a termalização do sistema flutuam em torno de um valor médio, de forma que podemos estimar o valor médio da energia e da magnetização tomando a média aritmética simples destas grandezas dentre todos os valores gerados. Estes valores podem, então, serem usados para se obter estimativas do calor específico e da susceptibilidade.

Uma questão que deve ser notada é que mesmo sendo as estimativas do calor específico e da susceptibilidade tomados com base em valores médios da energia e magnetização, essas grandezas também estarão sujeitas a flutuações estatísticas. Resta, então, a pergunta: como estimar o erro estatístico de uma estimativa para c_v, por exemplo, numa simulação com N_{mcs} passos de Monte Carlo já que, a princípio, em cada simulação teremos apenas um valor para essa grandeza? Podemos pensar em duas formas distintas: 1) realizar várias simulações (chamaremos de amostras diferentes) e comparar as estimativas entre amostras distintas. Neste caso, em cada amostra usaríamos uma sequência diferente de números aleatórios e uma configuração inicial diferente e realizaríamos os N_{mcs} passos em cada uma dessas amostra. Obteríamos, assim, uma estimativa diferente para cada amostra considerada. Outra opção, 2), seria realizar uma simulação muito longa e dividi-la em n

blocos, contendo N_{mcs}/n passos de Monte Carlo cada, e estimar as grandezas em cada um destes blocos e comparar estas estimativas. Estes métodos são, em muitos aspectos, equivalentes. O primeiro é capaz de diminuir um pouco possíveis erros sistemáticos. No entanto, no método que utilizaremos e com o conhecimento que já foi acumulado ao longo dos anos de simulações do modelo de Ising 2D pelo algoritmo de Metropolis, tais erros podem ser negligenciados sem efeitos significativos, de forma que utilizaremos o método 2, método de blocos para estimar os erros estatísticos. Da forma exposta aqui, também negligenciaremos o tempo de correlação entre passos de Monte Carlo, que deveria, a princípio, ser considerado. Ou seja, consideraremos que cada configuração obtida é estatisticamente independente das outras, o que não é verdade. Como frisado anteriormente, em situações mais delicadas estes fatores não poderiam ser desconsiderados e deveriam receber o tratamento adequado.

O método das caixas consiste, então, em estimar, no caso do calor específico, valores para $\langle E\rangle_i$ e para $\langle E^2\rangle_i$, correspondendo aos $m = N_{mcs}/n$ passos da caixa i, em cada uma das caixas e combinar os n valores assim obtidos para calcular a estimativa final de c_v, permitindo assim estimar o erro estatístico associado. Assim, a estimativa da energia relacionada à caixa i será

$$\langle E\rangle_i = \frac{1}{m}\sum_{j=im}^{(i+1)m-1} E_j,$$

onde E_j corresponde ao valor da energia da configuração $j \in \{N_{mcs}\}$. Na expressão acima, assumi que o índice j varia de 0 a $N_{mcs}-1$ e que o índice i varia de 0 a $n-1$, de acordo com os índices de arrays em python que começam em 0. De forma equivalente, podemos obter uma expressão para $\langle E^2\rangle_i$, bastando para isso trocar E por E^2 na expressão acima. O calor específico correspondente à caixa i será, então,

$$(c_v)_i = \frac{\beta^2}{N}(\langle E^2\rangle_i - \langle E\rangle_i^2).$$

O valor médio do calor específico será

$$\langle c_v\rangle = \frac{1}{n}\sum_{i=0}^{n-1}(c_v)_i,$$

e o erro estatístico associado será dado por

$$err(c_v) = \sqrt{\frac{\sum_{i=0}^{n-1}(\langle c_v\rangle - (c_v)_i)^2}{n(n-1)}}.$$

Esse mesmo procedimento pode ser replicado para as demais quantidades, inclusive para a energia e magnetização, onde para determinar o erro estatístico associado utilizaremos os valores médios encontrados em cada uma das n caixas.

Atividade (parte 2):

Vamos explorar as propriedades termodinâmicas do modelo de Ising 2D fazendo simulações pelo algoritmo de Metropolis. Para realizar as simulações você deverá escolher valores para os parâmetros da simulação. Especificamente, você deverá escolher o tamanho do sistema, L, a temperatura de simulação, T, o número de passos de Monte Carlo para termalização, N_{Term}, e o número de passos de Monte Carlo para calcular as médias termodinâmicas, N_{MCS}.

1) Quais critérios você utilizou para escolher os valores dos parâmetros descritos acima? Provavelmente suas escolhas iniciais precisarão ser revistas, não há problemas! Pelo

contrário, o ideal é que ao longo do trabalho você vá revendo suas escolhas, aprimorando-as, mas quero saber quais os principais fatores que nortearam suas escolhas finais.

2) Descreva o comportamento observado para as principais grandezas termodinâmicas – Energia por spin, Magnetização por spin, calor específico e susceptibilidade magnética – em função da temperatura. Ou seja, ao variar a temperatura, o que acontece com o valor destas grandezas? Quais são os limites para baixas e altas temperaturas? Há algum pico ou vale? O comportamento está em acordo com o que você esperava?
3) Ao variar o tamanho do sistema, como as curvas destas grandezas em função da temperatura se modifica? Há algum intervalo de temperaturas no qual as grandezas são independentes do tamanho do sistema? Em regiões onde há variação com o tamanho do sistema, como a grandeza é modificada quando L aumenta?
4) Como é o comportamento dos erros estatísticos à medida que a temperatura varia? Tem algum valor de temperatura em torno do qual os erros são maiores? Você enxerga algum motivo para isso? Os erros estatísticos dependem do tamanho do sistema? Como?
5) Com base no comportamento encontrado, identifique possíveis fases do sistema, descrevendo as principais características das fases encontradas.
6) Estime, utilizando os dados das suas simulações, a temperatura de transição de fase do sistema no limite termodinâmico, i.e., para o limite em que o tamanho do sistema é infinito.

Para embasar suas respostas, apresente gráficos das grandezas que você estudou, representando adequadamente as barras de erro encontradas.

Algumas dicas:

Ao iniciar uma simulação em uma temperatura alta com uma configuração aleatória, esperamos que a configuração esteja "mais próxima" da região do espaço de configurações correspondente às configurações de equilíbrio desta temperatura. De forma semelhante, se reduzirmos a temperatura considerando como configuração inicial a última configuração gerada na temperatura maior que foi simulada anteriormente, esperamos que estejamos mais próximos à região de equilíbrio do espaço de configurações, de forma que este procedimento, i.e., se manter a última configuração da temperatura anterior como primeira configuração da próxima temperatura a ser estudada, tende a reduzir o tempo necessário para o sistema termalizar. Mesmo assim, ao variar a temperatura é necessário refazer o processo de termalização, preferencialmente com o número de passos estimados para o caso independente da configuração inicial.

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)