O Método de Monte Carlo – Algoritmo de Metropolis

Conforme vimos, as propriedades termodinâmicas de sistemas físicos como o modelo de Ising podem ser obtidas numericamente através de enumeração exata ou do método de Wang-Landau. No entanto, estes métodos apresentam restrições que podem impossibilitar o estudo de alguns sistemas ou de determinadas situações através deles. No caso da enumeração exata, vimos que o tempo computacional necessário aumenta muito rapidamente com o tamanho do sistema, limitando a análise a sistemas extremamente pequenos. No caso do algoritmo de Wang-Landau, apesar do tempo computacional crescer de forma muito mais lenta com o tamanho do sistema, a simulação de redes muito grandes também é inviável. Outro ponto que merece nota é o baixo controle que temos sobre a simulação. De fato, uma vez que não temos como prever quando o histograma ficará flat, não temos como prever de antemão quantas iterações serão necessárias para se obter resultados com uma dada precisão. Inclusive, pode ocorrer de a simulação ficar "presa" em uma determinada região e a condição de histograma flat nunca ser atingida. 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 valo 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**. Note que o algoritmo de Wang-Landau que estudamos anteriormente não gera uma cadeia de Markov já que durante a simulação as taxas de transição são alteradas continuamente, violando o ponto 1 definido acima. 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 Botzmann. 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,

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

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.

Aplicação 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

```
@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
```

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. Desta forma, uma sugestão para uma rotina que executa um passo de Monte Carlo é

```
@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
```

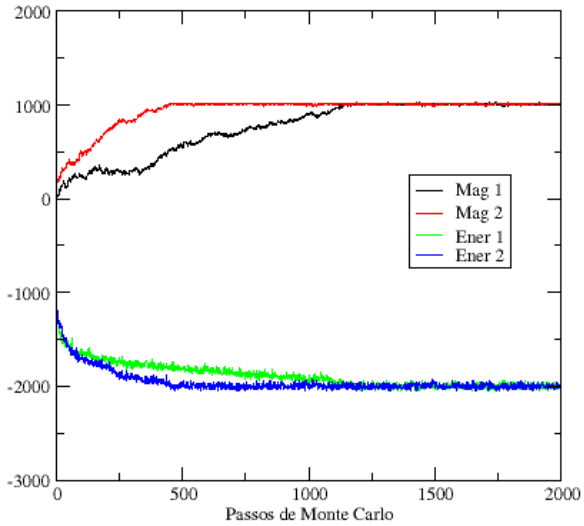
Não deixem de dedicar um tempo para entender os detalhes dessa rotina e do procedimento de Monte Carlo como um todo!

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 usadas anteriormente na simulação pelo método de Wang-Landau poderão ser aproveitadas.

Para estimar o número de passos necessários para termalizar o sistema, façam gráficos da energia e magnetização em função dos 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 redefinirem os valores de energia e

magnetização da configuração inicial. No exemplo da figura abaixo, mostro a evolução temporal da energia total e da magnetização total partindo de duas configurações iniciais aleatórias diferentes feitas para uma rede com $L = 32$ e na temperatura $1{,}5$. Como podem perceber, apenas após cerca de 1500 passos de Monte Carlo, os valores das grandezas passam a oscilar em torno de um mesmo valor médio. 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. Um teste interessante é considerar, também, que o sistema parte do estado fundamental com todos spins em $+1$ ou em -1.



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.