Uma abordagem didática sobre Monte Carlo

Matheus Roos

28 de dezembro de 2023

1 Introdução

A investigação teórica em física passa pela solução de equações, sejam elas diferenciais ou integrais, por exemplo. Uma ampla gama de problemas físicos envolve a resolução de integrais multidimensionais, e.g., no cálculo da função de partição em problemas de mecânica estatística.

Para ilustrar este caso, consideramos uma hamiltoniana genérica

$$\mathcal{H} = \frac{p^2}{2m} + U(\vec{r}, t) \ ,$$

onde a energia potencial U dependerá do problema em questão (para o caso de um gás ideal $U = 0$) e o termo $K = p^2/2m$ corresponde a energia cinética. Passamos agora para o cálculo da função de partição, para uma única partícula será

$$Z_1 = \int\int e^{-\beta\mathcal{H}} dp_1 dr_1 \ ,$$

e a função de partição do sistema como um todo pode ser escrito como

$$Z = Z_1 Z_2 \cdots Z_N = \int\int \cdots \int dr_1 dr_2 \cdots dr_N \int\int \cdots \int dp_1 dp_2 \cdots dp_N e^{-\beta\mathcal{H}} \ . \tag{1}$$

Temos o problema de resolver 2N integrais, na ausência de soluções analíticas devemos recorrer a métodos aproximativos/numéricos. Neste último caso, poderíamos pensar primeiro em utilizar um método de integração simples, como a regra de Simpson 1/3.

Vamos fazer um exercício mental conhecido como "estimativa de Fermi". Supomos que vamos calcular a função de partição como na Eq. (1) utilizando uma quadratura com 10 pontos (valor muito abaixo do usual) para um gás tridimensional, portanto vamos avaliar 10^{3N} pontos. Imaginemos um valor bastante modesto de partículas, $N = 20$. Um dos melhores supercomputadores, Fugaku, é capaz de realizar 10^{15} cálculos/s. Então o Fugaku levaria

$$t = \frac{10^{60}}{10^{15}} = 10^{45} s \ .$$

Para termos uma melhor noção deste número, a idade do universo é de cerca de alguns bilhões de anos, digamos $10^9 s$. Dessa forma, o Fugaku levaria 10^{36} a idade do universo para realizar o cálculos desta simples função de partição. Este exemplo dramático mostra que os métodos usuais de quadratura são são viáveis para integrais multidimensionais, devemos recorrer a um outro método.

O problema descrito acima era semelhante ao enfrentado por físicos durante a II Guerra Mundial no projeto Manhattan. Esse problema foi resolvido por *Stanislaw Ulam*, *John von Neumann* e

Nicholas Metropolis, dentre outros. O método foi chamado de Monte Carlo (MC), com a primeira publicação sendo 1949 [2]. O nome foi inspirado pelo famoso cassino Monte Carlo, em Mônaco. O nome é adequado devido a natureza estocástica e aleatória das simulações desenvolvidas pelos cientistas para resolver os problemas complexos e os jogos de azar do cassino.

A ideia fundamental das simulações de MC é usar a aleatoriedade para resolver problemas que podem ser determinísticos em princípio, mas cuja complexidade torna-os inviáveis de serem tratados pelos métodos convencionais.

Antes de explicar o método, façamos uma última comparação com os método usuais de quadratura. Pegamos como exemplo a quadratura Simpson 1/3, ela é mais precisa que Monte Carlo, desde que a dimensão seja $D < 4$ [4]. Além deste valor o método de MC se sobressai, além é claro, do fator tempo de processamento.

2 Integral de Monte Carlo

2.1 Caso unidimensional

Inciamos com o caso mais simples para ilustrar a ideia básica do método, a integral unidimensional. Consideremos o cálculo da seguinte integral

$$I = \int_a^b f(x)dx \ ,$$

para alguma função f bem comportada. Ao invés de dividirmos o intervalo de integração em subintervalos igualmente espaçados, consideremos tomar a média,

$$I \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \ ,$$

onde x_i serão os pontos escolhidos de forma randômica dentro do intervalo de integração [a,b].

A amostragem de Monte Carlo dará o resultado exato quando $N \to \infty$. No entanto isso não é possível de ser obtido na prática; temos que escolher um valor finito. Consequentemente, devemos fazer uma estimativa do erro estatístico, que é normalmente definido como o desvio padrão σ. Se repetirmos o procedimento por M vezes para o mesmo n conjunto de pontos, mas com uma sequência de números aleatórios diferentes (Fig. do erro por N), teremos então M médias independentes.

$$\langle A_i \rangle = \frac{1}{N}\sum_{j=1}^{N} A_{ij}$$

onde A_{ij} são os valores da função amostrada para a execução $i, j = 1, \cdots M$. O erro estatístico associado a cada uma destas médias individuais é o desvio padrão σ da distribuição destas médias, que pode ser estimado a partir dos dados gerados.

Como estamos tomando a média de uma medida estatística, devemos estimar o quanto nossos valores se distanciam da média, ou seja, calcular o desvio padrão da média. Do teorema do limite central [1], para um longo número N segue a seguinte lei estatística:

$$\sigma_I^2 \approx \frac{1}{N}\sigma_f^2 = \frac{1}{N}\left[\frac{1}{N}\sum_{i=1}^{N} f_i^2 - \left(\frac{1}{N}\sum_{i=1}^{N} f_i\right)^2\right] \ . \tag{2}$$

Como obtemos M médias diferentes, podemos claramente melhorar nossa estimativa se tomarmos a média das médias, ou seja

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

A incerteza sobre esta média é o desvio padrão da média (esta forma garante que se $M = 1$, ou seja, uma única varredura, a incerteza é indeterminada),

$$\langle \sigma \rangle = \frac{\sigma}{\sqrt{M-1}} = \sqrt{\frac{1}{M(M-1)} \sum_{i=1}^{M} (\langle A_i^2 \rangle - \langle A \rangle^2)} \ . \tag{3}$$

Podemos nos referir ao resultado de M execuções (*bins*), como $\langle A \rangle \pm \langle \sigma \rangle$.

O desvio padrão só possui significado se as médias $\langle A_i \rangle$ obedeça à uma distribuição gaussiana [6]. Embora amostas individuais de A_{ij} possam fugir de uma distribuição gaussiana, a "lei dos grandes números"garante que no infinito ($N \to \infty$) a média assumirá a forma da distribuição gaussiana. Isso é ilustrado na fig. (FAZER FIGURA DOS HISTOGRAMAS)

Podemos reduzir a variância ao multiplicar o integrando por um peso $w(x)$ normalizado,

$$\int_a^b w(x) dx = 1 \ .$$

A integral pode então ser reescrita como

$$I = \int_a^b dx w(x) \frac{f(x)}{w(x)}$$

se fizermos uma mudança de variável, obtemos (detalhes em [1])

$$I = \int_0^1 dy \frac{f(x(y))}{w(x(y))} \ .$$

A abordagem em MC é avaliar esta integral como

$$I \approx \frac{1}{N} \sum_{i=1}^{N} \frac{f(x(y_i))}{w(x(y_i))} \ .$$

O benefício desta mudança de variável é a redução da incerteza pelo seguinte mecanismo; devemos buscar encontrar um $w(x)$ que possua um comportamento semelhante a $f(x)$, i.e., que seja grande quando f também for, e pequeno quando f seja pequeno.

Uma outra maneira de entender esta melhoria é que a distribuição uniforme de pontos em y, implica que a distribuição de pontos em x é $w = dy/dx$, ou seja, que os pontos estarão mais concentrados nos valores mais importantes de x, onde w é grande e que dessa forma, pouco poder computacional será gasto no cálculo o integrando para pontos insignificantes.

Exemplo 1. *Consideremos a seguinte integral*

$$I = \int_0^1 \frac{dx}{1+x^2} = \frac{\pi}{4} = 0.7854 \ . \tag{4}$$

```
1  program Ex1
2  implicit none
3  integer :: seed = 987654321 !seed for random number generator
4  real :: exact = .78540      !exact answer
5  integer :: N, iX
6  real :: sumF, sumF2, fx
7  real :: f_ave, f2_ave, sigma
8
9  do
10   print *, 'Enter number of points (0 to stop)'
11   read *, N
12   if ( N == 0 ) STOP
13
14   sumF = 0.   !zero sums
15   sumF2 = 0.
16
17   do iX = 1, n    !loop over samples
18     fx = fun(ran(seed)) !integrand
19
20     !add contributions to sums
21     sumF = sumF + fX
22     sumF2 = sumF2 + fX**2
23   end do
24
25   f_ave = sumF/N
26   f2_ave = sumF2/N
27
28   sigma = sqrt((f2_ave - f_ave**2) / N)   !incerteza
29
30   print *, 'integral =', f_ave, '+/-', sigma, 'error =', exact - f_ave
31 end do
32 contains
33   real function fun(x)
34     implicit none
35     real, intent(in) :: x
36     fun = 1./(1. + x**2)
37   end function
38 end program Ex1
```

Listing 1: Script Fortran para calcular uma integral 1D via Monte Carlo (adaptado de [1]).

Neste caso, utilizamos a função **RAN** (intríseca do Fortran) para gerar números aleatórios, também fazemos uso de uma semente (*seed*) para inicializar a sequência de números pseudo-aleatórios. O processo encerra quando o usuário digita 0. As variáveis *sumF* e *sumF2* são inicializadas nas linhas 14 e 15 respectivamente. O *loop* sob as n amostras é inicializado na linha 17. O erro é estimado na linha 28, avaliada conforme a Eq. (2).

Exemplo 2. *Para a mesma integral do Exemplo 1, Eq. (4), escolhemos um peso*

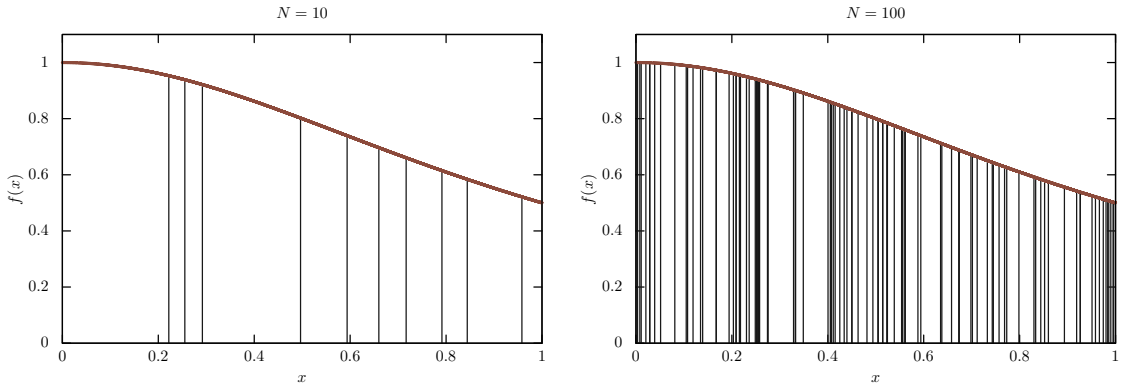
$$w(x) = (4 - 2x)/3 \ ,$$

com a nova variável de integração sendo

$$y = \frac{x(4-x)}{3} \ ,$$

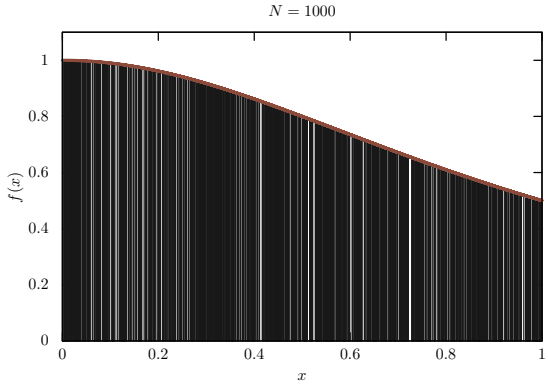
que pode ser invertida para

$$x = 2 - \sqrt{4 - 3y} \ .$$

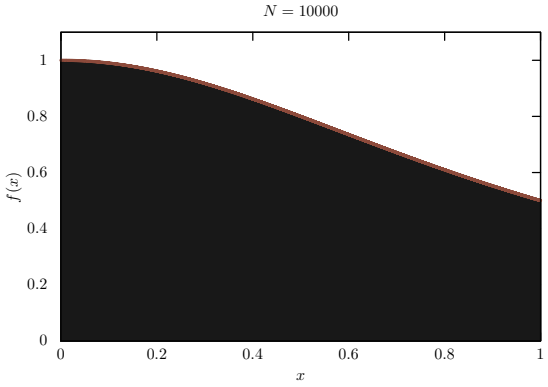


(a) $N = 10$

(b) $N = 100$



(c) $N = 1000$



(d) $N = 10000$

Figura 1: *Plot* da integral da Eq. (4) para diferentes passos n de Monte Carlo.

```
module Ex2_mod
contains
  real function fun(x)
    implicit none
    real, intent(in) :: x
    fun = 1./(1. + x**2)
  end function

  real function weight(x)
    implicit none
    real, intent(in) :: x
    weight = (4. - 2*x)/3.
  end function

  real function xx(y)
    !x as a function of y
    implicit none
    real, intent(in) :: y
    xx = 2. - sqrt(4. - 3*y)
  end function
end module Ex2_mod
```

```
22
23  program Ex2
24  use Ex2_mod
25  implicit none
26  integer :: seed = 987654321
27  real, parameter :: exact = .78540
28  integer :: N, iX
29  real :: sumF, sumF2, fX, f_ave, f2_ave, sigma
30  real :: x, y  !Add weight.
31
32  do
33    print *, 'Enter number of points (0 to stop)'
34    read *, N
35    if ( N == 0 ) STOP
36
37    sumF = 0.   !zero sums
38    sumF2 = 0.
39
40    do iX = 1, n
41      y = ran(seed)
42      x = xx(y)
43      fx = fun(x) / weight(x) !integrand
44
45      sumF = sumF + fX
46      sumF2 = sumF2 + fX**2
47    end do
48
49    f_ave = sumF/N
50    f2_ave = sumF2/N
51
52    sigma = sqrt((f2_ave - f_ave**2) / N)   !error
53
54    print *, 'integral =', f_ave, '+/-', sigma, 'error =', exact - f_ave
55  end do
56  end program Ex2
```

Listing 2: Integral de Monte Carlo com um peso específico w(x).

Escrevemos as funções dentro de um módulo por ser o procedimento mais adequado, pois dessa forma todas as interfaces serão explícitas, ou seja, a função **xx(y)** "conhece"a função **weight(x)**.

2.2 Caso bidimensional

Avançando em complexidade, passando a considerar agora o caso de uma integral bidimensional. A mudança de variável discutida anteriormente também pode ser generalizada para mais dimensões. Para uma função peso $w(x)$ normalizada sob os limites de integração, a então nova variável de integração será y, no qual o Jacobiano fornece $w(x) = |\partial y/\partial x|$. Isso é geralmente muito difícil (se não impossível) de se construir $x(y)$ explicitamente, então é mais conveniente pensar na mudança de variável para o caso multidimensional no sentido de distribuir os pontos $x_i(y_i)$ com uma certa distribuição w.

Exemplo 3. *Como "cavalo de guerra"escolhemos uma das formas de se estimar o valor da constate π, que é através do cálculo da área de um círculo, avaliada da seguinte maneira;*

$$\pi = 4\int_0^1 dx \int_0^1 dy\theta(1 - x^2 - y^2) \ ,$$

onde θ é uma função degrau para o primeiro quadrante de uma unidade de círculo. Neste exemplo vamos trabalhar com a questão de rejeição de dados, ao sortear pontos de coordenadas (x,y). Se um ponto sorteado satisfizer a relação

$$x^2 + y^2 \leqslant 1 \ ,$$

então aceitamos (pois pertencem a área do círculo) ele e caso contrário o rejeitamos. Esta relação está ilustrada na Fig. 2.

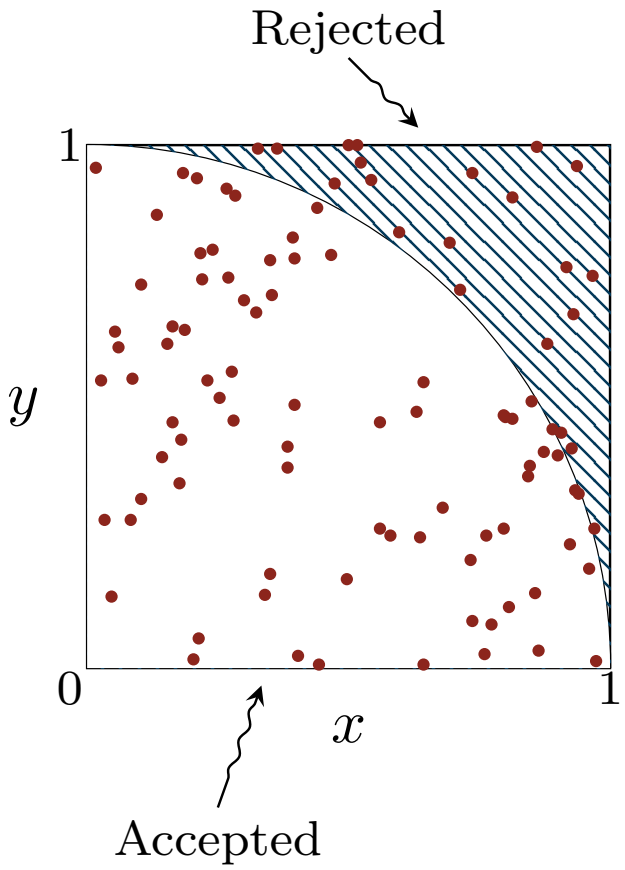


Figura 2: Estimativa do valor de π através do cálculo da área de um arco de circunferência. Os pontos na região hachurada são desprezados, enquanto que os demais serão aceitos.

```
1  program Ex_3
2  implicit none
3  integer :: seed = 3274927 !seed for random number generator
4  real, parameter :: exact = 4*atan(1.)
5  integer :: n, icount, ix
6  real :: x, y, pi4, sigma
7
8  do
```

```
9     print *, 'Enter number of points (0 to stop)'
10    read *, n
11    if ( n == 0 ) stop
12
13    icount = 0          !zero count
14    do ix = 1, n        !loop over samples
15      x = ran(seed)
16      y = ran(seed)
17
18      if ( (x**2 + y**2) <= 1. ) icount = icount + 1
19    end do
20
21    pi4 = real(icount)/n            !pi/4
22    sigma = sqrt(pi4*(1 - pi4)/n)   !error
23
24    print *, 'pi = ', 4*pi4, '+/-', 4*sigma, 'error = ', exact - 4*pi4
25  end do
26  end program Ex_3
```

Listing 3: Integral de Monte Carlo bidimensional: estimativa do valor de π.

O critério da Eq. (3) está presente no *scrpit* sob a condicional da linha 18.

CRIAR UM GRÁFICO DO ERRO SIMULAÇÃO-EXATO ()EIXO Y), POR NÚMERO DE PASSOS (N) (EIXO X), PLOTAR JUNTO A CURVA RAIZ DE N E MOSTRAR QUE NO INFINITO O ERRO DENTE A ISSO[6].

2.3 Geração de números aleatórios

A quadratura com Monte Carlo envolve duas tarefas básicas:

1. Gerar números aleatórios distribuídos sobre um volume de integração com uma distribuição específica $w(x)$;
2. Avaliar a função f/w.

Sobre a primeira tarefa, há diversas formas de gerar números ditos, pseudo-aleatórios, que são gerados através de alguma regra, equação determinística, mas mesmo assim, preservam propriedades estatísticas importantes (e suficientes para resolver nossos problemas) que são inerentes aos números aleatórios reais.

Um dos algoritmos mais comuns é utilizar um gerador linear congruente. Neste método a sequência de números aleatórios são gerados a partir de uma semente (*seed*). Portanto, a mesma semente produzirá sempre a mesma sequência idêntica de números aleatórios, permitindo assim reproduzir resultados obtidos em uma simulação. A qualidade destes números aleatórios gerados é essencial. Em suma, no gerador linear congruente teremos

$$x_{i+1} = (ax_i + c)|m| \ ,$$

onde a, c e m são chamados de "números mágicos".

3 Simulações de Monte Carlo

O termo "simulações"refere-se a duas características marcantes, que é o caráter aleatório e de simular "experimentos virtuais", diferindo-se portanto de "cálculos de campo médio", que são possuem um caráter determinístico.

3.1 O algoritmo *Metropolis*

Os métodos discutidos anteriormente para a geração de números aleatórios sob uma distribuição específica podem ser muito eficientes. No entanto, para o caso multidimensional devemos buscar uma outra abordagem, vamos considerar nesta seção um algoritmo amplamente conhecido, *Metropolis*. O algoritmo *Metropolis* pode ser implementado de diferentes formas, vamos começar descrevendo uma realização simples.

Suponha que desejamos gerar um conjunto de pontos em um espaço de variáveis x distribuídos com uma densidade de probabilidade $w(x)$. O algoritmo gera uma sequência de pontos $x_0, x_1, \cdots x_n$, como aqueles visitados sucessivamente por um caminhante aleatório, movendo-se no espaço x. Então à medida que o caminhada torna-se cada vez mais longa, os pontos que ela conecta se aproximaram da distribuição desejada. Esta caminhada possui as seguintes regras:

- Suponha que o caminhante esteja em um ponto x_n da sequência.
- Para gerar o ponto x_{n+1} ele realiza um passo experimental até um novo ponto x'. Este novo ponto pode ser escolhido de qualquer maneira conveniente, por exemplo, uniformemente aleatório dentro de um cubo multidimensional de lado pequeno δ em torno de x_n.
- Este passo pode ser "aceito"ou "rejeitado"de acordo com a seguinte razão

$$r = \frac{w(x')}{w(x_n)} \ . \tag{5}$$

- Se $r > 1$ o passo será aceito, i.e., $x_{n+1} = x'$;
- No entanto, se $r < 1$, o passo será aceito com probabilidade r. Neste caso, comparamos r com um número aleatório η uniformemente distribuído no intervalo [0,1];
  - Se $\eta < r$ o passo é aceito;
  - Mas se $\eta > r$ rejeitamos o passo, ou seja, $x_{n+1} = x_n$.
- Repetimos este procedimento para x_{n+2} e os demais passos(pontos). Qualquer ponto x_0 pode ser utilizado como ponto de partida para o nosso passeio aleatório.

Na Fig. 3 apresentamos um fluxograma com caráter didático que ilustra o algoritmo *Metropolis*.

Exemplo 4. *Algoritmo Metropolis para um caso com duas variáveis.*

A sub-rotina a seguir ilustra a aplicação do algoritmo Metropolis para amostrar uma distribuição bidimensional nas variáveis x e y. Cada chamada à sub-rotina executa mais um passo do passeio aleatório e retorna os próximos valores de x e y; o programa principal deve inicializar estas variáveis, bem como definir o valor de ***DELTA*** *e definir a distribuição* ***WEIGHT(x,y)****.*

```
1  module metrop
2  implicit none
3  integer :: seed = 39249187
4  contains
5    real function weight(x, y)
6    implicit none
7    real, intent(in) :: x, y
8    weight = x**2 + y**2
9    end function
10
```

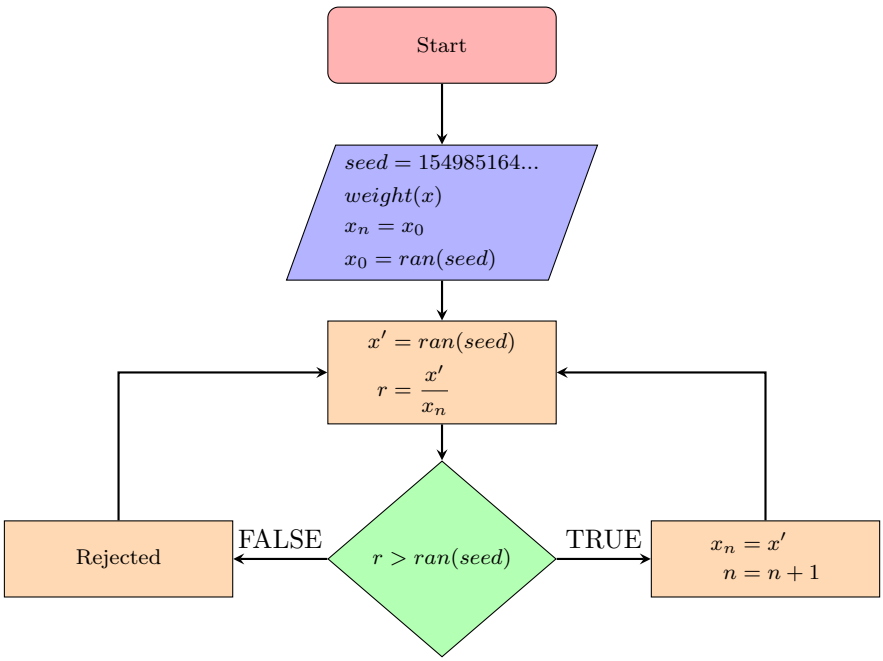


Figura 3: Fluxograma para o algoritmo *Metropolis*. O processo encerra quando $n = n_{max}$.

```
11      subroutine metropolis(x, y, delta)
12      implicit none
13      real :: x, y, delta
14      !local variables:
15      real :: new_x, new_y, ratio
16
17      !take a trial step in square about (x,y)
18      new_x = x + delta*(2*ran(seed) - 1)
19      new_y = y + delta*(2*ran(seed) - 1)
20      ratio = weight(new_x, new_y) / weight(x, y)
21
22      if ( ratio > ran(seed) ) then
23        x = new_x
24        y = new_y
25      end if
26      end subroutine
27    end module metrop
```

Listing 4: Módulo para o algoritimo *Metropolis*.

O cálculo do peso w costuma ser a parte mais demorada no algoritmo *Metropolis* [1].

Para provar que o algoritmo acima é capaz de gerar uma sequência de pontos distribuídos de acordo com o peso w. Consideremos um grande número de caminhantes partindo de diferentes pontos iniciais e movendo-se independentemente através do espaço x. Se $N_n(x)$ for a densidade desses caminhantes em x após n passos, então o número líquido de caminhantes movendo-se do

ponto x para o ponto y na próxima etapa será:

$$\begin{aligned}\Delta N(X) &= N_n(X)P(X \to Y) - N_n P(Y \to X) \\ &= N_n(Y)P(X \to Y)\left[\frac{N_n(X)}{N_n(Y)} - \frac{P(Y \to X)}{P(X \to Y)}\right] .\end{aligned}$$

Aqui, $P(X \to Y)$ é a probabilidade que o caminhante faça a transição de X para Y. O equilíbrio será atingido quando (e após um grande número de passos)

$$\frac{N_n(X)}{N_n(Y)} = \frac{P(Y \to X)}{P(X \to Y)} = \frac{N_e(X)}{N_e(Y)} .$$

A probabilidade de dar um passo de X para Y é

$$P(X \to Y) = T(X \to Y)A(X \to Y) ,$$

onde T é a probabilidade de dar um passo experimental de X para Y e A é a probabilidade de aceitar esse passo. Se a transição puder ser alcançada com um único passo, então

$$T(X \to Y) = T(Y \to X)$$

de modo então que a distribuição de equilíbrio dos caminhantes aleatórios de *Metropolis* satisfaçam

$$\frac{N_e(X)}{N_e(Y)} = \frac{A(Y \to X)}{A(X \to Y)} .$$

Se $w(X) > w(Y)$, então $A(Y \to X) = 1$ e

$$A(X \to Y) = \frac{w(Y)}{w(X)} ,$$

enquanto se $w(X) < w(Y)$ então

$$A(Y \to X) = \frac{w(X)}{w(Y)} ,$$

e $A(X \to X) = 1$. Em ambos os casos, a população de equilíbrio de caminhantes *Metropolis* satisfaz

$$\frac{N_e(X)}{N_e(Y)} = \frac{w(X)}{w(Y)} ,$$

Embora tenhamos tomado x' na vizinhança de x_0, poderíamos usar quaisquer regras de transição e aceitação desde que satisfaçam

$$\frac{w(X)}{w(Y)} = \frac{T(Y \to X)A(X \to Y)}{T(X \to Y)A(Y \to X)}$$

Na verdade, uma escolha limitante é $T(X \to Y) = w(Y)$, independente de X, e $A = 1$. Esta é a escolha mais eficiente, pois nenhuma etapa de teste é "desperdiçada"por rejeição. Contudo, esta escolha é um tanto impraticável, porque se soubéssemos como amostrar w para dar o passo experimental, não precisaríamos usar o algoritmo para começar.

Outra questão pertinente é o tamanho do passo. Se ele for muito grande, provavelmente $x(x') >> w(x)$, e então uma grande parcela dos passos experimentais será rejeitada. No entanto, se o passo for

muito pequeno, a maioria dos passos experimentais será aceito, mas não seremos muito audaciosos e resultando em uma amostragem pobre. Um bom ponto de partida é escolher o passo, tal que metade deles seja aceito.

Um problema no uso deste algoritmo, é que os pontos não são estatisticamente independentes, o próximo passo estará sempre na vizinhança do passo anterior, e portanto deve ser tomado cuidado ao utilizar para avaliar integrais.

Onde iniciar o passeio aleatório (x_0)? A princípio, qualquer valor é válido, pois "no final do dia"o processo irá "termalizar", não dependendo do ponto de partida. Mas podemos começar com um valor mais provável, onde $w(x)$ seja grande.

4 Aplicação: modelo de Ising 2D

4.1 Introdução

Existem diversos modelos nos quais os graus de liberdade residem em um rede e interagem localmente, e o mais simples deles é o modelo de Ising, o qual vamos utilizar para investigar as propriedades termodinâmicas de um sistema magnético.

Em termos magnéticos, o modelo de Ising pode ser descrito como uma abordagem para mapear um conjunto de graus de liberdade de *spins* que interagem entre si, podendo ter sofrer a ação de um campo externo. Estes graus de liberdade podem representar os *momenta* magnéticos dos átomos em um sólido a menos de uma constante de proporcionalidade [5].

Vamos considerar o caso particular bidimensional, onde os sítios (*spins*) são localizados em uma rede quadrada $N_x \times N_y$. Os *spins* podem ser rotulados como $S_{\alpha\beta}$, onde α e β localizam o *spin* na rede quadrada, podendo assumir os valores $S_i = \pm 1$. Dessa forma estaremos "imitando"os estados *up* e *down* do *spin* (que é quântico), embora os consideremos aqui este grau de liberdade como clássico, não impondo as regras de comutação do *momentum* angular que são inerentes em mecânica quântica (se fizéssemos, isso corresponderia ao modelo de Heisenberg).

A hamiltoniana para este sistema pode ser escrita como

$$\mathcal{H} = \sum_{(ij)} S_i S_j - H \sum_i S_i \ ,$$

onde a notação (ij) indica que esta é uma soma especial, e se estende para os primeiros vizinhos, J_{ij} é o termo de troca, medido em unidades de energia, se $J_{ij} > 0$ o anti-alinhamento entre spins é favorecido (antiferromagnético), enquanto que para $J_{ij} < 0$ o alinhamento entre primeiros vizinhos será favorecido (ferromagnético). O termo $H = \mu_B$ representa o termo de troca com o campo externo, e μ_B é o magneton de Bohr. Para o caso clássico (campo longitudinal) podemos escrever $H = h$ e para o campo transverso (caso quântico) $H =$
Gamma. É assumida condições periódicas de contorno, é dito que a rede possui a topologia de um toroide.

Como estamos interessados em obter a termodinâmica do sistema, é conveniente medir a energia em unidade de J e H em unidades de temperatura, de modo que o aquecimento do sistema corresponda à uma diminuição destes acoplamentos.

As configurações do sistema são especificadas fornecendo os valores de todas as variáveis de spin $N_x \times N_y = N_s$ e a ponderação sobre qualquer uma das 2^{N_s} configurações, que no ensamble canônico é

$$w(S) = \frac{e^{-\beta \mathcal{H}(S)}}{Z}$$

e a função de partição escrita como

$$\mathcal{Z}(J,h) = \sum_S e^{-\beta \mathcal{H}(S)}$$

As quantidades termodinâmicas de interesse são a magnetização, que pode ser obtida através da relação

$$M = \frac{\partial \ln \mathcal{Z}}{\partial h} = \sum_S w(S) \sum_\alpha S_\alpha \ ,$$

a suscetibilidade como

$$\chi = \frac{\partial M}{\partial h} = \frac{\partial^2 \ln \mathcal{Z}}{\partial h^2} = \sum_S w(S) \left(\sum_\alpha S_\alpha \right)^2 - M^2$$

a energia interna

$$U = \sum_S w(S) \mathcal{H}(S)$$

e o calor específico a campo constante,

$$C_h = \sum_S w(s) H^2(S) - U^2 \ .$$

No limite de uma rede infinita é possível obter a solução analítica, sendo mais simples na ausência de campo externo [3].

4.2 Configuração inicial

A solução numérica do modelo de Ising é útil tanto para ilustrar as técnicas que foram discutidas, como também para generalizar para hamiltonianas mais complicadas. Devido ao grande número de termos envolvidos para calcular as quantidades termodinâmicas, a realização direta de suas somas está fora de questão. Portanto, será mais eficiente gerar configurações de spin S com probabilidade $w(S)$ utilizando o algoritmo *Metropolis* e então calcular o valor esperado dos observáveis sobre estas configurações.

Exemplo 5. *Construindo a configuração inicial da rede de spins (tabuleiro de spins): Definimos uma rede quadrada de dimensão $N \times N$, percorremos elem. a elem. "jogando um dado", ou seja, sorteamos um número $0 < r < 1$, tal que se $r < 0.5$ então a variável de spin $S = -1$, caso contrário $S = 1$. Abaixo segue o script que gera essa tabela com a configuração inicial dos spins e na Fig. uma representação gráfica da configuração de spins dessa etapa inicial.*

```
1  PROGRAM Examp_5
2    implicit none
3    integer, parameter :: N = 20      !Size lattice
4    integer, dimension(N,N) :: S = 1 !spin variable
5    integer :: seed = 987654321
6    integer :: Nx, Ny                 !indices horizontal and vertical lattice
7    real :: r                         !storage random number
8
9    do Nx = 1, N
10     do Ny = 1, N
11       r = ran(seed)
```

```
12          if ( r < 0.5 ) S(Nx, Ny) = -1
13        end do
14      end do
15    END PROGRAM Examp_5
```

Listing 5: Criando a configuração incial de *spins*.

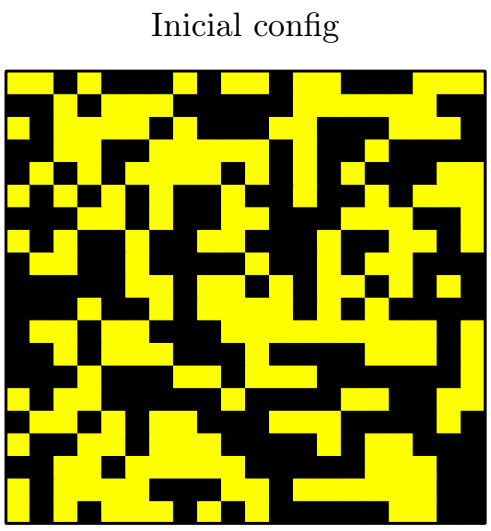


Figura 4: Configuração inicial de *spins*. Quadrados escuros representam os *spins up*, e os amarelos *spins down*.

4.3 O caminhante aleatório

Para implementar o algoritmo *Metropolis*, podemos realizar um passo experimental de S para S' alterando aleatoriamente todos os *spins*. Isto, no entanto levaria a uma configuração muito diferente de S, e portanto haveria uma grande probabilidade de rejeição. Portanto, é melhor dar passos menores, e por isso, consideramos configurações de testes que diferem da anterior apenas pela inversão de um *spin*.

Exemplo 6. *Caminhante aleatório em 1D: Consideremos um caso simples de um caminhante que move-se em uma dimensão, especificamente, ele anda em círculos devido as condições de contorno periódicas que impomos. Isto é, se ele passa da borda, trazemos o caminhante de volta pro início. O código a seguir mostra como podemos definir isso.*

```
1  PROGRAM Examp_6
2    implicit none
3    integer, parameter :: db = 8
4    integer, parameter :: n = 10
5    integer :: seed = 987654321
6    real(kind=db) :: r       !random number
7    real(kind=db) :: sweep   !sorteio/varredura do próximo passo
8    real(kind=db) :: delta   !tamanho do passo.
```

```
  integer :: x              !posição
  integer :: step           !próximo passo do caminhante

  delta = 1

  !Iniciamos a caminhada: sorteamos uma fração do espaço disponível 0 < x < n.
  x = int(N*ran(seed))
  print *, x

  do
    !sorteamos um número 0 < r < 1
    r = ran(seed)
    sweep = (2*r - 1)

    !Normalizamos para o tamanho do passo e pegamos a direção do passo
    step = int(sign(delta, sweep))

    x = x + step

    !Condições periódicas de contorno.
    if ( x >  n ) then
      !passou da borda superior, deve retornar pro início
      x = x - n
    end if
    if (x < 1 ) then
      !passou da borda inferior.
      x = x + n
    end if

    print *, x

    read(*,*)
  end do
END PROGRAM Examp_6
```

Listing 6: Caminhate aleatório em uma rede unidimensional.

Podemos facilmente estender nosso caminhante aleatório para duas dimensões e ao impor as condições periódicas de contorno, obtemos uma simetria toroidal.

Exemplo 7. *Consideramos agora o caminhante aleatório em uma rede de spins 2D (com condições de contorno) com uma configuração aleatória de spins. Consideramos primeiro um caminhante que se move como o rei do tabuleiro de xadrez. E depois mostramos um outro caso, onde o caminhante pula para um ponto aleatório da rede.*

```
PROGRAM Examp_7
implicit none
integer, parameter :: db = 8
integer, parameter :: n = 4
integer :: seed = 987654321
real(kind=db) :: delta       !tamanho do passo.
integer :: x, y              !posição
integer, dimension(n,n) :: S !variável de spin

  !Carrega a config. inicial
  call base(seed, n, S)

  delta = 1

```

```
  do
    !Iniamos a caminha num ponto qualquer da rede.
    x = int(n*ran(seed))
    y = int(n*ran(seed))

    !impedindo uma coord. nula
    if (x > 0 .AND. y > 0) exit
  end do

  ! Devemos escolher uma das duas formas para fazer a caminhada:

  !Passo do rei, em todas as direções, por uma casa apenas.
  call king_walk(seed, n, delta, x, y)

  !Pula para um ponto aleatórios da rede.
  !call crazy_walk(seed, n, x, y)
END PROGRAM Examp_7

subroutine king_walk(seed, N, delta, x, y)
  implicit none
  ! I/O variables:
  integer :: seed
  integer, intent(in) :: n
  real(kind=8), intent(in) :: delta
  integer, intent(inout) :: x, y
  ! Local variables:
  real, parameter :: pi = 4*atan(1.)
  real(kind=8) :: r

  r = ran(seed)

  !nint faz a conversão de forma adequada. nint(0.6) = 1
  x = x + int(delta*nint(cos(r*pi)))
  y = y + int(delta*nint(sin(r*pi)))

  !Condições de contorno:
  if ( x > n ) x = x - n
  if ( x < 1 ) x = x + n
  if ( y > n ) y = y - n
  if ( y < 1 ) y = y + n
end subroutine
```

Listing 7: Caminhate aleatório em uma rede bidimensional. Um passo de cada vez.

```
subroutine crazy_walk(seed, size_lattice, x, y)
  implicit none
  ! I/O variables:
  integer :: seed
  integer, intent(in) :: size_lattice
  integer, intent(inout) :: x, y

  do
    x = int( size_lattice*ran(seed) )
    y = int( size_lattice*ran(seed) )

    !Coord. nula não está definida.
    if (x > 0 .AND. y > 0) return
  end do
end subroutine
```

Listing 8: Caminhate aleatório que pode salta para pontos leatórios da rede.

4.4 O algoritmo *Metropolis* para uma rede Ising

Conseguimos construir o algoritmo do caminhante aleatório. Devemos agora atribuir uma função a este cainhante; que é definir se flipa o spin ou não. Isso será feito consideramos duas configurações, S e S', diferindo apenas pela inversão de um *spin*, $S_i = S_{\alpha\beta}$. A aprovação desta etapa experimental depende da proporção das funções de peso,

$$r = \frac{w(S')}{w(S)} = e^{-\beta[\mathcal{H}(S')-\mathcal{H}(S)]} \ .$$

Especialmente, se $r > 1$ o spin é invertido[$\Delta\mathcal{H} = \mathcal{H}(S') - \mathcal{H}(S) < 0$, ou seja, essa nova configuração leva a um estado de menor energia]. Caso $r < 1$, mas maior que um certo número aleatório uniformemente distribuído entre 0 e 1, então o *spin* é invertido (esta etapa pode ser interpretada como a introdução do conceito de entropia, i.e., permitindo com probabilidade que cresce com a temperatura que o sistema acesse configurações que levam um aumento da energia, ou seja, estados mais desordenados); caso contrário, nada é feito. Os termos envolvendo $S_{\alpha\beta}$ serão os mais relevantes em r. Podemos reescrever r como:

$$r = e^{-2\beta S_{\alpha\beta}(Jf+h)} \ ; \quad f = S_{\alpha+1\beta} + S_{\alpha-1\beta} + S_{\alpha\beta+1} + S_{\alpha\beta-1} \ .$$

Aqui f é a soma dos quatro *spins* primeiros vizinhos que pode ser invertido. Como f pode assumir apenas 5 valores distintos, 0, ± 2, ± 4, consequentemente, apenas 10 valores de r, pois ($S_{\alpha\beta} = S_i = \pm 1$); os quais podemos calcular antecipadamente e armazená-los em uma tabela, evitando assim de calcular repetidamente as exponenciais.

Exemplo 8. *No script a seguir realizamos uma simulação de Monte Carlo do modelo de Ising utilizando o algoritmo Metropolis. Iniciamos com uma configuração aleatória de spins para iniciar o passeio aleatório de Metropolis.*

```
1  PROGRAM Examp_8
2    implicit none
3    integer, parameter :: db = 8
4    integer, parameter :: n = 20
5    integer :: seed = 987654321
6    integer, parameter :: B = 0       !Campo mag. externo
7    integer, parameter :: sweep = 10000  !Varreduras
8    real(kind=db), parameter :: delta = 1.d0     !tamanho do passo.
9    real(kind=db), parameter :: T = 1.d0         !temperatura
10   real(kind=db), parameter :: J = 1            !Termo de troca
11   integer, dimension(n,n) :: S !variável de spin
12   integer :: x, y              !posição
13   integer :: i                 !indice das varreduras
14   integer :: NN_sum            !soma dos primeiros vizinhos: Si-1,j +...+Sij+1.
15   real(kind=db), dimension(-4:4, -1:1) :: ratio !razão entre o estado exp. e o antigo
16   integer :: accept   !Passos aceitos
17
18   !Carrega a config. inicial.
19   call base(seed, n, S)
20
21   !Visitando o primeiro estado
22   call first_step(seed, n, x, y)
23
24   !Definimos a hamiltoniana do modelo e a razão r=w(S')/w(S)
25   call hamiltonian(J, B, T, ratio)
26
```

```
  accept = 0

  do i = 1, sweep
    !Realizamos um passo experimental
    call king_walk(seed, n,delta, x, y)

    !Somamos as contribuições dos primeiros vizinhos:
    !nn_sum = Si-1j + Si+1j + Sij-1 + Sij+1
    call sum_neighbors(n, S, x, y, NN_sum)

    !Algoritmo Metropolis
    call metropolis(seed, NN_sum, ratio, S(x,y), accept)
  end do
END PROGRAM Examp_8

subroutine hamiltonian(J, B, T, ratio)
  implicit none
  integer, parameter :: db = 8
  ! I/O variables:
  real(kind=db), intent(in) :: J, T
  integer, intent(in) :: B
  real(kind=db), dimension(-4:4, -1:1), intent(out) :: ratio
  ! Local variables:
  real(kind=db) :: beta
  integer :: i

  beta = 1.d0 / T

  ! A razão 'ratio' neste caso pode assumir apenas os valores:
  !0, +/-2 e +/- 4.
  do i = -4, 4, 2
    ratio(i, -1) = exp(2*beta*(J*i - B))    !Spin down
    ratio(i, +1) = 1.d0 / ratio(i, -1)      !Spin up
  end do
end subroutine

subroutine sum_neighbors(size_lattice, S, Sx, Sy, NN_sum)
  implicit none
  ! I/O variables:
  integer, intent(in) :: size_lattice !Dimensão da rede
  integer, intent(in) :: Sx, Sy !Indice horizontal e vertical do sítio visitado
  integer, dimension(size_lattice, size_lattice), intent(in) :: S
  integer, intent(out) :: NN_sum
  ! Local variables:
  integer :: i, j !indice
  integer :: x, y !indice dos primeiros vizinhos
  integer, external :: Ccontour

  NN_sum = 0

  do i = -1, 1, 2
    do j = -1, 1, 2
      x = Sx + i
      y = Sy + j

      !Condições de contorno
      x = Ccontour(x , size_lattice)
      y = Ccontour(y , size_lattice)

      NN_sum = NN_sum + s(x,y)
```

```
87      end do
88    end do
89  end subroutine
90
91  subroutine metropolis(seed, NN_sum, ratio, spin, accept)
92    implicit none
93    integer, parameter :: db = 8
94    ! I/O variables:
95    integer :: seed
96    integer, intent(in) :: nn_sum
97    real(kind=db), dimension(-4:4,-1:1), intent(in) :: ratio
98    integer :: spin
99    integer :: accept
100
101   ! Através da variável nn_sum acessamos o valor da exponencial para
102   !o spin = S(x,y).
103   if ( ran(seed) < ratio(nn_sum, spin) ) then
104     !Flip the spin
105     spin = -spin
106
107     !update accept count
108     accept = accept + 1
109   end if
110 end subroutine
```

Listing 9: Algoritmo *Metropolis* para uma rede de Ising 2D.

A partir deste código geramos a sequência de *plots* a seguir, onde variamos o tamanho da rede ($L = 4$ e $L = 20$) e a temperatura ($T = 1$ e $T = 5$).

Começamos com a configuração inicial ($i = 1$) à $T = 1$ na Fig. 5(a) que sabemos de antemão corresponder a uma fase ordenada. Avançamos então com 10 passos *Metrópolis*, Fig. 5(b), onde o sistema começa a exibir um padrão. Na Fig. 5(c) aumentamos a quantidades de passo para 100, e encontrados todos os spins alinhados com valores $S_i = -1$. Então exploramos essa quantidade de passos para uma temperatura onde a fase é desordenada (para simulações de MC, cálculos de campo médio indicam que a fase ainda é ordenada) que é $T = 3$ e notamos que a ordem que anteriormente foi encontrada, agora foi destruída por flutuações térmicas.

Avançamos um pouco em nossa análise, e consideramos uma rede de dimensões mais realísticas para simulações. Neste caso, consideremos uma rede de 20×20 spins.

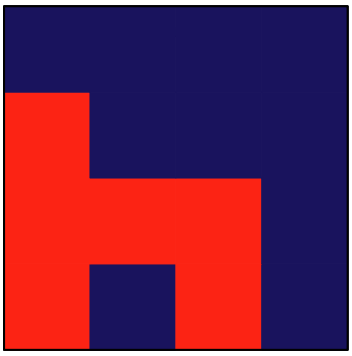
Podemos nota na Fig. 6(a) que os $n = 100$ passos que antes foram suficiente, agora já não são para uma rede de maior dimensão. Então na Fig. 6(b) aumentamos o número de passos *Metrópolis* para $n = 1000$ e notamos que começa a surgir um padrão de ordenamento. Seguimos então, e aumentamos para $n = 10$ mil passos na Fig. 6(c), que tem a mesma cara da Fig. 5(c), onde havíamos utilizado 100 passos apenas.

Então surge a pergunta: como determinar o número de passos onde o sistema vai *termalizar*? Nos casos abordados nas Fig. 5 e 6 nós já sabíamos de antemão qual deveria ser mais ou menos a resposta. Mas e para um sistema desconhecido, em que estamos de fato investigando e não sabemos a resposta? Devemos então definir um critério para isso (ou mais)!

4.5 Critérios de convergência

Existem diversas formas para determinar quando a simulação estabilizou, no entanto não há uma regra geral. Podemos olhar para algumas quantidades termodinâmicas (calcular médias) e analisar sua estabilidade.

Metropolis: $L = 4, T = 1, i = 1$

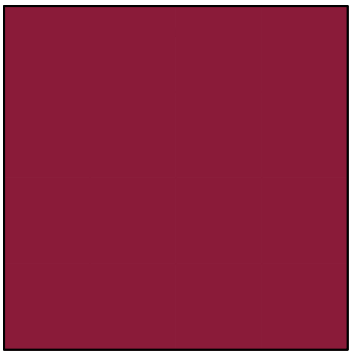


(a) Config. inicial:$T = 1$.

Metropolis: $L = 4, T = 1, i = 10$

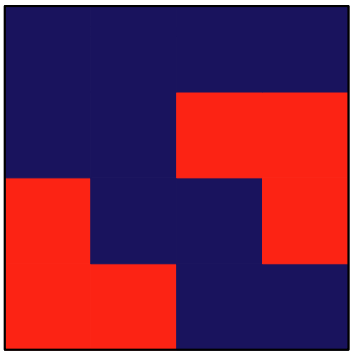
(b) Config. 10 passos:$T = 1$.

Metropolis: $L = 4, T = 1, i = 100$



(c) Config. 100 passos:$T = 1$.

Metropolis: $L = 4, T = 3, i = 100$



(d) Config. 100 passos:$T = 5$.

Figura 5: Diferentes configurações para uma rede de 4×4 spins à uma temperatura T. Quadrados vermelhos e azuis representam os estados de spin $S = -1$ e $S = 1$, respectivamente.

Como estamos interessados em obter a termodinâmica do sistema, é conveniente medir a energia em unidade de J e H em unidades de temperatura, de modo que o aquecimento do sistema corresponda à uma diminuição destes acoplamentos.

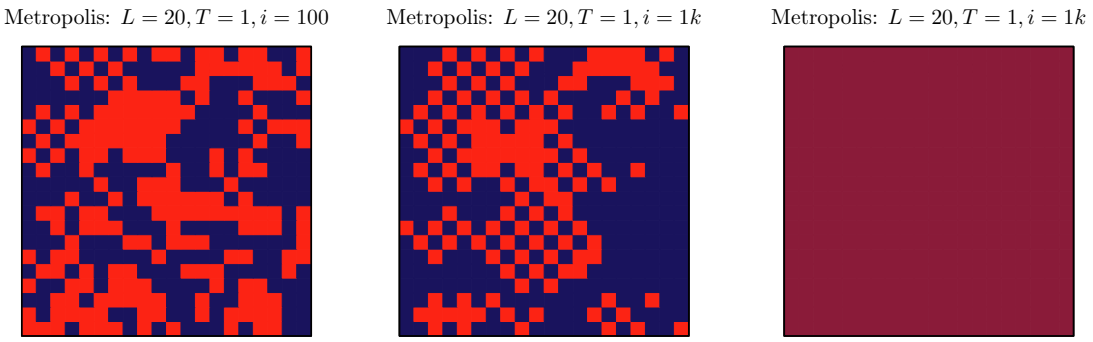
As configurações do sistema são especificadas fornecendo os valores de todas as variáveis de spin $N_x \times N_y = N_s$ e a ponderação sobre qualquer uma das 2^{N_s} configurações, que no ensamble canônico é

$$w(S) = \frac{e^{-\beta \mathcal{H}(S)}}{Z}$$

e a função de partição escrita como

$$\mathcal{Z}(J, h) = \sum_S e^{-\beta \mathcal{H}(S)}$$

As quantidades termodinâmicas de interesse são a magnetização, que pode ser obtida através da



(a) Config. 100 passos:$T = 1$. (b) Config. 1000 passos:$T = 1$. (c) Config. 10000 passos:$T = 1$.

Figura 6: Configurações de spins em uma rede de 20×20 spins à uma temperatura T. Quadrados vermelhos e azuis representam os estados de spin $S = -1$ e $S = 1$, respectivamente.

relação

$$M = \frac{\partial \ln \mathcal{Z}}{\partial h} = \sum_S w(S) \sum_\alpha S_\alpha \ ,$$

a suscetibilidade (flutuação) como

$$\chi = \frac{\partial M}{\partial h} = \frac{\partial^2 \ln \mathcal{Z}}{\partial h^2} = \sum_S w(S) \left(\sum_\alpha S_\alpha \right)^2 - M^2 \ .$$

a energia interna

$$U = \sum_S w(S) \mathcal{H}(S)$$

e o calor específico a campo constante,

$$C_h = \sum_S w(s) H^2(S) - U^2 \ .$$

No limite de uma rede infinita é possível obter a solução analítica, sendo mais simples na ausência de campo externo [3].

A rede é mostrada após cada varredura **NFREQ**, se solicitado. Varreduras de termalização (sem cálculo dos observáveis) são permitidas; estes permitem que o passeio aleatório "se estabilize"antes que os observáveis sejam calculados. Os valores de energia, magnetização, suscetibilidade e calor específico por *spin* são exibidos à medida que o cálculo avança, assim como a fração das etapas experimentais aceitas.

Uma característica deste programa requer mais explicações. Esta é uma técnica simples usada para monitorar as correlações varredura a varredura nos observáveis inerentes ao algoritmo *Metropolis*. Os observáveis básicos (energia e magnetização) são calculados a cada varredura **NFREQ**. Esses valores são então agrupados em "grupos" com membros **NSIZE**. Para cada grupo são calculadas as médias e desvios padrão da energia e da magnetização. À medida que mais grupos são gerados, suas médias são combinadas em uma média geral. Uma maneira de calcular a incerteza nesta média geral é tratar as médias do grupo como medidas independentes e usar a Eq. (2). Alternativamente,

a incerteza pode ser obtida calculando a média dos desvios padrão dos grupos em quadratura. Se a frequência de amostragem for suficientemente frouxa, estas duas estimativas concordarão. Contudo, se a frequência de amostragem for muito pequena e houver correlações significativas nas medições sucessivas, os valores dentro de cada grupo serão distribuídos de forma muito estreita e a segunda estimativa da incerteza será consideravelmente menor que a primeira. Estas duas estimativas da incerteza para a média geral da energia e magnetização por spin são, portanto, exibidas. Observe que esta técnica não é tão facilmente implementada para o calor específico e a suscetibilidade, pois eles próprios são flutuações na energia e na magnetização, e assim apenas as incertezas em suas grandes médias calculadas pelo primeiro método são exibidas.

Algoritmo: Este programa simula o modelo bidimensional de Ising usando o algoritmo de *Metropolis*. Antes de iniciar os cálculos, a taxa de aceitação r é calculada para os 10 casos diferentes (sub-rotina **PARAM**), e a configuração inicial é escolhida aleatoriamente (**DO loop 5** em sub-rotina **ARCHON**). Após a realização das varreduras de termalização (**loop 10**), inicia-se a coleta de dados para os grupos (**loop 20**). As duas sub-rotinas básicas são **METROP**, que realiza uma varredura *Metropolis* da rede, e **SUM**, que calcula a energia e a magnetização para a configuração da rede a cada varredura **NFREQ**. Durante a obtenção de dados, as somas totais e de grupo para a energia e a magnetização são acumuladas (sub-rotina **AVERAG**) para que os efeitos das correlações de varredura a varredura possam ser monitorados, conforme discutido imediatamente acima do Exercício 8.6. Após o número solicitado de grupos ter sido calculado, você será solicitado a informar o número de grupos adicionais a serem realizados [10.].

Entrada: Os parâmetros físicos são o campo magnético [0.] e a força de interação [.3], ambos em unidades de k_BT. Os parâmetros numéricos são o número de pontos da rede em x [20] e y [20], a semente de número aleatório [54767], o número de varreduras de termalização [20], a frequência de amostragem [5], o tamanho do grupo [10] e o número de grupos [10]. O número máximo de x pontos é fixado em **MAXX= 79**; e pontos y, por **MAXY = 20**. Esses limites são para garantir que a exibição de caracteres da configuração caiba na tela típica, que tem 24 linhas de comprimento e 80 caracteres de largura. Se o seu terminal não for 24 X 80, ou se você não estiver exibindo a configuração, você pode ajustar estes parâmetros. O parâmetro de saída de texto permite escolher a versão resumida da saída para a qual as somas de amostra não são exibidas. Consulte a nota em C.7 sobre a semente de números aleatórios e o armazenamento de números inteiros no compilador.

Saída: A cada varredura do **NFREQ**, a saída de texto exibe o índice do grupo, o índice da amostra, o tamanho do grupo, a taxa de aceitação, a energia e a magnetização. (Todas as grandezas termodinâmicas são por *spin*.) Se a saída curta for solicitada, esta saída não será impressa. A cada varredura **NFREQ*NSIZE** (quando um grupo é concluído), a saída de texto exibe o índice do grupo, as médias do grupo para energia, magnetização, suscetibilidade e calor específico, bem como a incerteza em energia e magnetização. Além disso, neste momento as grandes médias (incluindo todos os grupos calculados até agora) são exibidas com duas incertezas para energia e magnetização, conforme discutido no texto.

Se você solicitar gráficos para a tela ou arquivo, a cada varredura do NFREQ a configuração será exibida ou impressa usando caracteres: o *spin* para cima é indicado por um **X** e o *spin* para baixo é indicado por um espaço em branco. Não há saída gráfica mais sofisticada disponível, pois ela não pode dizer mais do que isso.

Referências

[1] Steven E. Koonin and Dawn C. Meredith. *Computational Physics-Fortran Version*. 1998.

[2] Nicholas Metropolis and S. Ulam. The monte carlo method. *Journal of the American Statistical Association*, 44(247):97–111, 1949.

[3] Lars Onsager. A two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65, 1944.

[4] José Pedro Rino and Bismarck Vaz da Costa. *ABC da Simulação Computacional*. 2013.

[5] J. J. Sakurai and Jim Napolitano. *Mecânica Quântica Moderna*. 2013.

[6] Anders W. Sandvik. Numerical integration and monte carlo integration. PY 502, Computational Physics, Fall 2023, 2023. Department of Physics, Boston University.