Física Estatística e Computacional: práticas computacionais

10 de Junho de 2009

1 Ambiente de trabalho

Existem vários ambientes de trabalho para o `Python.` O que foi usado para preparar estas sessões é descrito nesta secção e é apropriado para sessões interactivas.

Shell Foi usado o `ipyhton,` iniciado com a seguinte linha de comando:

```
$ ipython -pylab -p scipy
```

```
In [1]:
```

O `ipython` tem várias funcionalidades úteis—mantém história de comando anteriores a que se acede com as setas teclas ↑ e ↓, completa comandos com a tecla `tab`— e além disso, iniciado desta maneira, carrega um conjunto de módulos do `scipy (incluíndo o numpy) e` inicializa a bilioteca gráfica `matplotlib,` resolvendo questões de compatibilidade entre a linha de comando e as janelas onde surgem os gráficos da `matplotlib.` Com esta inicialização, uma chamada a uma função da `matplotlib` abre uma janela adicional, com o gráfico respectivo (ver fig. 1).

Editor O modo interactivo do `ipython` é muito útil para construir programas, mas está limitado a edição por linhas. Em qualquer trabalho sério é necessário manter o código em ficheiros editáveis. Uma maneira simples de trabalhar é manter o editor aberto (por exemplo: `emacs`) ; sempre que queremos executar o ficheiro que editámos basta escrever na shell do `ipython`

```
In [1]:  %run nome_ficheiro.py
```

O código correspondente é executado na sessão do `ipython.` Nota: o ficheiro deve estar na directoria onde foi iniciado o `ipython.`

2 Gerador de números (pseudo) aleatórios `do numpy`

A biblioteca `numpy`, incluída no `scipy,` tem um conjunto de funções associadas à geração de números pseudo-aleatórios (n.p.a.) com densidades de probabilidade (discretas e contínuas) muito variadas. Estas funções encontam-se no módulo `random (ver` listagem `1).`

Para obter mais informações sobre este módulo consultar `http://www.scipy.org/Numpy_Functions_by_Category` onde a documentação sobre o módulo está ligada com exemplos. A instrução `help(random)` gera uma lista das distribuições disponíveis, com a forma de chamada respectiva.

```
plot(arange(0,6,.1),exp(-arange(0,6,.1)))
```

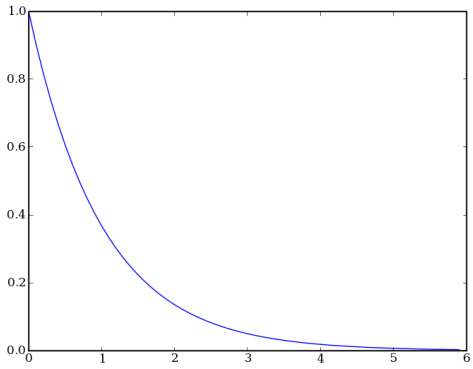

Figura 1: Exemplo de gráfico da `matplotlib`.

Algorithm 1 Função `rand()` gera n.p.a. com distribuição uniforme em $[0, 1)$.

```
In [1]: from numpy import *
In [2]: random.rand(10)
Out[2]:
array([ 0.8648737 ,  0.13031387,  0.15998715,  0.85403508,
0.95941483,         0.78582497,  0.27344148,  0.79753292,
0.57451419,  0.10908469])
```

2.1 Testes

2.1.1 Histograma

Dada uma série de dados $x_1, \ldots x_N$, para obter um *histograma* organizamos os dados por categorias e *contamos* o número de entradas por categoria. Para variáveis reais as categorias são em geral intervalos (*bins)* $[a_i, a_{i+1})$. A variável que conta as entradas no bin i de um histograma pode ser escrita na forma

$$X_i = \xi_1 + \ldots + \xi_N \tag{1}$$

em que $\xi_r = 1$ se $x_r \in [a_i, a_{i+1})$ (bin i) e $\xi_r = 0$ se $x_r \notin [a_i, a_{i+1})$. Desigando por p_i a probabilidade de ξ ser 1,

$$\langle X_i \rangle = N\langle \xi \rangle = Np_i \tag{2}$$

$$\Delta X_i^2 = N\Delta\xi^2 = Np_i(1-p_i) \tag{3}$$

Pelo teorema do limite central, a variável X_i tem uma distribuição gaussiana no limite $N \to \infty$. Podemos também definir histogramas *normalizados* em que as entradas são $h_i = X_i/N$ ($\sum_i h_i = 1$).

$$\langle h_i \rangle = \langle \xi \rangle = p_i \tag{4}$$

$$\Delta h_i^2 = \frac{1}{N}\Delta\xi^2 = \frac{p_i(1-p_i)}{N} \tag{5}$$

Actividade 1: histogramas de sequências de n.p.a Um histograma de uma sequência de n.p.a. permite uma inspecção visual da densidade de probabilidade com que os valores estão a ser gerados.

- Escrever uma função para fazer um histograma de uma série de dados com especificação dos valores limites e número de caixas (*bins).*
- Teste a função, calculando as entradas de um histograma de uma série 10000 n.p.a. gerados com a função `random.rand()` (sugerem-se 50 bins).Para uma distribuição gaussiana, é de esperar cerca de 1/3 da vezes desvios da média superiores a um desvio padrão.
- Usando a função `hist` do `matplotlib` (ver fig. 2), represente um histograma da série de dados $x_i, \ldots, x_N$ e das entradas do histograma $h_i, \ldots h_{nbins}$. Note que as entradas de um histograma devem ter uma distribuição gaussiana. No caso presente, as entradas dos diferentes *bins* têm o mesmo valor médio e variância .

Notas:

- É um erro comum calcular um histograma usando condições `if;` é muito mais eficiente usar o valor da varíável x para determinar o *índice* r da entrada h_r do histograma que deve ser incrementada.

```
In [1]: dados= random.rand(1000)
In [2]: pylab.hist(dados,10,1)  #third argument 1 for normalized histogram
```

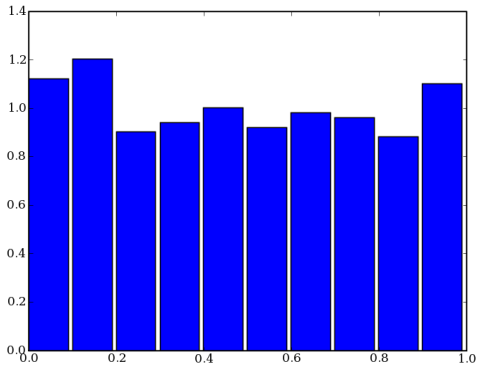

Figura 2: Exemplo da função `hist`

Actividade 2: momentos Para uma distribuição uniforme no intervalo $[0, 1)$ os momentos são (prove-o!)

$$\langle x^k \rangle = \frac{1}{k+1} \tag{6}$$

e a respectiva variância

$$\Delta(x^k)^2 = \langle x^{2k} \rangle - \langle x^k \rangle^2 = \frac{k^2}{(k+1)^2(2k+1)} \tag{7}$$

- Use o gerador `random.rand()` para gerar uma amostra com $N = 1000$ n.p.a ; calcule a médias de x^k da amostra e compare graficamente com as previsões aqui indicadas (a `matplotlib` dispõe de uma função `errorbar(x,y,dy)`, que permite representar a lista y em função da lista x com barras de erro dadas pela terceira lista dy: investigue esta função).
- Experimente calcular os momentos com k diferentes com uma só sequência de n.p.a ou com sequências "frescas" para cada k. Que diferença nota nos gráficos?

Algorithm 2 Removendo o primeiro ou último elemento de uma lista

```
In [6]: a=range(5)
In [7]: a[1:]
Out[7]: [1, 2, 3, 4]
In [8]: a[:-1]
Out[8]: [0, 1, 2, 3]
```

Actividade 3: Scatter plot Um modo visual de procurar correlações entre valores de uma sequência de n.p.a $x_1, \ldots, x_N$ consiste em representar gráficos de dispersão de pares (x_i, x_{i+1}),$i = 1, \ldots, N-1$ ou triplos (x_i, x_{i+1}, x_{i+2}), $i = 1, \ldots N-2$. Gere uma sequência de $N = 1000$ n.p.a e faça uma representação gráfica dos pontos (x_i, x_{i+1}), $i = 1, \ldots, N-1$.

Nota: investige a função `scatter` do `matplotlib`; confira o código da listagem `2`.

3 Amostragem (*sampling*)

3.1 Conceito

3.1.1 Caso Discreto

Dado um conjunto de eventos $\{e_1, \ldots, e_n\} = \{\mathbf{e}\}$, com probabilidades $\{p_1, \ldots, p_n\}$, pretendemos gerar uma sequência, sem correlações, em que cada evento e_i ocorra com a respectiva probabilidade p_i.

3.1.2 Caso contínuo

Dada uma variável aleatória com densidade de probabilidade $p(x)$ pretendemos gerar uma sequência de valores $x_1, x_2, \ldots$cuja densidade de probabilidade seja $p(x)$, e em que os valores sejam independentes (densidade de probabilidade de uma sequência determinada= $p(x_1)p(x_2)\ldots.$)

3.2 Método de inversão

Admitimos que dispomos de um gerador de números (pseudo)aleatórios que gera r com distribuição de probabilidade uniforme (d.p.u.) entre $[0, 1)$.

Exemplos:

- `random.rand()`, intervalo $[0, 1)$;
- `random.randint(0,n)` ; inteiros $0, 1, \ldots, n-1$ com igual probabilidade.

Distribuição de probabilidade cumulativa:

- $c_i = \sum_{j \leq i} p_j \Rightarrow p_i = c_i - c_{i-1} \quad (c_0 \equiv 0,\ c_n = 1)$;
- geramos r número aleatório com d.p.u. em $[0, 1)$ e, se $c_{i-1} \leq r \leq c_i$, seleccionamos e_i. A probabilidade de r ocorrer neste intervalo é $c_i - c_{i-1} = p_i$.
- O método é inconveniente se em cada passo de uma simulação os p_i variarem (necessário recalcular os c_i, $i = 1, \ldots, n$).

3.2.1 Método rejeição de Von Neumann (proposta/aceitação-recusa)

O método:

- proposta: escolher uniformemente (probabilidade $1/n$) um evento e_i;
- Aceitação-recusa: gerar n.p.a. r com d.p.u em $[0, 1)$;
  - Se $r \leq p_i$, aceitar e_i;
  - Se $r > p_i$, recusar e_i; nova proposta.

Sequência de eventos aceites tem a distribuição de probabilidade correcta.

Notas:

- O segundo passo é um exemplo de método de inversão, com dois eventos apenas:
  - e_i: probabilidade p_i;
  - recusa: probabilidade $1 - p_i$.

  As probabilidades cumulativas, são $c_0 = 0$, $c_1 = p_i$, $c_2 = 1$.
- Método gera cadeia de $n + 1$ eventos possíveis, $e_1, \ldots, e_n, R$, $R =$`recusa.` Na cadeia completa, com recusas, as probabilidades dos eventos são:

$$p_c(e_i) = \frac{1}{n} \times p_i \quad \text{(prob. proposta} \times \text{prob. aceitação)} \tag{8}$$

  o que dá

$$\sum_{i=1}^{n} p_c(e_i) = \frac{1}{n}. \tag{9}$$

  $1/n$ é a probabilidade de uma proposta ser aceite. A probabilidade de recusa é, $p_c(R) = 1 - 1/n$, que para n elevado ≈ 1. Método muito ineficiente. A maior parte dos n.p.a. gerados não dão origem a eventos da cadeia de eventos aceites.
- Ao restringirmos apenas aos eventos aceites, temos probabilidades condicionadas (a probabilidade na cadeia dos eventos aceites é proporcional ao número de vezes que um evento ocorre a dividir pelo número total de aceitações):

$$p(e_i|\text{aceite}) = \frac{p_c(e_i)}{p(\text{aceitação})} = \frac{p_i/n}{1/n} = p_i$$

- No segundo passo podemos substituir $p_i \rightarrow f_i$ em que $f_i = \lambda p_i$ desde que $0 \leq f_i \leq 1$ ($\lambda \leq \max\{p_i\}$). Por normalização

$$p_i = \frac{f_i}{\sum_i f_i} \tag{10}$$

  Se aceitarmos no segundo passo com f_i em vez de p_i, na cadeia sem recusas temos probabilidades

$$p(e_i|\text{aceite}) = \frac{p_c(e_i)}{p(\text{aceitação})} \tag{11}$$

$$= \frac{f_i \times 1/n}{\sum_i f_i \times 1/n} = \frac{f_i}{\sum_i f_i} = p_i \tag{12}$$

Note-se que mesmo assim,

$$p(\text{recusa}) = 1 - \frac{1}{n}\sum_i f_i = 1 - \frac{\lambda}{n}. \tag{13}$$

Se as probabilidades p_i forem da mesma ordem de grandeza $p_i \sim O(1/n)$ podemos ter $\lambda \sim O(n)$ e $p(\text{aceitação}) \sim O(1)$. Se uma das probabilidades $p_i \sim O(1)$ dominar, $p(\text{aceitação}) \sim O(1/n)$.

3.2.2 Generalização do método proposta aceitação/recusa: método misto

Proposta não tem que ser feita uniformemente; pode ser feita com probabilidades q_i (por exemplo, método de inversão) e a aceitação/recusa com f_i. Neste caso para a cadeia com recusas incluídas:

$$p_c(e_i) = f_i \times q_i. \tag{14}$$

O eventos aceites têm probabilidades de ocorrência:

$$p(e_i|\text{aceite}) = \frac{f_i q_i}{\sum f_i q_i} \equiv p_i. \tag{15}$$

Este método é conhecido como método misto quando usa o método da dist. prob. cumulativa no primeiro passo.

Actividade 4: passeio aleatório em 2D Gere um passeio aleatório discreto, numa rede quadrada, com saltos apenas para primeiros vizinhos, com probabilidades $p_1, p_{2,}p_3$ e p_4, não necessáriamente iguais, usando o método directo e o método de rejeição de Von-Neumann. Conte quantos n.p.a. gera em média por passo pelos dois métodos para diferentes valores das probabilidades. Represente graficamente algumas das trajectórias, e veja o efeito do enviesamento (*bias*) das probabilidades.

4 Distribuições contínuas

4.1 Método de inversão

x, variável aleatória (v.a.) com densidade de probabilidade (d.p.) $p(x)$, no intervalo $[a, b]$. Como gerar um sequência x_1, x_2 que constitua uma amostragem desta densidade?

- Distribuição de probabilidade cumulativa (d.p.c.)

$$c(s) = \int_a^s p(x)dx = (\text{prob. de } x < s) \tag{16}$$

- $c(a) = 0$; $c(b) = 1$ e , como $p(x) \geq 0$, $c(x)$ é monótona crescente:

$$c(s + ds) - c(s) = p(s)ds \Rightarrow p(s) = \frac{dc(s)}{ds}. \tag{17}$$

Seja $y \equiv c(x)$; qual é a densidade de probabilidade da v.a. y?

y tem valores no intervalo $[0, 1]$; probabilidade de $x \leq s$ é $c(s)$; Ora, se $x \leq s$, $y \leq c(s)$ pois $c(s)$ é monótona crescente. Por isso a probabilidade de $y \leq c(s)$ é $c(s)$.; ou seja, y tem uma distribuição uniforme no intervalo $[0, 1]$.Conclusão:

Sendo y uma v.a. com d.p.u. em $[0,1]$, $x = c^{-1}(y)$ é uma v.a. com densidade $p(x) = dc(x)/dx$.

O método de inversão depende da possibilidade de calcular o inverso da d.p.c., $c^{-1}(x)$.

- gerar r com d.p.u. em $[0,1)$ e calcular $x = c^{-1}(r)$.

Exemplo Para uma distribuição exponencial

$$\rho(x) = \lambda e^{-\lambda x} \tag{18}$$

a distribuição de probabilidade cumulativa é

$$c(x) = \int_0^x dy \lambda \exp(-\lambda y) = 1 - e^{-\lambda x} \tag{19}$$

Se $y = 1 - e^{-\lambda x}$,

$$x = -\frac{1}{\lambda} \log(1-y) \tag{20}$$

Assim, se y tiver uma distribuição uniforme no intervalo $[0,1)$, x tem densidade de probabilidade exponencial.

Actividade 5: teste do método de inversão. Usando este método de inversão, gere uma sequência de de n.p.a com a distribuição exponencial e inspecione um histograma dos resultados obtidos.

4.2 Integral por Monte-Carlo com importance sampling.

Considere o integral

$$I = \int_a^b dx f(x) \tag{21}$$

Este integral pode ser expresso em termos do valor médio $\langle f(x) \rangle = \int_{-\infty}^{+\infty} dx w(x) f(x)$ em que $w(x)$ é a distribuição uniforme no intervalo $[a,b)$:

$$w(x) = \begin{cases} \frac{1}{b-a} & \text{se} \quad a \leq x < b \\ 0 & \text{se} \quad x < a \,\text{ou}\, x > b \end{cases} . \tag{22}$$

O método de Monte-Carlo consiste em estimar este valor médio gerando uma amostra de valores x_i com distribuição uniforme, e usando a média da amostra como estimador do valor médio:

$$I = (b-a)\langle f(x) \rangle \approx \frac{b-a}{M} \sum_{i=1}^{M} f(x_i)$$

Mas suponhamos, por exemplo, que o integral que queremos calcular é

$$I = \int_0^\infty dx \exp(-x^2/2);$$

podemos aproximá-lo introduzindo um cutoff , $L = 50$; o resultado é na prática o mesmo,

$$I \approx \int_0^L dx \exp(-x^2/2). \tag{23}$$

Contudo, se usarmos o método referido acima a convergência será muito lenta. A maior parte dos valores de uma amostra uniforme no intervalo $[0, L)$ quase não contribui para o cálculo da média; a função integranda é picada junto à origem na região de integração.

Podemos, no entanto, usar uma amostra com distribuição arbitrária $w(x)$, se notarmos que

$$I = \int_0^\infty dx f(x) = \int_0^\infty dx w(x) \frac{f(x)}{w(x)} = \langle \frac{f(x)}{w(x)} \rangle$$

O método será mais eficiente se a função cuja média estamos a fazer, $f(x)/w(x)$, não for tão picada numa fracção pequena do intervalo. Os valores da amostra, x_i, são gerados com a densidade de probabilidade $w(x)$, e

$$I \approx \frac{1}{M} \sum_{i=1}^{M} \frac{f(x_i)}{w(x_i)}; \tag{24}$$

esta é a base da técnica de *importance sampling.* Em aplicações de Física Estatística, com espaços de fase de elevada dimensão, é essencial.

Actividade 6: ilustração da amostragem por importância Estime o integral da eq. 23 usando:

1. uma distribuição uniforme no intervalo $[0, L)$ com $L = 50$.
2. a distribuição exponencial $w(x) = \exp(-x)$ para fazer *importance sampling.* Use o método de inversão para gerar uma amostra com a densidade de probabilidade $w(x)$.

Compare a convergência dos dois métodos em função de M.

5 Decaímento radioactivo como fenómeno estocástico

A lei de decaímento radioactivo

$$N(t) = N_0 e^{-t/\tau}$$

é, na realidade, uma lei de valores médios, já que o processo de decaímento é estocástico. Num intervalo de tempo Δt, um núcleo tem uma probabilidade de sobrevivência $P_0(\Delta t) = \exp(-\Delta t/\tau)$ e de ter decaído $1 - P_0(\Delta t)$.

Actividade 7

- Construa uma simulação do processo de decaímento do seguinte modo. Escolha um passo de tempo menor que τ e um dado número inicial de núcleos; em cada passo de tempo decida probabilisticamente, para cada núcleo, se sobreviveu; represente graficamente os valores de $N(t)$ de núcleos sobreviventes em cada instante. Compare com a curva de decaímento exponencial. Varie o número inicial de núcleos.
- Em alternativa pode usar o método de inversão para gerar para cada núcleo (com uma chamada ao gerador de n.p.a.) o instante em que decaiu. Com estes dados pode reconstruir o valor de $N(t)$.

Algorithm 3 OOP em `python`. A classe `particle` define um objecto com duas propriedades, `position` e `velocity`.

```
...
class particle:
     def __init__(self,p,v):
     self.position = p
     self.velocity = v
     def translate(self,s)
          .
          .
     def rotate(self,theta)
          .
          .

pos=array([0.,0.])
vel=array([1.,0.])
p1=particle(pos,vel)     # creates an instance of particle
z=2.0
theta=pi/4.
p.translate(z)           # moves the particle: adds z*p.velocity
                         # to p.position
p.rotate(theta)          # rotates particle.velocity
```

6 Passeio aleatório contínuo

Nesta secção consideramos a propagação de uma partícula energética num meio. Supomos que a partícula se move com velocidade uniforme entre colisões; ao colidir, a sua direcção de movimento é alterada. Em duas dimensões podemos rodar a sua velocidade de um ângulo obtido de uma distribuição de probabilidade dada. A distância percorrida entre colisões é determinada pela probabilidade de colisão por unidade de comprimento, μ: a probabilidade de a partícula sofrer uma colisão entre z e $z + dz$ na direcção de movimento é μdz. Se $P_0(z)$ for a probabilidade de sobrevivência,

$$P_0(z + dz) = P_0(z)\,(1 - \mu dz) \Rightarrow P_0(z) = e^{-\mu z}.$$

A densidade de probabilidade da distância à proxima colisão é a distribuição exponencial

$$p(z) = \mu P_0(z) = \mu e^{-\mu z}. \tag{25}$$

Este problema oferece uma oportunidade para usar as características OOP (object oriented programming) de Python.

Podemos definir uma classe `particle` cujos objectos têm duas propriedades, `position e velocity.0` passeio aleatório pode ser construído definindo *métodos* que transladam a partícula (`translate)` na direcção da sua velocidade de um valor gerado com a densidade da eq.(25) e rodem a respectiva velocidade (`rotate,` ver listagem 3).

Actividade 8 Escreva um programa para gerar trajectórias estocásticas de partículas em duas dimensões, de acordo com este modelo. Use velocidade de módulo constante unitário. Eis algumas sugestões de exploração.

- Estude a evolução da densidade de probabilidade de posição para um conjunto de partículas com distribuição de velocidades iniciais isotrópica. Estude a rapidez com que evolui para

uma gaussiana. Varie a distribuição de probabilidade do ângulo de rotação da velocidade em cada colisão; pode variar desde difusão isotrópica (ângulo de rotação uniforme em $[0, 2\pi)$ até colisões dominada por pequenos ângulos (por exemplo distribuição uniforme em $[-\theta_1, +\theta_1]$.

- Meça a distância média quadrática $\langle R^2(t)\rangle$ e relacione-a com μ. Qual é a distância média entre colisões?
- Assuma um feixe inicial de partículas com a mesma direcção de propagação. Meça a variação da velocidade média das partículas com o tempo. Estude as variações temporais de $\langle x(t)\rangle$, $\langle \Delta x^2(t)\rangle$, $\langle y(t)\rangle$,$\langle \Delta y^2(t)\rangle$ (direcções paralelas e perpendiculares à velocidade inicial do feixe). No caso da coordenada na direcção de movimento inicial do feixe deve observar uma passagem entre um regime balístico e difusivo. Porquê?

7 Partículas num potencial $V(x)$: método de Metropolis-Hastings

Considere um gás de partículas colocadas num potencial $V(x)$. A densidade de probabilidade de equilíbrio de posições é

$$\rho(x) = \rho_o \exp\left(-\beta V(x)\right)$$

em que $\beta = 1/k_B T$.

O algoritmo de Metropolis-Hastings permite gerar um sequência de posições que tende **assimptóticamente** para a distribuição $\rho(x)$. O processo é o seguinte:

1. Gera-se uma posição inicial x_0.
2. Propõe-se uma transição
$$x_0 \to x_1 = x_0 + \Delta x.$$
em que Δx tem uma distribuição uniforme num intervalo $[-\Delta, +\Delta]$
3. Calcula-se a probabilidade
$$p = \min\left[1, \exp\left(-\beta V(x_1)\right) / \exp\left(-\beta V(x_0)\right)\right].$$
4. Aceita-se a transição ($x_0 = x_1$) com probabilidade p e rejeita-se ($x_0 = x_0$) com probabilidade $1 - p$ (método de aceitação/rejeição de Von-Neumann).
5. Volta-se a 2.

A sequência de valores assim gerados tem uma distribuição **assimptótica** dada por $\rho(x)$.

Actividade 9 Use este algoritmo para gerar a distribuição correspondente a um potencial de oscilador harmónico, $V(x) = kx^2/2$. Represente um histograma da posições obtidas. Verifique se a variação da temperatura conduz ao resultado esperado.

Estude o número de iterações requeridas para convergir para a distribuição assimptótica e o efeito da escolha de Δ, quer na convergência, quer no número de recusas. O que acontece se Δ não for escolhido convenientemente (demasiado grande ou demasiado pequeno)? Pode escolher sempre o mesmo Δ independentemente de β?

Sugestão de exploração Imagine que o potencial é um duplo poço

$$V(x) = \frac{1}{2}k(x^2 - 1)^2$$

e $k_B T \ll k$. O que pode acontecer quando a coordenada se aproximar de um dos mínimos, $x = \pm 1$? Experimente!

8 Simulação do Modelo de Ising

O modelo de Ising é um modelo de variáveis binárias $s_i = \pm 1$ (spins) , dispostas nos nodos de uma rede, com uma energia que depende da configuração relativa de vizinhos na rede:

$$E(\mathbf{s}) = -\sum_{\langle ij \rangle} s_i s_j$$

A soma $\langle ij \rangle$ é sobre vizinhos na rede. Cada par de spins vizinhos com o mesmo valor contribui $s_i s_j = -1$ para a energia total, e com valores opostos com $s_i s_j = +1$. Por isso, o sistema tem dois estados fundamentais $\{s_i = 1,\, i = 1, \ldots N\}$ e $\{s_i = -1,\, i = 1, \ldots N\}$. A função de partição é dada por

$$\mathcal{Z} = \sum_{\{\mathbf{s}\}} \exp\left(-E(\mathbf{s})/k_B T\right),$$

e o valor médio de qualquer variável do espaço de fase, $A(\mathbf{s})$,

$$\langle A \rangle = \frac{1}{\mathcal{Z}} \sum_{\{\mathbf{s}\}} A(\mathbf{s}) \exp\left(-E(\mathbf{s})/k_B T\right).$$

Contudo, estas somas têm 2^N termos e são impossíveis de fazer por enumeração exaustiva de estados para sistemas com um número aceitável de spins. Somos levados a usar um método de amostragem da soma, gerando uma sequência de M estados $\mathbf{s}_r$ com uma probabilidade de ocorrência $\propto \exp\left(-E(\mathbf{s}_r)/k_B T\right)$, e aproximando

$$\langle A \rangle \approx \frac{1}{M} \sum_{r=1}^{M} A(\mathbf{s}_r).$$

O algoritmo de Metropolis é usado para gerar esta cadeia:

1. É gerada uma configuração inicial de spin $\mathbf{s}_0$;
2. É escolhido à sorte um spin i; é calculada a variação de energia ΔE na transformação $s_i \to -s_i$;
3. Se $\Delta E \leq 0$ a inversão $s_i \to -s_i$ é realizada; se $\Delta E > 0$,o spin i é invertido com probabilidade $\exp(-\Delta E/k_B T)$, usando o método de rejeição de Von Neumann.
4. Volta-se a 2.

Na cadeia de estados assim obtida, uma configuração de spin $\mathbf{s}$ tem, assimpóticamente, uma probabilidade de ocorrência $\propto \exp\left(-E(\mathbf{s})/k_B T\right)$.

A listagem 4 mostra uma programa completo de simulação, com um varrimento de uma gama de temperaturas, e com o cálculo da energia e da magnetização por spin:

$$\begin{aligned} \epsilon &= \frac{1}{N}\langle E(\mathbf{s_r}) \rangle \\ m &= \frac{1}{N}\langle \sum_i s_i \rangle. \end{aligned}$$

Algorithm 4 Listagem de `ising_simulation.py`, para simulação do modelo de Ising.

```
#-----------------------------------------------------
#               Ising Model simulation
#-----------------------------------------------------
#                  Main Parameters
#
from ising import *
#
MCS=4000                              # Monte Carlo Steps
MCSTHERMAL=400                        # Thermalization steps
N=32                                  # Linear Lattice size
#
temperatures= arange(2.,3.,.05)       # Temperatures of simulation
#
f= open("teste.dat","w")              # Output file
#
#                   Temperature cycle
#
for t in temperatures:
    avgMag=0.0                        # accumulates  magnetization
    avgEnerg=0.0                      # accumulates energy
#
#   Create a list of N*N spins
    spin = createState(N,'ferro')
#
#   Tabulate Boltzmann wheights
    bws=boltzmanWeights(t)
#
#   Create an instance of  Lattice class
    s1=Lattice(N,spin,energy(spin,N),mag(spin))
#
#                     Monte Carlo cycle
#
#   Thermalize
#
    for r in range(MCSTHERMAL*N*N):
        s1.update(bws)
#
#   Measure
#
    for r in range(MCS*N*N):
        s1.update(bws)
        avgMag += s1.mag/float(N*N)
        avgEnerg += s1.energy/float(2.*N*N)
#
#                     normalize and print
#
    avgMag /= float(MCS*N*N)
    avgEnerg/=float(MCS*N*N)
    f.write('%6.2f\t%6.2f\t%6.2f\n'% (t,avgMag,avgEnerg))
f.close()
```

A estrutura deste programa é simples:

- Fixa os parâmetros da simulação: número de passos de Monte-Carlo por spin (`MCS);` o número de passos de termalização (`MCSTHERMAL`) para garantir que a distribuição dos estados da cadeia usados para cálculo de médias é a distribuição assimptótica , $\propto \exp(-E(\mathbf{s}_r)/k_BT)$; tamanho linear da rede `(N);` gama de temperaturas `(temperatures).`
- Faz um ciclo sobre as temperaturas (uma cópia da simulação para cada temperatura).
- Faz o ciclo da Monte-Carlo para cada temperatura, acumulando a magnetização (`avgMag`) e energia (`avgEnerg`).

Todos os detalhes da simulação estão escondidos no módulo `ising.py,` que proporciona:

- funções de criação de uma configuração inicial (`createState`); de medição da respectiva energia (`energy`) e magnetização (`mag`);
- uma função de inicialização dos pesos de Boltzmann requeridos pela simulação. Num modelo de Ising as variações de energia possíveis são discretas, e acelera muito a simulação tabelar $\exp(-\Delta E/k_BT)$ em vez de calcular exponenciais em cada passo.
- uma classe `Lattice`, que é o centro da simulação: esta classe tem como propriedades:
  - `N`, a dimensão **linear** da rede;
  - `state,` o estado dos spins;
  - `energy`, a energia ;
  - `mag`, a magnetização total;

  Tem um *método* `update`, que tem como argumento a lista de pesos de Boltzamnn, e que realiza um passo da simulação, actualizando as propriedades da instância de `Lattice:` o estado dos spins, as respectivas energia e magnetização.

Actividade 10 A tarefa desta actividade é escrever o módulo `ising.py.` As funções aí definidas estão documentadas na listagem 5. Neste caso foi implementado o modelo na rede quadrada. É mais simples começar por uma cadeia linear, embora esta só tenha estado ferromagnético a $T = 0$.

Notas

1. Num passo de Monte-Carlo só varia um spin. Não é necessário recalcular a energia ou a magnetização de toda a rede: basta calcular a respectiva variação. Por essa razão, a energia e magnetização, são propriedades da classe `Lattice.` São actualizadas a partir das respectivas variações.
2. As condições fronteira periódicas são uma das questão mais delicadas da simulação. Numa rede linear, o spin s_i só interage com s_{i-1} e s_{i+1}. A energia (multiplicada por 2) pode ser obtida somando sobre i a expressão

   $$s[i] * (s[i+1] + s[i-1]) .$$

   Contudo, esta expressão não funciona se $i = 0$ ou $i = N - 1$: os vizinhos de $i = 0$ são s_{N-1} e s_2, e os de s_{N-1} são s_0 e s_{N-2}. Este facto corrige-se facilmente usando divisão de inteiros. Como $i\%j$ é o resto da divisão inteira de i por j,

   $$s[i] * (s[(i+1)\%N] + s[(i-1+N)\%N])$$

   Uma técnica inteiramente semelhante pode ser usada em qualquer rede quadrada, cúbica, ou hipercúbica em d dimensões.

Actividade 11 Use o programa que escreveu para explorar os resultados de uma simulação de Monte-Carlo.

- registe a evolução de energia a magnetização durante a simulação, para várias temperaturas; verifique se as flutuações dependem da temperatura.
- varie o estado inicial (`'random'` ou `'ferro'`)
- compare simulações de tamanho variável (4×4,8×8, 16×16, 32×32)

Referências

[1] *Física Estatística e Computacional*, 2007, *Aulas na Web,* J. M. Nunes da Silva, `http://elearning.fc.up.pt/aulasweb0607/file.php/109985/moddata/forum/250473/15023/PC-FEC-v3.tar.bz2`

[2] *Física Estatística e Computacional*, Notas das teóricas, 2008, *Aulas na Web,* J. M. Nunes da Silva `http://elearning.fc.up.pt/aulasweb0607/mod/resource/view.php?id=411499`

[3] *Statistical Mechanics: Entropy, Order Parameters and Complexity*, James P. Sethna, Oxford Master Series in Condensed Matter, Oxforf University Press, 2006. Disponível *online* em `http://pages.physics.cornell.edu/sethna/StatMech/`

Algorithm 5 Listagem das docsctrings do módulo Ising.py

```
createState(N, type='random')
    Returns array of N*N spins +1 or -1

    if type='ferro' all spins +1
    if type='random' (default) equal probability
   +1 or -1

mag(state)
    returns unormalized magnetization of spin array
    M=s1+...+sN

energy(state, N)
    returns energy of N*N spins arranged in
    square lattice with periodic boundary
    conditions
    J=1

boltzmanWeights(temperature)
    Returns array[exp(-4/temperature),exp(-8/temperature)]
    for Ising model in square lattice
    Delta_E=8,4,0,-4,-8.
    simulation requires only exp(-4/T),exp(-8/T)

class Lattice
 |  Methods defined here:
 |
 |  __init__(self, N, state, energy, mag)
 |      Lattice is initialized with
 |
 |      N -linear dimension
 |      state - chain of N*N spins
 |      energy = energy(state,N)
 |      mag= mag(state)
 |
 |  update(self, bweights)
 |      performs one simulation update
 |      with Metropolis algorithm
```