Próbkowanie metodą Wang-Landau

Emil Śmiech, Amelia Królczyk

Styczeń 2021

Część I

Wstęp

Wraz z szybkim rozwojem procesorów komputerowych, symulacje numeryczne metodą Monte Carlo stały się uznanym narzędziem do badania białek, czy polimerów. W celu przyspieszenia symulacji zaproponowano kilka metod rozwiązywania problemów. Jedną z tych metod jest próbkowanie Wang-Landau.

Próbkowanie metodą Wang-Landau określa gęstość stanów i związaną z nimi funkcję podziału, nie zastępuje bezpośrednio algorytmu Metropolisa i jego symulacji fluktuacji termicznych. Jednakże ta metoda zapewnia równoważną symulację dla modelu Isinga.

Algorytm Metropolisa jest połączeniem techniki redukcji wariancji i techniki odrzucenia von Neumanna. Techniki te pokazują, jak zwiększyć efektywność integracji metodą Monte Carlo (próbkując głównie losowe punkty gdzie całka jest duża), oraz jak generować losowe punkty z dowolnym rozkładem prawdopodobieństwa.

W naszej pracy przetestujemy opisaną metodę w celu obliczenia gęstości stanów oraz energii wewnętrznej.

Część II

Opis próbkowania metodą Wang-Landau

Postępowanie w metodzie Wang-Landau jest podobe do algorytmu Metropolisa, jednak wykorzystuje się funkcję gęstości stanów $g(E_i)$, a nie współczynnik Boltzmanna:

1. Zaczynamy od dowolnej konfiguracji spinu $\alpha k = s_1, s_2, ..., s_N$oraz z dowolnymi wartościami gęstości stanów $g(Ei) = 1, i = 1, ..., M$, gdzie $M = 2N$, to liczba stanów systemu.

2. Generujemy konfigurację próbną $\alpha k + 1$ następująco:
a) wybieramy cząstkę „i" przypadkowo
b) odracamy spin „i"

3. Obliczamy energię $E\alpha tr$ konfiguracji próbnej.

4. Jeśli $g(E\alpha tr)g(E\alpha k)$, akceptujemy próbę, czyli ustawiamy $\alpha k + 1 = \alpha tr$.

5. Jeśli $g(E\alpha tr) > g(E\alpha k)$, akceptujemy próbę z prawdopodobieństwem $P = g(E\alpha k)/(g(E\alpha tr)$: a) wybieramy jednolitą liczbę losową $0r_i1$. b) ustawiamy $\alpha k+1 = \alpha tr$, jeśli Pr_j (akceptujemy) ; αk, jeśli $P < rj$ (odrzucamy) . Ta zasada akceptacji może być zwięźle wyrażona jako:

$$\mathcal{P}(E_{\alpha_k} \to E_{\alpha_{\text{tr}}}) = \min\left[1, \frac{g(E_{\alpha_k})}{g(E_{\alpha_{\text{tr}}})}\right]$$

który oczywiście zawsze akceptuje stany o niskiej gęstości (nieprawdopodobne).

6. Gdy mamy nowy stan, modyfikujemy gęstość prądu stanów $g(E_i)$ poprzez mnożnik f:

$$g(E_{\alpha_{k+1}}) \to f\, g(E_{\alpha_{k+1}}),$$

i dodajemy 1 do kosza na histogramie odpowiadającym nowej energii:

$$H(E_{\alpha_{k+1}}) \to H(E_{\alpha_{k+1}}) + 1.$$

7. Wartość mnożnika f jest empiryczna. Zaczynamy od liczby Eulera $f = e = 2,71828$, która wydaje się zachowywać dobrą równowagę między bardzo dużą liczbą małych kroków (małe f) a zbyt szybką serią skoków w przestrzeni energii (duże f). Ponieważ entropia $S = kBlng(E_i) \to kB[lng(E_i) + lnf]$, (15,24) odpowiada jednorodnemu wzrostowi entropii o kB.

8. Nawet przy rozsądnych wartościach f, powtarzane działania mnożenia prowadzą do wykładniczego wzrostu wielkości g. Może to powodować przepełnienia zmiennoprzecinkowe i zgodną utratę informacji (w końcu wielkość $g(E_i)$ nie ma znaczenia, ponieważ funkcja jest znormalizowana). Tych przepełnień można uniknąć, działając na logarytmach wartości funkcj. W tym przypadku aktualizacja gęstości stanów staje się:

$$\ln g(E_i) \to \ln g(E_i) + \ln f.$$

9. Trudność w przechowywaniu wartości $lng(E_i)$ polega na tym, że do obliczenia prawdopodobieństwa potrzebujemy stosunku wartości $g(E_i)$. Można to obejść, stosując tożsamość $x = exp(lnx)$ do wyrażenia stosunku jako:

$$\frac{g(E_{\alpha_k})}{g(E_{\alpha_{\mathrm{tr}}})} = \exp\left[\ln \frac{g(E_{\alpha_k})}{g(E_{\alpha_{\mathrm{tr}}})}\right] = \exp\left[\ln\, g(E_{\alpha_k})\right] - \exp\left[\ln\, g(E_{\alpha_{\mathrm{tr}}})\right].$$

10. Losowy spacer w E_i trwa do momentu uzyskania płaskiego histogramu odwiedzonych wartości energii. Płaskość histogramu jest regularnie testowana (co 10 000 powtórzeń), a spacer kończy się, gdy histogram jest dostatecznie płaski. Wartość f jest następnie zmniejszana, aby następny spacer zapewniał lepsze przybliżenie do $g(E_i)$.

11. Następnie zachowajemy wygenerowane $g(E_i)$ i resetujemy wartości histogramu $h(E_i)$ do zera.

12. Spacery są przerywane i rozpoczynane nowe, dopóki nie zostanie uzyskana żadna istotna korekta gęstości stanów. Mierzy się to wymagając mnożenia współczynnika $f \simeq 1$ w ramach pewnego poziomu tolerancji; na przykład $f1 + 108$. Jeśli algorytm się powiedzie, histogram powinien być płaski.

13. Ostatnim krokiem symulacji jest normalizacja wydedukowanej gęstości stanów $g(E_i)$. Dla modelu Isinga z N obrotami w górę lub w dół, warunek normalizacji wynika ze znajomości całkowitej liczby stanów:

$$\sum_{E_i} g(E_i) = 2^N \quad \Rightarrow \quad g^{(\mathrm{norm})}(E_i) = \frac{2^N}{\sum_{E_i} g(E_i)} g(E_i).$$

Ponieważ na sumę największy wpływ mają te wartości energii, w których $g(E_i)$ jest duże, może nie być precyzyjne dla gęstości o niskim E_i, które mają niewielki wpływ na sumę. W związku z tym bardziej precyzyjna normalizacja, polega na wymaganiu, aby istniały tylko dwa stany podstawy z energiami $E = 2N$ (jeden ze wszystkimi obrotami w górę i jeden ze wszystkimi obrotami na dół):

$$\sum_{E_i=-2N} g(E_i) = 2.$$

Część III

Kod i obliczenia

Rysunek 1: Ustawienia opcji grafiki, zadeklarowanie zmiennych oraz wielkości fizycznych

```
from cmath import sin, pi
from vpython import *
from numpy import zeros
import random

L = 8; N = (L * L)

#Set up graphics
entgr = graph(x = 0, y = 0, width = 500, height = 250, title = 'Density of States',\
    xtitle = 'E/N', ytitle = 'logg(E)', xmax = 2., xmin = -2., ymax = 45, ymin = 0)

entrp = gcurve(color = color.magenta)

energygr = graph(x = 0, y = 250, width = 500, height = 250, title = 'E vs T',\
    xtitle ='T', ytitle = 'U(T)/N', xmax = 8., xmin = 0, ymax = 0., ymin = -2.)

energ = gcurve(color = color.cyan)

histogr = canvas(x = 0, y = 500, width = 500, height = 300,\
    title = '1st histogram: H(E) vs. E/N, corresponds to log(f) = 1')

l = []
for i in range(0, L):
    for j in range(0, L):
        l.append(vector(2.25 * L * i - N, 2.25 * L * j - 40, 0))

histo = points(pos = l, color = color.red)
xaxis = curve(pos = [vector(-N, -50, 0), vector(N, -50, 0)])
minE = label(pos = vector(-N + 3, -60, 0), text = '-2', box = 0)
maxE = label(pos = vector(N-3, -60, 0), text = '2', box = 0)
zeroE = label(pos = vector(0, -60, 0), text = '0', box = 0)
ticm = curve(pos = [vector(-N, -55, 0), vector(-N, -53, 0)])
tic0 = curve(pos = [vector(0, -55, 0), vector(0, -53, 0)])
ticM = curve(pos = [vector(N, -55, 0), vector(N, -53, 0)])
enr = label(pos = vector(0, -75, 0), text = 'E/N', box = 0)

sp = zeros((L, L))          #Gridsize, spins
hist = zeros((N + 1))
prhist = zeros((N + 1))     #Histograms
S = zeros((N + 1), float)   #Entropy = logg(E)

def  iE(e):return int((e + 2*N)/4)

def  IntEnergy():
    exponent = 0.0
```

Rysunek 2: Inicjalizacja. Zdefiniowanie próbkowania Wang-Landau

```
46
47      for T in arange(0.2, 8.2, 0.2):         #Selectlambdamax
48          Ener = -2 * N
49          maxL = 0.0          #Initialize
50
51          for i in range(0, N + 1):
52              if S[i] != 0 and (S[i] - Ener/T) > maxL:
53                  maxL = S[i] - Ener/T
54                  Ener = Ener + 4
55
56          sumdeno = 0
57          sumnume = 0
58          Ener = -2 * N
59
60          for i in range(0, N):
61              if S[i] != 0:
62                  exponent = S[i]-Ener/T-maxL
63                  sumnume += Ener * exp(exponent)
64                  sumdeno += exp(exponent)
65                  Ener = Ener + 4.0
66
67          for j in range(0, N):
68              if S[j] != 0:
69                  U = sumnume/sumdeno/N           #internalenergyU(T)/N
70                  energ.plot(pos = [T, U])
71
72  def  WL():          #Wang-Landausampling
73      Hinf = 1.e10            #initialvaluesforHistogram
74      Hsup = 0.
75      tol = 1.e-3         #tolerance, stopsthealgorithm
76      ip = zeros(L)
77      im = zeros(L)           #BCRordown, Lorup
78      height = abs(Hsup-Hinf)/2.          #Initializehistogram
79      ave = (Hsup + Hinf)/2.          #aboutaverageofhistogram
80      percent = height/ave
81
82      for i in range(0, L):
83          for j in range(0, L):sp[i, j] = 1           #Initialspins
84
85      for i in range(0, L):
86          ip[i] = i + 1
87          im[i] = i - 1         #Caseplus, minus
88
89      ip[L - 1] = 0
90      im[0] = L - 1         #Borders
91      Eold = -2 * N         #Initializeenergy
92
```

Rysunek 3: Inicjalizacja entropii, wybór dowolnego spinu, inicjalizacja nowego histogramu

```
93      for j in range(0, N + 1):S[j] = 0           #Entropyinitialized
94      iter = 0
95      fac = 1
96
97      while fac > tol:
98          i = int(N*random.random())          #Selectrandomspin
99          xg = i%L
100         #Mustbei//L, noti/LforPython3:
101         yg = i//L           #Localizex, y, gridpoint
102
103         Enew = Eold + 2 * (sp[int(ip[xg]), yg] + sp[int(im[xg]), yg] +
104             sp[xg, int(ip[yg])] + sp[xg, int(im[yg])]) * sp[xg, yg]          #Changeenergy
105
106         deltaS = S[iE(Enew)]-S[iE(Eold)]
107
108         if deltaS <= 0 or random.random() < exp(- deltaS):
109             Eold = Enew;
110             sp[xg, yg] *= -1            #Flipspin
111
112         S[iE(Eold)] += fac;             #Changeentropy
113
114         if iter%10000 == 0 :            #Checkflatnessevery10000sweeps
115             for j in range(0, N + 1):
116                 if j == 0:
117                     Hsup = 0
118                     Hinf = 1e10         #Initializenewhistogram
119
120                 if hist[j] == 0:continue            #Energiesnevervisited
121                 if hist[j]>Hsup:Hsup = hist[j]
122                 if hist[j]<Hinf:Hinf = hist[j]
123
124             height = Hsup - Hinf
125             ave = Hsup + Hinf
126             percent = 1.0*height/ave            #1.0tomakeitfloatnumber
127
128             if percent<0.3:         #Histogramflat?
129                 print("iter", iter, "log(f)", fac)
130
131                 for j in range(0, N + 1):
132                     prhist[j] = hist[j]         #toplot
133                     hist[j] = 0         #Savehist
134
135                 fac *= 0.5          #Equivalenttolog(sqrt(f))
136
137         iter += 1
138         hist[iE(Eold)] += 1         #Changehistogram, add1, update
139
```

Rysunek 4: Przywołanie algorytmu Wang-Landau, obliczenia

```
140             if fac >= 0.5:          #justshowthefirsthistogram
141                 #Speedupbyusingarraycalculations:
142                 histo.x = 2.0 * arange(0, N) - N
143                 histo.y = 0.025 * hist - 10
144
145     deltaS = 0.0
146     print("wait because iter > 13 000 000")           #notalwaysthesame
147     WL()            #CallWangLandaualgorithm
148     deltaS = 0.0
149
150     for j in range(0, N + 1):
151         order = j * 4 - 2 * N
152         deltaS = S[j] - S[0] + log(2)
153         if S[j] != 0: entrp.plot(pos = [1. * order/N, deltaS])         #plotentropy
154
155     IntEnergy();
156     print("Done")
```

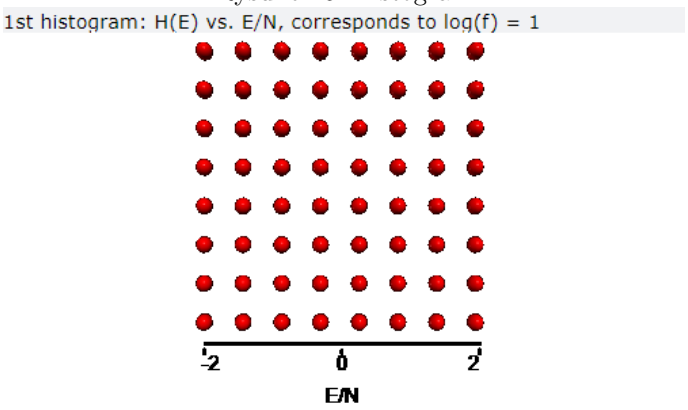
Rysunek 5: Obliczenia wykonane przez program:

```
wait because iter > 13 000 000
iter 0 log(f) 1
iter 50000 log(f) 0.5
iter 90000 log(f) 0.25
iter 130000 log(f) 0.125
iter 170000 log(f) 0.0625
iter 230000 log(f) 0.03125
iter 300000 log(f) 0.015625
iter 370000 log(f) 0.0078125
iter 450000 log(f) 0.00390625
iter 550000 log(f) 0.001953125
Done
```

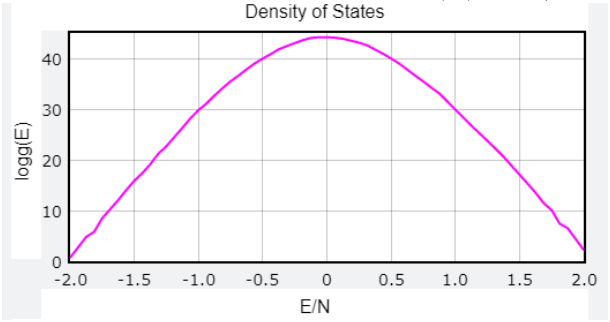
Część IV

Wyniki

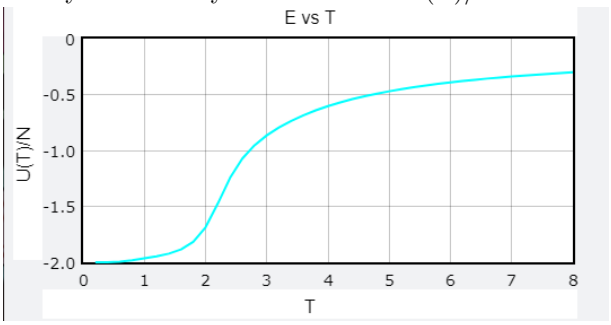
Rysunek 6: Histogram



Rysunek 7: Wykres zależności $logg(E)$ od E/N



Rysunek 8: Wykres zależności $U(T)/N$ od T



Wyniki pokazują, iż gęstość stanów uzyskana bezpośrednio przez metodę Wang-Landau umożliwia obliczenie właściwości termodynamicznych nawet dla dużych układów. Możemy również powiedzieć, że badania przeprowadzone tą

metodą są dokładniejsze niż te, które byłyby przeprowadzone w standardowej symulacji Monte Carlo. Wielkości termodynamiczne, takie jak energia swobodna i entropia są bezpośrednio szacowane na podstawie gęstości stanów.