Numerical simulations

$$Z = \sum_{\{s\}} e^{-\beta \mathcal{H}(s)}$$

think of the Ising model

$\left(\mathcal{H} = J \sum_{\langle ij \rangle} s_i s_j - B \sum s_i \qquad J<0 \right)$

expectation values:
(thermal average)

$$\langle X \rangle = \frac{1}{Z} \sum_{\{s\}} X(s) e^{-\beta \mathcal{H}(s)}$$

ex.: $X = s_i$ or $s_i s_{i+1}$

or $\frac{1}{N} \sum s_i$ $\quad \frac{1}{N_e} \sum_{\langle ij \rangle} s_i s_j$

one would like $\langle X \rangle$ as the fn. of β

& find critical exp., ~~ect~~ etc.

Can it be done <u>exactly</u>? On the lattice we have finite # dof — but

2d Ising, $N = 100 \times 100$, $2^N = 2^{10000}$ configurations!

that cannot be finished in reasonable time

Statistically: sample configurations:

$$P(\{s\}) = \frac{1}{Z} e^{-\beta \mathcal{H}(s)} = \frac{1}{Z} e^{-\beta E_s}$$

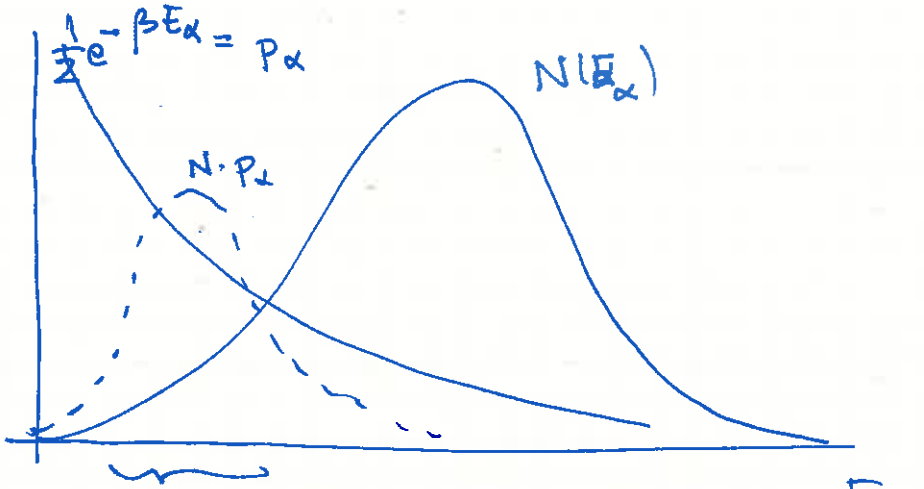
$E_{max} = J \cdot N_{links} > 0$

$E_s \nearrow$ $\quad P(E_s) \searrow$, but there are many configurations with the same energy: $\nearrow$ total contribution?

there's a balance of which configurations are most important at any fixed temperature.

$\beta = \infty$ $(T=0)$ only $E_s = \min(E)$ matters

$\beta = 0$ $(T = \infty)$ all configs are equally important



these are the important configs

Since $E_\alpha \sim$ Volume, the peak will get sharper with increasing volume!

Sampling

Notation : $\alpha, \alpha_2 \ldots \alpha$: possible configs

assume there are A total configs.

We sample the config α with probability S_α

$$\sum_{\alpha=1}^{A} S_\alpha = 1$$

and generate a sequence $\alpha_1, \alpha_2, \ldots$

and $P_\alpha = \frac{1}{Z} e^{-\beta E_\alpha}$ is the Boltzmann factor

$$\sum_{\alpha}^{A} P_\alpha = 1$$

Then the thermal average (exp. value) of an arbitrary operator X is

$$\langle X \rangle = \underbrace{\sum_{\alpha} P_\alpha X_\alpha}_{\text{physical definition}} = \frac{1}{B} \sum_{k}^{B} S_{\alpha_k} P_{\alpha_k} X_{\alpha_k}$$

as

Sampling

Notation:

- denote a config. by α; Assume there are A configs. all together.
- Boltzman factor: $p_\alpha = \frac{1}{Z} \sum_\alpha^A e^{-\beta E_\alpha}$

$$\sum p_\alpha = 1 \longrightarrow \text{defines } Z$$

- Thermal average of any operator X

$$\langle X \rangle = \sum_\alpha p_\alpha X_\alpha = \frac{\sum_\alpha X_\alpha e^{-\beta E_\alpha}}{\sum e^{-\beta E_\alpha}}$$

Assume we sample the configs with probability s_α and create a sequence $\alpha_1, \alpha_2, \ldots \alpha_B$.

$$\sum_\alpha^A s_\alpha = 1$$

An estimator of $\langle X \rangle$ is

$$X_B = \frac{1}{B} \sum_{k=1}^B \frac{1}{s_{\alpha_k}} p_{\alpha_k} X_{\alpha_k}$$

$$\lim_{B \to \infty} X_B = \langle X \rangle$$

How good an estimator is this?

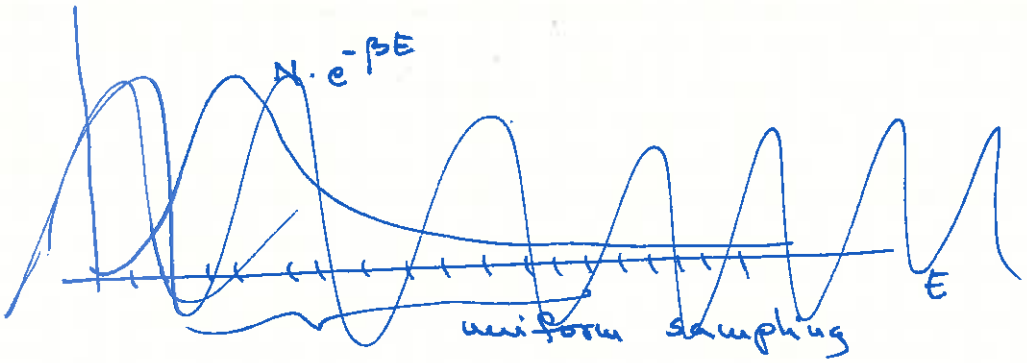
Variance of X_B:

$$\mathrm{Var}(X_B) = \langle X_B^2 \rangle - \langle X_B \rangle^2$$
$$= \langle (X_B - \langle X \rangle)^2 \rangle$$

↑ this is thermal average!

$X_B - \langle X \rangle$ = measures the fluctuations could be large due to $P_\alpha \sim e^{-\beta E_\alpha}$

Ex: $g_\alpha = \frac{1}{A}$ uniform (random) sampling

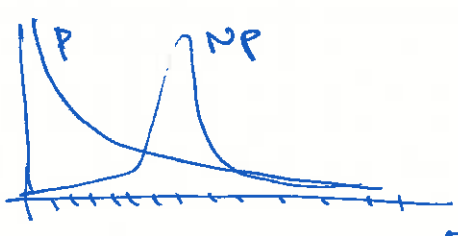


most terms will contribute very little a few will dominate

$$X_B = \frac{1}{BA} \sum P_{\alpha_k} X_{\alpha k}$$

P_{α_k}: varies exponentially

$X_{\alpha k}$: varies $O(1)$



Importance sampling

choose S_α to minimize the variance

Ex: $S_\alpha \approx P_\alpha X_\alpha \frac{1}{\langle X \rangle}$

$$X_B = \frac{1}{B} \sum \underbrace{\frac{1}{S} P_\alpha X_\alpha}_{\text{each term} \sim \langle X \rangle} = \quad \checkmark$$

This is impossible in practice, but

$$S_\alpha = P_\alpha$$

is pretty good choice. Most Monte Carlo updates do that or almost that.

If $S_\alpha = P_\alpha$

$$X_B = \frac{1}{B} \sum_k X_\alpha \quad : \text{simple average}$$

Monte Carlo methods:

Sample the Gibbs distribution $\{ p_\alpha \sim e^{-\beta E_\alpha}$ probability

$p_\alpha = P_\alpha$

$\xi_\alpha = X_\alpha \quad X_B = \frac{1}{B} \sum X_{\alpha_k}$ ↳ $\xi = X$

$\langle X \rangle = \frac{1}{A} \sum \xi$

Markov process: rule to create a sequence of configurations $\alpha_1, \alpha_2, \dots \alpha_k$, from $\alpha \to \alpha'$

$P(\alpha \to \alpha')$: probability that we'll go from $\alpha \to \alpha'$

Require:

1. $\sum_{\alpha'} P(\alpha \to \alpha'_i) = 1$: it goes somewhere

2. from any α, through a sequence, any other conf. must be reachable

3. reversibility or <u>detailed balance</u>:

$$p_\alpha P(\alpha \to \alpha') = p_{\alpha'} P(\alpha' \to \alpha)$$

If (1-3) are satisfied, the sequence created will (mimic — approximate) the Gibbs distribution at least <u>asymptotically</u>

Proof:

* (1) <u>if</u> $W(\alpha, n) = p_\alpha$: probability that the nth term in the sequence is α is p_α

i.e. the sampling is as desired

Then $W(\alpha, n+1) = \sum_{\alpha'} p_{\alpha'} P(\alpha' \to \alpha) = p_\alpha \sum_{\alpha'} P(\alpha \to \alpha') = p_\alpha$ ✓

(2)* even if $W(\alpha, n) \neq p_\alpha$ at a given n,

$D_n = \sum_\alpha |W(\alpha,n) - p_\alpha|^2$ decreases with n:

$$D_{n+1} = \sum_\alpha \left| W(\alpha, n+1) - p_\alpha \right|^2 =$$

$$= \sum_\alpha \left| \sum_{\alpha'} W(\alpha', n) P(\alpha' \to \alpha) - p_\alpha \right|^2$$

$$= \sum_\alpha \left| \sum_{\alpha'} W(\alpha', n) P(\alpha' \to \alpha) - p_\alpha \underbrace{\sum_{\alpha'} P(\alpha'' \to \alpha')}_{\sum_{\alpha'} p'_\alpha P(\alpha' \to \alpha'')} \right|^2$$

$$= \sum_\alpha \left| \sum_{\alpha'} \underbrace{P(\alpha' \to \alpha)}_{\text{pos.}} \left(W(\alpha', n) - p_{\alpha'} \right) \right|^2$$

$$\leq \underbrace{\sum_\alpha \sum_{\alpha'} \left| W(\alpha', n) - p'_\alpha \right| P(\alpha' \to \alpha)}_{1}$$

$$= \sum_{\alpha'} \left| W(\alpha' n) - p_{\alpha'} \right| = D_n$$

i.e. we get closer to the Gibbs distribution

It's all due to the <u>detailed balance</u> condition!

Metropolis

specific MC procedure : prescription of how to achieve db

detailed balance

$$\frac{\pi(\alpha \to \alpha')}{\pi(\alpha' \to \alpha)} = \frac{P_{\alpha'}}{P_\alpha} = e^{-\beta(E_{\alpha'} - E_\alpha)}$$

Metropolis : * choose α' randomly

* accept / reject the choice

$$P_{acc} = \min\left\{1,\ e^{-\beta(E_{\alpha'} - E_\alpha)}\right\}$$

i.e. : if the energy decreases, accept it always
$E_{\alpha'} - E_\alpha < 0$

if $E_{\alpha'} - E_\alpha > 0$, accept it with $e^{-\beta(E_{\alpha'} - E_\alpha)}$ probability.

This procedure satisfies detailed balance:

$$\pi(\alpha \to \alpha') = \pi_\alpha \cdot P_{acc}(\alpha \to \alpha')$$

$$\pi(\alpha' \to \alpha) = \pi_{\alpha'} \cdot P_{acc}(\alpha' \to \alpha)$$

$$\frac{\pi(\alpha \to \alpha')}{\pi(\alpha' \to \alpha)} = \frac{\pi_\alpha}{\pi_{\alpha'}} \cdot \frac{P_{acc}(\alpha \to \alpha')}{P_{acc}(\alpha' \to \alpha)} = e^{-\beta(E_{\alpha'} - E_\alpha)} \quad \checkmark$$

In practice, it will be efficient if

the acceptance is large

→ $\beta(E_{\alpha'} - E_{\alpha})$ not too ~~small~~ large

ex:

$E_\alpha = K \sum_{\langle i,j \rangle} s_i s_j$: extensive quantity

$\beta \Delta E \sim V$ if all ~~links~~ spins are changed

→ change only one spin at a time

It is not necessary to choose $\alpha \to \alpha'$ entirely randomly. If we choose it with prob. $p_p(\alpha \to \alpha')$, the transition prob is

$$P(\alpha \to \alpha') = p_p(\alpha - \alpha')\, P_{acc}(\alpha')$$

and DB

$$P_\alpha\, P(\alpha \to \alpha')\, P_{acc}(\alpha') = P_{\alpha'}\, p(\alpha' \to \alpha)\, P_{acc}(\alpha)$$

will be satisfied if $p(\alpha \to \alpha') = p(\alpha' \to \alpha)$ (reversibility)

Ex:

XY model

variable $\phi_n = e^{i\vartheta_n}$

$p(\vartheta \to \vartheta')$ if we update a single spin

$$\vartheta' = \vartheta + \Delta \cdot (r - 0.5)$$

"stepsize" ↑ ↑ random #, uniform $(0,1)$

If $\Delta = \pi$, ϑ' is random, independent of ϑ

Δ small: small change

Obviously reversible

The problem with Metropolis update is that for reasonable accept. one can change only locally, 1 spin at a time — leads to very large auto correlation.

Heat bath

fix $P(\alpha \to \alpha') = p_{\alpha 1}\, P(\alpha' \to \alpha)$

$P(\alpha \to \alpha') = \underbrace{p_p(\alpha \to \alpha')}_{\text{pick } \alpha' \text{ from } \alpha} P_{acc}(\alpha')$

$p_\alpha\, p_p(\alpha \to \alpha')\, P_{acc}(\alpha') = p_{\alpha'}\, p_p(\alpha' \to \alpha)\, P_{acc}(\alpha)$

How shall we pick α' if we want $P_{acc}(\alpha) = 1$?

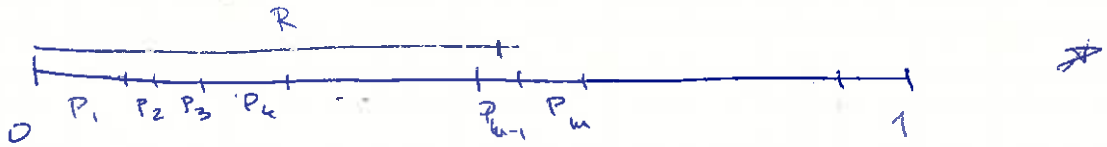
Heat bath algorithm. How to do it?

If the energy levels are local $\varepsilon_1 \dots \varepsilon_n$

$$p_j = e^{-\varepsilon_j \beta} \Big/ \sum_i e^{-\beta \varepsilon_i}$$

→ pick random # R; if $\sum_{j=1}^{m-1} p_j < R < \sum_{j=1}^{m} p_j$

choose state m



For continuous spins

$$R = \int_1^m dj\, p_j = F(m)$$

$$\hookrightarrow \underline{m = F^{-1}(R)}$$

This is useful if $F(m)$ is easily invertible.

Ising Heat Bath :

$$\beta H = -K \sum s_i s_j$$

Update spin s_i : $\beta H = -K\, s_i \sum_{j=\pm\mu} s_j$

$$\beta \varepsilon_1 = K \sum s_j$$

$$\beta \varepsilon_2 = -K \sum s_j$$

$$P_i = e^{-K \Sigma s_j} \Big/ \left(e^{K \Sigma s_j} + e^{-K \Sigma s_j} \right)$$

→ pick $s_i = \begin{matrix} -1 \\ +1 \end{matrix}$ if $\begin{matrix} R < P_i \\ R > P_i \end{matrix}$

Prove DB !

Generalize to XY model

Cluster update for spin models

Assign to config $\{s\}$ a bond config $\{B\}$
Update $\{B\}$ and transfer it back to $\{s'\}$

$\{s\}$ is a spin configuration

$$H = \sum_{\ell} H_{\ell} \qquad H_{\ell}(s_i, s_j) \quad \text{link interaction}$$

$$H_{\ell}(s_i s_j) = H_{\ell}(g s_i, g s_j) \qquad g \in G \text{ symmetry group}$$

B: bonds: connect lattice sites with

probability $P_{\ell}(s_i, s_j)$

$$0 \le P \le 1 , \quad P_{\ell}(g s_i, g s_j) = P_{\ell}(s_i, s_j)$$

satisfies the symm. but otherwise arbitrary

C: 

bonds ~ forms clusters:

— cluster boundary not ~~connected~~ active & connect cluster other points in different clusters

a set of points form a cluster if each point is connected to at least one other

Several bond configs. can give the same cluster

$$\mathrm{Prob}(B) = \prod_{\ell \in B} P_{\ell}(s_i s_j) \prod_{\ell \notin B} (1 - P_{\ell}(s_i s_j)) = w(s \to B)$$

$w(s \to B)$

Cluster: collection of spins

$$\mathrm{Prob}(C) = \prod_{\ell \notin B} (1 - P_{\ell}(s_i s_j)) \cdot [\text{ within clusters they might or might not be connected}]$$

$w(s \to C)$

(under $\ell \notin B$: cluster boundary)

An update of the spin system $s \to s'$ has to satisfy detailed balance:

$$e^{-H(s)} \; W(s \to s') = e^{-H(s')} \underbrace{W(s' \to s)}_{\text{transition / acceptance prob.}}$$

<u>Cluster update</u>: $s \to c \to s'$: transform the spin in one 'cluster', C_0

$$s_i' = s_i \quad \text{if} \quad i \notin C_0$$

$$s_i' = g s_i \quad \text{if} \quad i \in C_0$$

$W(s, c \to s')$: pick C_0, transform accept / reject the change

Does it satisfy detailed balance? We'll show that it satisfies an even stricter condition:

$$e^{-H(s)} \; W(s \to c) \; W(s, c \to s') = e^{-H(s')} \; W(s' \to c) \; W(s', c \to s)$$

for the original detailed balance we could sum over C's on both sides

s & s' differ only in one cluster, C_0

$W(s, c)$ is invariant under $s \to gs$ in pairs

$W(s \to c)$ & $W(s' \to c)$ differ only on the boundary of C_0

$H(s)$ and $H(s')$ differ only on the boundary as well

$$e^{-H(s)}\, W(s \to c) = \prod_{\ell \in \partial C_0} e^{-H_\ell(s_i s_j)} \left(1 - p_\ell(s_i s_j)\right) \cdot \underbrace{\prod_{\ell \notin \partial C_0} (\quad)}$$

$$e^{-H(s')}\, W(s' \to c) = \prod_{\ell \in \partial C_0} e^{-H_\ell(g s_i, s_j)} \left(1 - p_\ell(g s_i, s_j)\right) \underbrace{\prod_{\ell \notin \partial C_0} (}_{\text{same}}$$

DB condition that we need to satisfy:

$$\prod_{\ell \in \partial C_0} e^{-H_\ell(g s_i, s_j)} \left(1 - p_\ell(g s_i, s_j)\right) \cdot W(s, c \to s') =$$

$$\prod_{\ell \in \partial C_0} e^{-H_\ell(g s_i, s_j)} \left(1 - p_\ell(g s_i, s_j)\right) \underbrace{W(s' c \to s)}_{\text{acceptance prob. going from } (s,c) \text{ to } s'}$$

Notation

$$e^{-H_\ell(s_i s_j)} \left(1 - p_\ell(s_i s_j)\right) = e^{-Q(s_i s_j)} \qquad Q \geq H$$

with that DB is simple

$$\prod_{\ell \in \partial C_0} e^{-Q(s_i s_j)}\, W(s, c \to s') = \prod_{\ell \in \partial_0 C} e^{-Q(g s_i, s_j)}\, W(s', c \to s)$$

If we choose $Q = H$, this is just the Metropolis type update

We can do better than that

$$Q(s_i s_j) = \max_{g \in G} \{ \mathcal{H}(g s_i, s_j) \} = \mathcal{H}_{max}$$

Ex: Ising $\quad \mathcal{H}(s_i s_j) = -K s_i s_j = \pm K$ $\quad K>0$ for FM

$$\mathcal{H}(g s_i, s_j) = -K \cdot g \cdot s_i s_j = \pm K$$

$$g = \pm 1$$

$$\max(E) = |K|$$

in that case DG is just

$$W(s, c \to s') = W(s' c \to s)$$

(we can always accept the proposed change)

What is P_e then?

$$P_e(s_i s_j) = 1 - e^{H_e(s_i s_j) - Q_e(s_i s_j)}$$

$$= 1 - e^{+H_e(s_i s_j) - H_{max}} =$$

for Ising $\quad P_e = 0 \quad$ if $\quad s_i \neq s_j \quad \sim s_i s_j < 0$

$$P_e = 1 - e^{-2|K| s_i s_j} \quad \text{if} \quad s_i = s_j$$

Summarize the cluster update for Ising:

- put down bonds with prob.

$$P_e = \begin{cases} 0 & \text{if } s_i s_j < 0 \\ 1 - e^{-2K s_i s_j} & \text{if } s_i s_j > 0 \end{cases}$$

- identify clusters
- update = flip one cluster

drawback: if there are many small clusters, one will flip only a tiny part of the lattice

Generalize:

Consider all the clusters $C_0, C_1 \ldots C_n$

Flip each cluster with probability $\frac{1}{2}$

DB:

$$\prod_{l \in \partial C_\alpha} e^{-He(s_i s_j)} \left(1 - P_e(s_i s_j)\right) w(s, C \to s') =$$

$$\prod_{l \in \partial C_\alpha} e^{-He(g^\alpha s_i s_j)} \left(1 - P_e(g^\alpha s_i s_j)\right) w(s', C \to s)$$

$$g^\alpha = \begin{cases} 1 \\ -1 \end{cases} \quad \text{with } \tfrac{1}{2} \text{ probability}$$

However Q is independent of g^α, so we still get

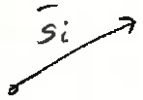
$$w(s, C \to s') = w(s', C \to s)$$

update now:

- put down bonds
- identify clusters
- flip / transform each cluster with $\frac{1}{2}$ probab.

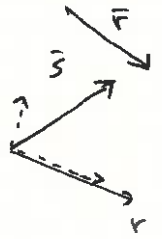
Generalize to O(n) spin models

$$H = -K \bar{S}_i \bar{S}_j$$



n-comp. vector

Choose a direction randomly



decompose $\bar{S} = \bar{S}_{\parallel} + \bar{S}_{\perp}$

flip only $\bar{S}_{\parallel} \to -\bar{S}_{\parallel}$

with $\frac{1}{2}$ probability

$$\bar{S} = \bar{S}_{\parallel} + \bar{S}_{\perp} \longrightarrow \bar{S}' = -\bar{S}_{\parallel} + \bar{S}_{\perp}$$

$$= \bar{S} - 2F(\bar{S}F)$$

$$Q = \max\left\{ -\beta(\bar{S}_i \bar{S}_j) \right\} = \max\left\{ -\beta\left(S_i^{\parallel} S_j^{\parallel} + S_i^{\perp} S_j^{\perp} \right) \right\}$$

$$= \max\left\{ -\beta\left(S_i^{\parallel} S_j^{\parallel} + S_i^{\perp} S_j^{\perp} \right), -\beta\left(-S_i^{\parallel} S_j^{\parallel} + S_i^{\perp} S_j^{\perp} \right) \right\}$$

$$= \beta \left| S_i^{\parallel} S_j^{\parallel} \right| - \beta S_i^{\perp} S_j^{\perp}$$

$$P = 1 - e^{+H(s_i s_j) + Q(s_i s_j)} = \begin{cases} 0 & \text{if } S_i^{\parallel} S_j^{\parallel} < 0 \\ 1 - e^{-2\beta S_i^{\parallel} S_j^{\parallel}} & \text{if } S_i^{\parallel} S_j^{\parallel} > 0 \end{cases}$$

This is the Wolf algorithm

- ~~Set~~ choose a random direction $\bar{F}$
- set up bonds with P above
- flip with $\frac{1}{2}$ prob. spins in a cluster $\parallel$ to $\bar{F}$

Identifying a single cluster

- bound config $b[n][\mu] = -1$
- choose a random site $\bar{n}$ ($n_x + L n_y + L^2 n_z$ ...)
  site = 0 ; check = 0
  - add $\bar{n}$ to cluster array : $cl[\text{site}] = n$
  - add $\bar{n}$ to check arrah : site ++ ; check ++ ; $ch[\text{check}] = n$
- ~~check~~

↳ pick last of check :

$np = ch[\text{check} - 1]$, check --

check the neighbors of np :

if $b[np, \mu] = -1$: new bound

↳ pick $b[np,\mu]$ with prob.

→ ~~b[n~~ no bound : $b[np,\mu] = 0$

- bound $b[np,\mu] = 1$

is $np+\mu$ in cluster?

no -> add $np+\mu$ to $cl[\text{site}] = np+\mu$

site ++

↳ add $np+\mu$ to check

$ch[\text{check}] = np+\mu$

check ++

repeat till check = 0

Cluster Algorithms for the Ising model

BY ULLI WOLFF

HU Berlin

1 Additional Variables

The Ising model is given by the partition function

$$Z = \sum_s \mathrm{e}^{-\beta H(s)}, \tag{1}$$

where s are configurations of spins $s(x) = \pm 1$ on a D-dimensional hypercubic torus, $\beta \geqslant 0$. The energy is

$$-H(s) = \sum_{x\mu} s(x)s(x+\hat{\mu}). \tag{2}$$

A convenient derivation of the Swendsen-Wang and other algorithms exploits the trivial identity

$$\mathrm{e}^{\beta\sigma\sigma'} = \mathrm{e}^{-\beta} \sum_{b=0,1} [\delta_{b,0} + (\mathrm{e}^{2\beta} - 1)\delta_{b,1}\,\delta_{\sigma,\sigma'}] \tag{3}$$

for two spin variables $\sigma, \sigma' \in \{-1, 1\}$. Proof: check all cases. By using the identity on all bonds of the Ising model and summing over bond variables $b_\mu(x) = 0, 1$ we find

$$Z = e^{-\beta VD} \sum_{s,b} \prod_{x\mu} [\delta_{b_\mu(x),0} + (\mathrm{e}^{2\beta} - 1)\delta_{b_\mu(x),1}\,\delta_{s(x),s(x+\hat{\mu})}] := \sum_{s,b} F(s,b), \tag{4}$$

where VD is the number of links. We now have more variables, but clearly if we generate a MC ensemble of (s,b) with probabilities $F(s,b)/Z$ then the s alone are Ising distributed, i.e. for spin observables $A(s)$ we have

$$\langle A(s)\rangle = \frac{1}{Z}\sum_s \mathrm{e}^{-\beta H(s)} A(s) = \frac{1}{Z}\sum_{s,b} F(s,b)A(s). \tag{5}$$

This follows simply by summing over b.

As usual, in a Monte Carlo, we update the variabes one after another, now bonds and spins. A heatbath for the bonds with fixed spins amounts to

- visit a bond $x\mu$
- locally, the probability for $b_\mu(x) = 1$ is

$$p(b_\mu(x) = 1) = \begin{cases} 0 & \text{if } s(x) \neq s(x+\hat{\mu}) \\ 1 - \mathrm{e}^{-2\beta} & \text{if } s(x) = s(x+\hat{\mu}) \end{cases}$$

- $b_\mu(x) = 0$ otherwise.

Note that at frozen spins the bond choices are independent of each other. Now we update spins at fixed bonds. A look at F shows:

- bonds with $b_\mu(x) = 1$ can only occur, where $s(x) = s(x + \hat{\mu})$. Any new choice of spins has to maintain this hard constraint.
- bonds with $b_\mu(x) \neq 1$ do not care about the relative position of their spins. All spin-orientations on these links have equal weight.

It looks as if bonds of strength β had been replaced by bonds of local strengths either ∞ or 0. Some thought shows that the following procedure amounts to a global heatbath (independent sampling) of spins at given bonds:

- solve the bond percolation problem of configuration b, i.e. decompose the sites into clusters c connected by $b_\mu(x) = 1$ bonds.
- choose randomly an orientation ± 1 with equal probability for each cluster
- give this value to all spins in the clusters.

Overall this procedure corresponds to one iteration of the Swendsen Wang update. Due to the auxiliary percolation problem, it is a nonlocal procedure (from the point of view of spins) with its greatly reduced critical slowing down. Note that the computational complexity ist still only of order VD. This is the algorithm proposed long ago be Swendsen and Wang[1].

2 Improved estimator

An additional bonus is an improved or noise-reduced estimator for spin correlations that makes use of the cluster information. We start from (5)

$$\langle A(s)\rangle = \frac{1}{Z}\sum_{s,b} F(s,b)A(s) = \frac{1}{Z}\sum_{s,b} F(s,b)\tilde{A}(b) \tag{6}$$

with

$$\tilde{A}(b) = \frac{\sum_s F(s,b)A(s)}{\sum_s F(s,b)}, \tag{7}$$

which is easy to show. For the two point function we find

$$A(s) = s(x)s(y) \Rightarrow \tilde{A}(b) = \sum_c \theta_c(x,y), \tag{8}$$

with the cluster incidence function θ_c which is one, if both $x \in c$ and $y \in c$ and vanishes otherwise.

It is easy to see that this is an improved estimator. If we assume that $\langle s(x)s(y)\rangle = \varepsilon = \left\langle \tilde{A} \right\rangle$, then the variances are

$$\left\langle (s(x)s(y) - \varepsilon)^2 \right\rangle = 1 - \varepsilon^2, \quad \left\langle (\tilde{A} - \varepsilon)^2 \right\rangle = \varepsilon(1-\varepsilon). \tag{9}$$

1. R.H.Swendsen and J.S.Wang, "Nonuniversal critical dynamics in Monte Carlo simulations," Phys. Rev. Lett. 58, 86 (1987)

All we needed here is $s^2 = 1$ and $\theta_c^2 = \theta_c$. The improvement is dramatic, where we have exponential decay of the 2-point function and ε is small.

Beside furnishing a reduced variance estimator for the correlation, relation (8) makes the effciency of the algortihm plausible: we may also read (8) as saying, that the size of the cluster is given by the size of the correlation length in a statistical sense. Hence we update collectively regions of the extent of a correlation volume. Heuristically it is plausible that this is what it takes to eliminate or significantly reduce critical slowing down.

3 Single cluster algorithm

Here we imagine another variant of the above algorithm. The first step of bond-setting and the implied cluster decomposition is unchanged. But for the spin update, we now proceed as follows:

- choose one of the clusters with a probability that may depend on the clusters in the decomposition in an arbitrary way
- flip all spins in (only) this cluster.

The last step may be considered as a Metropolis proposal with 100% acceptance (the Boltzmann factor at fixed bonds does not change under the flip). This general version is not an efficient algorithm, since one identifies all clusters in an effort that costs $\propto$ volume, but flips only one. But the following special case avoids this:

- pick a random spin
- create **only** the (single cluster) connected to it, i.e. by depth first search. Note that it is legal, to stochastically throw the bonds only as the correponding links are encountered
- flip all spins in (only) this cluster.

Obviously, the single cluster C is here effectively picked among the many Swendsen Wang clusters with a probability proportional to its size $|C|$ (defined as the number of sites in it due to the first step) and without actually buiding them. In $D = 2$ the first step is like throwing a dart on the plane (randomly, absolute beginner). So the effort is given by $|C|$, not by the volume. It has turned out that this preference of updateing larger clusters (without undue cost) leads to even less critical slowing down[2].

Also the improved estimator can be translated. The sum over clusters in (8) is now performed stochastically and we have

$$\langle (s(x)s(y)\rangle = \left\langle \frac{V}{|C|}\theta_C(x,y)\right\rangle, \tag{10}$$

2. U.Wolff, "COLLECTIVE MONTE CARLO UPDATING FOR SPIN SYSTEMS," Phys. Rev. Lett. 62, 361 (1989).

where on the r.h.s. we average over the clusters in the single cluster update steps. Note, that the inverse size ratio is needed to cancel the 'bias' in choosing the single cluster among the (virtual) Swendsen-Wang clusters.

Let us close with a trivial remark. If we succeed in estimating an autocorrelation time τ_{SW} referring to measurements seperated by SW updates and $\tilde{\tau}_{1C}$ for those with single cluster steps, then it is not fair to directly compare them. One rather has to form

$$\tau_{1C} = \tilde{\tau}_{1C} \times \frac{\langle |C| \rangle}{V} \tag{11}$$

to account for the smaller computational complexity. This can then be compared to general 'sweep-structured' algorithms.

```
/*****************************************************************************
  2-D Ising Model simulation using simple Monte Carlo Metropolis algorithm.
  Scans a given range of temperature values computing energy and
  magnetization expectation values.

  Usage:
  Input parameters at prompts, or use the syntax
    echo lat_size t_ini t_fin num_t threshold num_mc_sweeps swps_btwn_meas | ising2d > outfi
le
    eg:
    echo 64 2.29 2.29 1 .1 1000 1 | ising2d
  to get all parameters into simulations at once (ignore prompts).
  See routine init_params() below for parameter descriptions.

  Prompts and diagnostics are sent to stderr, data output to stdout.

  Data output format is:

  3  num_rows    # tags for IDL to know how many columns/rows to read for analysis
  T[1]  Energy(T[1])  Magnetization(T[1])
  T[2]  E(T[2])       M(T[2])
  T[3]  E(T[3])       M(T[3])
  T[4]  E(T[4])       M(T[4])
  ......................
  T[num_rows]  E(T[num_rows])  M(T[num_rows])

  This data can be read/analyzed with accompanying IDL file.

  Ref: Problem 8.4.1, Computational Physcs, N.J.Giordano.
  Homework # 4, Physics 7820.
  Fernando Perez
  10/28/98. Rev. 02/15/01
  Rev. by Archie Paulson 02/21/01
 *****************************************************************************/

#include <iostream.h>
#include <math.h>
#include "utils.hh" // some utilities from Numerical Recipes in C 2nd ed.

//*****************************************************************************
// parameter structure type definition
struct parameters
{
  int n_temps,n_sweeps,lat_size,sweep_gap;
  float temp_ini,temp_fin,d_temp,threshold;
};

//*****************************************************************************
// initialize all parameters of simulation
void init_params(parameters &p)
{
  cerr << "Enter lattice linear size: ";
  cin >> p.lat_size;
  cerr << "Enter initial T for range sweep: ";// T_c=2/ln(1+sqrt(2))~2.269
  cin >> p.temp_ini;
  cerr << "Enter final T for range sweep: ";
  cin >> p.temp_fin;
  cerr << "Enter # of T values to sample: ";
  cin >> p.n_temps;
  // p.threshold is the fraction (number in the 0..1 range) of MC sweeps which
  // are done only for thermalization but not used to make measurements
  cerr << "Enter fraction (0-1) of total sweeps thrown away for thermalization: ";
  cin >> p.threshold;
  cerr << "Enter total # of Monte Carlo sweeps: ";
  cin >> p.n_sweeps;
  cerr << "Enter # of sweeps between measurements: ";
  cin >> p.sweep_gap;
  // calculate step size in Temp range
  if (p.n_temps>1)
    p.d_temp=(p.temp_fin-p.temp_ini)/float(p.n_temps-1);
  else p.d_temp=0;
} // end of init_params()


//*****************************************************************************
// initialize the lattice with ordered array (everybody pointing up)
void init_lattice(short **lat,int lat_size,float &energy,float &mag)
{
  int i,j;

  for(i=0;i<lat_size;i++)
    for(j=0;j<lat_size;j++) lat[i][j]=1;
  mag=lat_size*lat_size;
  energy=-2*mag;
} // end of init_lattice()


//*****************************************************************************
// compute one full Monte Carlo sweep (Metropolis) at a given temperature
// uses Numerical Recipes' good but fast ran1() generator
// measures magnetization and energy, doesn't put error bars on them.
// Note: it expects the energy and magnetization of the lattice at the start
// to be supplied, it doesn't measure them directly (only computes the changes).
void compute_mc(short **spin,parameters par,float temp,
                float &energy_lat,float &mag_lat,
                float &energy_exval,float &mag_exval)
{
  int n_sweep,i,j,threshold,n_ave,size,gap;
  short s_ij;
  long seed=-1;  // for random # generator
  short delta_energy; // takes values +-8, +-4, 0
  float boltz_factor[5] = { exp(8.0/temp),  // lookup table for exp rather
                            exp(4.0/temp),  //    than computing in loop
                            1.0,
                            exp(-4.0/temp),
                            exp(-8.0/temp) } ;
    char tabs[]="\t\t";


  // initialize
  energy_exval=0;
  mag_exval=0;
  size=par.lat_size;  // to avoid accessing structure inside the loop
  gap =par.sweep_gap; //
  threshold=int(par.threshold*par.n_sweeps);
  n_ave=(par.n_sweeps-threshold)/gap;
  // do the required number of MC sweeps
  for(n_sweep=1;n_sweep<=par.n_sweeps;n_sweep++) {
    for(i=0;i<size;i++) // scan the lattice with (i,j)
      for(j=0;j<size;j++) {
        s_ij=spin[i][j]; // local for speed (compiler may optimize anyway)
        // compute energy change of flipping spin s[i][j]
        delta_energy=2*s_ij*(spin[(i-1+size)%size][j]+spin[(i+1+size)%size][j]+
                             spin[i][(j-1+size)%size]+spin[i][(j+1+size)%size]);
        // if energy goes down flip spins, else decide with Boltzmann factor
        if( (delta_energy<=0) ||
            ((boltz_factor[(delta_energy/4 +2)])>ran1(&seed)) ) {
          spin[i][j] = -s_ij;
          energy_lat += delta_energy;
          mag_lat -= 2*s_ij; // s_ij is the OLD value, so we subtract
        }
      }// end loop over lattice
```

```
    // start averaging values from selected lattices:
    if(n_sweep>threshold && (n_sweep%gap==0)) {
      energy_exval += energy_lat;
      mag_exval += mag_lat;
    }
    //cout  << n_sweep << tabs << mag_exval/n_sweep << endl;
  }// end MC sweeps
  // return final averaged values for energy and magnetization per spin
  energy_exval /= float(n_ave*size*size);
  mag_exval /= float(n_ave*size*size);
} // end of compute_mc()

//*************************************************************************
// Thermalize (Metropolis) at a given temperature
// uses Numerical Recipes' good but fast ran1() generator
// measures magnetization, doesn't put error bars on them.
// Note: it expects the energy and magnetization of the lattice at the start
// to be supplied, it doesn't measure them directly (only computes the changes).
void thermalize_mc(short **spin,parameters par,float temp,
                   float &mag_lat,float &mag_exval)

{
  int n_sweep,i,j,threshold,n_ave,size,gap;
  short s_ij;
  long seed=-1;  // for random # generator
  short delta_energy; // takes values +-8, +-4, 0.
  float boltz_factor[5] = { exp(8.0/temp),  // lookup table for exp rather
                            exp(4.0/temp),  //    than computing in loop
                            1.0,
                            exp(-4.0/temp),
                            exp(-8.0/temp) } ;
  char tabs[]="\t\t";

  // initialize
  mag_exval=0;
  size=par.lat_size;  // to avoid accessing structure inside the loop
  gap =par.sweep_gap; //
  threshold=int(par.threshold*par.n_sweeps);
  n_ave=(par.n_sweeps-threshold)/gap;
  // do the required number of MC sweeps
  for(n_sweep=1;n_sweep<=par.n_sweeps;n_sweep++) {
    for(i=0;i<size;i++) // scan the lattice with (i,j)
      for(j=0;j<size;j++) {
        s_ij=spin[i][j]; // local for speed (compiler may optimize anyway)
        // compute energy change of flipping spin s[i][j]
        delta_energy=2*s_ij*(spin[(i-1+size)%size][j]+spin[(i+1+size)%size][j]+
                             spin[i][(j-1+size)%size]+spin[i][(j+1+size)%size]);
        // if energy goes down flip spins, else decide with Boltzmann factor
        if( (delta_energy<=0) ||
            ((boltz_factor[(delta_energy/4 +2)])>ran1(&seed)) ) {
          spin[i][j] = -s_ij;
          mag_lat -= 2*s_ij; // s_ij is the OLD value, so we subtract
        }
      }// end loop over lattice
    // start averaging values after thermalization threshold
    if(n_sweep>threshold && (n_sweep%gap==0)) {
      mag_exval += mag_lat;
    }
    cout  << n_sweep << tabs << mag_lat << endl;
  }// end MC sweeps
  // return final averaged values for magnetization per spin
  mag_exval /= float(n_ave*size*size);
} // end of thermalize_mc()

//****************************** main program ********************************
int main(int argc, char *argv[])
{
  int i;
  char tabs[]="\t\t";
  char therm_flag =0; // flag to look at thermalization
  short **lat;  // matrix holding the spins
  float *energy_ave,*mag_ave,energy_lat,mag_lat,temp;
  parameters par;

  // set simulation parameters
  init_params(par);
  cerr << "\nIsing model simulation.\n";
  if (argc>1 && (argv[1][0] == 't')) {
    therm_flag = 1;
    cerr << "Thermalization measurements.\n ";
  }
  // send diagnostics to stderr, data output to stdout
  // mark the output with dimensions of arrays to be saved
  // so IDL can read them later
  // make data arrays
  lat=smatrix(0,par.lat_size-1,0,par.lat_size-1);
  energy_ave=vector(0,par.n_temps-1);
  mag_ave=vector(0,par.n_temps-1);

  // actual running over the temperature range
  for(i=0,temp=par.temp_ini; i<par.n_temps; i++) {
    cerr <<"T="<< temp << "\t("<<i+1<<" out of " <<par.n_temps<<")"<<endl;
    // initialize lattice "cold" (with +1 everywhere) for each run
    init_lattice(lat,par.lat_size,energy_lat,mag_lat);
    if (therm_flag) {
      cout  << "#m "<< i+1 << endl;
      thermalize_mc(lat,par,temp,mag_lat,mag_ave[i]);
    } else {
      compute_mc(lat,par,temp,energy_lat,mag_lat,energy_ave[i],mag_ave[i]);
      // output data
      cout  << temp << tabs << energy_ave[i] <<tabs << mag_ave[i] << endl;
      //cout  << temp <<tabs << mag_ave[i] << endl;
    }
    temp+=par.d_temp;
  }

  // put simulation info at end of output (if it goes at the start, IDL chokes)
  cout <<";\n; 2-D ISING MODEL SIMULATION\n; Data above is T, E(T), M(T)\n";
  cout <<"; First line marks number of data columns and rows.\n";
  cout <<";\n; SIMULATION PARAMETERS:\n";
  cout <<"; Lattice size     : "<<par.lat_size<<"x"<<par.lat_size<<endl;
  cout <<"; No. of Temps     : "<<par.n_temps<<endl;
  cout <<"; Temp. range      : "<<par.temp_ini<< " to "<<par.temp_fin<<endl;
  cout <<"; No. MC sweeps    : "<<par.n_sweeps<<endl;
  cout <<"; Reject threshold: "<<par.threshold<<endl;
  // free data arrays
  free_smatrix(lat,0,par.lat_size-1,0,par.lat_size-1);
  free_vector(energy_ave,0,par.n_temps-1);
  free_vector(mag_ave,0,par.n_temps-1);

  cerr << "\nIsing model simulation finished.\n\n";
  return(0);
}
//********************** end of file <ising2d_main.cc> ************************
```