Monte Carlo Simulations

The terminology “Monte Carlo” method addresses a wide range of problem solving techniques by using random numbers and the statistics of probability.

Name is indeed taken after the casino in the small Monegasque municipality, Where every game relies upon random events (roulette, dice etc).

One can then name, in principle, any method which uses random numbers to solve a problem a Monte Carlo method.

Monte Carlo usage in Science

Classical Monte Carlo (CMC) – used to obtain samples from a probability distribution to determine, for example, energy minimum structures.

Quantum Monte Carlo (QMC) – random walks can be used to determine quantum-mechanical energies.

Path-Integral Monte Carlo (PMC) – thermodynamics properties can be evaluated from quantum statistical mechanical integrals.

Simulation Monte Carlo (SMC) – algorithms used to evolve configurations based on various acceptance rules.

Monte Carlo usage in Science

Classical Monte Carlo (CMC) – used to obtain samples from a probability distribution to determine, for example, energy minimum structures.

Quantum Monte Carlo (QMC) – random walks can be used to determine quantum-mechanical energies.

Path-Integral Monte Carlo (PMC) – thermodynamics properties can be evaluated from quantum statistical mechanical integrals.

Simulation Monte Carlo (SMC) – algorithms used to evolve configurations based on various acceptance rules.

Molecular Dynamics or Monte Carlo

In Molecular Dynamics, properties are evaluated by tracking them over time. for a given microscopic state, macroscopic properties are calculated as time averages.

These time averages, however, include only the states which occur during the time scale of the MD simulation. Several important issues arise:

1. The length of the MD run is finite (eg. 10s or 100s of ns), and there may be processes/excitations which occur over longer times which would not be included in the MD time averages.

2. One needs to calculate properties averages, but there might not be interest in simulating, or knowledge of, the actual system dynamics (eg. a spin model). A considerably less demanding technique (CPU-wise) can be used to do the job.

In both cases, statistical sampling could be the better method, or MC.

Monte Carlo is NOT another form of dynamics.

Monte Carlo is a SAMPLING method.

The two most important aspects to be decided in Monte Carlo approaches are:

1. WHICH POPULATION TO SAMPLE FROM.

One needs to impose some constraints on the population of states sampled.

2. WITH WHAT PROBABILITY TO SAMPLE.

Biased or unbiased sampling can make a huge difference in efficiency.

What to sample?

The statistical ensemble gives the group of states over which one samples.

$$\begin{tikzcd}\boxed{\begin{array}{c}\textbf{MICROSCOPIC}\\ \text{(atoms, electrons)}\end{array}} \arrow[r, leftrightarrow, "\text{STATISTICAL}", "\text{MECHANICS}"'] & \boxed{\begin{array}{c}\textbf{MACROSCOPIC}\\ \text{Thermodynamics}\end{array}}\end{tikzcd}$$

Sample with Constraints

Fixed Variables

Macroscopic conditions (constant V, T, N) translate as boundary conditions, or constraints, in the microscopic universe.

Microscopic systems are then defined by the fixed thermodynamic variables in the macroscopic world (NVE), (NVT), (NPT) etc.

There are two types of thermodynamic variables:

Extensive variables – scale with size of system (V, N).

Intensive variables – don’t scale with size (T, P, μ)

Intensive variables are the conjugates of extensive variables.

The constraint used to sample the microscopic system is set by the fixed extensive thermodynamic variable.

The sampling probability depends on the relevant Hamiltonian.

The Hamiltonian in microscopic space corresponds to the free energy function in macroscopic space.

The conjugate, extensive and intensive variables, always “work” in pairs.

First law of thermodynamics, in the energy formulation, yields for the work terms, using the conjugate pairs:

$$dU = TdS + \left(-PdV\right) + \mu dN + \ldots$$

S is extensive, T is intensive;	TdS (heat flow term)
V is extensive, P is intensive;	PdV (mechanical work done term)
N is extensive, μ is intensive;	μdN (chemical work term)

One needs to always specify at least one variable for each pair of conjugate variables:

constant S or constant T

constant V or constant P

constant N or constant μ

This is how the constraints for the microscopic systems are defined.

This is also how the so-called valid thermodynamic ensembles are constructed.

In MC, thermodynamic quantities are averages over relevant set (population) of microscopic states (ensembles).

(NVE) – microcanonical ensemble
(NVT) – canonical ensemble
(μVT) – grand-canonical ensemble
(NPT) – isothermal-isobaric ensemble

Ensemble is the collection of all possible microscopic states the system can be in, for a given macroscopic condition.

This defines the population of states, including relevant constraints, which must be sampled in MC simulations.

As ensembles are determined by the extensive variables kept constant, the simplest one to construct is the (NVE) microcanonical ensemble.

It is ideally suited for Newtonian mechanics in a system closed in a box. If the box is closed, N cannot change, the volume is gain fixed, and in the case of Newtonian dynamics, the energy is fixed as well.

This is the reason why the (NVE) ensemble is the most natural ensemble for MD simulations. However, this is not the case in MC, where particle momenta are not involved.

How to sample?

The probability of states in any ensemble is proportional to $e^{-\beta H}$, where H is the Hamiltonian and $\beta = 1/k_B T$.

$$p \sim \exp(-\beta H)$$

This probability has to be normalized by the partition function Z, which is the sum of probabilities over all states ν in the ensemble:

$$Z = \sum_{\nu} \exp(-\beta H_{\nu}) \qquad\qquad p_{\nu} = \frac{\exp(-\beta H_{\nu})}{\sum_{\nu} \exp(-\beta H_{\nu})} = \frac{\exp(-\beta H_{\nu})}{Z}$$

P_{ν}, called the probability distribution function (PDF), yields in this manner the correct probability to sample essentially in any ensemble, provided one knows H.

The microscopic H should include everything that fluctuates in the system. Its correct form can be obtained by taking a Legendre transformation of the entropy of the system (H is essentially a Legendre transform), obtained from 1st law:

$$dS = \frac{1}{T} dU + \frac{P}{T} dV - \frac{\mu}{T} dN + \ldots$$

Note the conjugate pairs in the entropy formulation: (1/T, U), (– P/T, V), (μ/T, N).

The Hamiltonian, which corresponds to the relevant free energy function in macroscopic space, can be obtained for each microscopic ensemble.

Canonical Ensemble (NVT)

First law becomes: $dS = \frac{1}{T} dE$ or $d\left(S - \frac{1}{T} E\right) = 0$

the relevant free energy is F = E – TS, which is the Helmholtz free energy, and the Legendre transform of entropy yields: –F/T = S – E/T.

The Hamiltonian will thus contain only the – E/T term, and the PDF for the canonical (NVT) ensemble takes the following form:

$$p_\nu^{NVT} = \frac{\exp(-\beta E_\nu)}{\sum_\nu \exp(-\beta E_\nu)}$$

Isothermal-Isobaric Ensemble (NPT)

First law becomes: $dS = \frac{1}{T} dE + \frac{P}{T} dV$ or $d\left(S - \frac{1}{T} E - \frac{P}{T} V \right) = 0$

The free energy in this case is:

$$F = E - TS + PV$$

and the Legendre transform of entropy takes the form:

$$-F/T = S - E/T.$$

From this, only the $-(E + PV)/T$ term is taken in the Hamiltonian and the PDF for the isothermal-isobaric (NPT) ensemble becomes:

$$p_\nu^{NPT} = \frac{\exp[-\beta(E_\nu + PV_\nu)]}{\sum_\nu \exp[-\beta(E_\nu + PV_\nu)]}$$

Grand-canonical Ensemble (μVT)

First law becomes: $dS = \frac{1}{T}dE - \frac{\mu}{T}dN$ or $d\left(S - \frac{1}{T}E + \frac{\mu}{T}N\right) = 0$

The free energy for fixed (μ,V,T) becomes:

$$F = E - TS - \mu N$$

and the Legendre transform of entropy for this ensemble is:

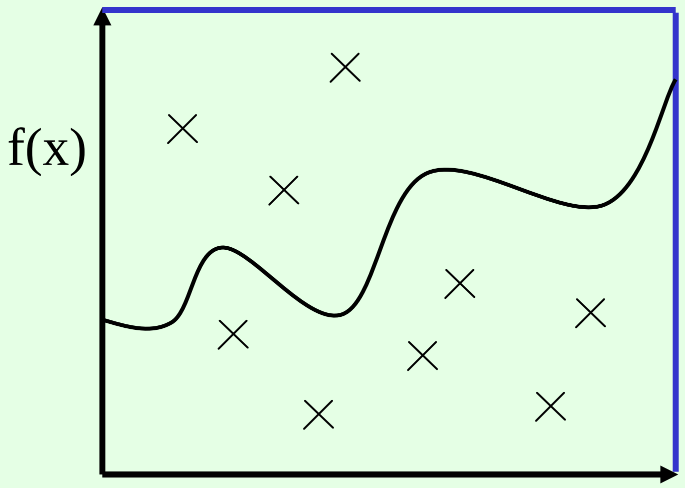
$$-F/T = S - E/T + \mu N/T.$$

Again, from this one takes only the – (E – μN)/T term in the Hamiltonian and the PDF for the isothermal-isobaric (μVT) ensemble becomes:

$$p_\nu^{\mu VT} = \frac{\exp\left[-\beta\left(E_\nu - \mu N\right)\right]}{\sum_\nu \exp\left[-\beta\left(E_\nu - \mu N\right)\right]}$$

Monte Carlo Integration

Originally, Monte Carlo was used as an integration method. Typically, the scheme for integrating a function F(x), consisted in:



- randomly obtain values of x below curve.
- determine value of f for that x.
- accumulate a sum of these values.
- divide the sum by number of trials to obtain the average.

$$I_{est} = \frac{1}{N}\sum_{i}^{N} f(x_i)$$

The procedure was easily extended to functions of two variables and multiple integrals. It is called **SIMPLE SAMPLING**.

Simple Sampling in MC (Simple MC)

The aim in MC simulations is to calculate average thermodynamic properties, $\langle A(r^N) \rangle$, which can be done by evaluating multidimensional integrals over the 3N degrees of freedom in an N particle system:

$$\langle A(r^N) \rangle = \int A(r^N) p(r^N) dr^N$$

where $p(r^N)$ is the appropriate PDF in the respective ensemble.

Here, one can concentrate on the (NVT) ensemble for two reasons:

1. **ALL OTHER ENSEMBLES** follow the same rational/approach as in (NVT).

2. **The NVT ensemble is the natural choice for MC simulations.**

In MD, Newton's EOM lead naturally to energy conservation, hence the NVE selection.

In MC, until recently, it was not possible to perform calculations in the NVE ensemble due to the absence of kinetic energy. Temperature, however, can be easily kept constant in the PDF, and the NVT-MC is simplest to implement.

Simple Sampling in MC (Simple MC)

As shown, the PDF in the NVT ensemble takes the form:

$$p(r^N) = \frac{\exp\left[-\beta E(r^N)\right]}{\int \exp\left[-\beta E(r^N)\right] dr^N}$$

These integrals cannot be evaluated analytically or numerically. Typical schemes for 3N-dimensional integrals require m^{3N} function evaluations, where m is the number of points required to evaluate the integral in each dimension.

In simple MC, a large number of trial configurations r^N are generated and the integrals are replaced by summations over a finite number of configurations:

$$\left\langle A(r^N) \right\rangle = \frac{\sum_{i=1}^{N_{trial}} A_i(r^N) \exp\left[-\beta E_i(r^N)\right]}{\sum_{i=1}^{N_{trial}} \exp\left[-\beta E_i(r^N)\right]}$$

Simple Sampling in MC (Simple MC)

With simple sampling, each trial configuration r^N corresponds to a randomly chosen state (point) ν. If one randomly picks M states, they need to be weighted with the correct probability p_ν:

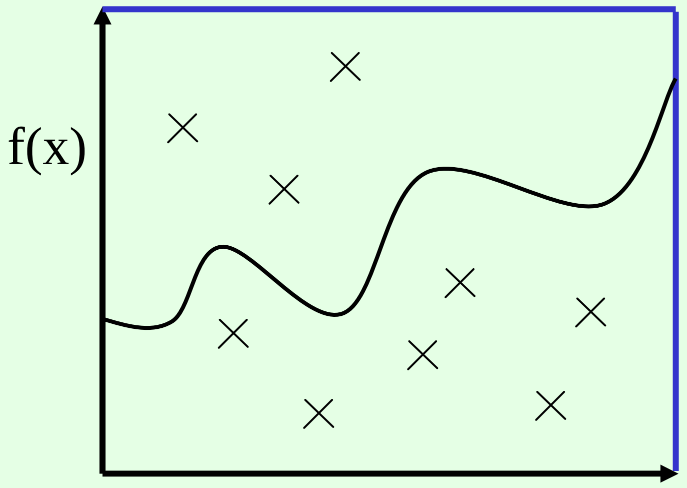
$$\langle A \rangle = \sum_{\nu=1}^{M} p_\nu A_\nu \qquad\qquad p_\nu = \frac{\exp(-\beta E_\nu)}{\sum_{\nu=1}^{M} \exp(-\beta E_\nu)} = \frac{\exp(-\beta E_\nu)}{Z_{M \neq NVT}}$$

Simple sampling does not work. The reason is states are picked essentially in proportion to their degeneracy. The higher the energy, the more states at that energy. One samples a great number of states but not the relevant ones.

This random, unbiased sampling of states yields too many configurations with low weight, or very small Boltzmann factor, which make very little contribution to the average which needs to be calculated.

One needs to bias the sampling method: **IMPORTANCE SAMPLING.**

Importance Sampling



$$I_{est} = \frac{1}{N}\sum_{i}^{N} f(x_i)$$ Simple Sampling

$$I_{est} = \sum_{i}^{N} f(x_i)/p(x_i)$$ Importance Sampling

In importance sampling points are chosen according to the anticipated importance of the value to the function (contribution it makes) and weighted by the inverse of the probability of choice.

The difference is that in importance sampling one on longer uses a simple average of all points sampled. In Importance sampling the sampling is biased by the use of a weighted average.

Importance Sampling

In MC, importance sampling translates into biasing the sampling towards the important, relevant, low energy states.

In other words, one picks states with a probability proportional to exp(–βE), instead of randomly picking them and weighing them later by a probability.

Random sample

Probability weighted sample

$$\langle A \rangle = \sum_{\nu=1}^{M} \frac{\exp(-\beta E_\nu)}{\sum_{\nu=1}^{M} \exp(-\beta E_\nu)} A_\nu \quad \Longrightarrow \quad \langle A \rangle = \sum_{\nu=1}^{M} A_\nu$$

If one has to calculate properties for which high energies are needed, sampling could be biased for those states.

Relevant thermodynamic ensembles fluctuate around states with low energy, so importance sampling is used in MC to sample mostly these states.

Markov Chains

Used to construct probability weighted samples.

A Markov chain is a sequence of trials in which the outcome of successive trials depends only on the immediately preceding trial.

The procedure ensures that one “walks” through the phase space and “visit” each state with proper probability.

In Markov chains, a new state is accepted only if it is more “favorable” than the existing state. For simulations of ensembles, this usually means that the new trial state is lower in energy.

In MC simulations Markov chains are required to accurately determine the properties of the system in the finite time available for simulation. The role of Markov chains is to sample those states which make the most significant contributions to the calculated thermodynamic averages.

Metropolis Sampling

Generates Markov chains which construct the probability weighted sample to explore the thermodynamical behavior around the energy minimum.

Metropolis sampling biases the generation of configurations towards those which make the most significant contribution to the integral/average of interest.

It generates states with a probability of $\exp[-\beta E(r^N)]$ and counts each of them equally.

This is in contrast to the simple MC integration/sampling, which generates states with equal probability and assigns them a weight of $\exp[-\beta E(r^N)]$.

Metropolis sampling generates Markov chains which satisfy to conditions:

1. The outcome of each trial belongs to a finite set of possible outcomes, called he state space, $\{\Gamma_1, \Gamma_2, \ldots, \Gamma_m, \Gamma_n, \ldots.\}$.
2. The outcome of each trial depends only on the outcome of the immediately preceding trial.

Metropolis Algorithm

The important aspect here is the transition probability, π_{mn}, which is the probability go from state Γ_m to Γ_n. Several conditions must be satisfied.

π_{mn} is a stochastic matrix, i.e. its rows must add to 1, $\sum_m \pi_{mn} = 1$ so that:

$\sum_m \rho_m \pi_{mn} = \rho_n$ where ρ_m and ρ_n are the probability densities for states m and n.

This is the general condition for an irreducible, or ergodic, Markov chain, in which every state can eventually be reached from another state. The elements of the matrix can be found by imposing the "microscopic reversibility" condition:

$$\rho_m \pi_{mn} = \rho_n \pi_{nm}$$

Summing over all states m, and using rule for π_{mn}, general condition is regained:

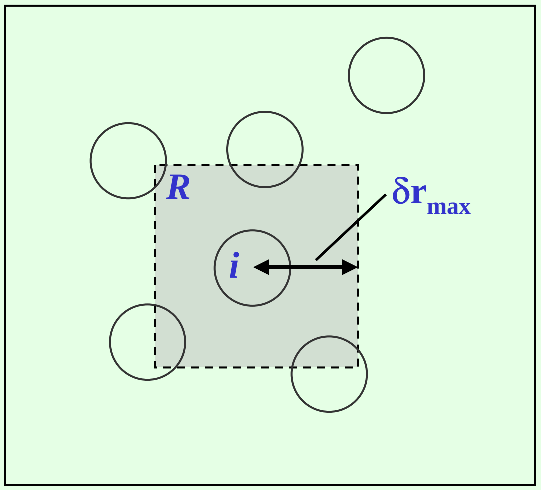
$$\sum_m \rho_m \pi_{mn} = \sum_m \rho_n \pi_{nm} = \rho_n \sum_m \pi_{nm} = \rho_n$$

Metropolis Algorithm

In 1953 Metropolis implemented first such scheme for distinct states m and n:

$\pi_{mn} = \alpha_{mn}$	$\rho_n \geq \rho_m$	$m \neq n$
$\pi_{mn} = \alpha_{mn}(\rho_n/\rho_m)$	$\rho_n < \rho_m$	$m \neq n$

where α_{mn} is the conditional probability of choosing n as the trial state. Method uses the condition that $\alpha_{mn} = \alpha_{nm}$, and schematically can be seen as:



State n obtained from state m by moving atom ***i*** to any point in ***R*** with uniform probability.

Size of square is $2\delta r_{max}$, centered on atom ***i*** (cube in 3D). In ***R***, there are a large, but finite number, $N_{\boldsymbol{R}}$, of possible new n states, Γ_n, denoted as r_i^n. The Metropolis scheme uses the following conditional probability:

$\alpha_{mn} = 1/N_{\boldsymbol{R}}$	if $r_i^n \in \boldsymbol{R}$.
$\alpha_{mn} = 0$	if $r_i^n \notin \boldsymbol{R}$.

Metropolis Algorithm

The aim is to compute <A> over measurements of A for configurations which are generated according to $p(r^N)$:

$$\left\langle A(r^N)\right\rangle = \int A(r^N)\frac{\exp\left(-\frac{U(r^N)}{k_BT}\right)}{Z}dr^N \qquad Z = \int \exp\left(-\frac{U(r^N)}{k_BT}\right)dr^N$$

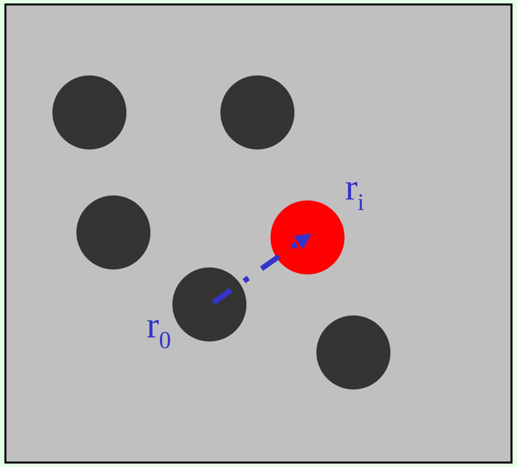
To generate configurations according to desired $p(r^N)$, the algorithm in Metropolis MC uses a Markov chain to sample the phase space with the ensemble distribution and a transition probability, to go from state **m** to state **n** equal to **1** if the move is downhill in energy ($\Delta U = U_{nm} = U_n - U_m < 0$).
If the move is uphill ($\Delta U > 0$), the move is accepted with a probability defined by the ratio of probabilities of initial and final states:

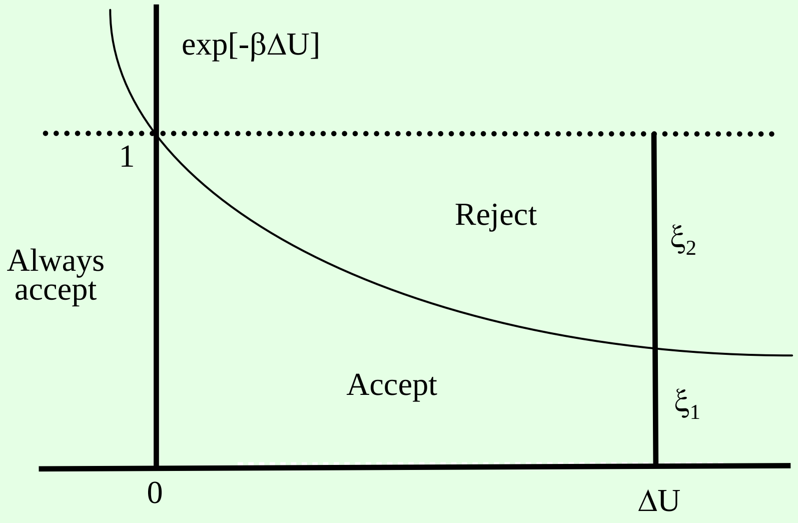
$$\frac{\rho_n}{\rho_m} \equiv \frac{p_n}{p_m} = \frac{\frac{1}{Z}\exp\left(-\frac{U_n}{k_BT}\right)}{\frac{1}{Z}\exp\left(-\frac{U_m}{k_BT}\right)} = \exp\left(-\frac{U_n - U_m}{k_BT}\right) = \exp\left(-\frac{U_{nm}}{k_BT}\right)$$

Metropolis Monte Carlo

1. ***Assign initial position*** to particles & calculate U.
2. ***Move one particle randomly*** & calculate new U' and $\Delta U = U' - U$.
3. If $\Delta U < 0$ - ***accept*** move.
4. If $\Delta U > 0$ - ***accept*** move if $\xi < \exp[-\beta \Delta U]$; $\xi \in (0,1)$ – random number.
5. If ***move rejected*** - take the old configuration as the new one
   - repeat 2 - 4 procedure for another arbitrarily chosen particle.
6. For each new configuration ***evaluate*** $\langle A \rangle$.

7. ***Repeat*** the whole procedure a few million times for adequate statistic





Algorithm 1: Basic Metropolis NVT MC Program

```
program mc                                  basic Metropolis algorithm

do icycl = 1, ncycl                         perform ncycl MC cycles
    call mcmove                             displace particle
      if (mod(icycl, nsamp) .eq. 0)
         call sample                        sample averages
enddo

end
```

Comments:

Typically, MC simulations are performed in cycles. During each cycle, a displacement is attempted for every particle.

The atom to be displaced can be chosen randomly or alternatively.

It is also possible to displace every atom and apply the acceptance criterion to the combined move.

Algorithm 2: Attempt to displace particle

```
subroutine mcmove                                  attempts to displace particle

o = int(ranf()*npart) + 1                          select a particle at random
  call ener(x(o), eno)                             energy old configuration
xn = x(o) + (ranf() – 0.5)*delr                    give particle random displacement
  call ener(xn, enn)                               energy new configuration
     if (ranf() .lt. exp(-beta*(enn – eno))        check acceptance rule
          x(o) = xn                                replace x(o) by xn
return
end
```

Comments:

There is a maximum allowed displacement (dMax). The choice of dMax will affect the acceptance rate (50% is most often derired).

Small values of dMax improve acceptance rate but slow the sampling of the phase space. Conversely, large values for dMax will reduce acceptance rate.

Algorithm 3: Use of Verlet List in a MC move

```
subroutine mcmove_verlet                      attempts to displace a particle
                                              using a Verlet list

o = int(ranf()*npart) + 1                     selects a particle at random

  if (abs(x(o) – xv(o)) .gt. (rv – rc)/2)     check to make new list
     call new_vlist
call en_vlist(o, x(o), eno)                   energy old configuration
xn = x(o) + (ranf() – 0.5)*delr               random displacement

  if (abs(xn – xv(o)) .gt. (rv – rc)/2)       check to make new list
     call new_vlist
call en_vlist(o, xn, enn)                     energy new configuration
arg = exp(-beta*(enn – eno)
  if (ranf() .lt. arg)                        check acceptance condition
     x(o) = xn                                if accepted, replace x(o) with xn

return
end
```

Algorithm 4: Calculating energy using Verlet lists

```
sunroutine en_vlist (i, xi, en)                      calculates energy using
                                                     the Verlet list
en = 0

   do jj = 1, nlist(i)                               loop over the particles in list
       j = list (i, jj)                              next particle in the list
            en = en + enij(i, xi, j, x(j))           get the energy
   enddo

return
end
```

Comments:

As in MD, averages in MC simulations are only accumulated after reaching equilibrium.

The number of required cycles for equilibration is not known beforehand. It is safe to disregard a large number of early cycles.

Algorithm 5: Use of Cell list in MC move

```
subroutine mcmove_neigh                      attempts to displace particle
                                             using a cell list

   call newnlist(rc)                         make the cell list
o = int(ranf()*npart) + 1                    select a particle at random

   call en_nlist(o, x(o), eno)               calculate energy old configuration
xn = x(o) + (ranf() – 0.5)*delr              give particle random displacement

   call en_nlist(o, xn, enn)                 calculate energy new configuration
arg = exp(-beta*(enn – eno))

   if (ranf() .lt. arg)                      check acceptance rule
            x(o) = xn                        if accepted, replace x(o) by xn

return
end
```

Algorithm 6: Calculate energy using Cell list

```
subroutine ennlist (i, xi, en)                calculates energy using cell list

en = 0
icel = int(xi/rn)                             determine the cell number

   do ncel = 1, neigh                         loop over the neighbor cells
      jcel = neigh(icel, ncel)                number of the neighbor
         j = hoc(jcel)                        head of chain in cell jcel

            do while (j .ne. 0)               loop over particles in cell
               if (i .ne. j)
                  en = en + enij(i, xi, j, x(j))   get the energy
                  j = link_l(j)               next particle in the list
            enddo
   enddo

return
end
```

Algorithm 7: MC using Combination of Verlet and Cell lists

```
subroutine mcmove_clist                      displace a particle using a combined list

o = int (ranf()*npart) + 1                   select a particle at random

   if (abs(x(o) – xv(o)) .gt. (rv – rc))     check to make a new list
       call new_clist
call en_vlist(o, x(o), eno)                  energy old configuration using Verlet
xn = x(o) + (ranf() – 0.5)*delr              random displacement

   if (abs (xn – xv(o)) .gt. (rv – rc))      check to make new list
       call new_clist
call en_vlist(o, xn, enn)                    energy new configuration using Verlet
arg = exp (-beta*(enn – eno))

   if (ranf() .lt. arg)                      check acceptance condition
       x(o) = xn                             if accepted, replace x(o) by xn

return
end
```

Smarter Monte Carlo

In conventional MC, all particles are moved with equal probability in randomly chosen directions.

The Metropolis method can be extended to achieve considerably even more efficient sampling:

$$\pi_{mn} = \alpha_{mn} \qquad \alpha_{nm}\rho_n \geq \alpha_{mn}\rho_m \qquad m \neq n$$

$$\pi_{mn} = \alpha_{mn}(\alpha_{nm}\rho_n/\alpha_{mn}\rho_m) \qquad \alpha_{nm}\rho_n < \alpha_{mn}\rho_m \qquad m \neq n$$

It can be shown that microscopic reversibility holds even for $\alpha_{mn} \neq \alpha_{nm}$. Markov chains can be generated, and moves from state m to state n, according to α_{mn}, are accepted with a probability given by $\min(1, \alpha_{nm}\rho_n/\alpha_{mn}\rho_m)$.

Preferential sampling – sampling different regions more often than others.
Force-bias Monte Carlo (FBMC) – biasing the movement of particles in the direction of forces acting on it.
Smart Monte Carlo (SMC) – motion due to random as well as systematic forces.
Virial-bias Monte Carlo – FBMC and SMC in NPT ensemble.