MÉTODOS DE COMPUTACIÓN EN FÍSICA DE LA MATERIA CONDENSADA

Máster en Física de la Materia Condensada y Nanotecnología

Curso 2010-2011

Rafael Delgado Buscalioni

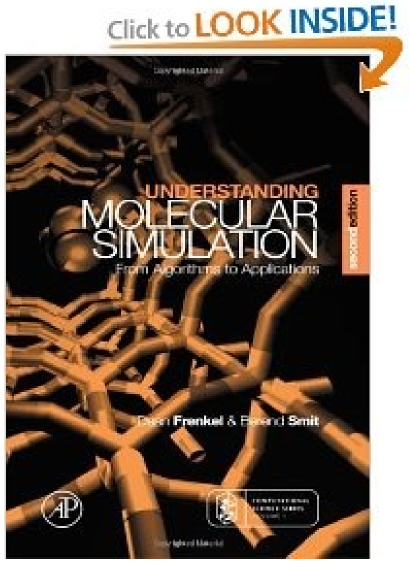
Métodos aplicados a Mecánica Clásica

Course material on the following web page:

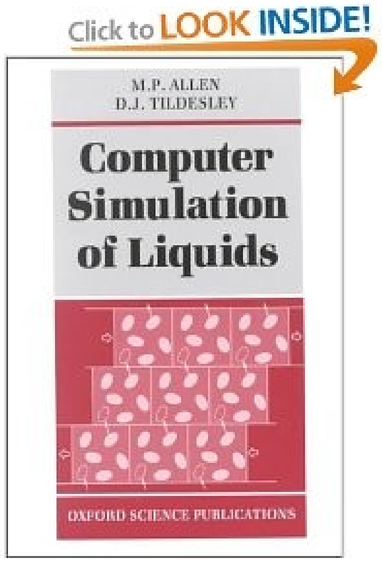
http://www.uam.es/otros/fmcyn/MetodosComputacionales.html

Reference books

Understanding Molecular Simulations
Daan Frenkel and Berend Smit
Academic Press (second Edition) (2002)



Monte Carlo
Free Energy, etc..



Computer simulations of Liquids
M.P Allen and D.J Tildesley
Oxford Science Publi. (1987)

Molecular Dynamics
(Equlibrium and non-equilibrium)
Monte Carlo, Brownian Dynamics, etc.

1. A BRIEF INTRODUCTION TO COMPUTATION IN CONDENSED MATTER PHYSICS

Computational methods essential to explore the behaviour of condensed phases of matter

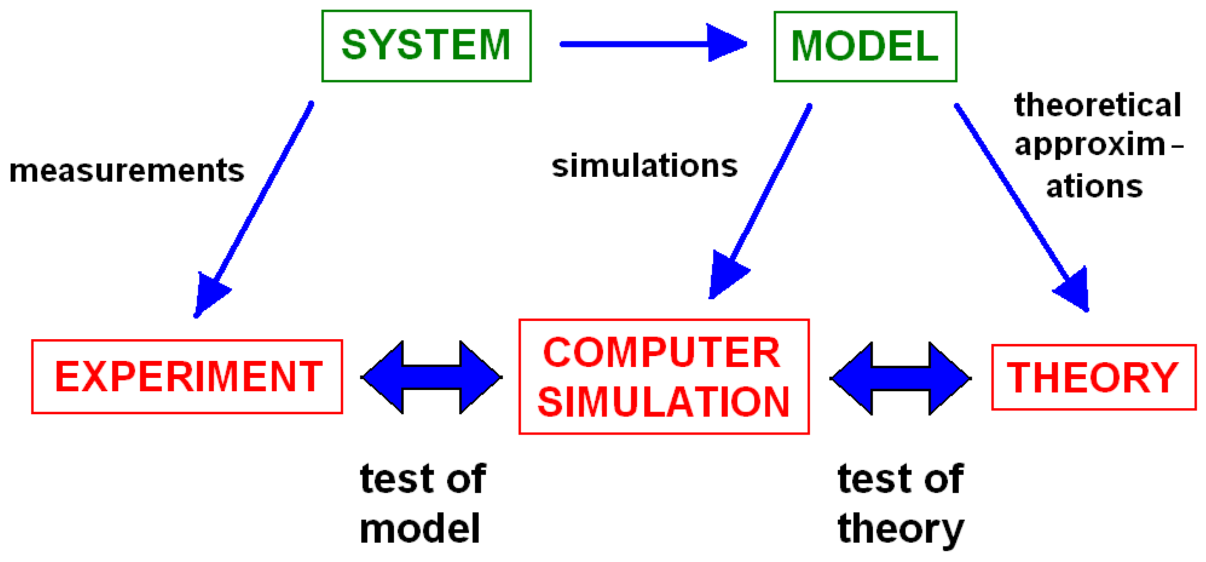
We review methods to solve dynamics of particle systems using the methods of classical and quantum mechanics

$\longrightarrow$ <u>COMPUTER SIMULATION</u>

Two basic methods:

- **Monte Carlo** (MC). Ensemble averages
- **Molecular Dynamics** (MD). Time averages

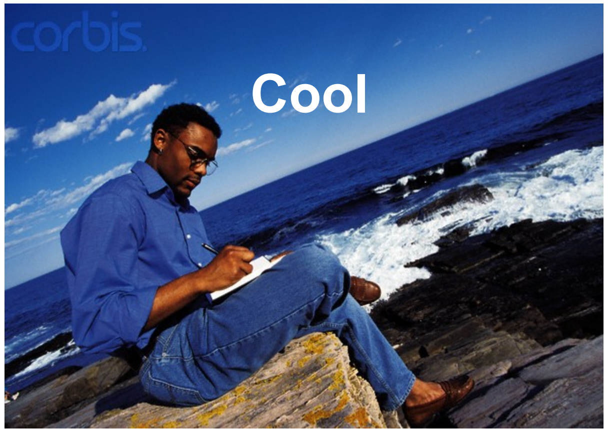
ROLE PLAYED BY COMPUTER SIMULATION



How to **start** a code

Bad...

...leads to **Desperado Coding**

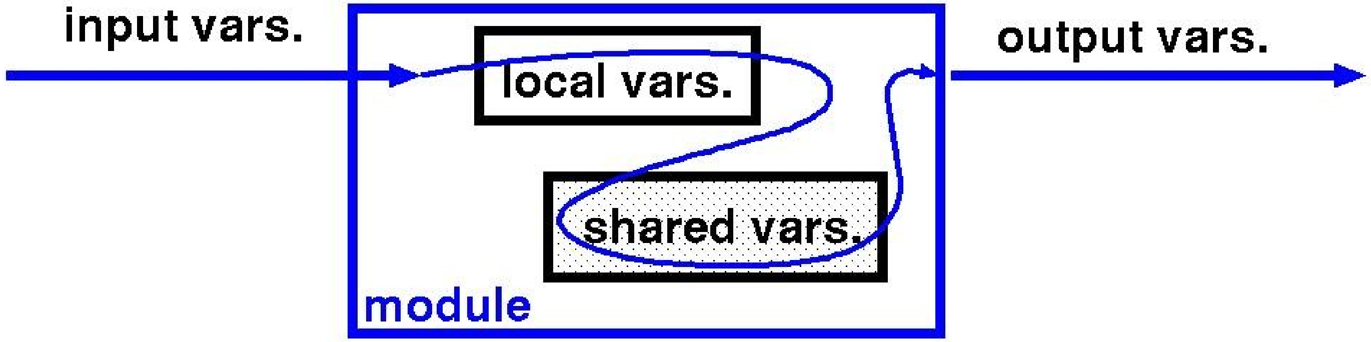


Think modular

Modular programming

Module (subroutine):
A closed set of operations

- Perform **only one** task
- **Transferable** to other codes:
  => Independent and Generic



Types of modules:

- **Evaluate** collective quantities (e.g. temperature, total energy)
- **Update** shared variables (forces, velocities, positions,..)
- **Sample**

Evaluate a collective quantity (e.g. temperature)

```
subroutine temperature(iout)
      implicit none
      include 'parameters.inc'
      include 'velocities.inc'
      integer i,iout
      real Temp_x,Temp_y,Temp_z,TEMP

      temp_x=0.
      temp_y=0.
      temp_z=0.
      do i=1,np
         temp_x=temp_x+ vx(i)*vx(i)
         temp_y=temp_y+ vy(i)*vy(i)
         temp_z=temp_z+ vz(i)*vz(i)
      end do

      TEMP=(TEMP_x+TEMP_y+TEMP_z)/float(3*np)

      write(IOUT,*) 'TEMP',temp_x/float(np),
     &                     temp_y/float(np),
     &                     temp_z/float(np)
      return
      end
```

$\longrightarrow$ load shared variables

$\longrightarrow$ perform operations with shared variables, but **not modify them.**

Update shared vars.

(e.g. integrate eq. of motion of particles)

```
      subroutine movea(dt)
      implicit none
c     ****************************************
c      FIRST PART OF VELOCITY VERLET TIME INTEGRATOR
c     ************************************
      include 'parameters.inc'
      include 'nparticles.inc'
      include 'positions.inc'
      include 'velocities.inc'
      include 'forces.inc'

      integer i
      real   dt,dt2
      dt2   = dt/2.0

      do 100 i = 1, np
         vx(i)=vx(i)+dt2*fx(i)
         vy(i)=vy(i)+dt2*fy(i)
         vz(i)=vz(i)+dt2*fz(i)

         rx(i)  = rx(i)  + dt*vx(i)
         ry(i)  = ry(i)  + dt*vy(i)
         rz(i)  = rz(i)  + dt*vz(i)

c ** periodic boundaries

        rx(i)=rx(i)-anint(rx(i)/lx)*lx
        ry(i)=ry(i)-anint(ry(i)/ly)*ly
        rz(i)=rz(i)-anint(rz(i)/lz)*lz

 100  continue

      return
      end
```

Sampling

histogram of x : histo
mean values for y(x) : yhisto

initialization call
sw=0

sampling calls
sw=1

output call(s)
sw=2

```
subroutine histogram(sw,x,y,xmin,xmax,iout)
 implicit none
 integer sw,nbin,nbin,ibin,iout
 real x,y,xL,dum,dx,xmax,xmin
 parameter (nbin=200)       ! number of bins
 real  histo(0:nbin)
 real  yhisto(0:nbin)
 integer icont,i
 save histo,yhisto,icont,xmax,xmin,dx,XL       !local variables need to be saved
 if(sw.eq.0) then
    do i=1,nbin                  !initialization to zero
       histo(i)=0.
       yhisto(i)=0.
    end do
    icont=0
    xL=xmax-xmin                 ! range of x
    dx=xL/real(nbin)             ! bin size -> dx
 else if(sw.eq.1) then
    icont=icont+1                ! counter, always count
    dum=(x-xmin)/xL
    ibin=int(nbin*dum)
    if(ibin.lt.nbin.and.ibin.gt.0) then       ! check x-range is ok
       yhisto(ibin)=yhisto(ibin)+y
       histo(ibin)=histo(ibin)+1.
    else
       write(*,*) 'warning x out of range',x
    end if
 else if(sw.eq.2) then     !normalize and write results
    open(iout,file='histogram.res')
    do ibin=0,nbin
       if(histo(ibin).ne.0) yhisto(ibin)= yhisto(ibin)/histo(ibin)
       histo(ibin)= histo(ibin)/float(iCONT)/DX
       write(iout,*) (ibin+0.5)*dx+xlimit(1),histo(ibin),yhisto(ibin)
    end do
    close(iout)

 end if
 return
 end
```

# A modular program
for molecular dynamics (MD)

```
      program md
      implicit none
      include 'parameters.inc'    
      include 'init.inc'
c-------------------------------
      call init
      call mean_thermo(0)
      call histogram(0)
      call force         !initial forces
      if(zero_momentum) call set_zero_momentum
      call save_config

      ttime=0
      step=0
      nstep = int(tmax/dt)
      do while (step.lt.nstep)
         step = step + 1
         call movea(dt)
         call force
         call moveb(dt)

c     *** SAVE ***
         if(mod(step,nsave).eq.0) call save_config
c     *** SAMPLE ****
         if(mod(step,nsamp).eq.0) then
            call total_momentum(6)
            call mean_thermo(1)
            call histogram(1)
         end if
c     *** WRITE OUTPUT ***
         if(mod(step,nbitac).eq.0) then
            call histogram(2)
         end if

      end do
      end
```

global parameters

shared vars. required for the main program

Data initializations

MD time loop

include file init.inc

integer nstep,nbitac,nsave,nsamp
real dt ,tmax
common/initio1/nstep,nbitac,nsave,nsamp
common/initio2/dt,tmax
real tempinit
common/temperature_input/tempinit
logical zero_momentum
common/init_momentum_zero/zero_momentum

Compiling with makefile

1> touch makefile
2> make -f make.md

file: make.md

```
F77=gfortran
F77FLAGS= -O3
FILE= bin/mdcode
OBJ= md.f init.f force.f move.f mean_thermo.f set_zero_momentum.f
total_momentum.f histogram.f save_config.f
$(FILE): $(OBJ)
        $(F77) $(F77FLAGS) $(OBJ) $(LDFLAGS) -o $(FILE)
$(OBJ): makefile
```

The Monte Carlo Method

Any method using random numbers

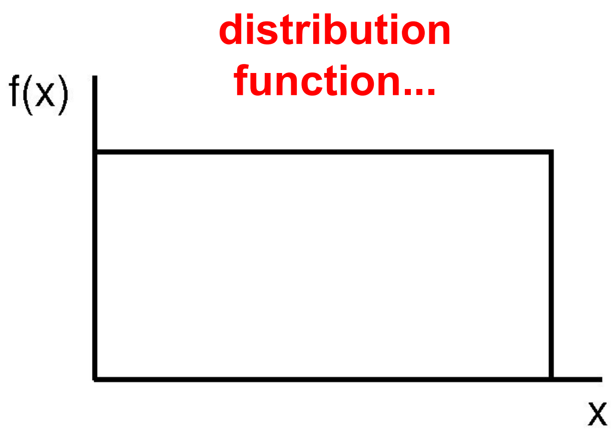
We would like to know how to generate random deviates x, distributed according to the probability function $f(x)$ in the interval $[a, b]$:

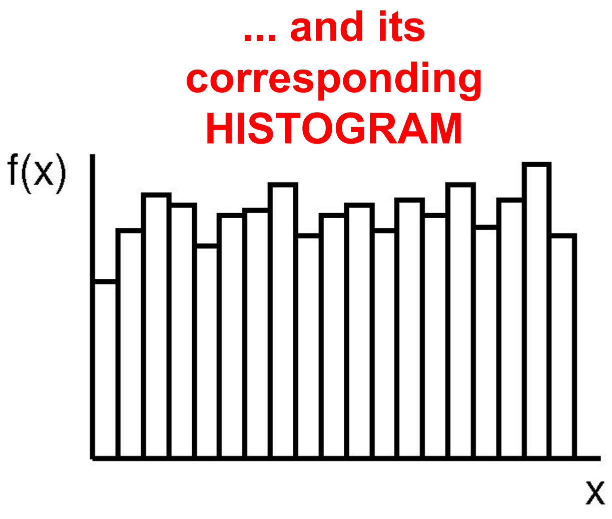
$$f(x)dx = \text{probability that } x \text{ lies within the interval } [x, x + dx]$$

All methods rely on the possibility to generate uniformly distributed random numbers, i.e. $f(x) = 1$ in the interval $[0, 1]$

1. How to generate uniformly distributed random numbers

All `FORTRAN` compilers have intrinsic functions to generate uniformly distributed random numbers, but very often they are not good enough





A possible method: The digits in the product of the two integer numbers

$$12345 \times 65539 = 809078955$$

look random. If we truncate the last five digits, we get

$$\begin{aligned} 08090|78955 &\rightarrow 78955 \times 65539 = 5174631745 \\ 51746|31745 &\rightarrow 31745 \times 65539 = 2080535555 \\ 20805|35555 &\rightarrow 35555 \times 65539 = \ldots \end{aligned}$$

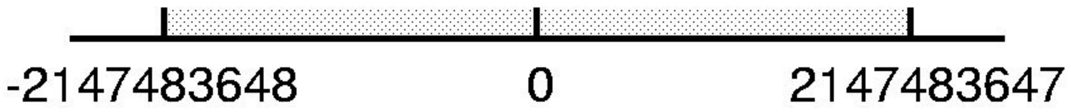
12345 is the *seed*, while 65539 is called the *base*

Dividing by 10^5, we obtain the sequence of numbers

$$0.78955, \quad 0.31745, \quad 0.35555, \ldots$$

which are *apparently* random; these are *pseudo-random numbers*

Special care that multiplication gives numbers in defined interval; for example, with 4-byte (32-bit) integers



Possible subroutine (the one we will use)

```
!
!  Random number generator
!
subroutine ggub(dseed,r)
real*8 z,d2p31m,d2pn31,dseed
data d2p31m/2147483647./,d2pn31/2147483648./
z = dseed
z = dmod(16807.*z,d2p31m)
r = z / d2pn31
dseed = z
end
```

16807 is the base
`dseed` is the seed (provided by the user)
`r` is a random number in $[0,1]$ (NOTE: single–precision variable)

Example of main program:

```
implicit real*8(a-h,o-z)
real*4 r
dseed=527321d0
call ggub(dseed,r)
end
```

2. How to generate random numbers distributed according to a general $f(x)$

(IMPORTANCE SAMPLING)

We review three methods

- **Acceptance–rejection method**

  Not very efficient (it involves generating 2 random numbers to obtain 1 number)
  If x and y are uniform random numbers, then

$$\{x / f(x) \leq y\}$$

  is distributed according to $f(x)$. The steps are:

  1. we generate uniform numbers x and y in $[a, b]$
  2. if $f(x) \leq y$, we accept x; otherwise x is rejected
  3. we go to step 1

- **Inversion method**

  This method uses the *cumulative function*:

$$P(x) = \int_a^x dx' f(x'), \qquad P(a) = 0, \quad P(b) = 1$$

  $P(x)$ is a monotonically increasing function, so it can be inverted to give $x = P^{-1}(y)$

  Then, if y is uniform in $[0, 1]$, then x is distributed with $f(x)$ in $[a, b]$

- **Method of Metropolis et al.**

  Very general and elegant method, very important in statistical and condensed–matter physics

  A sequence of numbers is generated

  $$x^{[0]}, x^{[1]}, x^{[2]}, \ldots$$

  such that, asymptotically, the numbers are distributed according to $f(x)$

  We begin with a *seed*, $x^{[0]}$, and iteratively obtain $x^{[n]} \quad \rightarrow \quad x^{[n+1]}$ by applying the following algorithm:

  – Obtain a <u>test</u> number as

  $$x^{[\mathrm{t}]} = x^{[n]} + \zeta \Delta x$$

  where $\zeta \in [-1, 1]$ is a uniform random number, and Δx a constant

  – Accept $x^{[\mathrm{t}]}$ as a new member of the sequence of random numbers with probability

  $$r = \frac{f(x^{[\mathrm{t}]})}{f(x^{[n]})}$$

  which means:

  $$\begin{cases} \text{si } r > \xi \quad \text{test value accepted, i.e.} & x^{[n+1]} = x^{[\mathrm{t}]} \\ \text{si } r < \xi \quad \text{test value not accepted, i.e.} & x^{[n+1]} = x^{[n]} \end{cases}$$

  where $\xi \in [0, 1]$ is a uniform random number

A few important points to bear in mind:

- Numbers are distributed according to $f(x)$, but only *asymptotically*
  A *warm–up* period, long enough to reach the asymptotic regime, is needed

- The value of Δx is adjusted so that approximately 50% of the tested numbers are accepted
  This insures a quick convergence to the asymptotic regime

- The method only rely on the ratio $r = f(x^{[\mathrm{t}]})/f(x^{[n]})$, which does *not* depend on the normalisation of the distribution function

  It is this feature that makes the Metropolis et al. algorithm so useful in applications to statistical physics

- The Metropolis et al. algorithm may be easily extended to more than one random variables

  $$\mathrm{x} = (\mathrm{x}_1, \mathrm{x}_2, \ldots)$$

  The sequence is

  $$\mathbf{x}^{[0]}, \mathbf{x}^{[1]}, \ldots$$

  and test vectors are generated as

  $$\mathbf{x}^{[\mathrm{t}]} = \mathbf{x}^{[n]} + \zeta \Delta \mathbf{x}$$

  following exactly the same steps as before

As an example, we apply the method to

$$f(x) = ce^{-x^4},$$

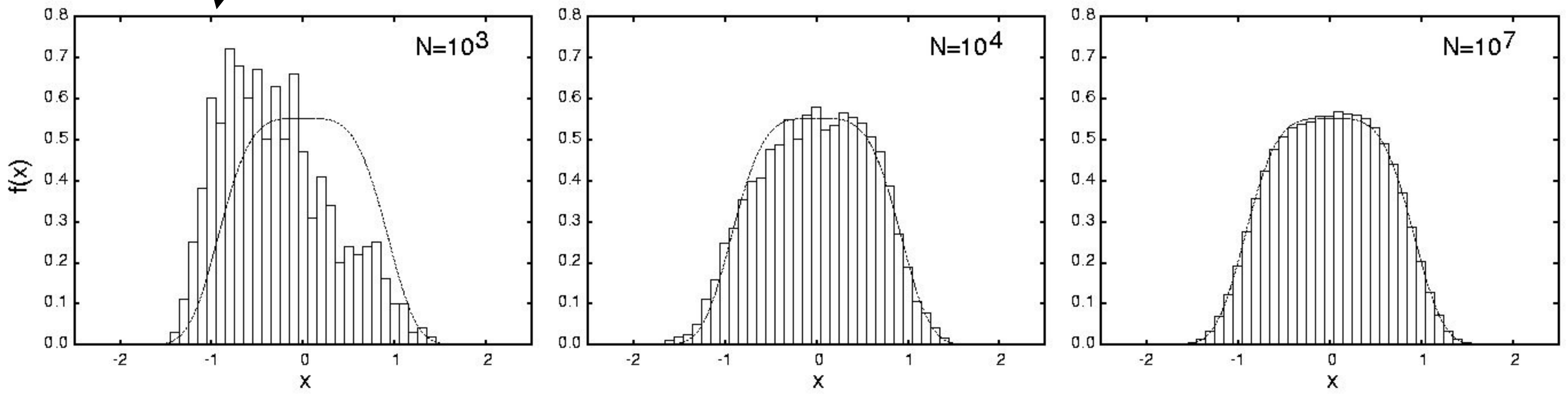
which is quite difficult for the first two methods. c is a normalisation constant, such that

$$\int_{-\infty}^{\infty} dx f(x) = 1$$

Its value *does not need to be known to implement the Metropolis et al. algorithm* (we just calculate c to compare the results with the normalised distribution)

$$\int_{-\infty}^{\infty} dx e^{-x^4} = 2\int_{0}^{\infty} dx e^{-x^4} = \frac{1}{2}\int_{0}^{\infty} dt t^{-3/4} e^{-t} = \frac{1}{2}\Gamma\left(\frac{1}{4}\right) = 1.8128...$$

here value of seed was $x^{[0]}=-1$



The Metropolis Method

Objetive: sample a system whose probability of being in configuration “c” is N(c)

How: Metropolis is based on an **importance-weighted** random walk:
Visited configurations: “c=o” means old config
“c=n” means new config

Transition probability: T(o,n).
Along the walk(s) the number of times any “o” is visited is m(o).
Thus, we want: m(o) proportional to N(o)

Detailed balance (sample without bias) **N(o) T(o,n)=N(n)T(n,o)**

Construction of T.
Probability of trial from o to n: **try(o,n)**
Probability of accept that trial: **acc(o,n)**
Indeed, **T(o,n)=try(o,n) acc(o,n)**

Metropolis choice for try: try(o,n)=try(n,o) _***symmetric***
Detailed balance: N(o) acc(o,n)=N(n) acc(n,o)
i.e, acc(o,n)/acc(n,o) = N(n)/N(o)

Metropolis choice for acc:
acc(o,n)= N(n)/N(o) if N(n)<N(o) (visiting less probable config.)
acc(o,n)= 1 if N(n)>N(o) (accept if n is more populated)

Transition probability

$$T(o,n) = acc(o,n) \quad \text{if } N(n) > N(o)$$
$$= acc(o,n)\, N(n)/N(o) \quad \text{if } N(n) < N(o)$$

$$T(o,o) = 1 - \sum_{n \neq o} T(o,n) \longrightarrow$$

It is important to also COUNT **not accepted moves** for the histogram normalization

The Metropolis Method for statistical physics

N(c) is **proportional** to the Boltzman factor exp(-U(c)/kT)

(note: the normalization constant is the partition function, Z, which is IMPOSIBLE to calculate in most cases)

However, Z is not needed here!

$$acc(o,n) = \exp(-(U[n]-U[o])/kT) < 1$$

Metropolis sampling

```
INITIALIZE
        call histogram(0, h)
            M =    number of MC steps
            nacc = 0,  number of accepted moves
            amp = amplitude of jumps
            x0 = initial configuration (value)


LOOP:   do i = 1, M
                    U = uniform in [-1,1]     !random uniform
                    xnew = xold + U*amp       !trial move
                     acc = f(xnew)/f(old)      !acceptance prob.
                                              !acceptance criterium
                     r = uniform in [-1,1]
                    if r <acc  then           !accept  move   :
                      xold=xnew
                       nacc=nacc+1
                    end if
        call histogram(1, xnew)                    ! sample
                    k=nint(xnew/h)+1
                    his(k)=his(k)+1

            end loop
NORMALIZE HISTOGRAM AND PRINT: call histogram(2,dummy)
```

FORTRAN code that implements the algorithm for this problem:

```
!
!  Metropolis et al.  algorithm
!  for f(x)=c exp(-x**4)
!
parameter (n=1000)
real*8 dseed,his(-n:n),h,x,rr,xt,amp,darg,anorma
real*4 r
integer i,m,j,k,n
h=0.1d0
dseed=53211d0
amp=0.2d0
m=10**6
nacc=0
do i=1, n
his(i)=0
end do
x=0.2
do j=1, m
call ggub(dseed,r)
xt=x+(2*r-1)*amp
darg=xt**4-x**4
rr=dexp(-darg)
call ggub(dseed,r)
if(dble(r).lt.rr) then
x=xt
nacc=nacc+1
end if
k=dnint(x/h)+1
if(k.gt.n.or.k.lt.-n) stop 'k fuera de intervalo'
his(k)=his(k)+1
end do
anorma=0d0
do k=-n, n
anorma=anorma+his(k)
end do
do k=-n, n
write(*,*) (k-1)*h,his(k)/(anorma*h)
end do
end
```

```
if(dble(r).lt.rr) then
x=xt
nacc=nacc+1
end if
k=dnint(x/h)+1
if(k.gt.n.or.k.lt.-n) stop 'k fuera de intervalo'
his(k)=his(k)+1
end do
anorma=0d0
do k=-n, n
anorma=anorma+his(k)
end do
do k=-n, n
write(*,*) (k-1)*h,his(k)/(anorma*h)
end do
end
```

Some
Monte Carlo Applications
In Statistical Physics of liquids

3. Applications in statistical physics

Problems in statistical and condensed–matter physics usually have many degrees of freedom

There are two types of problems:

- **Non–thermal problems**: configurations with constant statistical weight

  Examples:

  a. Random walk
  b. Modified random walks
  c. Percolation problem

- **Thermal problems**: configurations are weighted by Boltzmann distribution

$$P_i \propto e^{-E_i/kT}$$

  and configurations are generated using the Metropolis et al. algorithm

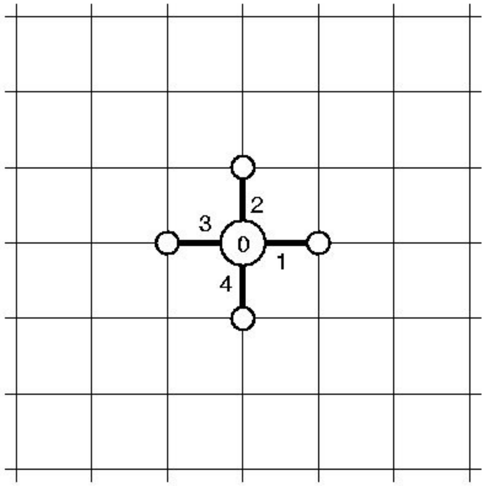
  Examples:

  a. Ising model in 2D (discrete variables)
  b. Hard spheres (continuous variables)

Random walk (RW)

This model is used to describe a number of problems in statistical physics:

- Brownian motion
- a single polymer macromolecule in solution (limit of high flexibility)
- diffusion of a particle
- ...



We start from an origin '0', and define four unit vectors

$$\mathbf{v}_1 = (+1, 0), \quad \mathbf{v}_2 = (0, +1), \quad \mathbf{v}_3 = (-1, 0), \quad \mathbf{v}_4 = (0, -1)$$

An N–step random walk (RW) is generated as follows:

1. set $\mathbf{r}_0 = (0, 0)$ and $n = 0$
2. choose an integer random number m_n from the set $\{1, 2, 3, 4\}$
3. make $\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_{m_n}$
4. if $n = N$, set $\mathbf{R} = \mathbf{r}_n$; otherwise we go back to step 2

The vector $\mathbf{R}$ connects the origin with the last point generated in the random walk

Clearly, if we generate two different random walks, the vector $\mathbf{R}$ will be different.

If the lattice has coordination (number of neighbours) z, the number of different N–step random walks is

$$Z_N^{\text{RW}} = z^N$$

For a polymer chain, Z_N^{RW} would be the partition function of the polymer (assuming the configurations of the polymer had zero energy).

<u>Statistical behaviour of $\mathbf{R}$</u>

Clearly:

$$\langle \mathbf{R} \rangle = \mathbf{0} \qquad \langle ... \rangle \text{ average over different chains}$$

$$\left\langle R^2 \right\rangle = \left\langle |\mathbf{R}|^2 \right\rangle = \left\langle \left| \sum_{n=1}^{N} \mathbf{v}_{m_n} \right|^2 \right\rangle = \left\langle \sum_{n=1}^{N} |\mathbf{v}_{m_n}|^2 \right\rangle + \left\langle \sum_{n=1}^{N} \sum_{\substack{p=1 \\ n \neq p}}^{N} \mathbf{v}_{m_n} \cdot \mathbf{v}_{m_p} \right\rangle = N$$

moves are independent and therefore uncorrelated. We then have

$$\left\langle R^2 \right\rangle_{\text{RW}} = N, \qquad r_N \equiv \sqrt{\langle R^2 \rangle_{\text{RW}}} = \sqrt{N}$$

where r_N defines an end-to-end distance. This result holds in the limit $N \to \infty$.

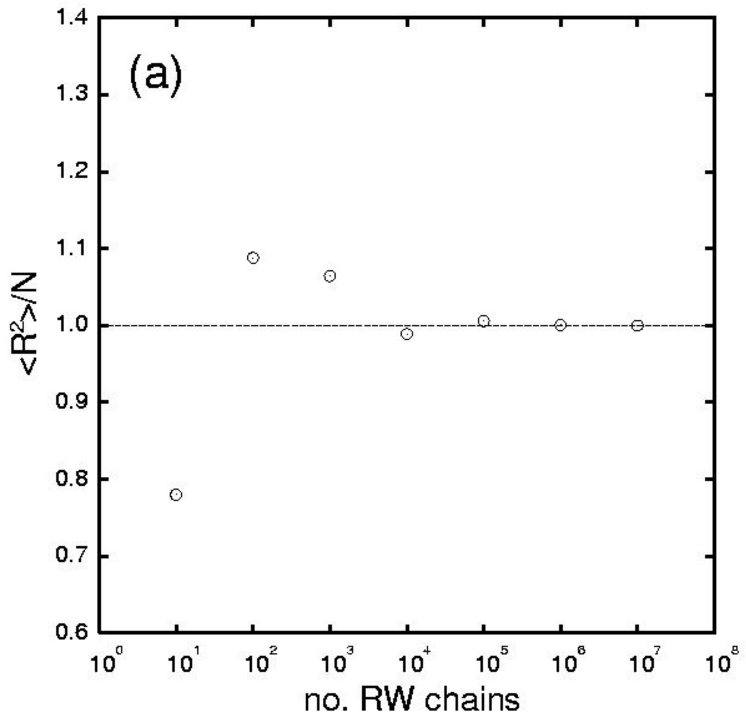
Code to generate random walks of length Ntot:

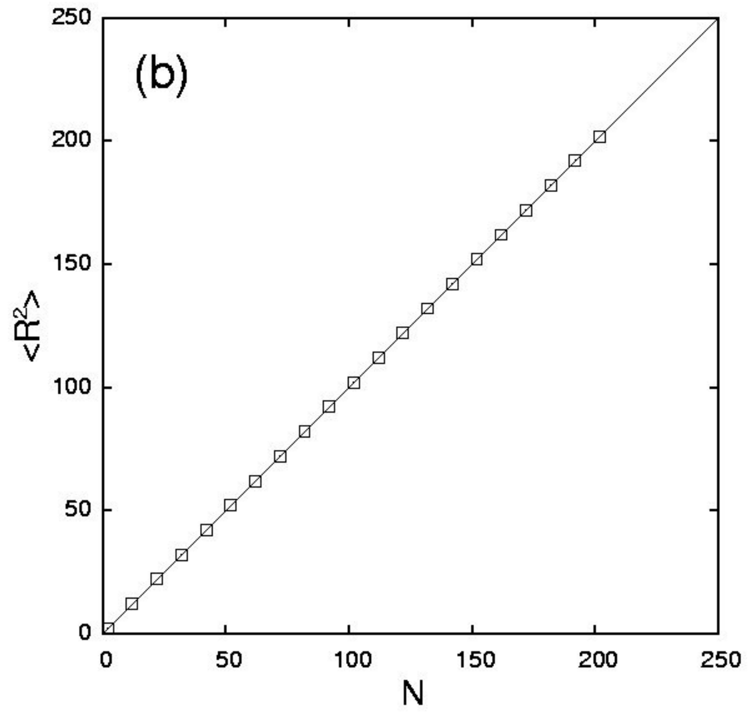
```
!
!  Random walk
!
implicit real*8(a-h,o-z)
real*8 dseed
real*4 r
dimension i(4),j(4)

data i/1,0,-1,0,
data j/0,1,0,-1,

dseed=51323d0
Ntot=20
M=10**6
ar2=0

do l=1,M
x=0
y=0
do n=1,Ntot
call ran (dseed,r)
k=4*r+1
x=x+i(k)
y=y+j(k)
end do
r2=x**2+y**2
ar2=ar2+r2
end do
write(*,'(''N, M, <r2>='',2i8,f12.5)') Ntot,l,ar2/M
end do

end
```

With this code the following results were obtained.

(a) value of $\langle R^2 \rangle / N$ with respect to number of chains generated for a value $N = 20$. It can be seen that the results tend to unity as M, the number of chains, increases.

(b) value of $\langle R^2 \rangle$ with respect to N. We can also see how the results tend to confirm the theoretical prediction that $\langle R^2 \rangle$ increases linearly with N (the continuous line has slope one).





Modified random walks

The random walk has some shortcomings in polymer physics:

- a polymer chain cannot turn back on itself
- a polymer chain cannot intersect itself

These problems stem from neglect of the excluded volume between the units

The non-reversal random walk (NRRW) model corrects for the first problem, by first defining

$$\mathbf{v}_{n\pm4} = \mathbf{v}_n$$

and then modifying step 2 of the RW for $n > 1$:

1. make $\mathbf{r}_0 = (0,0)$ and $n = 0$
2. choose an integer random number m_n from the set $\{m_{n-1} - 1, m_{n-1}, m_{n-1} + 1\}$
3. make $\mathbf{r}_{n+1} = \mathbf{r}_n + \mathbf{v}_{m_n}$
4. if $n = N$ we set $\mathbf{R} = \mathbf{r}_n$; otherwise go to step 2

The partition function is now

$$Z_N^{\mathrm{NRRW}} = (z-1)^N$$

To avoid the second problem, we can introduce the so–called *self-avoiding random walk* (SARW)

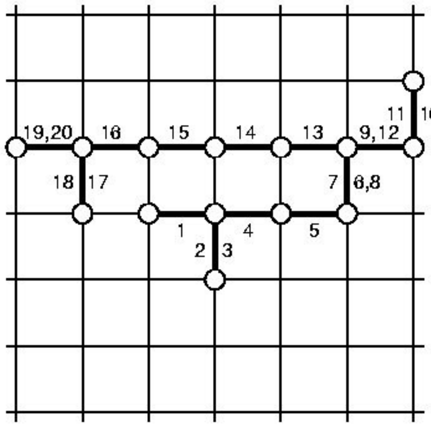
The following additional condition in step 3 of the NRRW algorithm is added:

3′ if the node $\mathbf{r}_{n+1}$ has been already visited, the process stops and we start it over again.

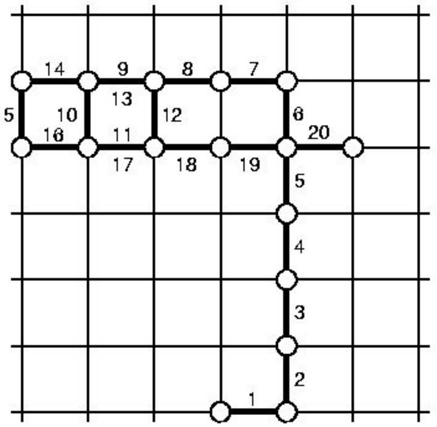
The partition function of the SARW model is much more complicated; actually an analytic expression is known only in the limit $N \to \infty$:

$$Z_N^{\text{SARW}} \xrightarrow[N \to \infty]{} N^{\gamma-1} z_{\text{eff}}^N \,, \qquad z_{\text{eff}} \leq z-1$$

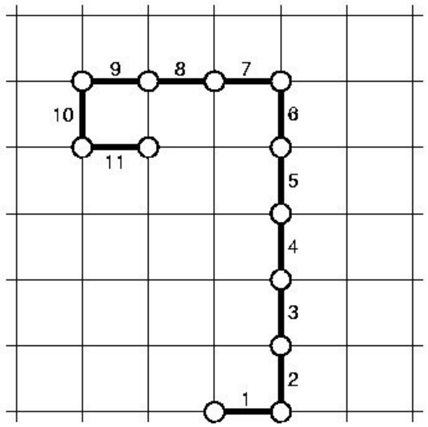
where γ is a critical exponent, and z_{eff} is the *effective coordination number*
(in 1D $z_{\text{eff}} = z-1$, with $z=2$)



RW



NRRW



SARW

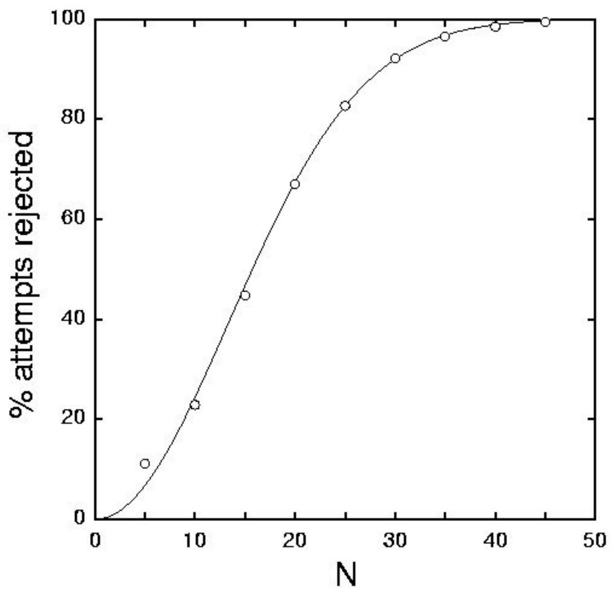
When N is large, the SARW algorithm becomes very inefficient since most of the attempts to make the chain grow are rejected. In this limit, the ratio of accepted to attempted steps can be calculated as:

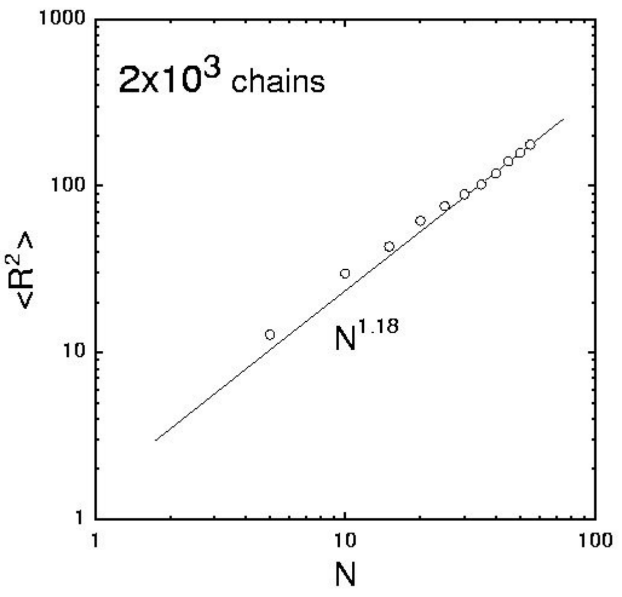
$$P_N = \frac{Z_N^{\text{SARW}}}{Z_N^{\text{NRRW}}} \underset{N\to\infty}{\longrightarrow} N^{\gamma-1}\left(\frac{z_{\text{eff}}}{z-1}\right)^N = e^{-N\log\frac{z-1}{z_{\text{eff}}}+(\gamma-1)\log N}$$

The algorithm becomes exponentially inefficient (impractical for $N \simeq 100$)

From Monte Carlo simulations it is known that

$$\left\langle R^2\right\rangle_{\text{SARW}} = N^{2\nu}, \quad \nu \simeq 0.59 \quad \nu \text{ critical exponent}$$





The percolation problem

Percolation is a simple but non–trivial geometrical problem. The simplest form of percolation is:

- a square lattice with N sites
- each site can be occupied with probability p (left empty with probability $1-p$)
- a cluster of size l is defined as a group of l connected nearest neighbours

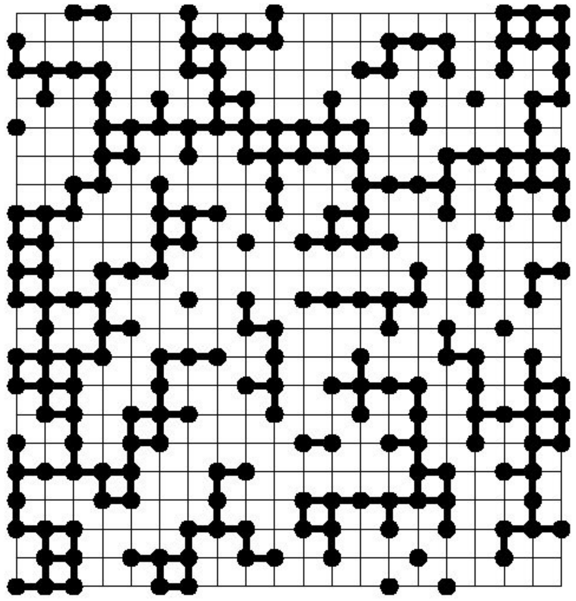
- let $n_l(p)$ be the number of clusters of size l
- let $P(p)$ be the fraction of occupied sites (number of occupied sites with respect to the total number of sites N) that belong to the largest cluster

Percolation transition with respect to p:

there appears a cluster in the system that *percolates*, i.e. spans the whole system, at

$$p = p_c$$

where p_c is a critical probability



Percolating cluster: cluster with a size similar to that of the system

The percolation problems can be analysed by means of the Monte Carlo method:

- each site of the lattice is occupied with probability p:
  1. sites are visited sequentially (one after the other)
  2. for each site we generate a uniform random number ζ in $[0,1]$
  3. If $\zeta < p$, we occupy the site with an atom, otherwise we leave it empty

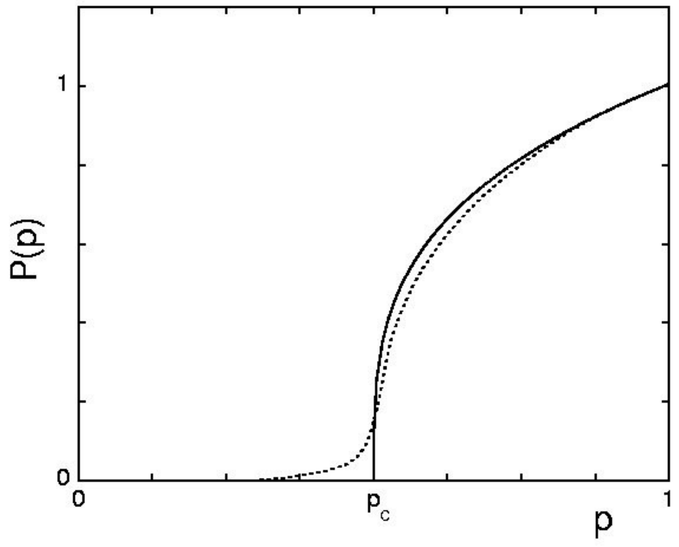
  After visiting the N sites we will have generated one configuration

- Clusters in each configuration are identified
- $P(p)$ is obtained by averaging over a large number of configurations

For the 2D square lattice, simulations give $p_c \simeq 0.59$

The percolation problem is rather similar to a second–order transition if we make

$$\begin{aligned} P(p) &\rightarrow \text{order parameter} \\ G(p) = \sum_l n_l(p) &\rightarrow \text{free energy} \end{aligned}$$



The problem has critical and scaling properties:

$$P(p) \sim (p - p_c)^{\beta}, \qquad G(p) \sim (p - p_c)^{2-\alpha}, \qquad \text{where } \alpha \text{ and } \beta \text{ are critical exponents}$$

Numerical integration with many variables

Traditional numerical integration methods cannot be used here

Let the integration volume be a cube of side D, and define a square lattice with n nodes per axis. Then the ratio of nodes on the surface to nodes in the bulk is

$$\frac{2Dn^{D-1}}{n^D} = \frac{2D}{n} \quad \underset{D \to \infty}{\longrightarrow} \quad \infty$$

As D increases the method becomes highly inefficient!

The Monte Carlo method approximates $\int_{V^D} g(x) d^D x \simeq \frac{W^D}{M} \sum_{i=1}^{M} g(x_i)\zeta_i$

- $x \equiv (x_1, x_2, ..., x^D)$ is a vector in D dimensions
- V^D is the D–dimensional integration volume
- $W^D \supseteq V^D$ is a volume that contains the volume V^D
- M is the number of uniform random numbers x_i generated in W^D
- and ζ_i is a number such that

$$\zeta_i = \begin{cases} 1, & \text{if } x_i \in V^D \\ 0, & \text{otherwise} \end{cases}$$

The method is especially powerful when there is a normalised distribution function in the integrand. Consider the 1D integral

$$I = \int_a^b dx f(x) g(x), \qquad f(x) \geq 0, \qquad \int_a^b dx f(x) = 1$$

The Monte Carlo method with uniform sampling gives

$$I = \int_a^b dx f(x) g(x) \simeq \frac{b-a}{M} \sum_{i=1}^{M} f(x_i) g(x_i), \qquad \{x_i\} \text{ uniformly distributed numbers in } [a,b]$$

Making use of the cumulative function $P(x)$:

$$y = P(x) = \int_a^x dx' f(x') \quad \rightarrow \quad dy = f(x) dx$$

If we make the change in variables $x \rightarrow y$, we have:

$$I = \int_a^b dx f(x) g(x) = \int_0^1 dy g\left(P^{-1}(y)\right) \approx \frac{1}{M} \sum_{j=1}^{M} g(\overbrace{P^{-1}(y_j)}^{x_j})$$

Now if y is *uniformly* distributed in $[0,1]$, then $P^{-1}(y)$ is a random variable distributed according to $f(x)$

$$I = \int_a^b dx f(x) g(x) \simeq \frac{1}{M} \sum_{j=1}^{M} g(x_j), \qquad \{x_j\} \text{ distributed as } f(x) \text{ in } [a,b]$$

This may be directly extended to multidimensional integrals with an integrand weighted by a distribution function.

Thermal problem: Ising model in 2D

We consider square lattice, with a 1/2–spin on each site, $s_i = \pm 1$ (spin up $+1$, spin down -1)
The Hamiltonian of the system is

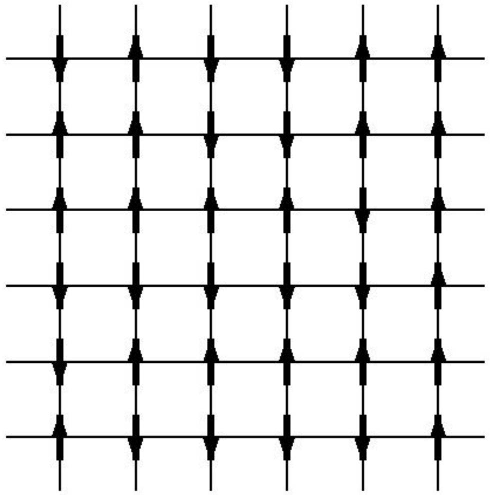
$$H(\{s_i\}) = -J \sum_{nn} s_i s_j - B \sum_i s_i, \qquad J > 0, \qquad B \text{ magnetic field}$$

nn stands for *nearest neighbours* (four for the square lattice)
We represent each configuration of N spins by

$$s \equiv \{s_i\} = \{s_1, s_2, ..., s_N\}$$

Technically this is a more complicated problem, since configurations have to be weighted by the normalised Boltzmann factor:

$$P(s) = \frac{e^{-\beta H(s)}}{\sum_s e^{-\beta H(s)}}, \qquad \beta = \frac{1}{kT}$$



How to generate configurations? We will use an importance–sampling technique, the Metropolis et al. algorithm

Uniform sampling will produce configurations that, in most cases, will have a negligible statistical weight (i.e. small Boltzmann factor), which will give rise to very inefficient (or even helpless!) simulations.

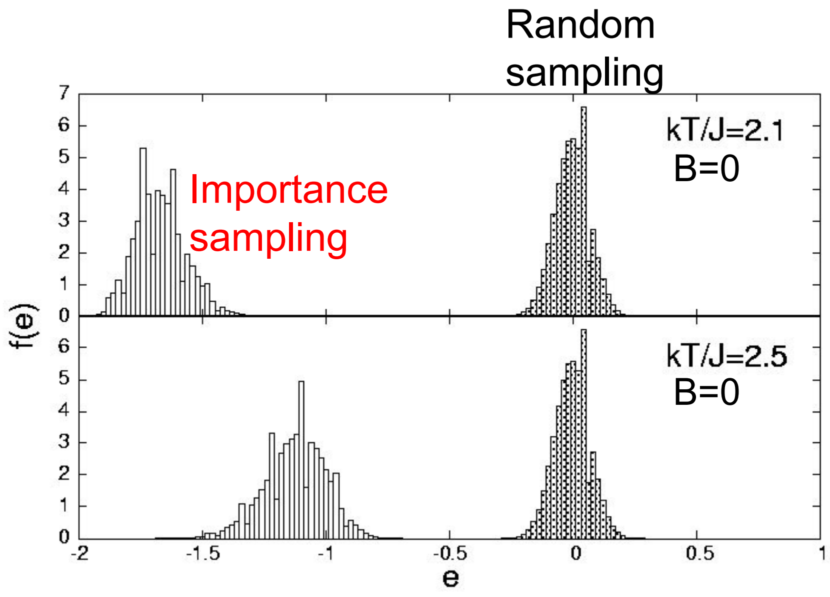
Uniform sampling

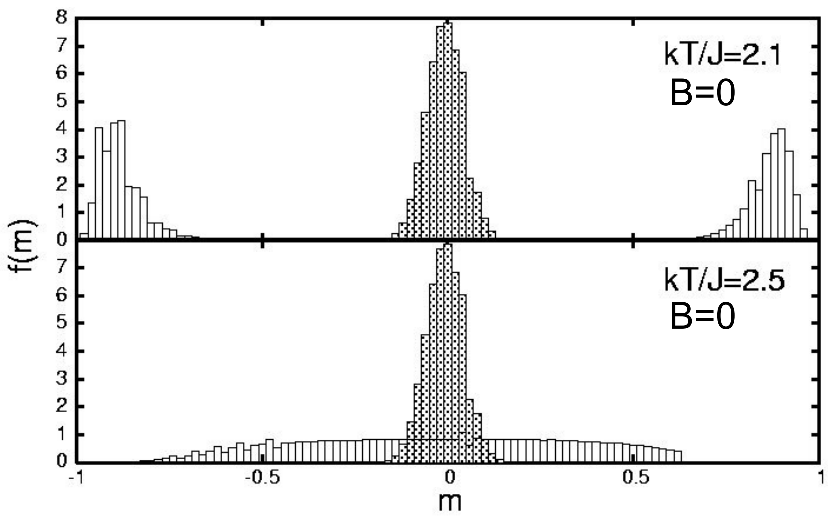
The sum $\sum_s$ extends over the 2^N configurations of the spins. For example, for $N = 10^4$

$$2^N = 2^{10^4} \simeq 10^{3010.3} \quad \rightarrow \quad \text{out of question!}$$

A reduced set of configurations is chosen, and these are randomly and uniformly selected:

1. We choose a uniform random number $\zeta \in [0, 1]$
2. If $\zeta < 0.5$, we set $s_i = +1$; otherwise, we set $s_i = -1$
3. We repeat for all spins $i = 1, 2, ..., N$





Monte Carlo code for the 2D Ising model

Details on how to implement the importance-sampling technique:

- **Trial moves**: spins are moved one at a time. Spins are visited sequentially or randomly, and switched from up to down or from down to up. Change of energy involved in switching the i–th spin is

$$\Delta E = E_t - E_0 = 2 s_i \sum_{n.n.} s_j$$

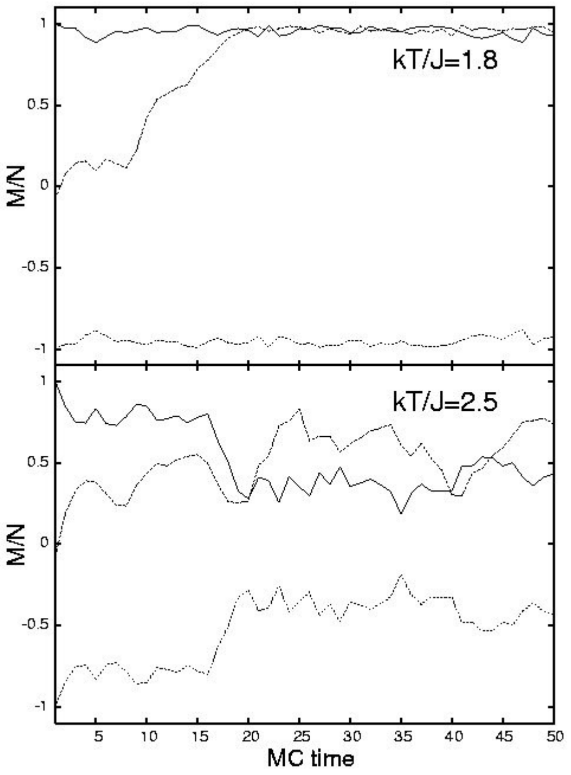
  Move accepted with probability

$$r = \frac{e^{-\beta E_t}}{e^{-\beta E_0}} = e^{-\beta \Delta E} \quad \text{partition function not needed!}$$

  After we have attempted to turn the N spins, we have one MC step.

- **Initialisation**: we may start from all–up, all–down, or random configurations.

- **Warming up**: the Metropolis et al. algorithm generates configurations asymptotically, so a warm–up period is needed



```fortran
!
!  2D-Ising model Monte Carlo with
!  importance sampling
!
parameter (n=20)
integer*1 is(n,n)
real*8 dseed
dimension histm(101),histe(101)

data dseed /186761d0/
nsteps = 100000000
nblock=10

n2 = n*n
T=2.1

ama = 0
amaa = 0
ene = 0
enea = 0

do i=1,101
histm(i)=0
histe(i)=0
end do

do i = 1, n
do j = 1, n
is(i,j)=1
call ran (dseed,random)
if(random.lt.0.5) is(i,j)=-1 !  comment to set ordered state
```

```
ama = ama + is(i,j)
end do
end do

do i = 1, n
ip = i + 1
if (ip .gt.  n) ip = 1
do j = 1, n
jp = j + 1
if (jp .gt.  n) jp = 1
ene = ene - is(i,j)*(is(ip,j)+is(i,jp))
end do
end do

do k = 1, nsteps/nblock
do l = 1, nblock

do i = 1, n
do j = 1, n
sij = is(i,j)
ip = i + 1
if (ip.gt.n) ip=1
im = i - 1
if (im.lt.1) im=n
jp = j + 1
if (jp.gt.n) jp=1
jm = j - 1
if (jm.lt.1) jm=n
de = 2*sij*(is(ip,j)+is(im,j)+is(i,jp)+is(i,jm))
call ran (dseed,random)
```

initial energy

MC steps: 1 step = 1 sweep over all spins

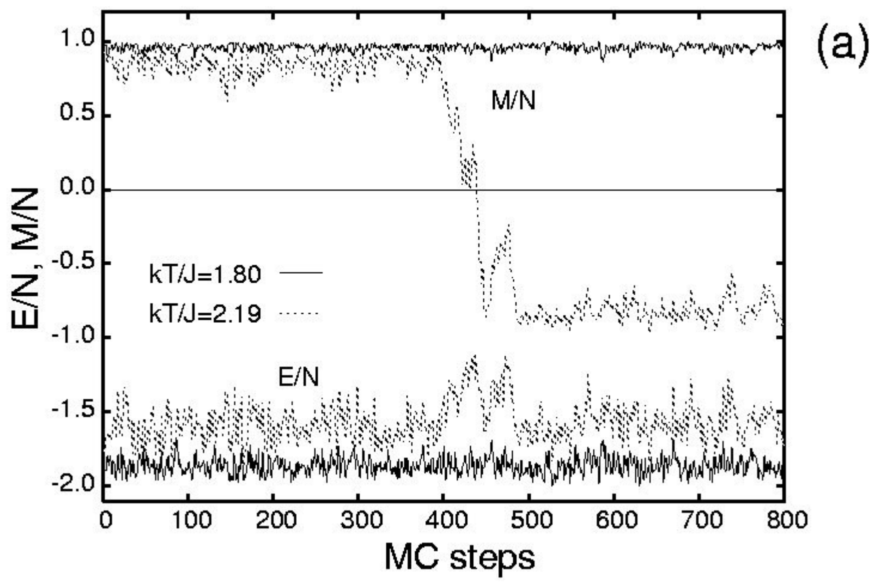
Here done in blocks of `nblock` steps

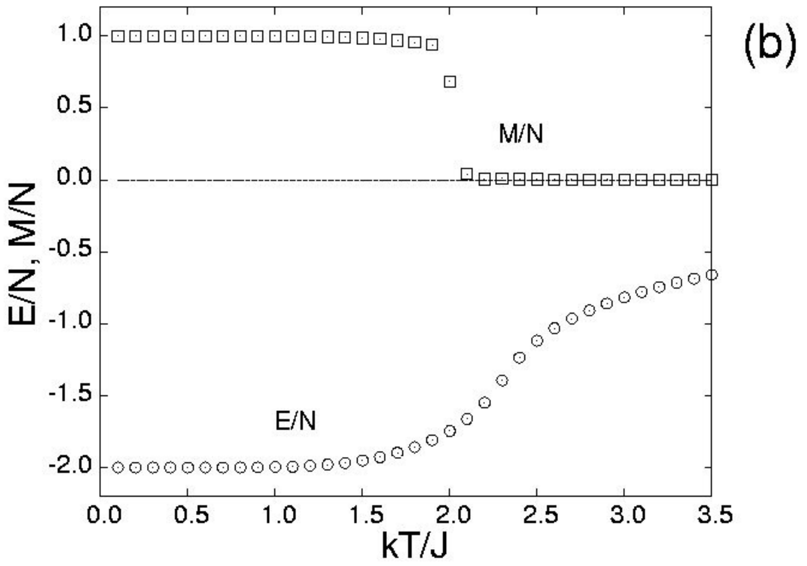
i,j: spin to be flipped

energy change

```
if (exp(-de/T) .gt.  random) then
is(i,j) = -sij
ama = ama - 2*sij                          check for acceptance
ene = ene + de
end if
end do
end do

end do

enea = enea + ene
amaa = amaa + ama

im=(ama/n2+1)*0.5e2+1
histm(im)=histm(im)+1

ie=(ene/n2+2)*50.0+1
histe(ie)=histe(ie)+1
end do

write(*,'(''T= '',f5.3,'' M/N= '',f5.3,'' E/N= '',f6.3)')
>T, abs(amaa/(nsteps/nblock*n2)),enea/(nsteps/nblock*n2)

open(3,file='histe.dat')
sum=0
do i=1,101
sum=sum+histe(i)
end do
do i=1,101
emm=2*(i-1)/100.0-2
write(3,'(2f9.3)') emm,histe(i)/sum*50.0
end do
close(3)
```

```
open(3,file='histm.dat')
sum=0
do i=1,101
sum=sum+histm(i)
end do
do i=1,101
amm=2*(i-1)/100.0-1
write(3,'(2f9.3)') amm,histm(i)/sum*50.0
end do
close(3)

end
```





(a) MC time evolution of magnetisation per spin, M/N, and energy per spin, E/N, at two different temperatures. $N = 20 \times 20$.

(b) Phase diagram (M/N vs. T) and dependence of energy per spin, E/N, on T. Number of MC steps $= 10^7$.

Model with continuous variables: hard spheres

Simple but important model in the development of the statistical mechanics of fluids and solids. It mimicks the interaction potential between two spherical molecules, $\phi(r)$:

$$\phi(r) = \begin{cases} \infty, & r < \sigma \\ \\ 0, & r > \sigma \end{cases}$$

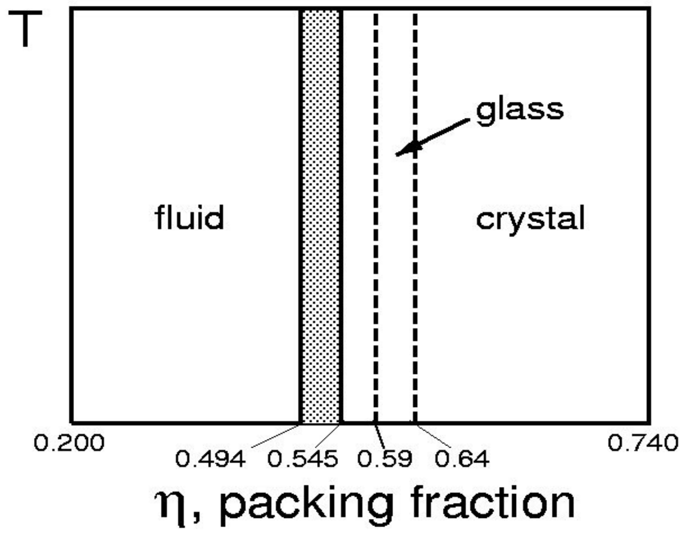
It is an *athermal* model (i.e. thermodynamic properties depend trivially on temperature) since the corresponding Boltzmann factor does not depend on temperature T:

$$e^{-\beta\phi(r)} = \begin{cases} 0, & r > \sigma \text{ (spheres overlap)} \\ \\ 1, & r > \sigma \text{ (spheres do not overlap)} \end{cases}$$

Only configurations where there is no overlap between the spheres are allowed. These have the same statistical weight, and possess zero energy.

Despite its 'trivial' appearance, the hard–sphere model contains highly non–trivial physics.

- fluid phase with long–range positional disorder
- crystalline phase with long–range (fcc) positional order
- amorphous (metastable) phase with long–range positional disorder but negligible viscosity coefficient and virtually zero diffusion



packing fraction, η

ratio of volume occupied by N spheres to total volume V:

$$\eta = \frac{Nv}{V} = \rho v = \frac{\pi}{6}\rho\sigma^3, \quad v = \left(\frac{4\pi}{3}\right) \times \left(\frac{\sigma}{2}\right)^3 = \frac{\pi}{6}\sigma^3$$

(molecular volume)

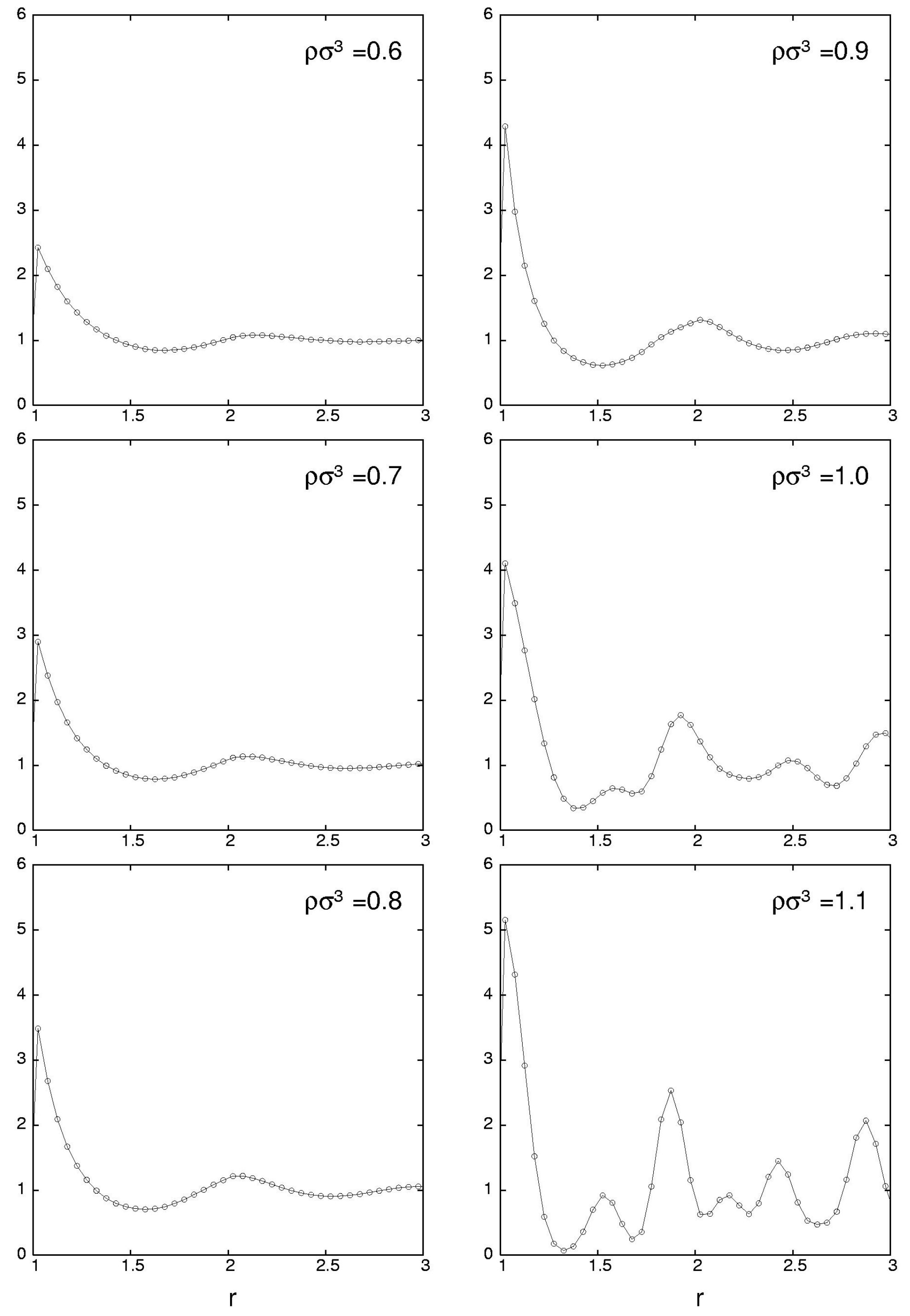
Pressure and radial distribution function. The radial distribution function $g(r)$ is a function of the distance between a pair of particles. It is a measure of the local order of the fluid.

$$g(r) = \frac{\text{mean number of particles around a given one at distance } r}{\text{expected number with interactions turned off}}$$

For a dense fluid, the function shows a strong first peak at distances $\sim \sigma$, and a damped oscillatory structure tending to one as r increases.

The reduced pressure is given by

$$\frac{p}{\rho kT} = 1 - \frac{2\pi\rho}{3kT}\int_0^\infty dr r^3 \phi'(r) g(r) = 1 + \frac{2\pi\rho}{3}\sigma^3 g(\sigma^+)$$



ρσ^3 =0.6
ρσ^3 =0.9
ρσ^3 =0.7
ρσ^3 =1.0
ρσ^3 =0.8
ρσ^3 =1.1
r
r

Approximations for the fluid and the crystal:

- **fluid**: virial equations of state (virial coefficients)
- **crystal**: free–volume approximation

Virial equation of state

It is a density expansion, based on experimental data, in terms of the *virial coefficients* B_n:

$$\frac{pV}{NkT} = 1 + \sum_{n=1}^{\infty} B_{n+1}\rho^n = 1 + \sum_{n=1}^{\infty} B^*_{n+1}\eta^n, \quad B^*_n = \frac{B_n}{v^{n-1}}$$

Expressions for virial coefficients based on statistical mechanics:

$$B_2 = -\frac{1}{2}\int d\mathbf{r} f(r)$$

$$B_3 = -\frac{1}{3}\int d\mathbf{r}\int d\mathbf{r}' f(r)f(r')f\left(|\mathbf{r}-\mathbf{r}'|\right)$$

$$\ldots$$

where $f(r) = \exp\left[-\beta\phi(r)\right] - 1$ is the *Mayer function*. For hard spheres $f(r) = \begin{cases} -1, & r < \sigma \\ 0, & r > \sigma \end{cases}$

The second virial coefficient is analytical:

$$B_2 = \frac{1}{2} \times 4\pi \int_0^\sigma dr r^2 = \frac{2\pi}{3}\sigma^3 = 4v$$

Let us evaluate the B_3 coefficient using multidimensional Monte Carlo integration. B_3 is known exactly, $B_3 = 5\pi^2\sigma^6/18 \simeq 2.74156$. In spherical coordinates:

$$B_3 = -\frac{1}{3} \times \int_0^\sigma dr r^2 \int_{-1}^1 d(\cos\theta) \int_0^{2\pi} d\phi$$

$$\times \int_0^\sigma dr' r'^2 \int_{-1}^1 d(\cos\theta') \int_0^{2\pi} d\phi' f(r) f(r') f(|\mathbf{r}-\mathbf{r}'|)$$

- place a sphere at the origin
- choose a location for a second sphere so that it overlaps with the first
- choose a location for a third sphere so that it overlaps with the first
- check whether the second and third spheres overlap, i.e. whether $f(|\mathbf{r}-\mathbf{r}'|) = -1$ or 0

Then:

$$B_3 = \frac{1}{3} \times \frac{1}{M} \left[(\sigma) \times (2) \times (2\pi)\right]^2 \sum_{i=1}^{M} r_i^2 r_i'^2 \xi_i = \frac{(4\pi\sigma)^2}{3M} \sum_{i=1}^{M} r_i^2 r_i'^2 \xi_i$$

where $i = 1, ..., M$ are Monte Carlo steps ($\xi = 1$ or 0 depending on whether the second and third spheres overlap)

M	B_3^{MC}	$B_3^{\mathrm{MC}}/B_3^{\mathrm{exacto}}$
10^2	2.51933	0.91894
10^3	2.65935	0.97002
10^4	2.77402	1.01184
10^5	2.74633	1.00174
10^6	2.74411	1.00093
10^7	2.74242	1.00032
10^8	2.74170	1.00005

```
!
!  Evaluation of B3 by Monte Carlo integration
!  hard-sphere diameter sigma=1
!
implicit real*8(a-h,o-z)
real*4 r,t,f
integer*8 M,i

dseed=173211d0
pi=4.0d0*datan(1.0d0)
pi2=2*pi

do mpot=2,8
M=10**mpot
sum=0
do i=1,M
call ggub (dseed, r)
call ggub (dseed, t)
call ggub (dseed, f)
cp=dcos(pi2*f)
sp=dsin(pi2*f)
ct=2*t-1
st=dsqrt(1-ct**2)
x=r*st*cp
y=r*st*sp
z=r*ct
r1a=x**2+y**2+z**2
call ggub (dseed, r)
call ggub (dseed, t)
call ggub (dseed, f)
```

```
cp=dcos(pi2*f)
sp=dsin(pi2*f)
ct=2*t-1
st=dsqrt(1-ct**2)
x1=r*st*cp
y1=r*st*sp
z1=r*ct
r1b=x1**2+y1**2+z1**2
r2=(x-x1)**2+(y-y1)**2+(z-z1)**2
if(r2.lt.1.0d0) sum=sum+r1a*r1b
end do
vol=4*pi
B3=sum*vol**2/(3*M)
B3ex=5*pi**2/18
write(*,'(''M='',i9,'' B3='',f8.5,'' B3/B3ex='',f8.5)')
> M,B3,B3/B3ex
end do
end
```

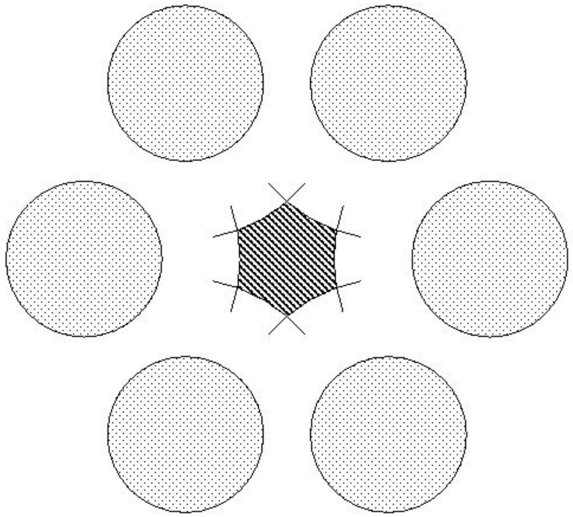
Free–volume equation for the crystalline phase

The free–volume approximation assumes the neighbours of a given sphere to be fixed in their equilibrium positions. The central sphere moves in the 'cage' left by the neighbours, within a 'free volume' v_f.

$$F = -kT \log Z = -NkT \log \left(\frac{v_f}{\Lambda^3}\right), \qquad \frac{F}{NkT} = \log \left(\frac{\Lambda^3}{v_f^3}\right), \qquad p = -\left(\frac{\partial F}{\partial V}\right)$$

To calculate v_f we can use the Monte Carlo integration method:

$$v_f \simeq \frac{W}{M} \sum_{m=1}^{M} \xi_i, \qquad \xi_i = \begin{cases} 1, & \text{no overlap} \\ 0 & \text{overlap} \end{cases}$$



```
!
!  Free volume for the fcc lattice
!
implicit real*8(a-h,o-z)
dimension xx(12),yy(12),zz(12)
data xx/+0.0d0,+0.0d0,+0.0d0,+0.0d0,+0.5d0,-0.5d0,+0.5d0,
>-0.5d0,+0.5d0,-0.5d0,+0.5d0,-0.5d0/
data yy/+0.5d0,-0.5d0,+0.5d0,-0.5d0,+0.0d0,+0.0d0,+0.0d0,
>+0.0d0,+0.5d0,+0.5d0,-0.5d0,-0.5d0/
data zz/+0.5d0,+0.5d0,-0.5d0,-0.5d0,+0.5d0,+0.5d0,-0.5d0,
>-0.5d0,+0.0d0,+0.0d0,+0.0d0,+0.0d0/
real*4 r
pi=4d0*datan(1d0)
sq2=dsqrt(2d0)

rho=1.3d0
a=(dsqrt(2d0)/rho)**(1d0/3d0)
eta=rho*pi/6
dseed=173211d0
M=2e7
sum=0

do i=1,M
call ggub (dseed, r)
x=a/2*(2*r-1)
call ggub (dseed, r)
y=a/2*(2*r-1)
call ggub (dseed, r)
z=a/2*(2*r-1)

```

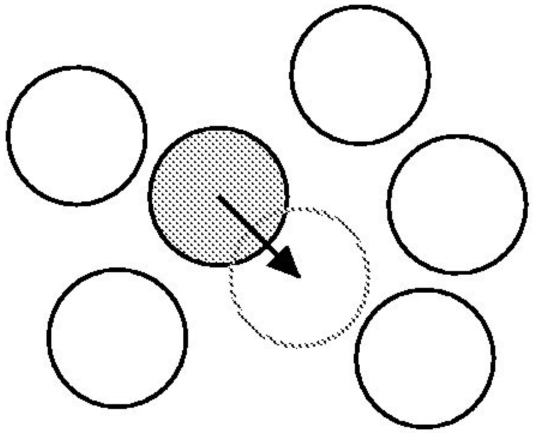
```
do k=1,12
xk=a*xx(k)*sq2
yk=a*yy(k)*sq2
zk=a*zz(k)*sq2
r2=(xk-x)**2+(yk-y)**2+(zk-z)**2
if(r2.lt.1d0) go to 1
end do
sum=sum+1
1 end do

vf=sum/M*a**3
write(*,'(''eta='',f10.5,'' vf='',f10.5,'' ff='',f12.7)')
>eta,vf,-rho*dlog(vf)

end
```

Monte Carlo code for thermal simulation of hard spheres

We consider N hard spheres contained in a rectangular box.

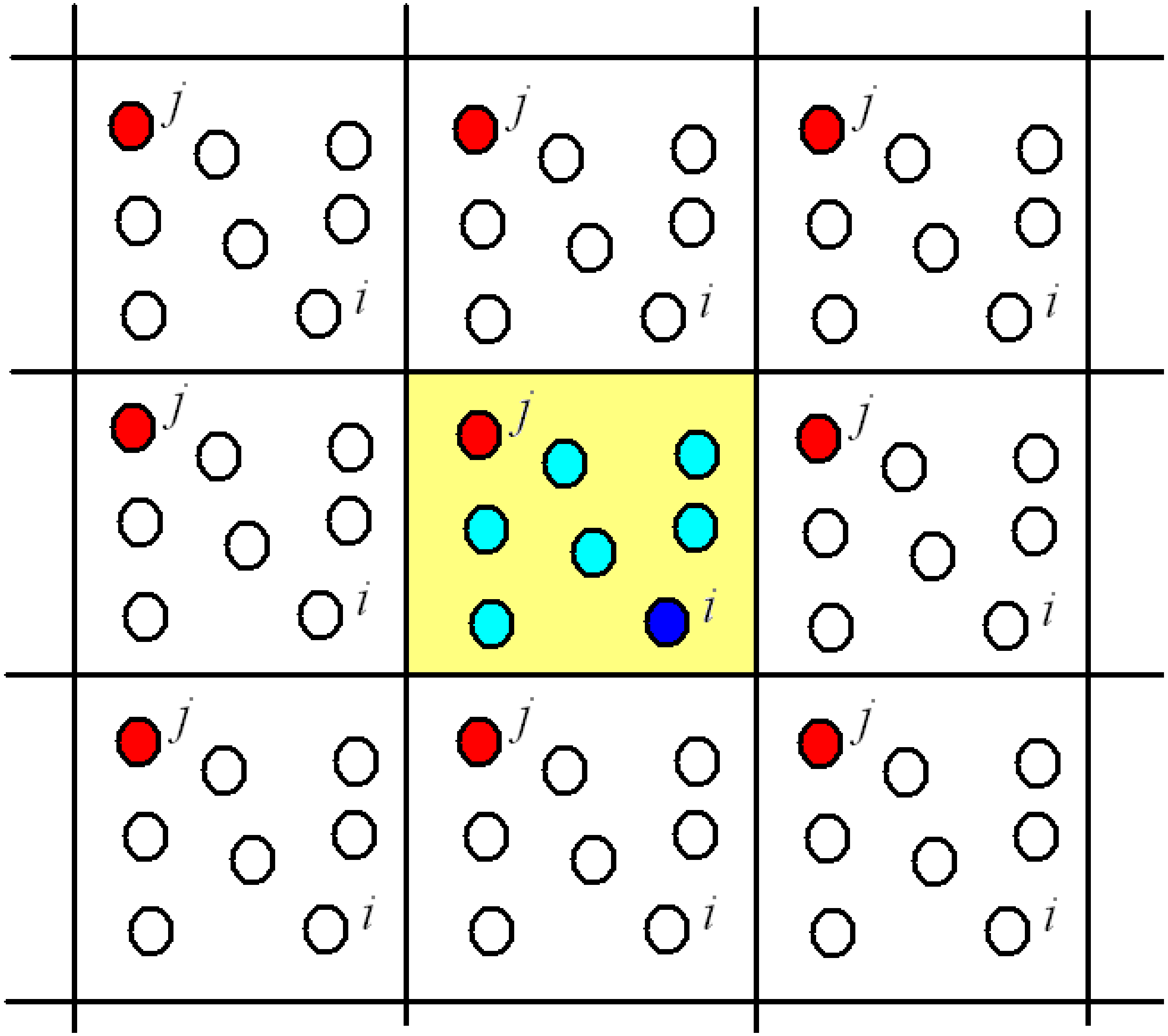
A move consists of randomly moving a sphere and checking whether this displacement gives an overlap or not. If there is no overlap, the move is accepted; otherwise it is rejected:

$$e^{-\beta \Delta E} = \begin{cases} 0, & \text{there is at least one overlap} \\ 1, & \text{there are no overlaps} \end{cases}$$



a. Periodic boundary conditions

b. Minimum image convention


j
i
j
i
j
i
j
i
j
i
j
i
j
i
j
i
j
i

c. Calculation of radial distribution function.

$g(r)d^3r$ = number of particles to be found in a differential volume element d^3r a distance r from that sphere, with respect to number without interactions:

$$g(r) = \frac{1}{N} \times \frac{\left\langle \sum_{i=1}^{N} n_i\left(r, \Delta r\right) \right\rangle}{4\pi r^2 \rho \Delta r}$$

$\langle ... \rangle$ is average over different configurations of the spheres

- $n_i(r, \Delta r)$ is the number of particles within a spherical shell of width Δr, centred on the i–th particle, and at a distance r from this particle
- $4\pi r^2 \Delta r$ is the volume of the spherical shell:

$$\frac{4\pi}{3}\left[\left(r + \frac{\Delta r}{2}\right)^3 - \left(r - \frac{\Delta r}{2}\right)^3\right]$$

```
!
!  MC simulation of hard spheres
!
implicit real*8(a-h,o-z)
parameter (num=4000, ngrx=1000)

real*4 rr
dimension r1x(num), r1y(num), r1z(num)

common gr(ngrx)

data dseed /1467383.d0/
data hr /0.05d0/
data pi /3.141592654d0/

open (1, file = 'hs.par')
read (1,*) ini, lmn
read (1,*) nstepeq, nstep0, nblock0
read (1,*) dens, amp
close (1)

if (lmn .gt.  num) stop 'lmn .gt.  num'

open (2, file = 'hs.inp')
open (3, file = 'hs.out')
open (4, file = 'hs.gr')

if (ini .eq.  0) then
call fcc (lmn, dens, r1x, r1y, r1z, xlx, yly, zlz)  ←—  particles on sites of fcc lattice
else
do i = 1, lmn
read(2,*) r1x(i), r1y(i), r1z(i)
end do
```

```fortran
read(2,*) xlx,yly,zlz
end if

xll = xlx/2
Yll = yly/2
zll = zlz/2

volume = xlx*yly*zlz
ro = lmn/volume

nox = xll/hr
if (nox .gt.  ngrx) stop 'nox .gt.  ngrx'

do istage = 1, 2

if (istage .eq.  1) then
nblock = nstepeq
nstep = nblock
else
nblock = nblock0
nstep = nstep0
end if

do j = 1, lmn
call pbc (r1x(j),r1y(j),r1z(j),xlx,yly,zlz,xll,yll,zll)
end do

do j = 1, ngrx
gr(j) = 0
end do

naccept = 0
numgr = 0
```

$\leftarrow$ periodic boundary conditions applied

```

do ii = 1, nstep/nblock

numgr = numgr + 1
call gdr (lmn,r1x,r1y,r1z,xlx,yly,zlz,xll,yll,zll)  <----- histogram for g(r) accumulated

nacceptb=0

do jj=1,nblock

do j = 1, lmn
call pbc
> (r1x(j),r1y(j),r1z(j),xlx,yly,zlz,xll,yll,zll)
end do

do k = 1, lmn
call ggub(dseed, rr)       \
desx = (rr-0.5)*amp         |
call ggub(dseed, rr)        |
desy = (rr-0.5)*amp         |
call ggub(dseed, rr)        |
desz = (rr-0.5)*amp          >   test position for k-th particle
r1xk = r1x(k)               |
r1yk = r1y(k)               |
r1zk = r1z(k)               |
r3x = r1xk + desx           |
r3y = r1yk + desy           |
r3z = r1zk + desz          /

```

```
do j = 1, lmn
if (j .ne.  k) then
xi = r3x
yi = r3y
zi = r3z
xx = xi - r1x(j)
yy = yi - r1y(j)
zz = zi - r1z(j)
call mic (xx,yy,zz,xlx,yly,zlz,xll,yll,zll)
r = xx*xx + yy*yy + zz*zz
if (r .lt.  1d0) go to 1
end if
end do
```

check overlap with other particles

```

naccept = naccept + 1
nacceptb = nacceptb + 1
r1x(k) = r3x
r1y(k) = r3y
r1z(k) = r3z

1 end do
end do

acceptratio = real(nacceptb)/(nblock*lmn)*100
if (acceptratio .gt.  50.0) amp = amp * 1.05
if (acceptratio .lt.  50.0) amp = amp * 0.95
end do
```

adjust acceptance ratio to 50%

```

do i=1,lmn
write(3,*) r1x(i),r1y(i),r1z(i)
end do
write(3,*) xlx,yly,zlz

write(8,'(''% Mov acept ='',f10.6)') naccept*100./(nstep*lmn)

if (istage .eq.  2) then

do i = 1, ngrx
r = (i-1+0.5d0) * hr
dvol = 4*pi/3d0 * ((r+hr/2)**3-(r-hr/2)**3)
grr = gr(i)/(dvol*lmn*numgr*(lmn-1)/volume)
write(4,*) r,grr
end do

end if

end do

end
```

normalisation of g(r)

```
!
!  Periodic boundary conditions
!
subroutine pbc (r1x,r1y,r1z,xlx,yly,zlz,xll,yll,zll)

implicit real*8 (a-h,o-z)

if (dabs(r1x-xll) .ge .xll) then
if (r1x .lt.  0d0) then
r1x = r1x + xlx
else
r1x = r1x - xlx
end if
end if
if (dabs(r1y-yll) .ge.  yll) then
if (r1y .lt.  0d0) then
r1y = r1y + yly
else
r1y = r1y - yly
end if
end if
if (dabs(r1z-zll) .ge.  zll) then
if (r1z .lt.  0d0) then
r1z = r1z + zlz
else
r1z = r1z - zlz
end if
end if

end
```

```
!
!  Minimum image convention
!
subroutine mic (xx,yy,zz,alx,aly,alz,alx2,aly2,alz2)
implicit real*8(a-h,o-z)

if (dabs(dble(xx)) .ge.  alx2) then
if (dble(xx) .lt.  alx2) then
xx = xx + alx
else
xx = xx - alx
end if
end if

if (dabs(dble(yy)) .ge.  aly2) then
if (dble(yy) .lt.  aly2) then
yy = yy + aly
else
yy = yy - aly
end if
end if

if (dabs(dble(zz)) .ge.  alz2) then
if(dble(zz) .lt.  alz2) then
zz = zz + alz
else
zz = zz - alz
end if
end if

end
```

```
!
!  Generation of fcc lattice
!
subroutine fcc (lmn, dens, r1x, r1y, r1z, xlx, yly, zlz)

implicit real*8 (a-h, o-z)
parameter (num = 4000)

dimension r1x(num), r1y(num), r1z(num)
dimension sx(4), sy(4), sz(4)

data sx /0d0, 0.5d0, 0.5d0, 0d0/
data sy /0d0, 0.5d0, 0d0, 0.5d0/
data sz /0d0, 0d0, 0.5d0, 0.5d0/

data sh /0.01d0/

a = (4d0/dens)**(1d0/3d0)
n = (lmn/4)**(1d0/3d0) + 0.001

xlx = n*a
yly = n*a
zlz = n*a

m = 1
do i = 1, n
do j = 1, n
do k = 1, n
do l = 1, 4
r1x(m) = (i - 1 + sx(l) + sh) * a
r1y(m) = (j - 1 + sy(l) + sh) * a
r1z(m) = (k - 1 + sz(l) + sh) * a
m=m+1
end do
end do
end do
end do

end
```

```
!
!  Histogram for g(r)
!
subroutine gdr(lmn,r1x,r1y,r1z,xlx,yly,zlz,xll,yll,zll)

implicit real*8(A-H,O-Z)
parameter (num=4000, ngrx=1000)

dimension r1x(num), r1y(num), r1z(num)
common gr(ngrx)
data hr /0.05d0/

do i = 1, lmn-1
do j = i + 1,lmn
xx = r1x(i) - r1x(j)
yy = r1y(i) - r1y(j)
zz = r1z(i) - r1z(j)
call mic (xx,yy,zz,xlx,yly,zlz,xll,yll,zll)
r = xx*xx + yy*yy + zz*zz
rr = dsqrt(r)
if (rr .lt.  xll) then
k = rr/hr + 1
gr(k) = gr(k) + 2
end if
end do
end do

end
```

THE METHOD OF MOLECULAR DYNAMICS

The equations of motion of a system of N interacting particles are solved numerically

The system is classical, and can be described by means of the potential energy

$$U = U\left(\left\{\vec{r}_k\right\}\right)$$

The equations of motion are:

$$\begin{cases} \dfrac{d\vec{r}_i}{dt} = \vec{v}_i \\ \dfrac{d\vec{v}_i}{dt} = -\dfrac{\nabla_i U}{m} \end{cases}$$

Since the energy is conserved, the system evolution can be visualised as a trajectory on a hypersurface in phase space,

$$H\left(\left\{\vec{r}_k\right\},\left\{\vec{v}_k\right\}\right) = T\left(\left\{\vec{v}_k\right\}\right) + U\left(\left\{\vec{r}_k\right\}\right) = const.$$

where T is the kinetic energy

Integration methods

Based on finite difference schemes, in which time *t* is discretised using a time interval *h*

Knowing the coordinates and the forces at time *t*, the state at a later time *t*+*h* is obtained

The simplest (but still powerful!) algorithm is the *Verlet algorithm*

VERLET ALGORITHM

We write the following Taylor expansions:

$$\vec{r}_i(t+h)=\vec{r}_i(t)+h\vec{v}_i(t)+\frac{h^2}{2m}\vec{F}_i(t)+\frac{h^3}{6}\ddot{\vec{v}}_i(t)+\ldots$$

$$\vec{r}_i(t-h)=\vec{r}_i(t)-h\vec{v}_i(t)+\frac{h^2}{2m}\vec{F}_i(t)-\frac{h^3}{6}\ddot{\vec{v}}_i(t)+\ldots$$

Adding, neglecting terms of order $O(h^4)$, and rearranging:

$$\vec{r}_i(t+h) = 2\vec{r}_i(t) - \vec{r}_i(t-h) + \frac{h^2}{m}\vec{F}_i(t)$$

This is a recurrence formula, allowing to obtain the coordinates at time *t+h* knowing the coordinates at times *t* and *t-h*

It is a formula of third order (the new positions contain errors of order h^4)

The velocities can be obtained from the expansion:

$$\vec{r}_i(t+h) = \vec{r}_i(t-h) + 2h\vec{v}_i(t) + O(h^2)$$

as

$$v_i(t) = \frac{\vec{r}_i(t+h) - \vec{r}_i(t-h)}{2h}$$ (which contains errors of order h^2)

The kinetic energy at time *t* can now be obtained as

$$E_c(t) = \sum_{i=1}^{N} \frac{1}{2} m \left|\vec{v}_i(t)\right|^2$$

Using the equipartition theorem, the temperature can be obtained as

$$\left\langle \frac{1}{2} m v_{\alpha}^{2} \right\rangle = \frac{kT}{2} \quad \longrightarrow \quad T = \frac{m}{N_f k} \left\langle \sum_{i=1}^{N} \left| \overset{e}{v}_i \right|^2 \right\rangle = \frac{2}{N_f k} \left\langle E_c \right\rangle$$

where

α= arbitrary degree of freedom

N_f= number of degrees of freedom

LEAP-FROG VERSION

Numerically more stable version than standard algorithm. We define:

$$\vec{v}_i\left(t-\frac{h}{2}\right)=\frac{\vec{r}_i(t)-\vec{r}_i(t-h)}{h},\quad \vec{v}_i\left(t+\frac{h}{2}\right)=\frac{\vec{r}_i(t+h)-\vec{r}_i(t)}{h}$$

Positions are updated according to

$$\vec{r}_i(t+h)=\vec{r}_i(t)+h\vec{v}_i\left(t+\frac{h}{2}\right)$$

Using the Verlet algorithm,

$$\vec{r}_i(t+h)-\vec{r}_i(t)=\vec{r}_i(t)-\vec{r}_i(t-h)+\frac{h^2}{m}\vec{F}_i(t)$$

so that

$$\vec{v}_i\left(t+\frac{h}{2}\right)=\vec{v}_i\left(t-\frac{h}{2}\right)+\frac{h}{m}\vec{F}_i(t)$$

Therefore leap-frog version or Hamiltonian version is

$$\begin{cases} \vec{v}_i\left(t+\frac{h}{2}\right)=\vec{v}_i\left(t-\frac{h}{2}\right)+\frac{h}{m}\vec{F}_i(t) \\ \vec{r}_i(t+h)=\vec{r}_i(t)+h\vec{v}_i\left(t+\frac{h}{2}\right) \end{cases}$$

The velocities are obtained as

$$\vec{v}_i(t)=\frac{\vec{v}_i\left(t+\frac{h}{2}\right)+\vec{v}_i\left(t-\frac{h}{2}\right)}{2}$$

Stability of trajectories

Systems with many degrees of freedom have a tendency to be unstable

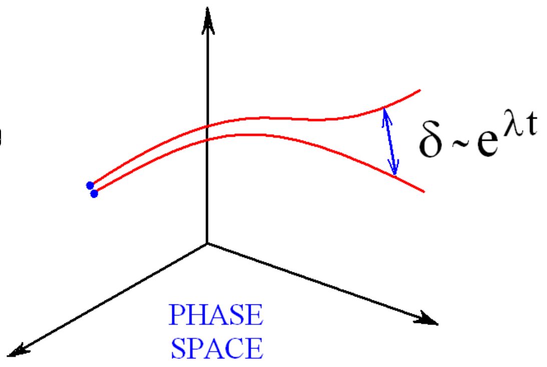
If δ is the distance in phase space between two trajectories that initially are very close, and write

$$\delta(t) = Ce^{\lambda t}$$

λ= Lyapounov coefficient

then the system is

- STABLE if δ< 0 or decreases faster than exponentially
- UNSTABLE if δ> 0; the system is said to be *chaotic*



Lyapounov instability is important because:

1. it limits the time beyond which an accurate trajectory can be found
2. to reach high accuracy after time *t* we need too many accurate decimal digits in the initial condition

$$e^{\lambda t} = c_0 10^{-\varepsilon} \longrightarrow \varepsilon = \frac{\log c_0 - \lambda t}{\log 10}$$

Since systems with many degrees of freedom are intrinsically unstable, very accurate integration algorithms are useless

The basic requirements that an algorithm should satisfy are:

1. **Time reversibility:**

$$\begin{cases} \dfrac{d\vec{r}_i}{dt} = \vec{v}_i \\ m\dfrac{d\vec{v}_i}{dt} = -\vec{\nabla}_i U \end{cases}$$

The equations are invariant under the transformations

$$t \to -t$$

$$\vec{v}_i \to -\vec{v}_i$$

The Verlet algorithm is time reversible

$$\begin{tikzcd} \vec{r}_i(t+h) = 2\vec{r}_i(t) - \vec{r}_i(t-h) + \dfrac{h^2}{m}\vec{F}_i(t) \arrow[r, "h \to -h"] & \vec{r}_i(t-h) = 2\vec{r}_i(t) - \vec{r}_i(t+h) + \dfrac{h^2}{m}\vec{F}_i(t) \arrow[d] \\ & \vec{r}_i(t+h) = 2\vec{r}_i(t) - \vec{r}_i(t-h) + \dfrac{h^2}{m}\vec{F}_i(t) \arrow[ul] \end{tikzcd}$$

Irreversibility in some algorithms induces intrinsic energy dissipation so that energy is not conserved

2. Symplecticity:

The probability distribution $f\left(\{\vec{r}_i\},\{\vec{v}_i\},t\right)$ evolves like an incompressible fluid, which implies $\dot{f} = 0$

A symplectic algorithm conserves the volume of phase space

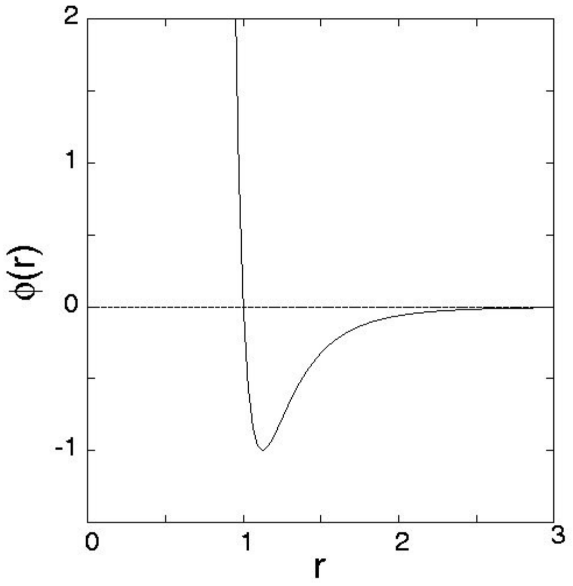
The Lennard–Jones potential

The Lennard–Jones potential, $\phi_{\text{LJ}}(r)$, is a pair potential that depends on the distance r between two particles:

$$\phi_{\text{LJ}}(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

- ϵ, energy parameter
- σ, zero of the potential (roughly the diameter of the particles)

The central problem of Molecular Dynamics is the calculation of forces. In this case the forces can be calculated analytically by differentiation of the potential.



Let N be the number of particles in the system. The potential energy of the system is:

$$U(\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N) = \frac{1}{2}\sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} \phi_{\mathrm{LJ}}\left(|\mathbf{r}_j - \mathbf{r}_i|\right) = \sum_{i=1}^{N}\sum_{j<i}^{N} \phi_{\mathrm{LJ}}\left(|\mathbf{r}_j - \mathbf{r}_i|\right)$$

The first version includes a $1/2$ prefactor in order not to count the same pair of particles twice. The force on particle i is:

$$\mathbf{F}_i = -\nabla_{\mathbf{r}_i} U = -\sum_{j\neq i} \nabla_{\mathbf{r}_i}\phi_{\mathrm{LJ}}\left(|\mathbf{r}_j - \mathbf{r}_i|\right) \equiv \sum_{j\neq i} \mathbf{F}_{ij}$$

Note that $\mathbf{F}_{ij} = -\nabla_{\mathbf{r}_i}\phi_{\mathrm{LJ}}\left(|\mathbf{r}_j - \mathbf{r}_i|\right)$ is the contribution of particle j to the force on particle i, and that, by Newton's third law, $\mathbf{F}_{ij} = -\mathbf{F}_{ji}$; this property helps save valuable computer time. Now:

$$\nabla_{\mathbf{r}_i}\phi_{\mathrm{LJ}}\left(|\mathbf{r}_j - \mathbf{r}_i|\right) = \phi'_{\mathrm{LJ}}\left(|\mathbf{r}_j - \mathbf{r}_i|\right)\nabla_{\mathbf{r}_i}|\mathbf{r}_j - \mathbf{r}_i| = \phi'_{\mathrm{LJ}}\left(|\mathbf{r}_j - \mathbf{r}_i|\right)\frac{\mathbf{r}_j - \mathbf{r}_i}{|\mathbf{r}_j - \mathbf{r}_i|}$$

Since

$$\phi'_{\mathrm{LJ}}(r) = -\frac{48\epsilon}{\sigma^2}\left[\left(\frac{\sigma}{r}\right)^{13} - \left(\frac{\sigma}{r}\right)^{7}\right]$$

we finally have

$$\mathbf{F}_i = \frac{48\epsilon}{\sigma^2}\sum_{j\neq i}\left[\left(\frac{\sigma}{r}\right)^{14} - \left(\frac{\sigma}{r}\right)^{8}\right](\mathbf{r}_j - \mathbf{r}_i)$$

A molecular-dynamics FORTRAN code

- declare variables
- starting positions and velocities
- initialise parameters
- MD loop
  - apply periodic boundary conditions
  - loop over pairs of particles
    - calculate relative position (using minimum image convention)
    - accumulate forces
    - accumulate energies, virial, etc.
  - update positions and velocities (leap frog)
- averages and print out

```
!
!  MD for Lennard-Jones particles
!
implicit real*8(a-h,o-z)
parameter (n=256)
dimension x(n),y(n),z(n)
dimension vx(n),vy(n),vz(n)
dimension fx(n),fy(n),fz(n)
dimension ig(1000)
real*4 r
common//hgr,cutr2,ngr

rho=0.8
T=2
npasos=2000

dum = 17367d0
pi = 4d0 * datan(1d0)
call init (n, rho, T, x, y, z, aL)
do i=1,n
vr = dsqrt(3*T)
call ggub(dum,r)
cost = 2*r-1
sint = dsqrt(1-cost**2)
call ggub(dum,r)
fi = r*2*pi
vx(i) = vr*sint*dcos(fi)
vy(i) = vr*sint*dsin(fi)
vz(i) = vr*cost
end do

```

Particles initially on nodes of a fcc lattice

Maxwell-Boltzmann distribution for velocities

```
aL2 = aL/2d0
cut2 = (2.5d0)**2
cutr2 = (aL/2)**2
ngr = 100
hgr = aL2/ngr
dt = 0.01d0

ec = 0
u = 0
ap = 0

do i = 1,1000
ig(i) = 0
end do

do k = 1,npasos

do i = 1, n
call pbc (x,y,z,aL,aL,aL,aL2,aL2,aL2) ←—— periodic boundary conditions
end do
do i = 1, n
fx(i) = 0
fy(i) = 0
fz(i) = 0
end do
epot=0
```

```
do i = 1, n-1
xi = x(i)
yi = y(i)
zi = z(i)
do j = i+1, n
xx = xi-x(j)
yy = yi-y(j)
zz = zi-z(j)
call mic (xx,yy,zz,aL,aL,aL,aL2,aL2,aL2)
r2 = xx**2+yy**2+zz**2
if (r2 .lt.  cut2) then
r1 = 1/r2
r6 = r1**3
pot=4*r6*(r6-1)
u = u+pot
epot=epot+pot
rr = 48*r6*r1*(r6-0.5d0)
fxx = rr*xx
fyy = rr*yy
fzz = rr*zz
ap = ap+rr*r2
fx(i) = fx(i)+fxx
fy(i) = fy(i)+fyy
fz(i) = fz(i)+fzz
fx(j) = fx(j)-fxx
fy(j) = fy(j)-fyy
fz(j) = fz(j)-fzz
end if
end do
end do

```

minimum image convention

FORCE calculation

add up forces to each member of *ij* pair

```
ekin=0
do i=1,n
vxi = vx(i)+dt*fx(i)
vyi = vy(i)+dt*fy(i)
vzi = vz(i)+dt*fz(i)
vxx = 0.5d0*(vxi+vx(i))
vyy = 0.5d0*(vyi+vy(i))
vzz = 0.5d0*(vzi+vz(i))
en = vxx**2+vyy**2+vzz**2
ekin = ekin+en
ec = ec+en
vx(i) = vxi
vy(i) = vyi
vz(i) = vzi
x(i) = x(i)+dt*vx(i)
y(i) = y(i)+dt*vy(i)
z(i) = z(i)+dt*vz(i)
end do
```

update velocities and positions

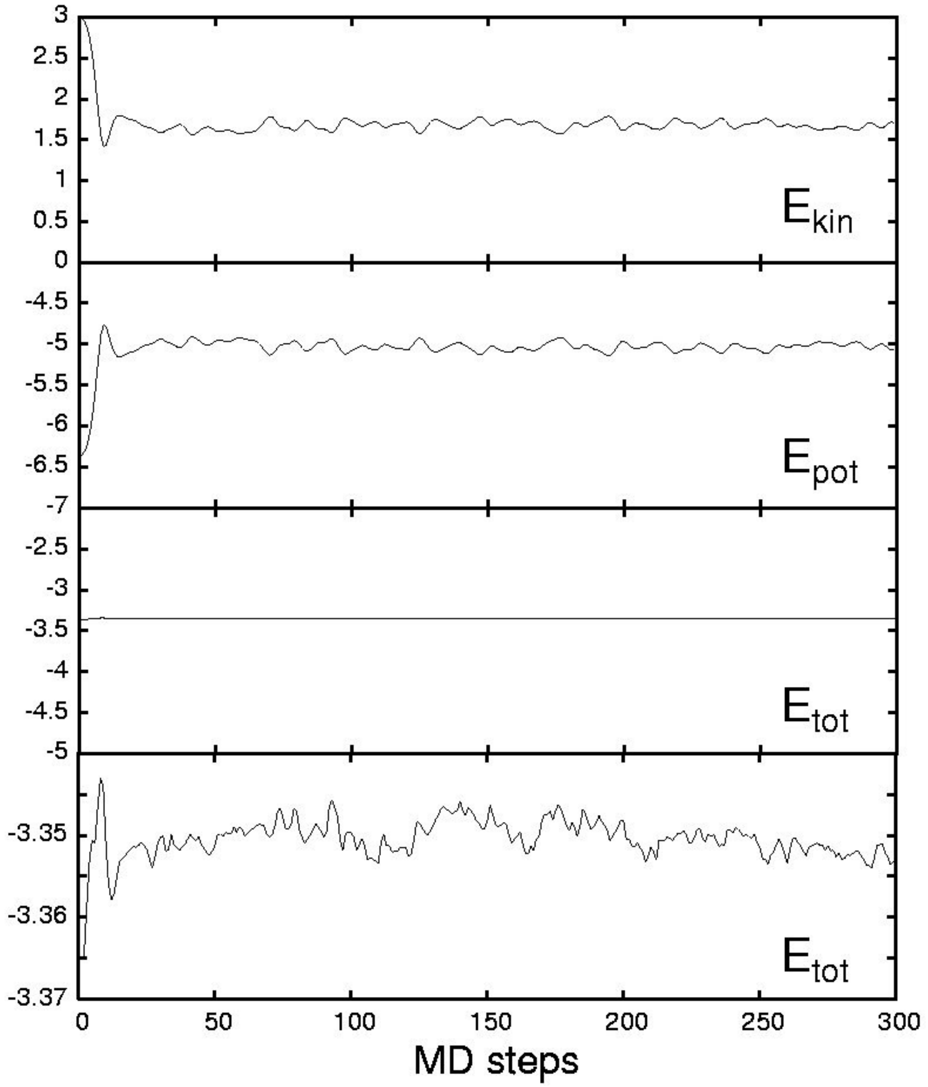
```

call gr(x,y,z,aL,aL2,ig)
```

$\leftarrow$ histogram for *g(r)*

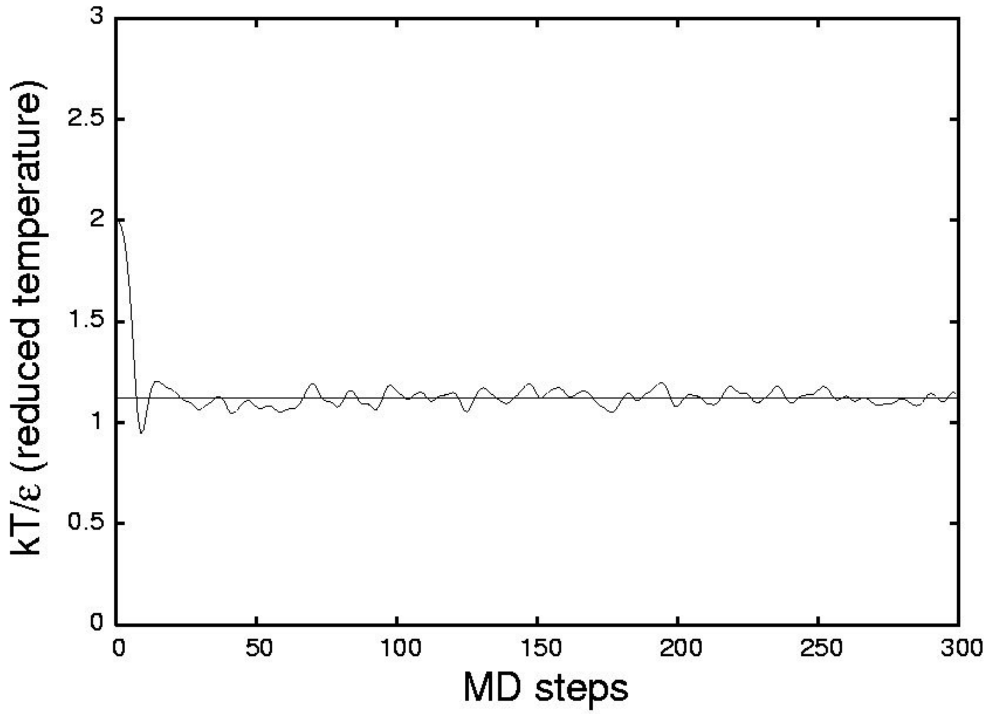
```

end do

temp =ec/((3*n-3)*npasos)
u = u/(n*npasos)
ap = rho*temp+ap/(3*aL**3*npasos)

write(*,*) 'temperature=',temp
write(*,*) 'energy=',u
write(*,*) 'pressure=',ap

```

```
do k=1,ngr
r=(k-1)*hgr+hgr/2d0
vol=4*pi/3*((r+hgr/2d0)**3-(r-hgr/2d0)**3)
gdr=2*ig(k)/(n*(n-1)/aL**3*npasos*vol)
write(1,*) r,gdr
end do

end
```

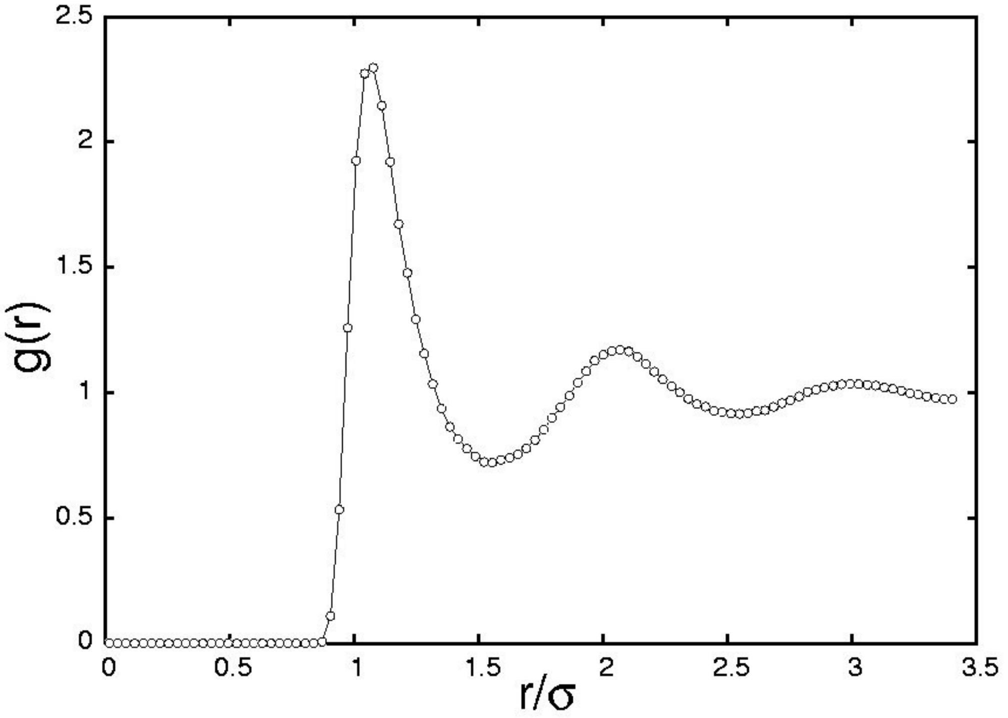
normalise radial distribution function



3
2.5
2
1.5
1
0.5
0
E_{kin}
-4.5
-5
-5.5
-6
-6.5
-7
E_{pot}
-2.5
-3
-3.5
-4
-4.5
-5
E_{tot}
-3.35
-3.36
-3.37
E_{tot}
0
50
100
150
200
250
300
MD steps



kT/ε (reduced temperature)
MD steps
0
0.5
1
1.5
2
2.5
3
0
50
100
150
200
250
300



g(r)
r/σ
0
0.5
1
1.5
2
2.5
3
3.5

FIN

MÉTODOS COMPUTACIONALES EN FíSICA DE LA MATERIA CONDENSADA
Curso de doctorado del programa de Física de la Materia Condensada
Curso 2006-2007

E. Velasco
Departamento de Física Teórica de la Materia Condensada
Facultad de Ciencias
Universidad Autónoma de Madrid

CONTENTS

EL MÉTODO DE DINÁMICA MOLECULAR

Se resuelven numéricamente las ecuaciones de movimiento de un sistema de N partículas interactuantes que, clásicamente, se pueden escribir en términos de la energía potencial

$$U = U(\{\mathbf{r}_k\}), \qquad q \equiv \{\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_N\}$$

como

$$\begin{cases} \dfrac{d\mathbf{r}_i}{dt} = \mathbf{v}_i \\ \\ \dfrac{d\mathbf{v}_i}{dt} = -\nabla_i U(\{\mathbf{r}_k\}) \, /m \end{cases}$$

Con una dinámica que conserva la energía, la evolución del sistema se puede visualizar como una trayectoria sobre una hipersuperficie

$$H(\{\mathbf{r}_k\}, \{\mathbf{v}_k\}) = T(\{\mathbf{r}_k\}) + U(\{\mathbf{r}_k\}) = \text{const.}$$

en el espacio de fases, donde $T(\{\mathbf{v}_k\})$ es la energía cinética

Métodos de integración

Se basan en esquemas en diferencias finitas, en los que el tiempo t se discretiza a través de un intervalo h

Sabiendo las coordenadas en t y las fuerzas $\mathbf{F}_i(t)$ se trata de obtener el estado en el instante posterior $t+h$

El algortimo más simple es el algoritmo de Verlet:

- **Algoritmo de Verlet**. Hacemos los siguientes desarrollos de Taylor:

$$\mathbf{r}_i(t+h) = \mathbf{r}_i(t) + h\mathbf{v}_i(t) + \frac{h^2}{2m}\mathbf{F}_i(t) + \frac{h^3}{6}\ddot{\mathbf{v}}_i(t) + ...$$

$$\mathbf{r}_i(t-h) = \mathbf{r}_i(t) - h\mathbf{v}_i(t) + \frac{h^2}{2m}\mathbf{F}_i(t) - \frac{h^3}{6}\ddot{\mathbf{v}}_i(t) + ...$$

  Sumando, despreciando términos $O(h^4)$, y reorganizando los términos que quedan:

$$\mathbf{r}_i(t+h) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t-h) + \frac{h^2}{m}\mathbf{F}_i(t)$$

  Esta fórmula de recurrencia nos permite, conociendo $\mathbf{r}_i(t)$ y $\mathbf{r}_i(t-h)$, obtener la nueva posición $\mathbf{r}_i(t+h)$. Se trata de un algoritmo de $O(h^3)$ [es decir, la nueva posición contiene errores de $O(h^4)$].

La velocidad se puede obtener del desarrollo

$$\mathbf{r}_i(t+h) = \mathbf{r}_i(t-h) + 2h\mathbf{v}_i(t) + O(h^2)$$

de donde

$$\mathbf{v}_i(t) = \frac{\mathbf{r}_i(t+h) - \mathbf{r}_i(t-h)}{2h}$$

que contiene errores de $O(h^2)$. La energía cinética, en el instante t, se puede calcular de

$$E_c(t) = \sum_{i=1}^{N} \frac{1}{2} m \left|\mathbf{v}_i(t)\right|^2$$

y la temperatura, utilizando el teorema de equipartición:

$$\left\langle \frac{1}{2} m v_\alpha^2 \right\rangle = \frac{kT}{2} \quad \rightarrow \quad T = \left\langle \sum_{i=1}^{N} \frac{m \left|\mathbf{v}_i(t)\right|^2}{kN_f} \right\rangle$$

$\alpha =$ grado de libertad cualquiera, y $N_f =$ número de grados de libertad del sistema

- **Versión de** *leap-frog*. Es una versión numéricamente más estable que el algoritmo de Verlet. Se definen:

$$\mathbf{v}_i\left(t-\frac{h}{2}\right)=\frac{\mathbf{r}_i(t)-\mathbf{r}_i(t-h)}{h}, \quad \mathbf{v}_i\left(t+\frac{h}{2}\right)=\frac{\mathbf{r}_i(t+h)-\mathbf{r}_i(t)}{h}.$$

  Las posiciones se actualizan con la expresión

$$\mathbf{r}_i(t+h)=\mathbf{r}_i(t)+h\mathbf{v}_i\left(t+\frac{h}{2}\right)$$

  Usando el algoritmo de Verlet,

$$\mathbf{r}_i(t+h)-\mathbf{r}_i(t)=\mathbf{r}_i(t)-\mathbf{r}_i(t-h)+\frac{h^2}{m}\mathbf{F}_i(t)$$

  de donde

$$\mathbf{v}_i\left(t+\frac{h}{2}\right)=\mathbf{v}_i\left(t-\frac{h}{2}\right)+\frac{h}{m}\mathbf{F}_i(t)$$

De aquí obtenemos la versión *hamiltoniana* del algoritmo de Verlet:

$$\begin{cases} \mathbf{r}_i(t+h)=\mathbf{r}_i(t)+h\mathbf{v}_i\left(t+\frac{h}{2}\right) \\ \\ \mathbf{v}_i\left(t+\frac{h}{2}\right)=\mathbf{v}_i\left(t-\frac{h}{2}\right)+\frac{h}{m}\mathbf{F}_i(t) \end{cases}$$

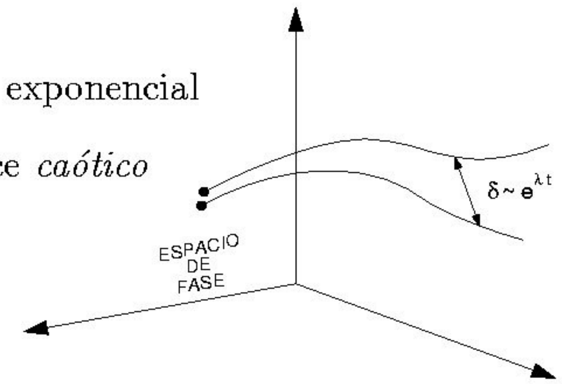
$$\mathbf{v}_i(t)=\frac{\mathbf{v}_i\left(t+\frac{h}{2}\right)+\mathbf{v}_i\left(t-\frac{h}{2}\right)}{2}$$

Estabilidad de las trayectorias

Los sistemas de muchos grados de libertad tienen una tendencia a ser *inestables*

Si escribimos la distancia entre dos trayectorias que empezaron muy juntas como $\delta(t) \sim e^{\lambda t}$, el sistema decimos que es:

- ESTABLE si $\lambda < 0$ ó δ decrece más lentamente que una exponencial
- INESTABLE si $\lambda > 0$; en esta situación el sistema se dice *caótico*



El coeficiente λ se conoce como *coeficiente de Lyapunov*

La inestabilidad de Lyapunov, que se da cuando $\lambda > 0$, es importante por dos razones íntimamente relacionadas:

- limita el tiempo en el que se puede encontrar una solución numérica precisa, en la que podamos confiar
- para alcanzar una precisión alta en la trayectoria, tras un tiempo t, se necesita un número de dígitos desproporcionadamente alto en la condición inicial; en concreto, este número crece linealmente con el tiempo: si ϵ es el número de dígitos,

$$e^{\lambda t} \sim 10^{-\epsilon} \qquad \rightarrow \qquad \epsilon \sim \frac{\lambda t}{\log 10}$$

Los sistemas de muchos grados de libertad son *intrínsecamente inestables* y, por tanto, *intrínsecamente caóticos*. Los algoritmos numéricos de integración muy precisos son inútiles

Las propiedades básicas exigibles a un algoritmo de integración son:

1. Que sea reversible en el tiempo

$$\begin{cases} \dfrac{d\mathbf{r}_i}{dt} = \mathbf{v}_i \\ \\ m\dfrac{d\mathbf{v}_i}{dt} = -\nabla_i U \end{cases}$$

   Si hacemos la transformación $t \to -t, \mathbf{v}_i \to -\mathbf{v}_i$ las ecuaciones no cambian de forma
   El algoritmo de Verlet, al hacer $h \to -h$, respeta esta propiedad:

$$\mathbf{r}_i(t+h) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t-h) + \frac{h^2}{m}\mathbf{F}_i(t)$$

$$\longrightarrow \quad \mathbf{r}_i(t-h) = 2\mathbf{r}_i(t) - \mathbf{r}_i(t+h) + \frac{h^2}{m}\mathbf{F}_i(t)$$

   La irreversibilidad de algunos algoritmos de integración introduce una *disipación de energía* intrínseca que provoca que la energía del sistema no se conserve.

2. Que sea simpléctico
   Esto quiere que la función de probabilidad del sistema, $f(\{\mathbf{r}_k\}, \{\mathbf{v}_i\}, t)$ evoluciona en el espacio de fases como un fluido incompresible, lo cual implica que $\dot{f} = 0$

   Un esquema numérico *simpléctico conserva el volumen del espacio de fases.*

```
do n=1,Ntot
call ran (dseed,r)
k=4*r+1
x=x+i(k)
y=y+j(k)
end do
r2=x**2+y**2
ar2=ar2+r2
end do
write(*,'(''N, M, <r2>='',2i8,f12.5)') Ntot,M,ar2/M
end do

end
```