7. Random walks

[G+T 7.3, G+T 12]

Now, for the first time on this course, we start to deal with actual simulation of physical processes, as opposed to using MC for integration or data analysis. Let us start with a typical problem in physics, a couple of drunken sailors.

7.1. Introduction

7.1.1. The two drunken sailors

Consider a sailor who gets out of a corner pub completely drunk, in a city where all blocks are in perfect squares. The drunken sailor walks from one corner of a block to the next, but then in this corner forgets where he was coming from, and where he is going. He keeps going, picking the next direction where to go randomly among the four possible ways. Then in the following corner he again forgets where he was going, but keeps on walking, forming a random walking pattern.

Consider another drunken sailor, who has drunk even more, and fallen asleep in the middle of a huge square. He also starts walking, but after a few steps he stops, changes direction randomly, takes another few steps, again changes direction, and so on.

We obviously could use Monte Carlo methods to simulate the walk of the two drunken sailors, and answer questions like how far they on average have come after N steps. But why on earth would we want to, being physicists?

7.1.2. ...and what this has to do with physics

The answer is that exactly the same kind of **random walk** processes are surprisingly prevalent in physics. The random walk performed by the sailor walking among the square blocks is migration on a lattice, and can correspond exactly to the motion of an atom migrating on a (square) lattice. The second case, the movements of the sailor on the square, is migration in a continuum, and can correspond e.g. to atom movement in a two-dimensional liquid. The extension to non-square lattices, 3D lattices or 3D liquids is obvious.

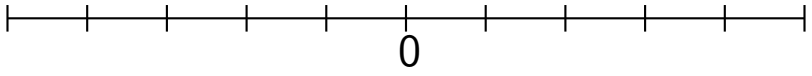
Because of such relations to the natural sciences, the basic "drunken sailor" random walk problem is widely used in many branches of science. Answering e.g. the question of average expected distance for a given N has a direct correspondence to the definition of the diffusion constant!

7.2. Simple random walks and diffusion

[G+T 7.3]

7.2.1. One-dimensional walk

Let us first consider the simplest possible case, a random walker in a one-dimensional lattice:



Say that a walker begins at $x = 0$, and that all steps are of equal length l. After each time interval τ the walker has an equal probability of moving left or right. The direction of each step is independent of the previous one. Let us denote the displacement at each step by s_i, for which

$$s_i = \begin{cases} +l \text{ with 50\% probability} \\ -l \text{ with 50\% probability} \end{cases}$$

Then after N steps (time $N\tau$) in the random walk, the position (and displacement) x of the walker is

$$x(N) = \sum_{i=1}^{N} s_i$$

and the displacement squared is

$$x^2(N) = \left(\sum_{i=1}^{N} s_i \right)^2 \tag{1}$$

Let us now think about what the average distance the walker has moved is. It is immediately obvious that with the equal probabilities to go left and right,

$$\langle x(N) \rangle = 0$$

that is, the average **position** will always be at the origin. But this does of course not mean that the particle always is at zero. It means that the probability of finding the particle somewhere is $centered$ at $x = 0$, but naturally the probability distribution gets wider with increasing numbers of steps N.

To get a handle on the broadening, let us consider the squared displacement, Eq. 1. We can rewrite this as

$$x^2(N) = \left(\sum_{i=1}^{N} s_i \right)^2 = \sum_{i=1}^{N} s_i \sum_{j=1}^{N} s_j = \sum_{i=1}^{N} s_i^2 + \sum_{i=1}^{N} \sum_{\substack{j=1 \\ j \neq i}}^{N} s_i s_j$$

Then consider the pair $s_i s_j$ for a given pair $i, j, j \neq i$. This quantity will be

$$s_i s_j = \begin{cases} +l^2 \text{ with 50\% probability} \\ -l^2 \text{ with 50\% probability} \end{cases}$$

so on average the sum over $s_i s_j$ will be zero! But on the other hand

$$s_i^2 = l^2$$

independently of whether s_i is $+l$ or $-l$!

Hence the average after N steps will be

$$\langle x^2(N) \rangle = l^2 N \tag{2}$$

7.2.1.1. Coding the random walk

Before we proceed to look at deeper into the physical significance of this, let us see how we can do a Monte Carlo simulation of this. There is nothing hard here: we simply generate N uniform random numbers between 0 and 1, and if the uniform number is below 0.5 we walk downwards, otherwise we walk upwards. We then collect the result, and repeat for a large number n_{times} times to get a representative average. Here is an implementation in Fortran90:

```
program randomwalk1d
  implicit none

  integer :: N,nwalks,iwalk,istep
  integer :: x,seed
  double precision :: xsum,xsqsum,dir

  character :: buf*80
  integer, external :: iargc
  double precision, external :: uniformrand

  if (iargc() < 3) then                                  |  PRELIMINARIES
     print *,'Usage: randomwalk N nwalks seed'           |
     STOP                                                |
  endif                                                  |
                                                         |
  call getarg(1,buf); read (unit=buf,fmt=*) N            |
  call getarg(2,buf); read (unit=buf,fmt=*) nwalks       |
  call getarg(3,buf); read (unit=buf,fmt=*) seed         |
                                                         |
  print *,'Doing  random walk to',N,'steps',nwalks,' times'  _|

  xsum=0.0; xsqsum=0.0;                                  |  RANDOM WALK
  do iwalk=1,nwalks          ! LOOP OVER WALKS           |
```

```
    ! New walk                                                    |
    x=0;                                                         |
    do istep=1,N           ! LOOP OVER STEPS IN WALK iwalk       |
       dir=uniformrand(seed)                                     |
       if (dir<0.5) then                                         |
          x=x-1                                                  |
       else                                                      |
          x=x+1                                                  |
       endif                                                     |
    enddo                                                        |
    xsum=xsum+x                                                  |
    xsqsum=xsqsum+x*x                                            |
  enddo                                                         _|

  print *,'<x(N)>',xsum/nwalks,' <x^2(N)>',xsqsum/nwalks

end program randomwalk1d
```

When ran, this gave e.g. the following result:

```
beam.helsinki.fi randomwalk> nice randomwalk1d 100 100000 12313
 Doing  random walk to  100 steps  100000  times
 <x(N)> -2.648000000000000E-002  <x^2(N)>  100.260080000000
```

```
beam.helsinki.fi randomwalk> nice randomwalk1d 100 100000 7142
 Doing  random walk to  100 steps  100000  times
 <x(N)>  2.7340000000000000E-002  <x^2(N)>  100.126840000000
```

and with a hundred times better statistics:

```
 Doing  random walk to  100 steps  10000000  times
 <x(N)> -1.9328000000000000E-003  <x^2(N)>  99.9611160000000
 Doing  random walk to  100 steps  10000000  times
 <x(N)>  1.7720000000000000E-004  <x^2(N)>  99.9962144000000
```

So we see that indeed $\langle x(N) \rangle$ is on average 0 and $\langle x^2(N) \rangle \approx N$, as predicted by our analytical derivation!

7.2.2. Continuum limit: the diffusion equation

[G+T 12.5; G+T 7A]

This basic random walk can be rewritten as a continuum diffusion equation by taking the limit in which the lattice spacing l and the time step τ go to zero.

Let us begin by writing the random walk behaviour in terms of a so called **master equation.** Let $P(i, N)$ denote the probability that a walker is at site i after N steps. Since walkers have an equal probability to walk left and right, it is clear that

$$P(i, N) = \tfrac{1}{2}P(i + 1, N - 1) + \tfrac{1}{2}P(i - 1, N - 1)$$

To get a continuum limit with familiar names for the variables, we can identify

$$t = N\tau \qquad \text{and} \qquad x = il$$

Now we can rewrite the previous equation as

$$P(x/l, t/\tau) = \tfrac{1}{2}P(x/l + 1, t/\tau - 1) + \tfrac{1}{2}P(x/l - 1, t/\tau - 1)$$

but since the probability is independent of the length or time scales, we have

$$aP(x,t) = P(ax,t) \qquad \text{or} \qquad bP(x,t) = P(x,bt)$$

for any constants a, b. So we can multiply the equation with l and τ to obtain

$$P(x,t) = \tfrac{1}{2}P(x+l,t-\tau) + \tfrac{1}{2}P(x-l,t-\tau)$$

We rewrite this by subtracting $P(x,t-\tau)$ and dividing by τ

$$\frac{P(x,t) - P(x,t-\tau)}{\tau} = \frac{P(x+l,t-\tau) + P(x-l,t-\tau) - 2P(x,t-\tau)}{2\tau} \quad (3)$$

The left hand side already clearly resembles the definition of a derivative, if we take the limit $\tau \longrightarrow 0$. To see what happens on the right hand side, we expand the P functions as Taylor series about x and t with l and τ as the deviation. We only write out the terms needed:

$$P(x,t-\tau) \approx P(x,t) - \frac{\partial P(x,t)}{\partial t}\tau$$

$$P(x \pm l,t-\tau) \approx P(x,t) \pm \frac{\partial P(x,t)}{\partial l}l + \tfrac{1}{2}\frac{\partial^2 P(x,t)}{\partial x^2}l^2 - \frac{\partial P(x,t)}{\partial t}\tau$$

and we can rewrite eq. 3 as

$$\frac{P(x,t) - P(x,t-\tau)}{\tau} \approx \frac{1}{2\tau} \qquad [P(x+l,t-\tau) + P(x-l,t-\tau) - 2P(x,t-\tau)]$$

$$\frac{P(x,t) - P(x,t) + \frac{\partial P(x,t)}{\partial t}\tau}{\tau} \approx \frac{1}{2\tau} \qquad \left[P(x,t) + \frac{\partial P(x,t)}{\partial l}l + \tfrac{1}{2}\frac{\partial^2 P(x,t)}{\partial x^2}l^2 - \frac{\partial P(x,t)}{\partial t}\tau \right.$$

$$+P(x,t) - \frac{\partial P(x,t)}{\partial l}l + \tfrac{1}{2}\frac{\partial^2 P(x,t)}{\partial x^2}l^2 - \frac{\partial P(x,t)}{\partial t}\tau$$

$$\left. -2P(x,t) + 2\frac{\partial P(x,t)}{\partial t}\tau\right]$$

and after we do all the obvious cancellations we get

$$\frac{\partial P(x,t)}{\partial t} \approx \frac{l^2}{2\tau}\frac{\partial^2 P(x,t)}{\partial x^2}$$

In the limit $\tau \longrightarrow 0, l \longrightarrow 0$ but where the ratio l^2/τ is finite, this becomes an exact relation. If we

further define $D \equiv l^2/2\tau$ we get the equation

$$\frac{\partial P(x,t)}{\partial t} = D\frac{\partial^2 P(x,t)}{\partial x^2} \tag{4}$$

But this is just the well-known macroscopic diffusion equation in 1 dimension, with D being exactly the diffusion constant. So we have from a very simple microscopic model derived a well-known macroscopic equation!

For those of you who do not know diffusion, what the equation does is describes the time dependence of e.g. particle or heat diffusion in a system with a concentration or temperature gradient.

If we furthermore compare with the previous equation 2, $\langle x^2(N)\rangle = l^2 N$ and use the identities $D \equiv l^2/2\tau$ and $t = N\tau$ we can rewrite eq. 2 as

$$\langle x^2(t)\rangle = 2Dt \tag{5}$$

This equation is called the **Einstein relation** after a certain physicist who first derived it back in 1905 [A. Einstein, Ann. Phys. 17 (1905) 549].

The 3-dimensional generalizations, which are straightforward to derive in a similar manner, are:

$$\frac{\partial P(x,y,z,t)}{\partial t} = D\nabla^2 P(x,y,z,t) \tag{6}$$

and for the Einstein relation:

$$\langle R^2(t)\rangle = 6Dt \tag{7}$$

where

$$\langle R^2(t)\rangle = \langle x^2(t) + y^2(t) + z^2(t)\rangle$$

and the factor of 2 has been replaced by a 6.

Furthermore, it even turns out that these results are not specific to being on a lattice, as you will find yourself in one of the exercises.

7.2.3. Microscopic and macroscopic diffusion

What is the relation between using MC simulations to deal with a random walk and the macroscopic equation? This depends on the outlook. If we are interested in macroscopic diffusion, we can use MC simulations of random walk essentially as an advanced numerical means to solve the diffusion equation 6.

Of course ordinary numerical solution of partial differential equations is in many cases likely to be perfectly adequate for dealing with diffusion, and much more efficient than random walk simulations. But the more complicated the diffusion problem gets, the harder it becomes to set up and solve the diffusion equation, and at some point it may actually become both easier and more efficient to use a microscopic random walk model. By utilizing the kinetic Monte Carlo approach (dealt with in the next section of this course) to random walks the microscopic simulations actually can become quite efficient.

On the other hand, if we are primarily interested in understanding the microscopic processes themselves, we can do random-walk like simulations on the microscopic level to see what is really going on. The relation with the diffusion equation can then just be used check that the microscopic

mechanisms will lead to the expected macroscopic limit if we expand our number of walkers towards infinity and look at larger length scales.

For instance, if we look at atom migration, the l can naturally be the known interatomic distance, and τ the average jump frequency of the atoms. In this case taking the limit $l \longrightarrow 0$ and $\tau \longrightarrow 0$ has no physical meaning since the real l and τ are finite; instead we can think of obtaining the macroscopic, continuum diffusion equation by looking at time scales $t >> \tau$ and length scales $L >> l$.

The Einstein relation is extremely useful for this kind of problems in that they allow determining the macroscopic diffusion constant in a simple way from microscopic data of Δx and τ. In the pure random walk as the ones here, there is actually not much information to gain as one has to know τ in advance. But if for instance MD simulations are used to predict the defect motion, it is sometimes truly possible to $predict$ the diffusion coefficient.

It is also interesting to note that present-day physics very often deals with truly atomistic length scales — in commercial Si microprocessors the smallest parts are already today only a few ten atom layers thick. In this kind of system, the assumption that the system length scale $L >> l$ may no longer be true, and the continuum approximation starts to become increasingly inaccurate. In such

cases a random walk look on migration may be the most appropriate way to deal with diffusion, and MC simulations of random walk the best tool to work with theoretically. This has now lead to the semiconductor industry being directly involved in atom-level diffusion simulations.

7.3. Modified random walks

In addition to the completely isotropic random walk described above, there are numerous variation where the walk directions are non-isotropic or constrained somehow.

7.3.1. Scaling law

In considering other types of random walks, one often uses the concept of a **scaling law** to describe the basic properties of the generator. For the basic random walk, we considered the dispersion

$$\langle \Delta x^2(N) \rangle = \langle x^2(N) \rangle - \langle x(N) \rangle^2$$

and found that

$$\langle \Delta x^2(N) \rangle = \langle x^2(N) \rangle \propto N^1$$

where N is the number of steps.

In other, more complicated walks it need not be true that $\langle x(N) \rangle = 0$, nor does the exponent on N need to be 1. Hence one can write for a general random walk that

$$\langle \Delta x^2(N) \rangle \propto N^{2\nu}$$

and use the exponent ν to characterize the walk. A similar scaling law dependence can also be valid for other quantities f of interest. It is customary to have 2ν in the exponent, because $\sqrt{f}$ then gives ν directly.

To determine the exponent in practice, it is often useful to make statistics of $\langle \Delta x^2(N) \rangle$ as a function of N while N is growing in the simulation. The one can output $f(N)$ and use a fitting routine to determine the exponent.

(*Hint for novices in data analysis*: a useful trick to do an extremely quick test of whether something follows a scaling law: if $f(N)$ is plotted in a log-log graph, it will appear linear if it follows a power law. The slope gives the exponent. For `xgraph` the options `-lnx -lny` give a log-log plot).

7.3.2. Examples

[G+T 12.2 –]

In here we will first list a few common types of modified random walks, very briefly describing what they are. In the next two subsections we will look at two examples in greater detail, also describing what real-life cases they correspond to.

7.3.2.1. Nonequal probabilities

It is of course possible to forgo the requirement that a step in all directions is taken with equal probability. Say for instance that in the 1-dimensional example the probability to move right is p and to the left is $q = 1 - p$. This will introduce a **drift** in the system, and we will have

$$\langle x(N) \rangle = (p - q) l N$$

instead of 0. This is called a **biased random walk,** and corresponds to the macroscopic diffusion equation with an extra term which can be interpreted as a velocity:

$$\frac{\partial P(x,t)}{\partial t} = D \frac{\partial^2 P(x,t)}{\partial x^2} - v \frac{\partial P(x,t)}{\partial x} \tag{8}$$

The physical interpretation can e.g. be a raindrop falling in a swirling breeze.

7.3.2.2. Persistent random walks

In the simple random walk, the probability that a step is taken into a certain direction is independent of the previous steps, and in fact constant all the time. In case the jump probability depends on the previous transition, one can talk about a **persistent** random walk.

A simple example is again given in a 1D system: if a jump has just been made at step $N-1$, let us say that the probability that the next jump occurs $in\ the\ same\ direction$ is α, and the probability it occurs in the opposite direction is $1-\alpha$. In case $\alpha = 0.5$ we have just the ordinary random walk. But if $\alpha \neq 0.5$ the two jumps will be correlated. If $\alpha > 0.5$ there is a simple physical interpretation of the correlation: once an object starts moving, it is more likely it will keep moving in the same direction on successive steps.

This has e.g. been observed in the study of molecules in a chromatographic column. These molecules can either be in an immobile, trapped phase with zero velocity, or in a mobile phase with velocity v. For steps of unit length the position will then change at each step by v or 0. But once the molecule is in a mobile state, it tends to stay there, so one could describe the motion using an $\alpha > 0.5$ to determine how it behaves.

7.3.2.3. Restricted random walks

Another important modification of the walk is to consider a system with certain points or regions which somehow restrict the motion of the atom. Such a restriction could e.g. be a line which reflects particles, a region which absorbs them or that two particles can not occupy the same site.

The simplest possible restriction is probably a **trap** in a lattice. It is simply a site which absorbs every random walker it encounters.

Traps are important in condensed matter physics. For instance, we can consider a simple model for light energy transport in a solid. A lattice has "hosts" and traps. When an incoming photon reaches a host, the light energy will be transferred into an electron excitation, which can travel in the lattice somewhat like a particle, an "exciton". When the exciton reaches a trap site it is absorbed there, and a chemical reaction can occur there due to the added energy from the exciton.

For defects in solids, a surface often acts as a "trap line" which absorbs all defects which reach it. In the exercises, you get to code this problem (although just for fun I mask it as a drunken sailor problem).

In subsection below I describe in detail how to model a system where particles can not occupy the same site.

7.3.2.4. Continuum walks

There is no reason why a random walk should only be on a lattice with a constant step size. If you e.g. think about atom motion in a gas, it is quite clear that the distance between collisions is not very likely to de constant, and the atoms can of course move in any direction.

There are **continuum** walks, where the steps can be in any direction. The step length may be fixed, or of variable length with some distribution.

As a simple example, if you do a 2D random walk in a random direction with steps of unit length, it turns out that for large N the distribution of displacements will become a Gaussian.

Finally, it is of course also possible to have a walk on a lattice where the displacement may be longer than to the nearest neighbour sites. Certain interstitial atom diffusion mechanisms can have such a character [1].

[1] For those of you who know about diffusion in metals, I am thinking of crowdion motion in BCC metals

7.4. Lattice gas

Consider a finite lattice with some density ρ of N_p particles. The particles can move on the lattice by jumps to the nearest sites, but two particles can not occupy the same site. This is a simple example of a **restricted** random walk (see above). The physical interpretation is e.g. vacancies moving in a lattice.

To simulate this kind of system, we need a bit more of an advanced approach than before. First of all, we need to simulate the motion of all the particles at the same time, not taking the average over many independent single-particle motions as was done before.

To be able to meet the criterion that two particles should not occupy the same site, we can do two things. One is to make an array which contains all possible lattice sites. The other is to, at each move, find the distance to all other particles and check that no one occupies the site to be moved to. In case the lattice is small enough to fit the available computer memory using the former solution is much easier and faster.

In case a particle jumps on average after every time Δt, then if we have N_p particles we should

make a move of one of the particles at time intervals of

$$\Delta t_p = \Delta t / N_p$$

But it is possible that a particle can not move at all, if all neighbouring sites are occupied. Should the time in this case be advanced by Δt_p or not? This depends on the physical situation we are involved with. In this case, we will choose to advance the time.

Another choice to make is whether we should always just try a jump in one direction, or for a given particle choice choose one of the available jumps if any is available. This time, we choose the latter one, because it is $harder$ to code.

Finally, we have to choose how to treat the borders. Now we will choose to use **periodic boundaries.** This simply means that if a particle exits the boundary on one side, it comes out on the other side. This mimics a section of an infinite system.

There is one non-trivial practical thing to consider when coding this. That is that we can not now simply evaluate

$$\Delta x = x(t = 0) - x(t)$$

because the particles may move several times over the periodic boundaries around the cell. In this case the above equation could give a much too small answer. What we have to do is introduce another array dx which sums up the displacement of the particle, before consideration of the periodic boundaries.

(Note that if you read Gould-Tobochnik about this, their code does not take account of this!)

So to summarize, here is a sketch of an algorithm on how to simulate this kind of a walk.

1° Choose number of particles N_p, number of steps $N_{\rm steps}$, side length L. Set Δt and lattice size a.

2° Set all positions in the $L \times L$ grid to be empty

3 a° Generate N_p particle coordinates randomly on the grid, checking that no two particles end up on the same points.

3 b° Mark the points with the particles in the $L \times L$ grid as filled.

4° Loop over MC steps of time Δt

- **5°** Loop from 1 to N_p
  - **6°** Pick one particle i at random
  - **7°** Find which positions it can jump to. If none, return to step 6°
  - **8°** Let the particle jump to one of the allowed directions j by a displacement $x_i = x_i + \delta x_j$, $y_i = y_i + \delta y_j$, enforce periodic boundaries on x and y
  - **9°** Set $dx_i = dx_i + \delta x$, $dy_i = dy_i + \delta x$ (where periodic boundaries **do not** play a role!)
- **10°** End loop from 1 to N_p
- **11°** Update time $t = t + \Delta t$

12° End loop over MC steps

13° Output $\langle \Delta R^2 \rangle = \langle dx_i^2 + dy_i^2 \rangle$ and calculate diffusion coefficient.

The crucial difference here to the previous random walk algorithms is that the outer loop goes over MC steps, the inner one over particles. When the walkers are independent of each other ("non-interacting") we can deal with one walker at a time, saving memory since storage of all particles is not needed.

How to do this in practice is illustrated in the Fortran90 code below. It gives out the diffusion constant D as the final answer, having reasonable values of 1 ns for the average jump time and 2 Å for the jump distance.

Note that this program fails for large numbers of particles using the Park-Miller "minimal standard" random number generator. In that case, the diffusion coefficient will keep growing instead of stabilizing at large times. The code below used the Mersenne twister, the Fortran version available from the course home page. (Yes, I did find this out the hard way, spending almost an entire Sunday debugging my code before I realized the random number generator was the culprit!)

```
! To compile use e.g.
! for Linux/Absoft Fortran:
! f90 -O -o randomwalk_latticegas randomwalk_latticegas.f90 -lU77 -lfio
! For Alphas with Compaq Fortran:
! f90 -O -o randomwalk_latticegas randomwalk_latticegas.f90
!
```

```
program randomwalk_latticegas
  implicit none

  logical, allocatable :: lattice(:,:)
  integer, allocatable :: x(:),y(:)
  double precision, allocatable :: dx(:),dy(:)

  integer :: Nsteps,Np,L,narg
  integer :: istep,isubstep
  integer :: dir,i,j,seed,nfail,njumps

  integer, parameter :: MAXINT=1000000000

  ! Variables for counting allowed directions
  integer ::  free(4),nfree
  integer :: dxtrial(4),dytrial(4)
  integer :: xnew(4),ynew(4)

  double precision :: dxsum,dysum,dxsqsum,dysqsum
  double precision :: t,deltat,drsqave,D,a,help

  character :: buf*80
  integer, external :: iargc
  double precision, external :: uniformrand,grnd


  ! Set average time between jumps and jump length Units is s and cm
  ! although actually this is not needed for the simulation
  deltat=1d-9;  ! 1 ns
```

```
a=2e-8;       ! 2 Å

if (iargc() <2) then
   print *,'Usage: randomwalk_latticegas Nsteps Np L seed'
   STOP
endif

call getarg(1,buf); read (unit=buf,fmt=*) Nsteps
call getarg(2,buf); read (unit=buf,fmt=*) Np
call getarg(3,buf); read (unit=buf,fmt=*) L
call getarg(4,buf); read (unit=buf,fmt=*) seed

call sgrnd(seed)

print *,'Doing lattice gas walk to',Nsteps,'MC steps, initial seed',seed
print *,'using',Np,' particles on a',L,'^2 square lattice'

if (Np >= L*L) then
   print *,'Are you crazy? Number of particles >= number of sites'
   STOP 'Too small lattice'
endif

print *,'Allocating arrays'
allocate(lattice(0:L-1,0:L-1))
allocate(x(Np),y(Np))
allocate(dx(Np),dy(Np))

! Mark all positions as empty
do i=0,L-1
```

```
   do j=0,L-1
      lattice(i,j) = .false.
   enddo
enddo

! enumeration of directions: 1 left 2 right 3 up 4 down
dxtrial(1)=+1; dytrial(1)= 0;
dxtrial(2)=-1; dytrial(2)= 0;
dxtrial(3)= 0; dytrial(3)=+1;
dxtrial(4)= 0; dytrial(4)=-1;

nfail=0; njumps=0;
! Generate particles on lattice
do i=1,Np
   do ! Loop until empty position found
      ! To be on safe side, check that upper limit not reached
      x(i)=int(grnd()*L); if (x(i)>=L) x(i)=L-1;
      y(i)=int(grnd()*L); if (y(i)>=L) y(i)=L-1;
      if (lattice(x(i),y(i))) then
         ! Position already filled, loop to find new trial
         cycle
      else
         lattice(x(i),y(i))=.true.
         ! Success, go to next particle
         exit
      endif
   enddo
   dx(i)=0.0d0; dy(i)=0.0d0;
enddo
```

```
t=0.0;
do istep=0,Nsteps-1    ! Loop over MC steps
   do isubstep=1,Np    ! Do all particles on average once every MC step

      ! Pick one particle at random
      i=int(grnd()*Np)+1; if (i>Np) i=Np;

      ! Find possible directions, store it in free()
      nfree=0
      do j=1,4
         xnew(j)=x(i)+dxtrial(j);
         if (xnew(j) >= L) xnew(j)=0; if (xnew(j)<0) xnew(j)=L-1;
         ynew(j)=y(i)+dytrial(j);
         if (ynew(j) >= L) ynew(j)=0; if (ynew(j)<0) ynew(j)=L-1;
         if (.not. lattice(xnew(j),ynew(j))) then
            ! Success: position free
            nfree=nfree+1
            free(nfree)=j
         endif
      enddo
      ! If no possible directions, get new particle
      if (nfree == 0) then
         nfail=nfail+1
         cycle
      endif
      njumps=njumps+1

      ! Pick one of the possible directions randomly
```

```
    ! Note that the dir>nfree check here really is needed!
    dir=int(grnd()*nfree)+1; if (dir>nfree) dir=nfree
    j=free(dir)

    ! Now x(i),y(j) is old position and xnew(j),ynew(j) new
    ! Double check that new site really is free
    if (lattice(xnew(j),ynew(j))) then
       print *,'HORROR ERROR: THIS SHOULD BE IMPOSSIBLE'
       print *,i,j,dir,nfree
       print *,free
       print *,x(i),y(i),xnew(j),ynew(j)
       STOP 'HORROR new site bug'
    endif
    !Empty old position and fill new
    lattice(x(i),y(i))=.false.
    lattice(xnew(j),ynew(j))=.true.

    x(i)=xnew(j); y(i)=ynew(j);
    dx(i)=dx(i)+dxtrial(j); dy(i)=dy(i)+dytrial(j);
enddo

 if (mod(istep*Np,1000000) == 0) then
    ! Calculate and print intermediate results every now and then
    ! Get total displacement from dx,dy
    dxsum=0.0d0; dysum=0.0d0;
    dxsqsum=0.0d0; dysqsum=0.0d0;
    do i=1,Np
       dxsum=dxsum+dx(i); dysum=dysum+dy(i);
       dxsqsum=dxsqsum+dx(i)*dx(i); dysqsum=dysqsum+dy(i)*dy(i);
```

```
      enddo
      drsqave=(dxsqsum+dysqsum)/(1.0*Np)

      if (t>0.0) then
         ! Get diffusion coefficient by proper scaling
         D=drsqave*a*a/(4*t)
         print *,'At',t,' drsqave',drsqave*a*a,' D',D,' cm^2/s'
      endif

   endif

   t=t+deltat

enddo

! Get total displacement from dx,dy
dxsum=0.0d0; dysum=0.0d0;
dxsqsum=0.0d0; dysqsum=0.0d0;
do i=1,Np
   dxsum=dxsum+dx(i); dysum=dysum+dy(i);
   dxsqsum=dxsqsum+dx(i)*dx(i); dysqsum=dysqsum+dy(i)*dy(i);
enddo
print *,'dxsum',dxsum,' dysum',dysum
print *,'dxsqsum',dxsqsum,' dysqsum',dysqsum
drsqave=(dxsqsum+dysqsum)/(1.0*Np)
print *,'drsqave',drsqave
print *,'Number of failed jumps',nfail,' number of successes',njumps
! Get diffusion coefficient by proper scaling
D=drsqave*a*a/(4*t)
```

```
  print *,'At',t,' drsqave',drsqave*a*a,' D',D,' cm^2/s'

end program randomwalk_latticegas

((After this the Mersenne twister source code should follow.
In that code, you have to change the comment character from
``*'' to the Fortran90 ``!''.))
```

This will be animated during the lecture.

[[Lecturers own reminder on animation, reader can ignore:

```
cd opetus/mc/tests/randomwalk
f90 randomwalk_latticegas_output.f90 -lU77
 a.out 100 2 20 12278 | grep "^ P" | dpc msleep 100 x -1 21 y -1 21 m 1 d 21 sd 440 440 erase 2 3 4 5 _
```

and then increase second argument.]]

And here is a series of results:

Np	L	Np/L^2	D (cm^2/s)	nfail	njumps
10	100	0.001	9.769973881166823E-008	0	10000000
10	100	0.001	1.127346430730184E-007	0	10000000
100	100	0.01	1.028685543050629E-007	0	10000000
100	100	0.01	9.469519884885580E-008	0	10000000

10000	1000	0.01	9.899003879678247E-008	0	10000000
1000	100	0.1	9.111043889255736E-008	292	9999708
1000	100	0.1	9.427090885414200E-008	279	9999721
100000	1000	0.1	9.403952985695557E-008	3127	99996873
3000	100	0.3	8.284148565973272E-008	109626	29890374
3000	100	0.3	7.915751903784448E-008	110196	29889804
6000	100	0.6	5.895798261670045E-008	1152902	10847098
6000	100	0.6	5.913229928124830E-008	1154808	10845192
9000	100	0.9	1.771291645136659E-008	11574471	6425529
9000	100	0.9	1.786338311620434E-008	11571431	6428569
900000	100	0.9	1.824779088931029E-008	57886778	32113222
9900	100	0.99	1.831247452488705E-009	19013835	786165
9900	100	0.99	1.860272704661156E-009	19015892	784108

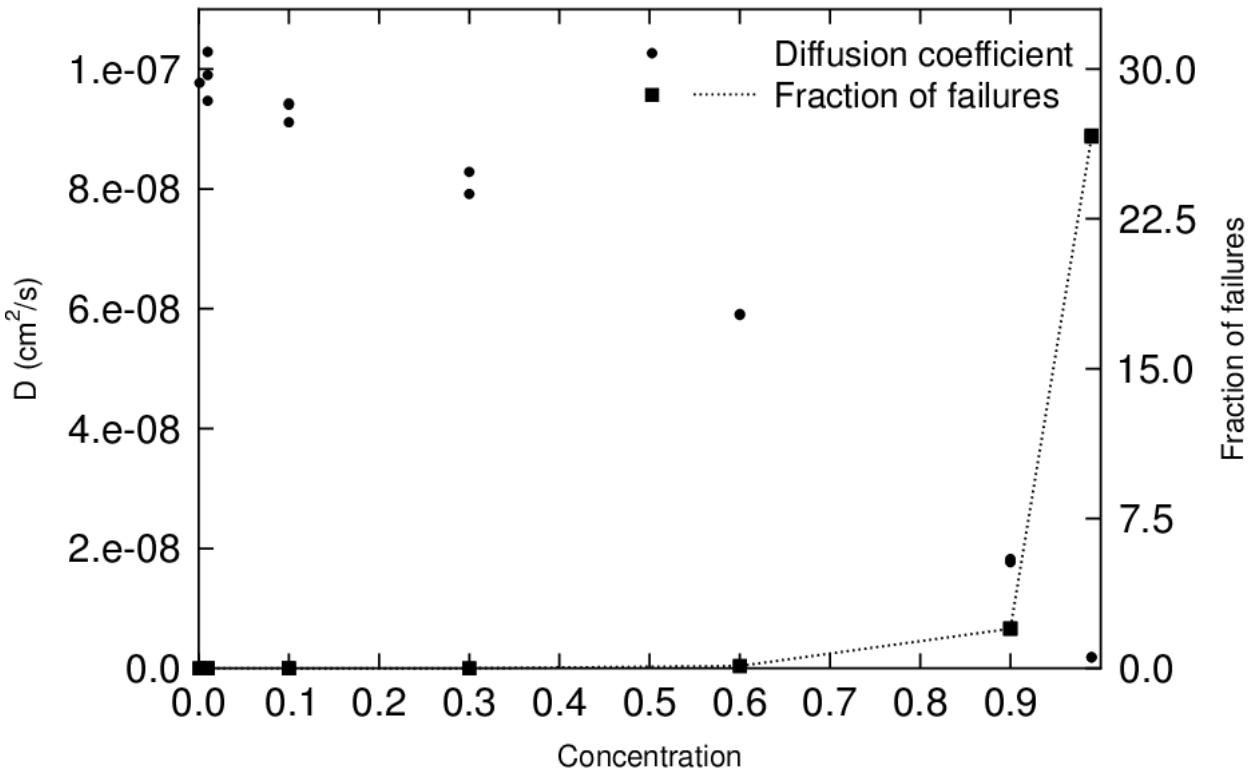
What does this mean? At small concentrations, the system behaves essentially as an unconstrained random walk. For that one, we know that $\langle \Delta R^2 \rangle$ should be equal to $a^2 N$, where N is the number of steps, and a is the jump distance, and the result for the diffusion coefficient should be

$$D = \frac{\langle \Delta R^2 \rangle}{4t} = \frac{(2\ \text{Å})^2 N}{4N\Delta t} = \frac{(2\ \text{Å})^2}{4 \times 1\ \text{ns}} = 10^{-7} \frac{\text{cm}^2}{\text{s}}$$

which is exactly what we get, within the uncertainty. But for larger concentrations, the number of failed jumps starts to grow, which reduces the diffusion coefficient. It is interesting to note,

however, that the diffusion coefficient and number of failed jumps do not follow an obvious linear dependence, so doing the MC simulation really is worthwhile.

Here is still a plot of $D(\rho)$, where ρ is the particle concentration.



7.5. Length of polymers in a good solvent

[Gould-Tobochnik 12.3]

As a final example of where random walks can be useful, we consider the length of polymers lying in a good solvent. A polymer is a molecule which consists of a long chain of basic building blocks. For instance, polyethylene can be represented as

$$\cdots\text{–}CH_2\text{–}CH_2\text{–}CH_2\text{–}CH_2\text{–}\cdots$$

When a polymer is placed in a good solution, where it basically can float around freely, it will after some time adopt a twisted form which is neither straight nor completely curled up. Imagine a single noodle (spaghetti strain) in warm water.

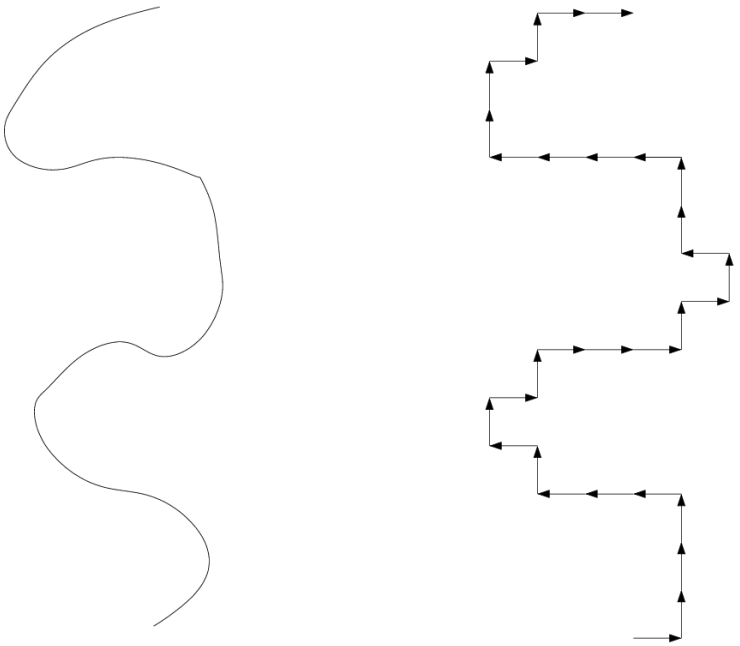
Experiments on polymers have shown that when the length of the "noodle" is measured, the average mean square end-to-end length has been found to have the following dependence:

$$\langle \Delta R^2(N) \rangle \propto N^{2\nu} \quad \text{with} \quad \nu \approx 0.592$$

for a wide range of polymers, regardless of their other properties. Here N is the number of monomers. The fact that such very general behaviour is observed indicates that there is some very simple general reason for the behaviour.

This lead Rosenbluth and Rosenbluth [Ref. J. Chem. Phys 23 (1955) 356 according to G+T] to think that a very simple random walk approximation could explain this behaviour. The main idea here is as follows. Let us think of making a random walk whose path would give the shape of the particular polymer.

The figure below illustrates a polymer and a random walk which would approximate the shape of this polymer.



A simple random walk clearly can not explain the behaviour, since it gives an exponent of $\nu = 0.5$. But it is also clear that the ordinary random walk does not describe the physical situation sensibly either, because it can return to the same positions many times. But in a polymer the monomers certainly can not occupy the same space – atoms can just not be squeezed on top of each other without extreme kinetic energies or pressures. So let us modify the basic random walk as follows: we add the simple requirement that the walker should never return to a position it has already occupied. This kind of random walk is called the **self-avoiding walk** (SAW).

If we know what the minimum-energy angles between the monomers are (and today this usually is known quite well), we could take these angles to construct the walk. But since the same scaling behaviour is observed for many kinds of polymers, this suggests that the result will be fairly independent of the angles. So we can start with using a square lattice (2D) or simple cubic lattice (3D) with all angles of 90° to simulate the walk. And it turns out that this gives almost exactly the right answer! You get to find this out yourself in the exercises.

This is a really beautiful example of how a very simple physical model can lead to understanding of nature!.

7.5.1. Coding self-avoiding random walks

Coding the SAW efficiently is not quite trivial. It could seem easy. Just do random walks one at a time as usual for non-interacting walkers, store all the positions visited to an 2D or 3D array, and if a new step enters a previously visited site, disregard it.

But herein lies the problem. We can not just disregard the last step, and look for a new direction. This would skew the statistics; we have to disregard the whole walk. If we want to make a walk of any sensible length (say, $N > 30$), the probability of the walker entering a previously visited site becomes enormously large. Hence we end up disregarding almost all walks.

In my tests, in 3D walks for $N = 20$ only roughly 2 % of the walks were successful, for $N = 30$ only 0.1%, and for $N = 40$ only 0.04%. So the efficiency is terrible even for small N, and quickly becomes truly horrendous for increasing N.

Hence it is very important to have some way to make the simulation of the SAW's more efficient. We will here present the simplest one, the optimization scheme of Rosenbluth and Rosenbluth.

The idea is that once a walk attempts to do a step N that is impossible, we do not disregard the

whole walk, but only this step. We then pick some other, possible step. A weight factor $W(N)$ is used to ensure that the statistics remains correct.

The way the weighting is done can be written as follows. Consider a situation where we have just done step $N-1$, and look at step N. Three outcomes are possible:

1° No step is possible. Set $W(N) = 0$ and restart

2° All steps other than the step right backward are possible, set W(N)=W(N-1)

3° Only m steps are possible with $1 \leq m < 3$ (2D) or $1 \leq m < 5$ (3D). In this case we choose randomly one of the possible steps, and set $W(N) = \frac{m}{3} W(N-1)$ (2D) or $W(N) = \frac{m}{5} W(N-1)$(3D).

The correct value of the final average $\langle \Delta R^2(N) \rangle$ is obtained by weighting $R_i^2(N)$ for each step N in a given walk i with the weight $W_i(N)$ obtained in the same walk. To get the average, we

have to divide by the sum of $W_i(N)$ instead of N, i.e.

$$\langle \Delta R^2(N) \rangle = \frac{\sum_{i=1}^{N_{\mathrm{walks}}} W_i(N) R_i^2(N)}{\sum_{i=1}^{N_{\mathrm{walks}}} W_i(N)}$$

$W(0)$ is initialized to 1.0.

This is tremendously much more efficient than the direct approach because it is actually quite rare to end up in a situation where no step at all is possible. Hence the number of failures is vastly smaller than before: in my tests I obtained only 8 failures for 3D walks of length 30 simulating 10000 walks.

Note that in handling step 1 there are actually two choices. One is disregarding the whole walk, the other just disregarding the steps after the N we have managed to reach. If the goal is to deal with walks (polymers) always reaching the full length, one should do the former. But which one to do

may actually depend on how the experiments are exactly carried out. But as long as there are very few failed walks, this does not in any case make much of a difference.

There are other, more advanced ways of speeding up the SAW (see e.g. Gould-Tobochnik). But already the Rosenbluth-Rosenbluth scheme is easily good enough to test the basic result of whether $\nu = 0.592$.