Random Systems

Mojtaba Alaei

November 10, 2013

Content of the course

Random systems

Example: Diffusion

Brownian motion

i

Random numbers in computer

To simulate random processes we need to produce random numbers. We can not really produce random numbers in compuers and in fact we produce and use pseudorandom numbers which are not truely random!

Pseudorandom number generator

A pseudorandom number generator is an algorithm for generating a sequence of numbers that approximates the properties of random numbers. The sequence is not truely random in that it is completely determined by a relatively small set of initial values.

Pseudorandom number generator; an example

Consider the following equation (linear congruential generator):

$$x' = (ax + c) \bmod m \tag{1}$$

Where a, c, and m are integer constants and x is an integer variable.

```
from pylab import plot, show
N = 100
#Paramters for psudorandom genrator:
a = 1664525
c = 1013904223
m = 4294967296
x = 1
results = []

for i in range(N):
    x = (a*x+c)%m
    results.append(x)
plot(results,"o")
show()
```

- The numbers are not actually random (By knowing a, c, m and x you can calculate the next (random) number!!)
- It matters what values you choose for the constants a, c, m
- By choosing different starting values for x, you can get different sequences of random numbers.
- The initial value is called the seed for the random number generator; it specifies where the squence will start.

Random numbers in python

In the linear congruential generator ($x' = (\,ax\,+\,c\,)\,mod\,m$) there are correlations between the values of successive numbers, whereas true random numbers would be independent.
But these days physicist use a better random number generator so-called Mersenne twister. In the random pakage in python, Mersenne twister genrator is provided, which contains the following useful functions:

`random()`	Gives a random floating—point number uniformly distributed in the range from zero to one, including zero but not including one
`randrange(n)`	Gives a random integer from 0 to n—1, inclusive
`randrange(n,m)`	Gives a random integer from m to n, inclusive
`randrange(n,m,k)`	Gives a random integer in the range m to n—1 in steps of k

Random number seeds

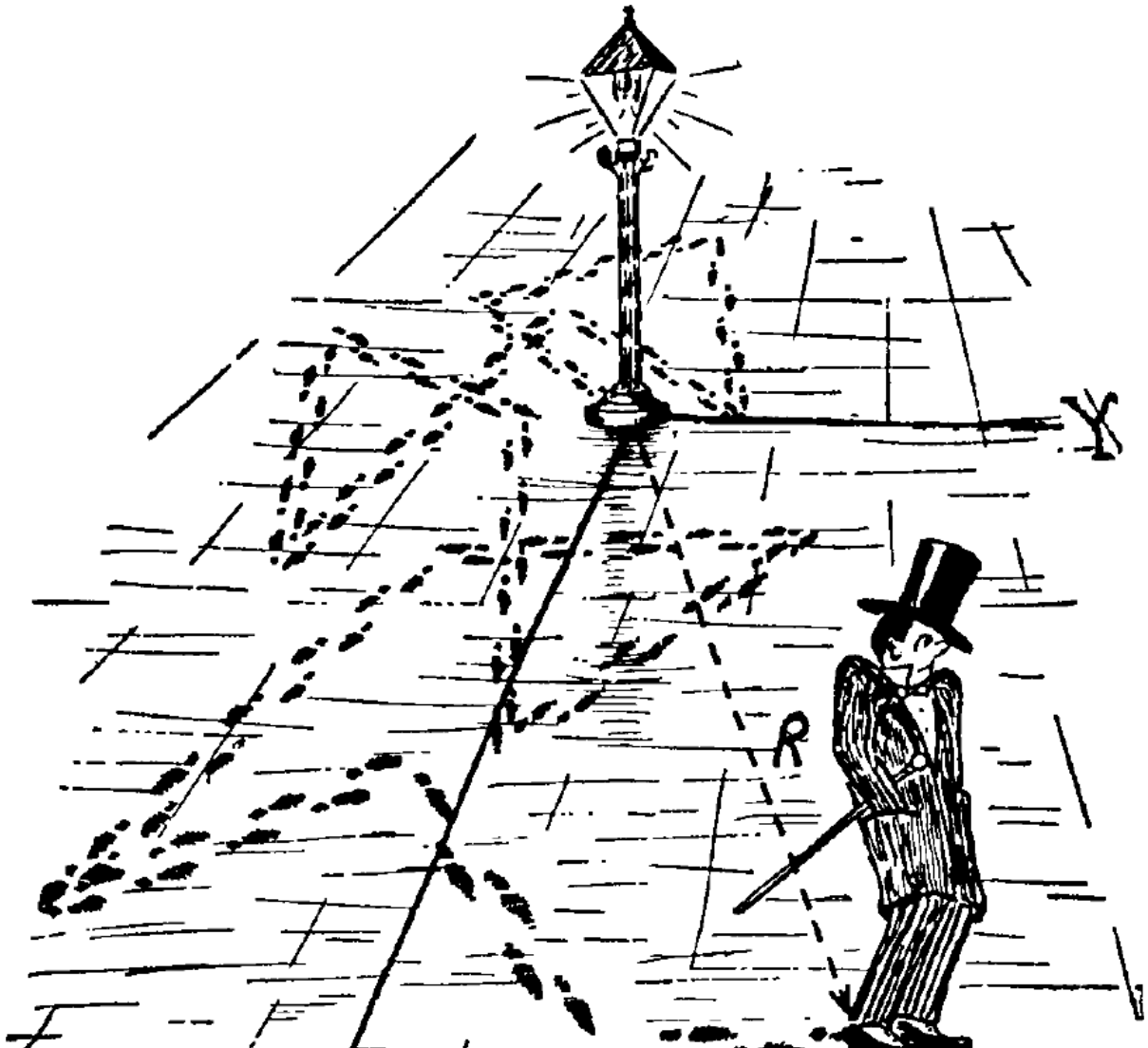
Run the following program several times:

```
from random import randrange, seed
seed(42)
for i in range(4):
    print(randrange(10))
```

- What is the role of seed?
- Where do you need to use seed?

Random walk

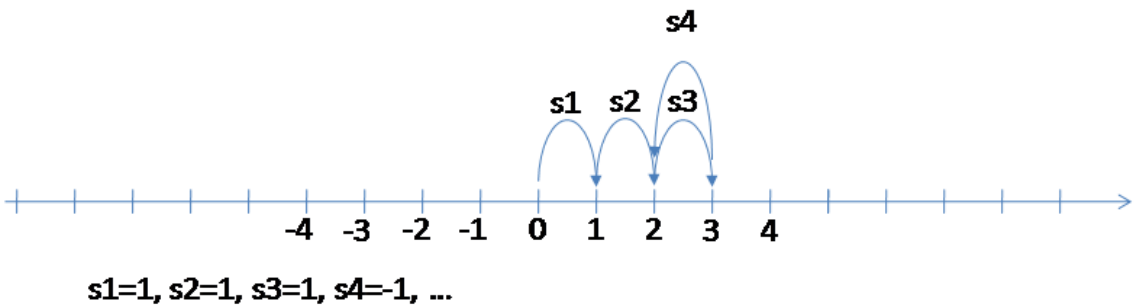
Random walk is a simple model to describe many random processes.



Random walk in one dimension

Suppose a walker who walks

- toward +x direction with a probability p (for example $p = 1/2$)
- and toward -x dirction with an another probability, $q = 1 - p$ (for example $q = 1/2$).
- His steps have same length, equal to a ($s_i = \pm a$)



So after N steps, the position of walker is:

$$x_N = \sum_{i=1}^{N} s_i \tag{2}$$

$< x_N >$ and $< x_N^2 >$

for a random walk (the steps are independent of each other):

$$\begin{aligned} < x_N > &= < \sum_{i=1}^{N} s_i > = \sum_{i=1}^{N} < s_i > = N < s > \\ &= N(pa - qa) = N(p-q)a \end{aligned} \tag{3}$$

For $p = q = 1/2$, $< x_N > = 0.0$.

$$\begin{aligned} < x_N^2 > &= < \sum_{i=1}^{N} (\sum_{j=1}^{N} s_i s_j) > \\ &= \sum_{i=1}^{N} \sum_{j=1}^{N} < s_i s_j > \end{aligned} \tag{4}$$

The terms $s_i s_j$ with $i \neq j$ will be ± 1 with equal probability (for $p = q = 1/2$). So the terms $< s_i s_j >$ for $i \neq j$ are equal to zero. Therefore:

$$< x_N^2 > = \sum_{i=1}^{N} s_i^2 = N\, a^2 \tag{5}$$

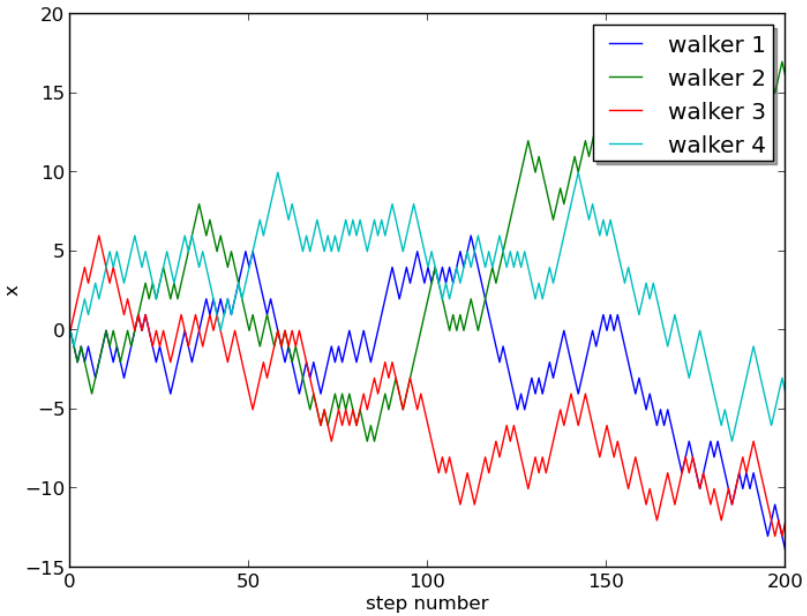
Simulate random walk

```
from random import random

x=0

for i in range(N):
    p = random()
    if p < 0.5:
        x=x+1
    else:
        x=x—1
```

What is the difference of x_N and $< x_N >$?

What is the difference of x_N and $< x_N >$?

```
from random import random
import matplotlib.pyplot as plt
N=200; M=4; x1=0
x=[x1]
for j in range(M):
    x1=0
    x=[x1]
    for i in range(N):
        p= random()
        if p < 0.5:
            x1=x1+1
        else:
            x1=x1-1
        x.append(x1)
    plt.plot(x)

plt.show()
```



exercise 5, part 1

- random walk in one-dimension
  - Calculate $< x_N >$ for arbitrary N steps
  - Plot $< x_N^2 >$ vs. N
- random walk in two-dimesion
  - Calculate $< r_N >$ for arbitrary N steps ($r_N = x_N + y_N$)
  - Plot $< r_N^2 >$ vs. N

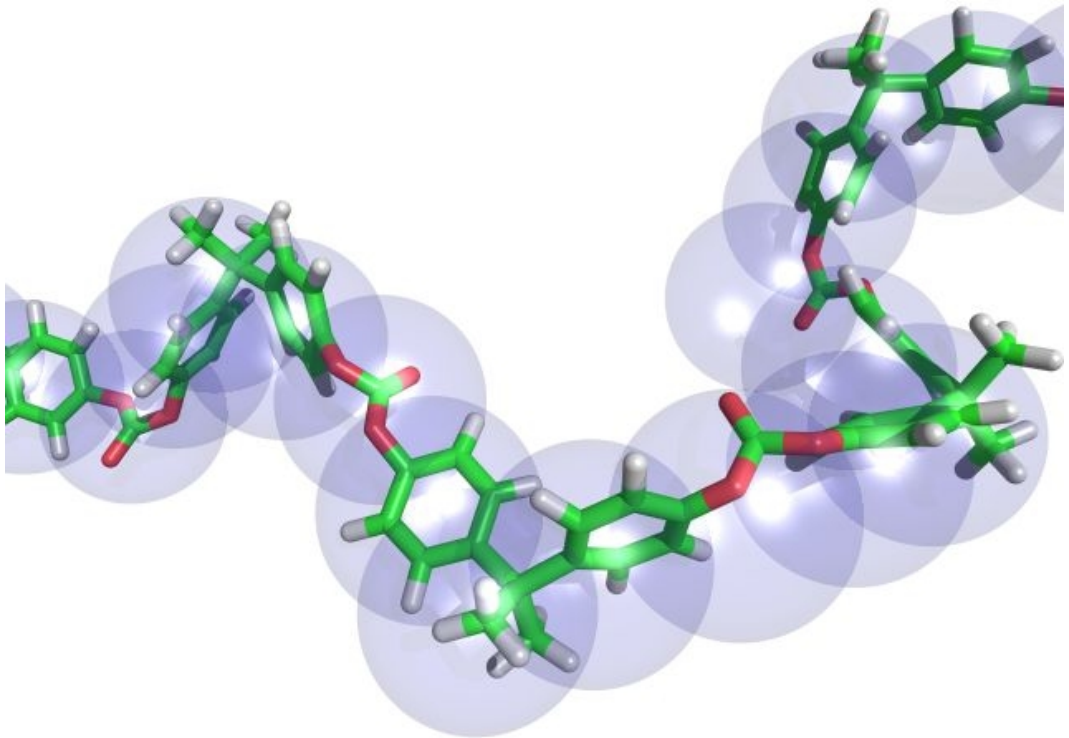
Hint

For two dimensions we can use randrange(4):

```
for step in range(N):


    p = random.randrange(4)
    if   p == 0:
        x += 1
    elif p == 1:
        y += 1
    elif p == 2:
        x -= 1
    elif p == 3:
        y -= 1
```

polymers

A polymer is a large molecule composed of many repeated subunits, known as monomers.



random walks and polymers

A random walk is an obvious candidate for modeling flexible polymers. BUT!

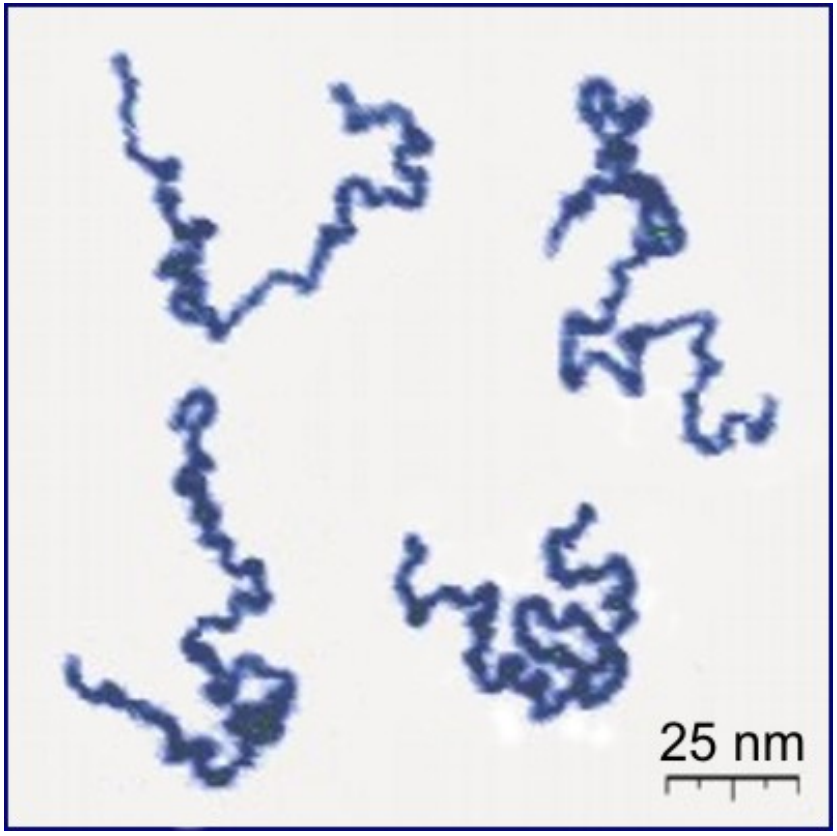


Figure: Appearance of real linear polymer chains as recorded using an atomic force microscope on a surface, under liquid medium. Chain contour length for this polymer is 204 nm; thickness is 0.4 nm (JACS,Roiter, Y.; Minko, S. (2005),downloaded from wikipedia

Self-avoiding walks and polymers

Each link in the polymer chain corresponds to one step in the walk, and scince the polymer is flexible, each step is independent of the one immediately befor it. However, for this problem a random walk ingores an important piece of physics.

Solution

The path followed by our polymer molecule must not be allowed to intersect itself. Only one segment of the polymer is allowed to occupy any particular region of space. A random walk that subject to this constraint is called a a self-avoiding walk, or SAW.

Self-avoiding walks

The "true" self-avoiding walk is defined as the statistical problem of a traveller who steps randomly, but tries to avoid places he has already visited.

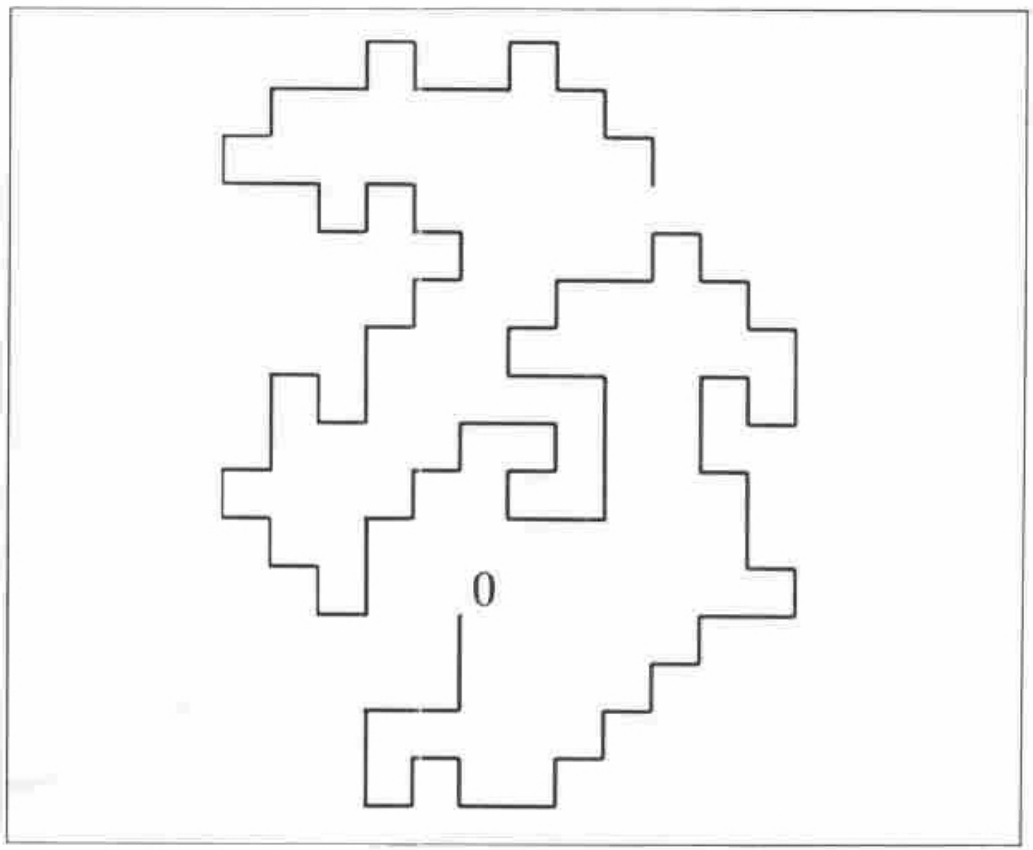
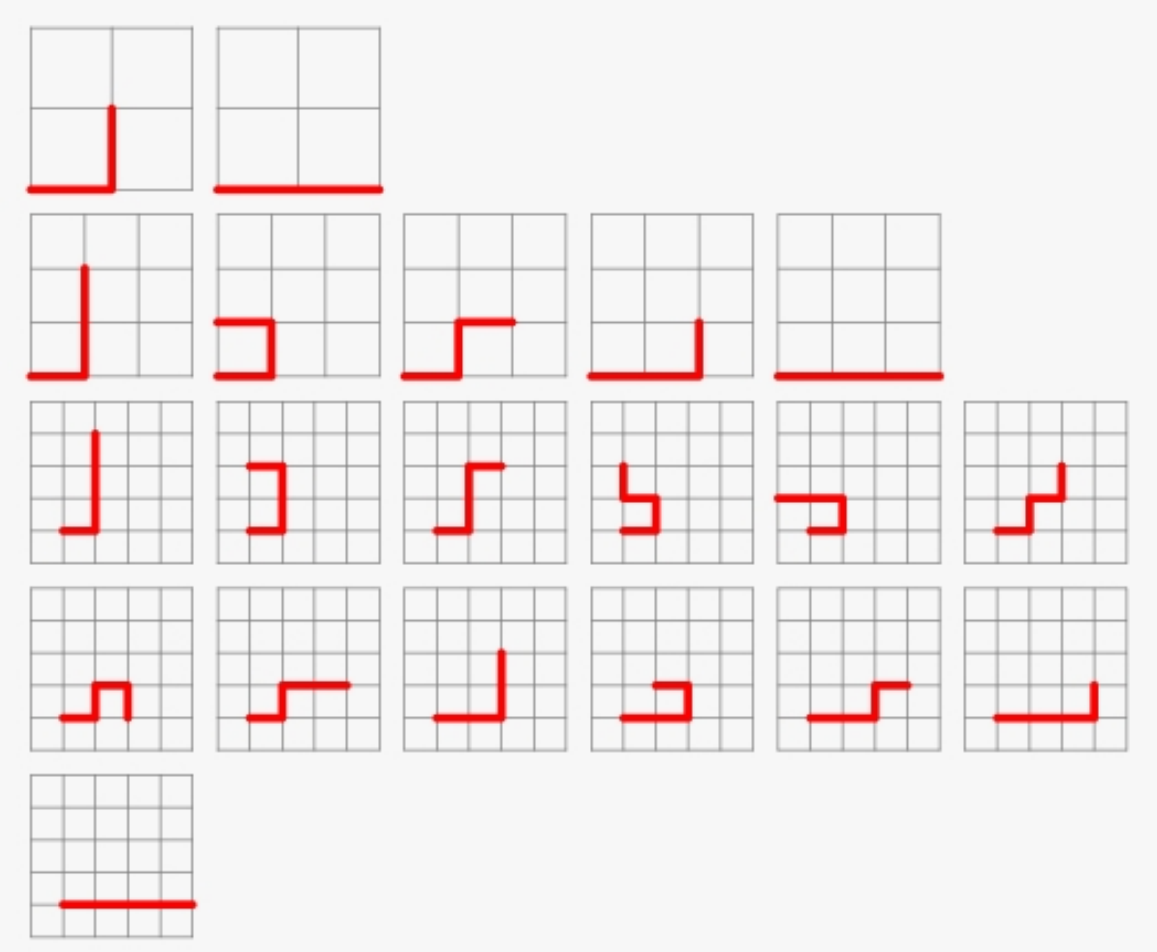


Figure 1. A 2-dimensional self-avoiding walk with 105 steps, beginning at 0. A subsequent step to the left would eventually lead to a trap.

Self-avoiding walks



Self-avoiding walks in one dimension

Self-avoiding walks in one dimension are trivial. There are only two different self-avoiding walks with fixed step size in 1-D. If the first move is to the left, every subsequent step is to the left. If the first move is to the right, the walk continues to the right.

$$|x| = N \tag{6}$$

Self-Avoiding Walk in Two Dimensions

In the simple random walk:

$$< x_N^2 > \sim N \tag{7}$$

But for a self-avioding walk:

$$< x_N^2 > \sim N^{\alpha} \tag{8}$$

Where $1 < \alpha < 2$.

Simplistic Self-Avoiding Walk Algorithm

- Generating self-avoiding walks is tricky! Consider a self-avoiding walk on a 2-D square lattice. The walker must step North, South, East or West with equal probability. At the same time the walker must avoid previously visited locations.
- Unfortunately, these two requirements are incompatible! A walker who steps with equal probability NSEW cannot avoid previously visited locations. The first step has 4 allowed directions, NSEW. Every subsequent step has 3, 2, 1, or 0 allowed directions.
- The fundamental problem is to ensure that every SAW chain in an equilibrium ensemble occurs with the same probability.

Simplistic Self-Avoiding Walk Algorithm

We can model a self-avoiding random walk in two dimensions by having the the walker step NSEW with equal probabilities, and simply discarding failed walks.

Running the codes shows that the algorithm is extremely inefficient. The fraction of discarded walks increases exponentially with length N:

$$\frac{\text{Walks Generated}}{\text{Total Number of Attempts}} \sim e^{-\lambda N} \tag{9}$$

where λ is a positive attrition constant.

exercise 5, part 2

For a salf-avioding walk in two (and three dimension), calculate α

- To derive α, you should plot $log < r_N^2 >$ vs. N, the slope of the line is α
- Because this algorithm is time-consuming, chose the maximum of N equal to 25 ($N_max = 25$) and take the average over 200 walkers
- For fitting the date to a line, you can use numpy.polyfit.
  (see http://docs.scipy.org/doc/numpy/reference/generated/numpy.polyfit.html)
  or
  a softwere such as gnuplot or xmgrace

The Reptation Method

The reptation model of polymer diffusion was developed by Pierre de Gennes who won the Nobel Prize in 1991 for his work on liquid crystals and polymers.

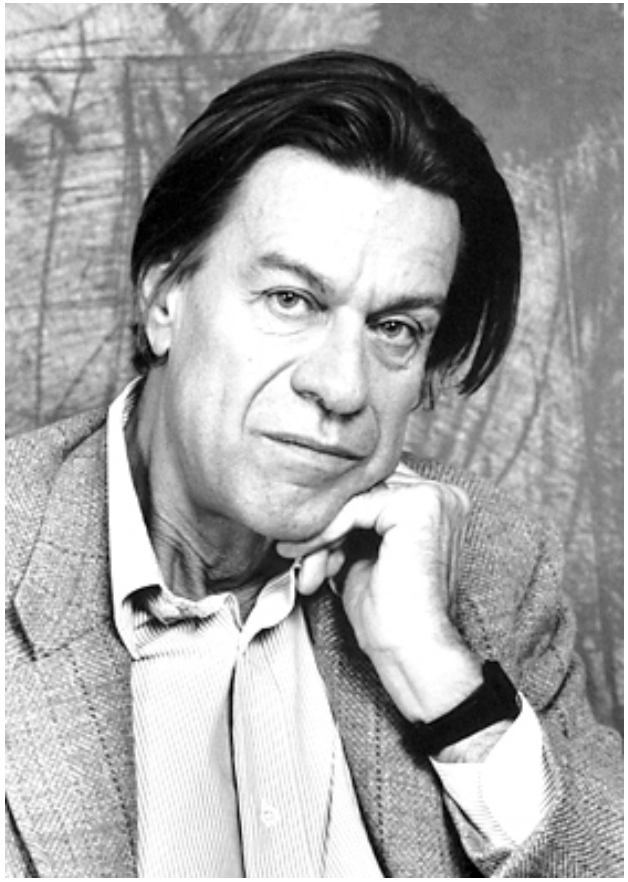


Figure: Pierre de Gennes

The Reptation Method

Reptation is the thermal motion of very long linear, entangled macromolecules in polymer melts or concentrated polymer solutions. Derived from the word reptile, reptation suggests the movement of entangled polymer chains as being analogous to snakes slithering through one another.

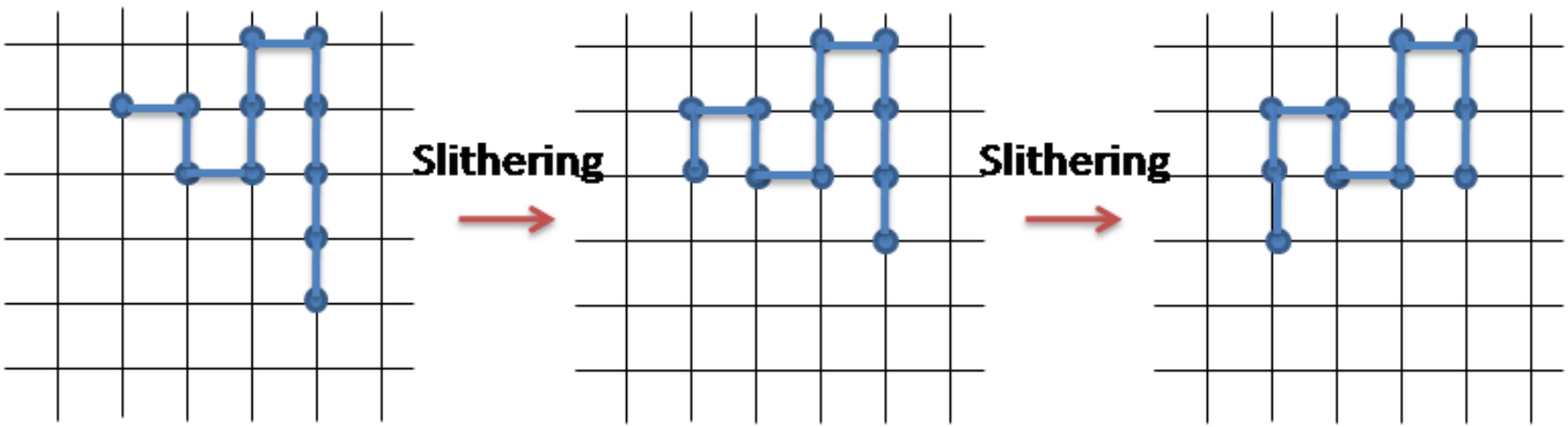
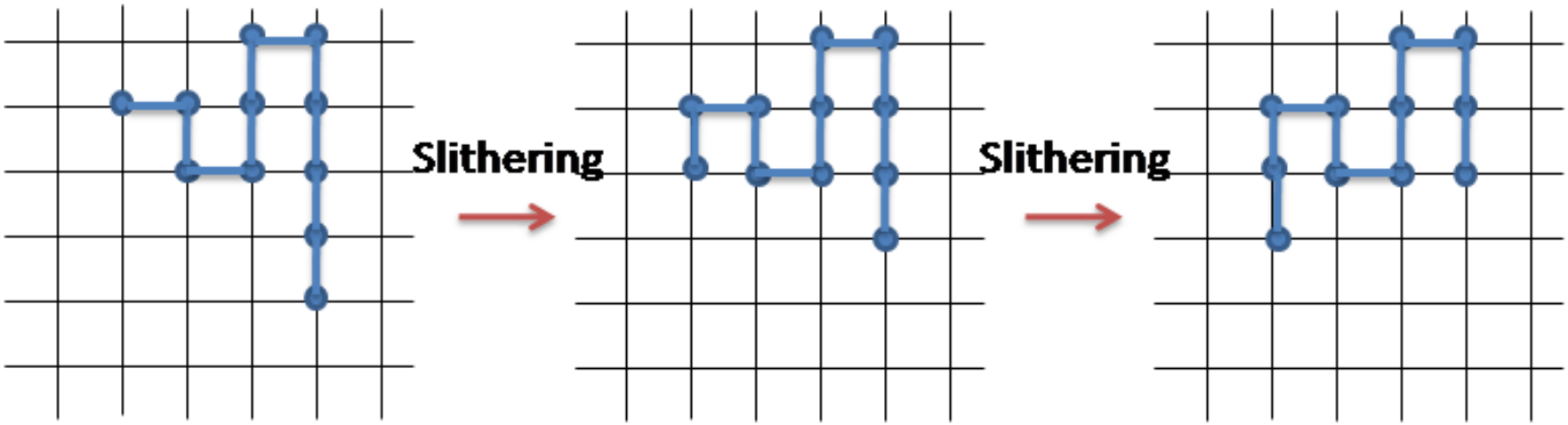


Figure:

The Reptation Method



Reptation Algorithm

- Assume that we have generated a random walk.
- Choose one of the end points at random and delete this point.
- Choose one the end points at random.
- Add the delete point to the choosen end with a random direction.

Percolation