Introduction to diffusion and random walks

4

In this chapter we will bridge insights from random walks with diffusion and stochastic processes. We will start by describing and modeling random walks, we will address their applications, their structure and their scaling properties. Then we will address the probability distribution for the end point of a random walk. This will be the starting point for discussion diffusion. We will map random walks onto a diffusion process, and describe the same process using both a random walk model and a diffusion model, learning about advantages and disadvantages of both. We will then address the stationary solution of the diffusion process — the Poisson equation.

4.1 Random walks

A random walk is a fundamental physical process in which a random walker — a particle, an atom, a measurement, an individual — moves in random steps. A typical example is the Browian motion of small particles due to the thermal motion of atoms. Another example is the spreading of an atom throughout a fluid.

ams 1: Show diffusion model simulation in Atomify-Mac

4.1.1 Random walks as sum of random variables

In one dimension, we can describe a random walk by its random individual steps, u_i, where i is the step number. The step u_i is a random variable that may have some distribution. The position, x, of the walker after N steps is the sum of the individual steps:

$$x(N) = \sum_{i=1}^{N} u_i \ . \tag{4.1}$$

Modeling a random walk in Python. We can model a random walk in Python by selecting each of the steps, u_i, randomly from some distribution, and then finding the sum. For example, we may flip a coin, so that it is equal probabilities to go left and right:

$$u_i = \begin{cases} -1 \ p = 1/2 \\ +1 \ p = 1/2 \end{cases} \tag{4.2}$$

We can select an integer number between 0 and 1 using the Python command `randint(2)`. Thus we can generate a sequence of random number u_i by

```
from pylab import *
N = 10
u = randint(0,2,size=N)
u
Out: array([0, 1, 0, 1, 1, 0, 0, 0, 1, 0])
```

Ooops. This generates two possible outputs, but not -1 and 1, which is what we wanted. We transform u_i to its wanted range

```
u = 2*randint(0,2,size=N)-1
u
Out: array([ 1,  1, -1,  1,  1,  1, -1,  1,  1,  1])
```

This looks reasonable. How far does the walk get after N steps? This is simply the sum of u_i:

```
x = sum(u)
x
Out: 6
```

We can illustrate the walk by its position, x, as a function of the number of steps, N, it has gone. Thus we need to find the sum over u_i first, for $i = 1$, then for $i = 1, 2$, and so on. This is done automatically by the function `cumsum`:

```
xs = cumsum(u)
xs
Out:array([1, 2, 1, 2, 3, 4, 3, 4, 5, 6])
```

We can now illustrate $x(N)$:

```
N = 1000
u = 2*randint(0,2,size=N)-1
xs = cumsum(u)
plot(xs), xlabel('N'),ylabel('x(N)')
```

The resulting plot is shown in Fig. 4.1. This is an interesting figure with many features. We will soon discuss the basic properties of this figure.

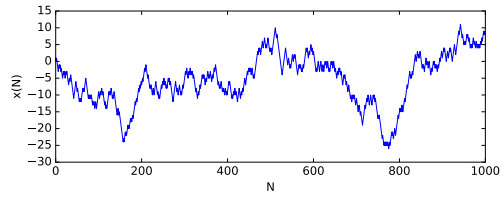

Fig. 4.1 Plot of the position $x(N)$ of a random walk after N steps.

Random walks in two dimensions. We can define random walks in any dimension, simply by assuming the individual steps can occur in any dimension. In two dimensions we may introduce $\mathbf{u}$, which may for example be

$$\mathbf{u} = \begin{cases} (1,1) & p = 1/4 \\ (-1,1) & p = 1/4 \\ (1,-1) & p = 1/4 \\ (-1,-1) & p = 1/4 \end{cases} \tag{4.3}$$

This simply corresponds to selecting one u_i in the x-direction and an v_i in the y-direction with the same distribution. It is therefore very simple to create such a walk in two dimensions:

```
u = 2*randint(0,2,size=(N,2))-1
r = cumsum(u,axis=0)
```

Where the `axis` option specifies along which axis the sum should be taken. We plot the walk, producing the nice structure in Fig. for $N = 10, 100, 1000$, and 10000 in Fig. 4.2.

Characterizing these structures. The structues generated by a random walk in two- or three dimensions corresponds to the structures generated by for example a randomly coiling polymer. How can we characterize the structure in one-, two- or higher dimensions?

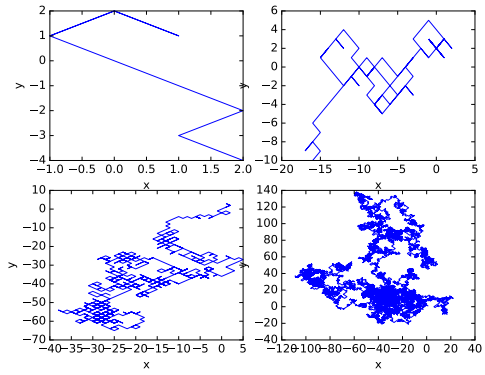

Fig. 4.2 Plot of the positions **r** of a random wal.

If we look at either Fig. 4.1 or Fig. 4.2 we see that the random walk does not stray very far from its initial position. We can characterize the two dimensional structure by its extent, for example by its deviation. We could measure this as the maximum deviation, $r_{\max}$ in two-dimensions or $x_{max} - x_{min}$ in one dimension. However, it is customary to describe the deviations by the quadratic deviation (corresponding to the variance or the standard deviation). That means that we characterize the extent of the walk by

$$\Delta x^2 = \langle (x_i - \langle x_i \rangle)^2 \rangle \ , \tag{4.4}$$

where the $\langle .. \rangle$ means the average. We calculate this directly as

$$\bar{x} = \langle x \rangle = (1/N) \sum_i x_i \ , \tag{4.5}$$

and

$$\Delta x^2 = (1/N) \sum_i (x_i - \bar{x})^2 \ . \tag{4.6}$$

Measured deviation in one dimension. We can now implement and measure the deviation, Δx, for one walk. However, each walk is different, but we can observe statistical trends by analyzing many walks. We would therefore like to measure $\Delta x(N)$ by averaging over M walks.

We implement this in a Python program using the function `std` to calculate the deviation in one go:

```
# Find deviation of random walks
Nvalues = [10,20,40,80,160,320,640]
walkdev = zeros(len(Nvalues))
for i in range(len(Nvalues)):
```

```
    N = Nvalues[i]
    M = 10000 # Nr of samples
    for im in range(M):
        x = cumsum(2*randint(0,2,size=N)-1)
        xdev = std(x)**2
        walkdev[i] = walkdev[i] + xdev
    walkdev[i] = walkdev[i]/(M*1.0)
plot(Nvalues,walkdev,'-o'),xlabel('N'),ylabel('dx^2(N)')
```

The resulting plot in Fig. 4.3 is astonishing. It is a completely straight line! This indicates that $\Delta x^2 \propto N$.

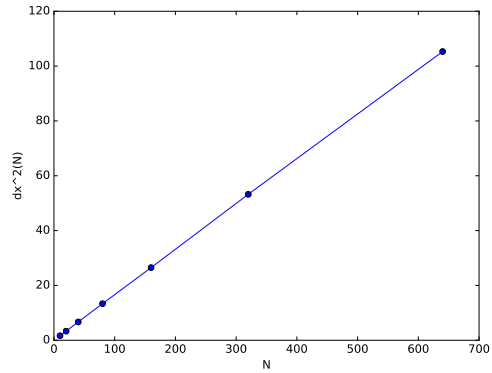

Fig. 4.3 Plot of the deviation, Δx^2, as a function of the number of steps, N.

Exercise: Measure extent of a two-dimensional walk. Write a program to perform the same measurement in two dimensions. Plot the deviation as a function of N and comment on the results.

Hint: Notice that the deviation is independent in the x and the y directions:

$$\Delta r^2 = \langle(\mathbf{r}_i - \mathbf{r}_{avg})^2\rangle = \langle(x_i - x_a)^2 + (y_i - y_a)^2\rangle = \langle(x_i - x_a)^2\rangle + \langle(y_i - y_a)^2\rangle. \tag{4.7}$$

Theory for the extent of a random walk. Because the observed behavior is so clear, we look for a theoretical argument.

First, we find the average of x:

$$\langle x\rangle = \langle\sum_i u_i\rangle = \sum_i \langle u_i\rangle = 0 \tag{4.8}$$

as long as there is no net movement in one direction. (In that case we can subtract the net movement).

Then we look at the deviation of x:

$$\langle x_i^2 \rangle = \langle \sum_i \sum_j u_i u_j \rangle = \langle \sum_i u_i u_i \rangle = N \langle u_i^2 \rangle \ . \tag{4.9}$$

This argument is independent of dimensionality, and independent of the individual steps u_i as long as they are uncorrelated. (This is why $\langle u_i u_j \angle = 0$ when $i \neq j$). The argument is valid for any distribution of u_i — as long as the distribution has finite variance. (The number $\delta^2 = \langle u_i^2 \rangle$ must be finite).

Thus we have found very generally that the variance of the random walk is proportional to the number of steps. This means that it moves slowly away from zero!

Structure and dimensionality. If we think of the random walk as representing a polymer, we could argue that the mass of the polymer corresponds to the number of monomers — the number of steps. Thus, the mass would be $M = m_0 N$, where m_0 is the mass of one step in the walk. We have therefore found that the mass of the polymer scales with its extent Δr^2 according to:

$$M = m_0 N = m_0 (\Delta r / \delta)^2 \ . \tag{4.10}$$

This means that the mass is proportional with the exent to the second dimension. In two dimensions this means that it fills the same. However, this result also holds in three-dimensions. The random walk reprensents a two-dimensional structure also in three dimensions.

May illustrate other physical fractal structure, such as percolation clusters, here.

Other types of random walks. The random walk may cross itself, but a polymer cannot. Instead, a polymer will behave like a self-avoiding random walk. But how do we expect such as walk to behave? (Indicate scaling behavior for small and large dimensions).

4.1.2 Distribution of positions of a random walker

We have only looked at how wide the walk goes. What about the distribution of end-points, x of a walk of N steps. (Actually, you may know that this problem has an exact solution for some processes and an exact limit for most distribution of individual steps, u_i, but we will concentrate on studying the properties numerically here).

Let us generate a random walk with the following probabilities for a single step:

$$u_i = \begin{cases} \frac{1}{3} & +1 \\ \frac{1}{3} & 0 \\ \frac{1}{3} & -1 \end{cases} \tag{4.11}$$

We want to measure the probability $P_N(x)$ that the walker lands in the position x after N steps. Or, we may want to find the probabilty for the walker to stop in an interval dx from x to $x + dx$. We measure this by finding the frequency of occurence of such an event. We perform M experiments, and count how many times N_x the walker ends in boxes in the interval from x to $x + dx$. Then we estimate the probability as

$$P_N(x)dx = \frac{N_x}{M} \Rightarrow P_N(x) = \frac{N_x}{M\,dx} \,. \tag{4.12}$$

We did this in detail to ensure that we remember to divide by the box size when we generate a histogram. We can count how many events falls into a box of width dx by using the `histogram` function in Python. We can either specify the boundaries for all the boxes or let Python specify the box positions.

We prefer to choose the box positions ourselves. Since the possible positions after N steps span from $-N$ to N we choose the box size to be 1 and generate a set of edges for the boxes spanning from $-N - 1/2$, $-N + 1/2$ to $N + 1/2$ using the `range` command. This is implemented in the program walk1dscaling00.py[1]:

```
# Random walk in one dimension - measurement
from pylab import *
N = 100
nsample = int(10000000/N)
x = zeros(nsample)
for i in range(nsample):
    z = randint(3,size=N)-1
    x[i] = sum(z)
edges = array(range(-N,N+1))-0.5
Nx,e = histogram(x,edges)
x = (edges[:-1]+edges[1:])/2.0
dx = diff(edges)
Px = Nx/(dx*nsample)
plot(x,Px), xlabel('x'), ylabel('P(x)')
show()
```

The resulting plot of $P(x)$ as a function of x is shown in Fig. 4.4.

[1] http://folk.uio.no/malthe/compcourse/walk1dscaling00.py

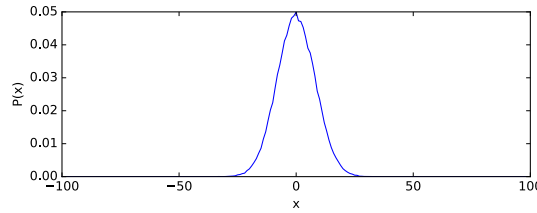

Fig. 4.4 Plot of $P(x)$ for $N = 100$.

Finding $P_N(x)$ as a function of N. How does this distribution change as the number of steps increases. We plot $P_N(x)$ for various values of N using walk1dscaling01.py[2]:

```
# Random walk in one dimension - scaling
from pylab import *

Nvalues = [10,50,100,200,400]

for N in [10,50,100,200,400]:
    nsample = int(1000000/N)
    x = zeros(nsample)
    for i in range(nsample):
        z = randint(3,size=N)-1
        x[i] = sum(z)
    edges = array(range(-N,N+1))-0.5
    Nx,e = histogram(x,edges)
    x = (edges[:-1]+edges[1:])/2.0
    dx = diff(edges)
    Px = Nx/(dx*nsample)
    plot(x,Px)
show()
```

Zooming in provides some insight into the structure. The distribution becomes wider as N increases. Indeed, we have found that the deviation increases with N. Can we rescale the plot to include this insight?

We plot $P_N(x)$ as a function of $x/\sqrt{N}$. However, this does not produce the wanted behavior, because we see that the plots are too high. We also need to rescale the other axis.

Our theory is that the probability distribution depends on $\sqrt{N}$ in the following way

$$P_N(x) = N^a f(x/\sqrt{N}) \, . \tag{4.13}$$

We can then use the normalization condition to find out how to rescale the other axis. Since the integral of $P_N(x)$ must be normalized, we find

[2] http://folk.uio.no/malthe/compcourse/walk1dscaling01.py

$$\int_{-\infty}^{\infty} P_N(x)dx = \int_{-\infty}^{\infty} N^a f(x/\sqrt{N})dx \tag{4.14}$$

$$= \int_{-\infty}^{\infty} N^a f(u)\sqrt{N}du = 1 \tag{4.15}$$

which requires that $N^a = 1/\sqrt{N}$. We rescale with this as well in walk1dscaling02.py[3]:

```
# Random walk in one dimension - scaling
from pylab import *

Nvalues = [10,50,100,200,400]
clf()
for N in [10,50,100,200,400]:
    nsample = int(1000000/N)
    x = zeros(nsample)
    for i in range(nsample):
        z = randint(3,size=N)-1
        x[i] = sum(z)
    edges = array(range(-N,N+1))-0.5
    Nx,e = histogram(x,edges)
    x = (edges[:-1]+edges[1:])/2.0
    dx = diff(edges)
    Px = Nx/(dx*nsample)
    plot(x/sqrt(N),Px*sqrt(N))
show()
```

Proving that this is indeed the scaling behavior of the probability distribution. Notice that this of course corresponds to the exact solution from the Central Limit theorem. However, it is nice to see how we can obtain such results in a more empirical fashion. Indeed, this is how we often look for scaling structures in nature.

Measuring $P_N(x)$ for two-dimensional walks. Generating and characterizing the probability distribution for the position $\mathbf{r}_N$ of a random walker after N steps follows the same principles as in one dimension. We generate M walks, each of length N. Each walk consists of a series of independent steps, $\mathbf{u}_i$. We select the x- and y-component of $\mathbf{u}_i$ independently of each other, each is chosen in the same way as for the one-dimensional walk:

$$u_{x,i} = \begin{cases} \frac{1}{2} & +1 \\ \frac{1}{2} & -1 \end{cases} \quad u_{y,i} = \begin{cases} \frac{1}{2} & +1 \\ \frac{1}{2} & -1 \end{cases} \tag{4.16}$$

We find the position, $\mathbf{r}_N$ after N steps as the sum of the individual steps:

[3] http://folk.uio.no/malthe/compcourse/walk1dscaling02.py

$$\mathbf{r}_N = \sum_{i=1}^{N} \mathbf{u}_i \ . \tag{4.17}$$

In order to find the probability distribution, we count the number of walkers $N(n_x, n_y)$ that end up in the box (n_x, n_y). We prescribe the edges of the boxes so that all integers fall into a specific box by giving the edges in the x- and the y-direction as spanning from $-N - 1/2$ to $N + 1/2$ in steps of 1. The implementation contains the following steps:

```
# Random walk in two dimensions
from pylab import *

dim = 2
N = 100 # nr of steps
nsample = 100000 # nr of samples
x = zeros((nsample,dim))
n = zeros(nsample)
for i in range(nsample):
    # Generate random walk of N steps
    z = 2*randint(2,size=(N,dim))-1
    x[i,:] = sum(z,axis=0)
    nrm = norm(x[i])
    n[i] = nrm
#plot(n)
Nn,edges = histogram(n)
#3hist(n)
#%%
xedges = array(range(int(min(x[:,0])),int(max(x[:,0])),2))+0.5
yedges = array(range(int(min(x[:,1])),int(max(x[:,1])),2))+0.5
Nn,xedges,yedges = histogram2d(x[:,0],x[:,1],[xedges,yedges])
imshow(Nn,interpolation='nearest', origin='low',
                extent=[xedges[0], xedges[-1], yedges[0], yedges[-1]])
axis('equal')
```

Notice that we generate all the $\mathbf{u}_i$ values with the command

```
    z = 2*randint(2,size=(N,dim))-1
```

which returns a matrix consisting of two-dimensional vector specifying the individual steps of the walker. Also notice the use of the `sum` function to sum all the x-displacements and all the y-displacements. That is, the command

```
    x[i,:] = sum(z,axis=0)
```

returns a vector containing $\sum_i u_{xi}$ as the x-component and $\sum_i u_{yi}$ as the y-component.

Notice also the use of the `histogram2d` function to generate the two-dimensional histogram with the specified edges. The resulting plot is

shown in Fig. 4.5. We could perform a similar scaling analysis for the two-dimensional system as we did for the one-dimensional system. **ams 2**: Include in next version

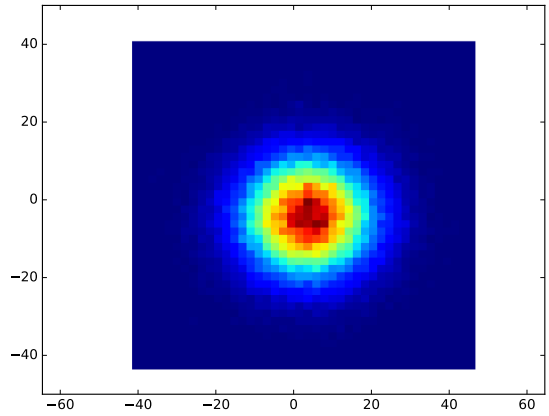

Fig. 4.5 Image showing the local probability $P_N(x, y)$ for the walker to be in the position (x, y) aften N steps when it starts at $(0, 0)$.

4.2 Random walks and diffusion

How can we relate random walks to the number of atoms/particles in a given region in space. Fig. 4.6 shows random walkers on a course grained grid on the left. There is initially a number $N_{i,j}$ walkers in the center grid box, but after a few time steps, corresponding to some steps of the random walkers, some of the walkers have walked from $N_{i,j}$ and to its surrounding boxes, and some walkers have walked from the surrounding boxes and into $N_{i,j}$.

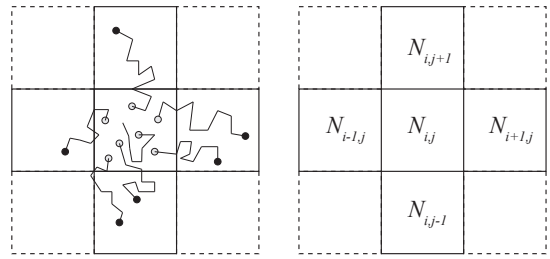

Fig. 4.6 Illustration of the motion of random walkers and the number of random walker per box.

Analysis of one-dimensional system. We can address this process in detail in a one-dimensional system. In this case, we have a given number of walkers, $N_i(t)$ in box i at time t. We assume that each walker has a probability $p = R\Delta t$ to leave the box and move into box $i+1$ during a time interval Δt. This means that the number of walkers that moved from box i into box $i+1$ will be pN_i in the time interval Δt. Similarly, there is a number pN_i that moves from box i and into box $i-1$. There will also be walkers walking from box $i-1$ and into box i, and walkers moving from box $i+1$ into box i. Let us include these effects into an equation for N_i:

$$N_i(t+\Delta t) = N_i(t) - \underbrace{R\Delta t N_i(t)}_{\text{(moving into } i-1\text{)}} - \underbrace{R\Delta t N_i(t)}_{\text{(moving into } i+1\text{)}} \tag{4.18}$$

$$+ \underbrace{R\Delta t N_{i-1}(t)}_{\text{(coming from } i-1\text{)}} + \underbrace{R\Delta t N_{i+1}(t)}_{\text{(coming from } i+1\text{)}} . \tag{4.19}$$

We collect term on each side to get

$$\frac{N_i(t+\Delta t) - N_i(t)}{\Delta t} = R\Delta x^2 \frac{(N_{i+1}(t) - N_i(t)) - (N_i(t) - N_{i-1}(t))}{\Delta x^2} \tag{4.20}$$

We recognize this as the discrete form of a partial differential equation in $N(x,t)$:

$$\frac{\partial N}{\partial t} = R\Delta x^2 \frac{\partial N^2}{\partial x^2} . \tag{4.21}$$

If we instead describe the system by the number of particles per box, $c(x,t) = N/V_b$, where V_b is the volume per box, we get the diffusion equation:

$$\frac{\partial c}{\partial t} = D\frac{\partial c^2}{\partial x^2} . \tag{4.22}$$

where $D = R\Delta x^2$ is called the diffusion constant.

Two approaches to address diffusion. This points to two possible approaches to address diffusion. We may consider an ensamble of many particles the diffuse individually, and then count how many particles have ended up in a particular box/position after a given time, or we can describe the number of particles per box, and then calculate how this number changes with time.

Mapping random walks onto diffusion. We can map the random walk model directly onto the diffusion equation. We know that the probability for a particle to move a distance Δx, which is the box size, over a time

Δt is $p = R\Delta t$. This allows us to solve the random walk problem or the diffusion problem and compare the results directly.

Comparing random walks and diffusion in one dimension. We compare the two descriptions directly by describing the diffusion processes by the number of walkers, N_i, in each box. We start the system by all M walkers starting from $x = 0$, which means that $N_0 = M$ and $N_i = 0$ $(i \neq 0)$ initially.

Each time step, Δt, each walker has a probability p to move to the box to its left, and p to move to the box to its right. (It means that it has the probabilty $1 - 2p$ not to move at all). Notice that we here describe this as a single step for the random walker, but we may think of this process as a result of many small steps that lead to the walker moving out of the box (with probability p) or remaining in the box (with probability $1 - 2p$).

Each time step we also update the numbers N_i of particles in boxes i accordring to the diffusion equation in (4.20):

$$N_i(t + \Delta t) - N_i(t) = R\Delta t \left((N_{i+1}(t) - N_i(t)) - (N_i(t) - N_{i-1}(t)) \right) , \tag{4.23}$$

which gives

$$N_i(t + \Delta t) = N_i(t) + R\Delta t \left(N_{i+1}(t) - 2N_i(t) + N_{i-1}(t) \right) , \tag{4.24}$$

where we can replace $R\Delta t = p$.

Simultaneous implementation in Python. We implement both methods at the same time in Python in order to compare the results. This is done in the program diffwalk1d.py[4]:

```
# Comparing random walks and diffusion in 1d
from pylab import *

M = 10000 # Nr of walkers
L = 100 # Max size of lattice

# Each time step - move walkers and propagate diffusion solution

p = 0.1 # Prob for motion
pinv = 1.0-p
nsteps = 3001 # Nr of timesteps

# Initialize walkers
x = zeros(M) # Initial position of walkers
edges = array(range(-L,L+1))-0.5
```

[4] http://folk.uio.no/malthe/compcourse/diffwalk1d.py

```
xc = 0.5*(edges[:-1]+edges[1:])

# Initialize concentrations
c = zeros((2*L+1,2))
i0 = 0
i1 = 1
c[L] = M # c[L] corresponds to x = 0
cx = range(-L,L+1)
D = p

#%%
ion()
noutput = 10
for it in range(nsteps):
    # First update positions of all random walkers
    for iw in range(M):
        rnd = rand(1)
        dx = -1*(rnd<p)+1*(rnd>pinv)
        x[iw] = x[iw] + dx

    # Perform explicit step for diffusion equation
    for ix in range(1,len(c)-1):
        # use i0 and generate i1
        c[ix,i1] = c[ix,i0] + D*(c[ix-1,i0]-2*c[ix,i0]+c[ix+1,i0])
        # Notice how the end points are avoided - issues?
    # Flip i0 and i1
    ii = i1
    i1 = i0
    i0 = ii

    # Plot the two concentrations
    if (mod(it,noutput)==0):
        Nx,e = histogram(x,edges)
        clf()
        plot(cx,c,'-r',xc,Nx,'-b')
        xlabel('x'),ylabel('N'),pause(0.001)
```

Notice the nice way the step `dx` that we add to each walker includes the possibility to stay and the possibilities to move either left or right.

```
        dx = -1*(rnd<p)+1*(rnd>pinv)
```

The resulting plots after $M = 100$, 500, 1000 and 2000 time steps are shown in Fig. 4.7. We notice that the two solutions follow each other closely, but that the random walker model has more noise. However, the amount of noise would be reduced if the number of walkers was increased.

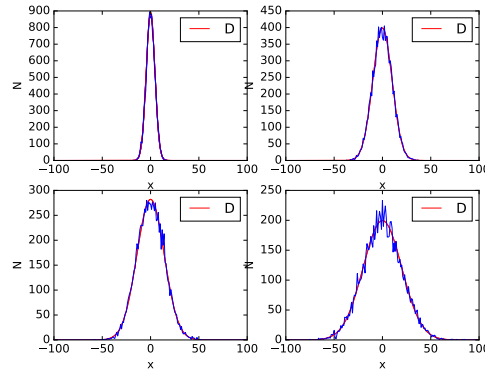

Fig. 4.7 Illustration of the distribution $N(x)$ of walkers for the diffusion model (red) and the random walker model (blue).

4.2.1 The diffusion equation

We motivated the diffusion equation from the number of random walkers passing from one box to another:

$$N_i(t + \Delta t) = R\Delta t \left(-N_i(t) - N_i(t) + N_{i-1}(t) + N_{i+1}(t)\right) \ . \tag{4.25}$$

This was done for the one-dimensional case, but a similar argument can be made in two (or higher dimensions):

$$N_{i,j}(t+\Delta t) = N_{i,j}(t)+R\Delta t \left(-4N_{i,j}(t) + N_{i-1,j}(t) + N_{i+1,j}(t) + N_{i,j-1}(t) + N_{i,j+1}(t)\right) \ . \tag{4.26}$$

We can again rearrange this into a set of derivatives:

$$\frac{\partial N}{\partial t} = R\Delta x^2 \left(\frac{\partial^2 N}{\partial x^2} + \frac{\partial^2 N}{\partial y^2}\right) \ , \tag{4.27}$$

where we again may divide by the volume, V_b, per boxing, resulting in an equation for the concentration $c(x, t)$ of particles:

$$\frac{\partial c}{\partial t} = D \left(\frac{\partial c^2}{\partial x^2} + \frac{\partial c^2}{\partial y^2}\right) = D\nabla^2 c \ . \tag{4.28}$$

Solving the diffusion equation is efficient. Solving the diffusion equation is an efficient way to model the changes in concentration — more efficient than using random walkers. We can solve it using the explicit scheme presented for $N_{i,j}$ above. This is not the most robust or efficient method to solve the diffusion equation, but it is simple and easy to understand.