1 Background

Today we will continue with random walks, making our way to the diffusion problem know as Brownian motion. Before getting to that point we need to understand more sophisticated ways of generating non-uniform random distributions. In the previous two lectures we used the function `numpy.random.random()` which generates random numbers in the range $(0, 1)$ with the same probability, this means that every number in this range has the same change of being generated. This is called a *uniform* process, denoted by $\mu(0, 1)$. We already know how to generate uniform numbers in arbitrary ranges, in python this can also be accomplished with the function

```
numpy.random.uniform(a, b)
```

The result of the random walk, on the other hand, has a very distinctive shape. If $p = 0.5$ we found that the change of finishing the walk close to $x = 0$ is larger than finishing the walk at, e.g. $x = 10$. In other words, the probability of generating the final positions of the walkers is *not* uniform. In fact, it is called a *binomial* process, denoted by $B(p, N)$. And the result is a bell-like shape as we discovered last session. In python, binomial random numbers can be generated using the function

```
numpy.random.binomial(N, p)
```

There is another random process of interest for us, which is called a *normal* (Or Gaussian) process $\mathcal{N}(\mu, \sigma)$

```
numpy.random.normal(mu, sigma)
```

where `mu` is the mean and `sigma` is the deviation. We can visualize how these different methods work by using *histograms*, the same way we did last class for the random walk in one dimension.

```
numpy.histogram(x, bins=<bins>)
```

This is an example of how to generate $n = 50000$ random numbers using these three distributions

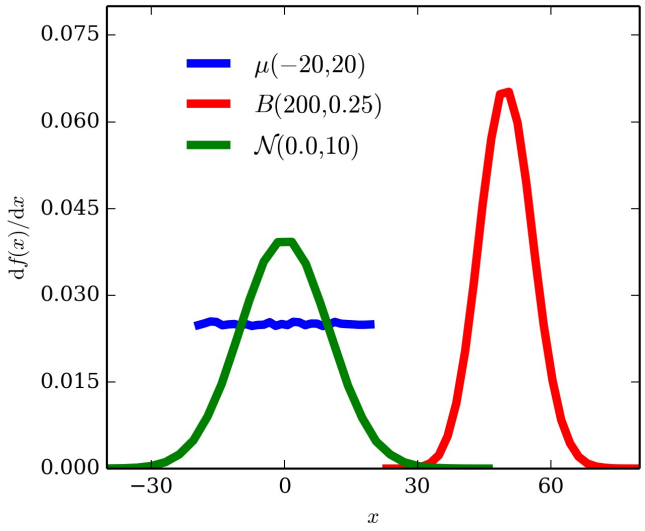
```
# draws n samples from different random distributions
n = 50000
xu = np.random.uniform(-20, 20, n)
xb = np.random.binomial(200, 0.25, n)
xn = np.random.normal(0.0, 10, n)

# generates the histograms
bins = 30
y1, x1 = np.histogram(xu, bins=bins, density=True)
y2, x2 = np.histogram(xb, bins=bins, density=True)
y3, x3 = np.histogram(xn, bins=bins, density=True)

# shifts the edges
x1 = x1[0 : -1] + 0.5 * (x1[1] - x1[0])
x2 = x2[0 : -1] + 0.5 * (x2[1] - x2[0])
x3 = x3[0 : -1] + 0.5 * (x3[1] - x3[0])

# plot
```

```
plt.plot(x1, y1, 'b-')
plt.plot(x2, y2, 'r-')
plt.plot(x3, y3, 'g-')
```



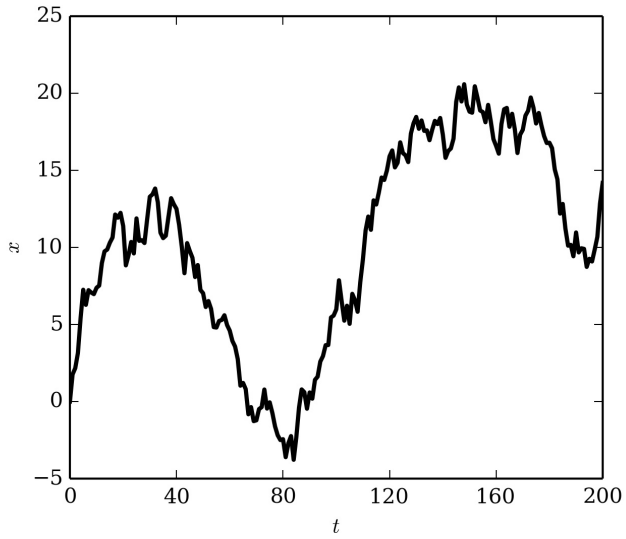
With this, we have enough elements to generate our first Brownian process. One defining property of the process is that successive steps are independent and follow a normal distribution

2 Problems

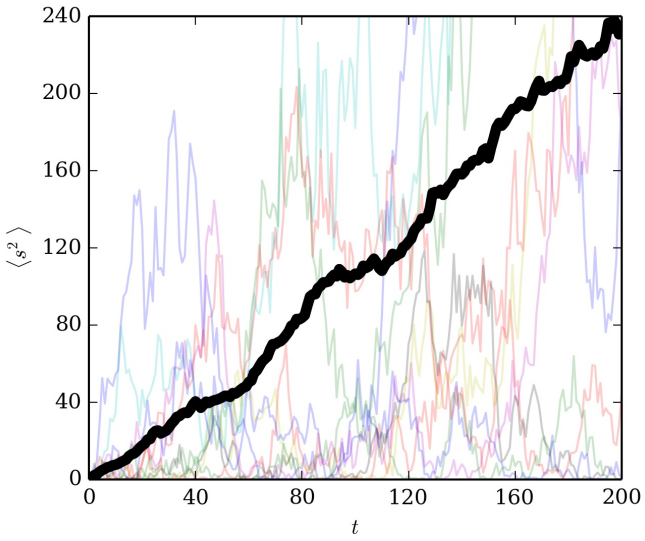
1. Investigate the use of the functions `sum` and `numpy.cumsum`
2. Create a function that returns a Brownian random walk in 1 dimension with n steps. Assume that every point in the walk the next step follows the distribution $\mathcal{N}(0, 1)$. As before, we will start our walks from $x = 0$

   ```
   brownian(n)
   ```

   For the examples use $n = 200$



3. At a given time t (iteration) calculate the distance to the center s. Calculate $\langle s^2 \rangle$ for N random walks an plot it as a function of t. Use $N = 100$



4. The diffusion coefficient D is defined such that

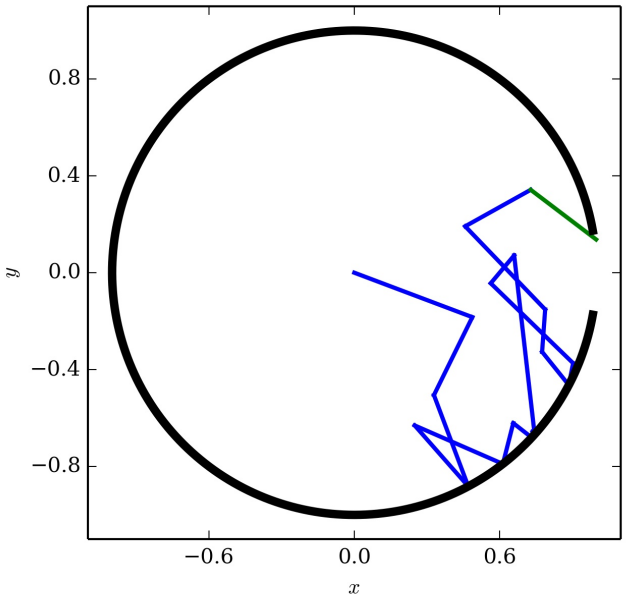
$$\langle s^2 \rangle = 2Dt$$

Find D for this walk (hint: use the function `scipy.stats.linregress`)

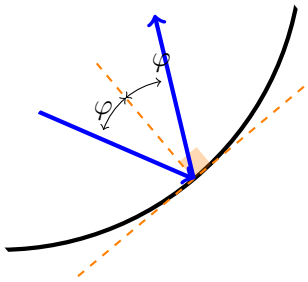
$D = 0.61$

5. Advanced problem: Narrow Escape A particle in Brownian motion is confined in a circle of radius R. Assume that each step is drawn from the distribution $\mathcal{N}(0, \Delta)$ and that at $t = 0$ the particle is located at the origin.

(a) The circle is open as shown in the figure. Assume that the angle subtended by the missing arc is $\Delta\theta$, such that the length of the orifice is $\Delta s = R\Delta\theta$.



(b) If the particle hits the circle, it bounces elastically. That is, the impact angle is the same as the reflected angle



(c) Define the time that it takes for a single particle to leave the circle as τ. Calculate the average $\langle\tau\rangle$ over a set of n random walks.

(d) Plot $\ln\langle\tau\rangle$ as a function of $\ln\Delta s$. use $R = 1$, $\Delta = 0.2$, $N = 1000$.

3 Solutions

1. `sum` calculates the sum of all the elements in an array. `cumsum` returns the cumulative total up until a given position of an array

2. ```
   # Brownian random walk
   np.random.seed(0)

   def brownian(n):
       delta = np.random.normal(0, 1, n)
       x = np.append(0, np.cumsum(delta))

       return x

   x = brownian(200)
   plt.plot(x)
   ```

3. ```
   # Plots the displacements

   N = 100
   n = 200

   s2 = np.zeros(n + 1)
   for i in range(N):

       x = brownian(n)
       s2 = s2 + x**2

   s2 /= float(N)
   t = range(n + 1)

   plt.plot(t, s2, 'r-')
   plt.show()
   ```

4. ```
   slope, intercept, r_value, p_value, std_err = stats.linregress(t, s2)
   print 'Diffusion coefficient D =', 0.5 * slope

   >>> Diffusion coefficient D=0.61
   ```
   ```