Lecture 5: Randomness

- Random numbers, probability distributions, and histograms

- Stochastic processes

Random numbers

- Pseudo-randomness and seeds

  Random numbers are uncorrelated (cannot guess the next number knowing the previous numbers)

  Typically random numbers generated on a computer are not truly random (needs special hardware).

  Algorithms are deterministic in nature,
  same starting point → same results
  python defaults to seeding off of system time.

- Probability distribution from which random numbers are selected
  - Uniform, Gaussian, etc.

```
np.random.rand(10)
```

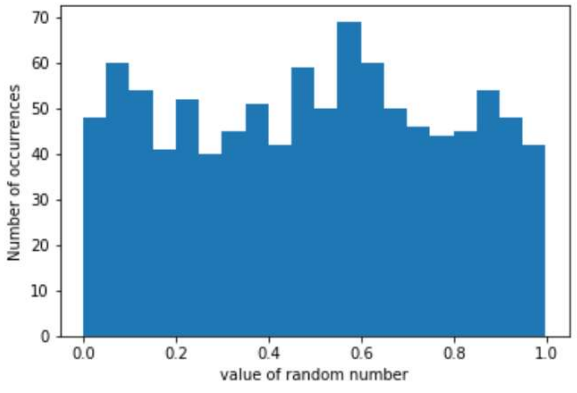
```
array([0.9255095 , 0.75993944, 0.8229656 , 0.60445265, 0.70088504,
       0.68478212, 0.74377771, 0.53734852, 0.16260794, 0.91740962])
```

Uniform Distribution (every number in the range is equally probable).

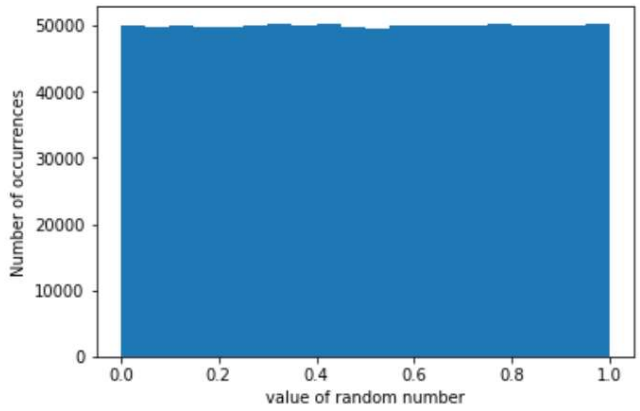
Selecting from **uniform** distribution in range [0,1):
np.random.rand() returns random number
np.random.rand(N) returns array of N random numbers

Histograms show the number of times a value was obtained.

```
mynums=np.random.rand(1000)
plt.hist(mynums,20)
plt.ylabel("Number of occurrences")
plt.xlabel("value of random number")
plt.show()
```



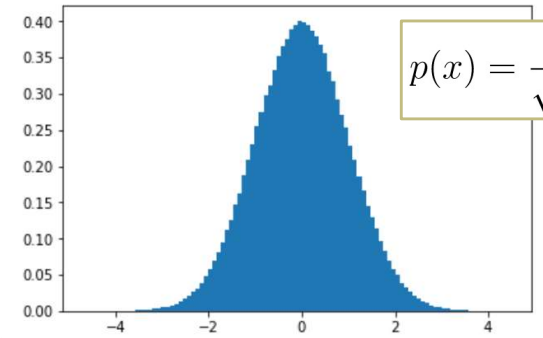

```
mynums=np.random.rand(1000000)
plt.hist(mynums,20)
plt.ylabel("Number of occurrences")
plt.xlabel("value of random number")
plt.show()
```



density='true' will normalize histogram so it is directly comparable to the probability distribution

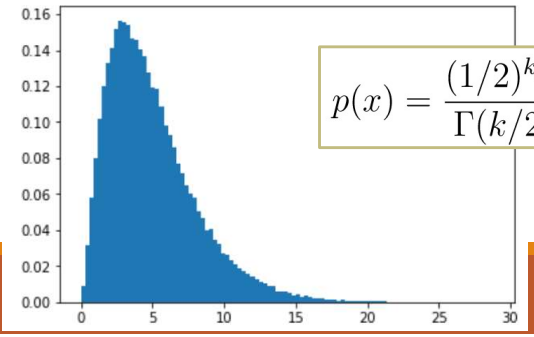
Probability distribution

```
mynums=np.random.normal(0,1,1000000)
plt.hist(mynums,100,density='true')
plt.show()
```



$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

```
mynums=np.random.chisquare(5,100000)
plt.hist(mynums,100,density='true')
plt.show()
```



$$p(x) = \frac{(1/2)^{k/2}}{\Gamma(k/2)} x^{k/2-1} e^{-x/2}$$

np.random can sample from many different distributions

beta(a, b[, size])
binomial(n, p[, size])
chisquare(df[, size])

dirichlet(alpha[, size])
exponential([scale, size])

f(dfnum, dfden[, size])
gamma(shape[, scale, size])
geometric(p[, size])

gumbel([loc, scale, size])
hypergeometric(ngood, nbad, nsample[, size])

laplace([loc, scale, size])

logistic([loc, scale, size])
lognormal([mean, sigma, size])

logseries(p[, size]) ...

Stochastic processes

They posses inherent randomness (examples include radioactive decay, diffusion, magnetic domain orientation, many statistical mechanics problems)

How do we describe such processes?

Radioactive decay

- Spontaneous decay is a natural process in which a particle, with no external stimulation, and at one instant in time, decays into other particles

- Since the exact moment when any one particle decays is random, it does not matter how long the particle has been around or what is happening with other particles. (independent events)

Probability of decay

Lets say in 1 second, a particle has a 1% chance of decaying.

numerical route

Start with initial number of particles N_0

Loop over total time (with a time step of 1 sec):

for each particle roll some dice (1% of the time, destroy particle)

analytic route

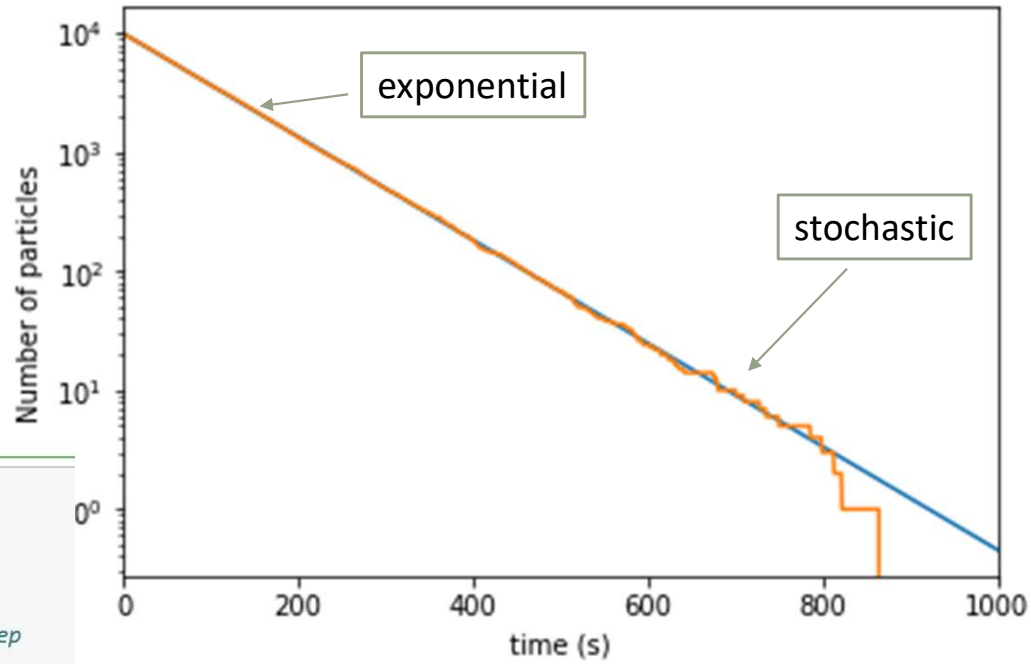
$$\frac{dN}{dt} = -.01 * N$$
$$\int dN/N = \int -.01 * dt$$
$$\ln(N) = -.01t + C$$
$$N = N_0 e^{-.01t}$$

(we assume that 1% of our particles disappear each second). This really only makes sense if we are talking about a large number of particles

Simple Monte-Carlo type calculation

Start with initial number of particles N

Loop over total time (with a time step of 1 second):
    Loop over every remaining particle:
    for each particle roll some dice (1% of the time, destroy that particle)



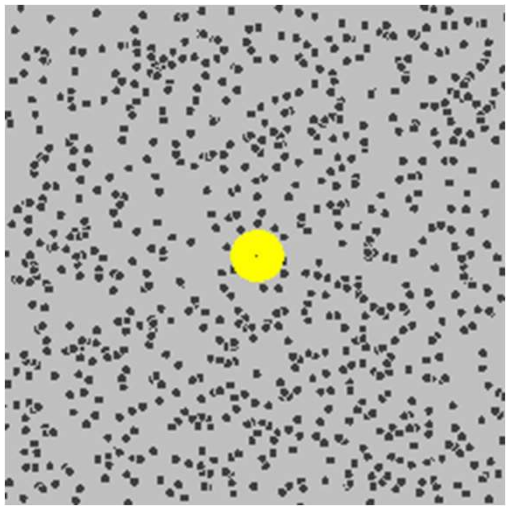

```
import numpy as np
import matplotlib.pyplot as plt
N=10000  # initial number of particles
Ns=[]
for i in range(10000):  #time step of 1sec
    Ns.append(N)        #keep track of how many particles exist at each time-step
    for j in range(N):  #loop over remaining particles
        if (np.random.random()<.01):  # if random is < 0.01 (happens 1% of the time)
            N=N-1                     # decrease the number of particles by 1
```

Diffusion (random walk)

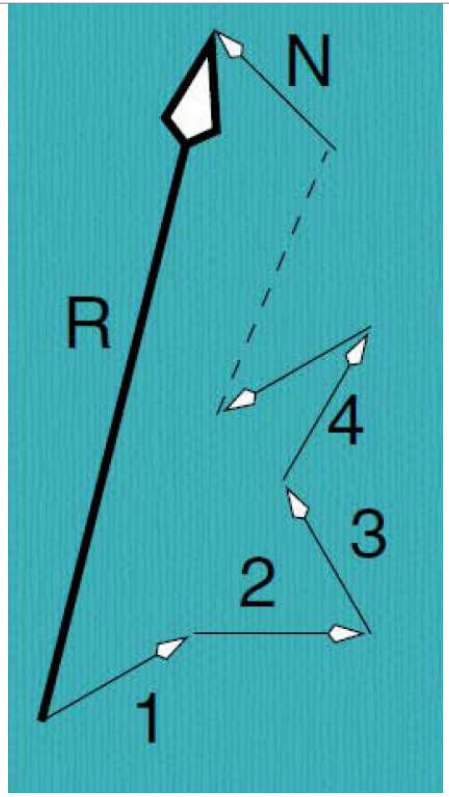
Brownian motion: Pollen undergoes erratic motion when in a fluid.

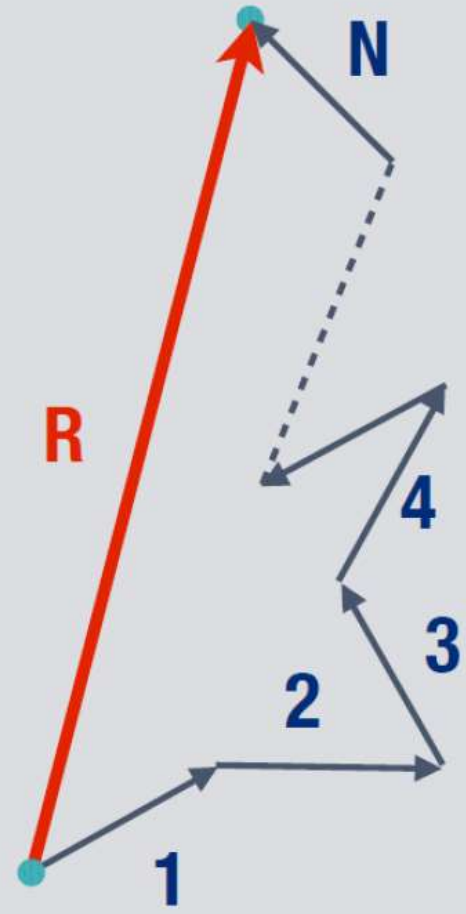
This is due to the instantaneous imbalance of net forces arising from collisions with the environment.

Often modeled as a random walk



(note this can be modeled with a random force taken from a Gaussian distribution)

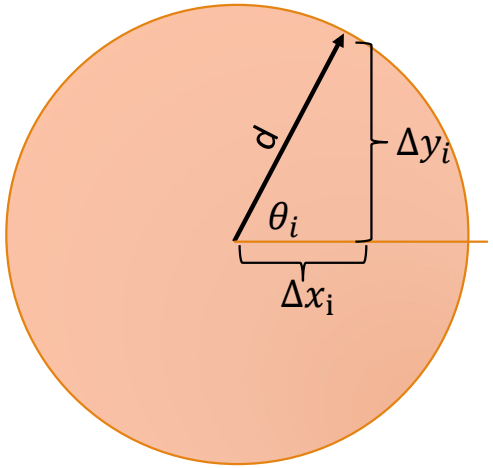




R for N steps?

2D Random Walk

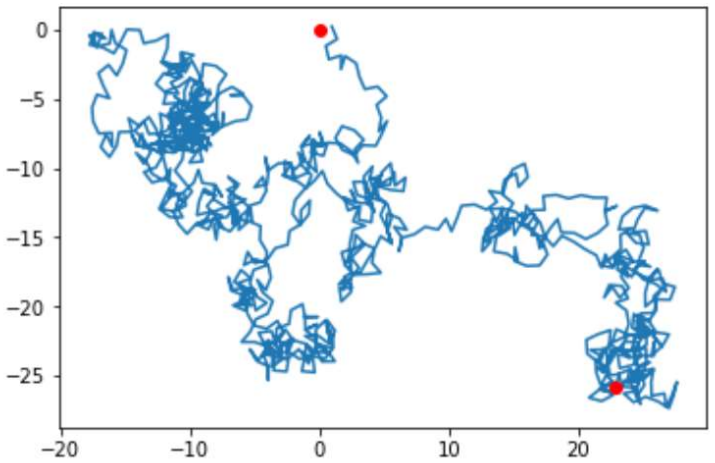
- Each step is the same length d
- Each step is in a random direction → θ_i = random number $[0,2\pi)$.



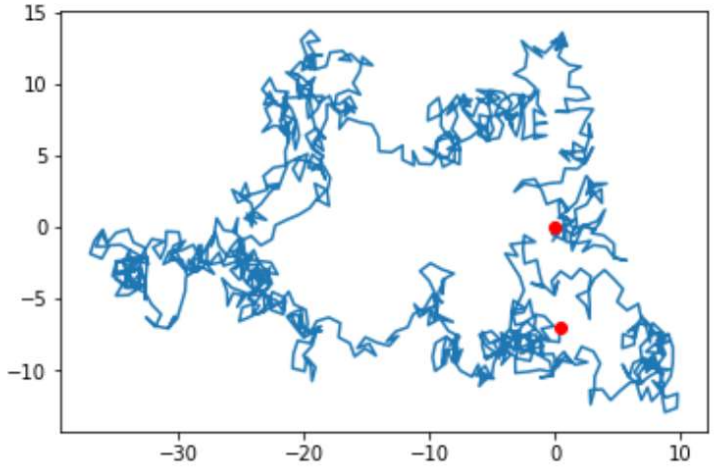
After N steps,

- $x_{final} = \Delta x_1 + \Delta x_2 + \cdots + \Delta x_N$
  $= \sum_{i=1}^{N} \cos(\theta_i) d$
- $y_{final} = \Delta y_1 + \Delta y_2 + \cdots + \Delta y_N$
  $= \sum_{i=1}^{N} \sin(\theta_i) d$

```
thetas=np.random.rand(1000)*2*np.pi
dx=np.cos(thetas)
dy=np.sin(thetas)
xpos=dx.cumsum()
ypos=dy.cumsum()
plt.plot(xpos,ypos) #plot trajectory
plt.plot(xpos[-1],ypos[-1],'ro') #show final point
plt.plot(0,0,'ro') #show initial point
plt.show()
```




```
thetas=np.random.rand(1000)*2*np.pi
dx=np.cos(thetas)
dy=np.sin(thetas)
xpos=dx.cumsum()
ypos=dy.cumsum()
plt.plot(xpos,ypos) #plot trajectory
plt.plot(xpos[-1],ypos[-1],'ro') #show final point
plt.plot(0,0,'ro') #show initial point
plt.show()
```



Every realization of the walk is different. What useful information can be extracted?

Note: cumsum = cumulative sum
dx.cumsum() returns array: [dx[0], dx[0]+dx[1], dx[0]+dx[1]+dx[2],…]

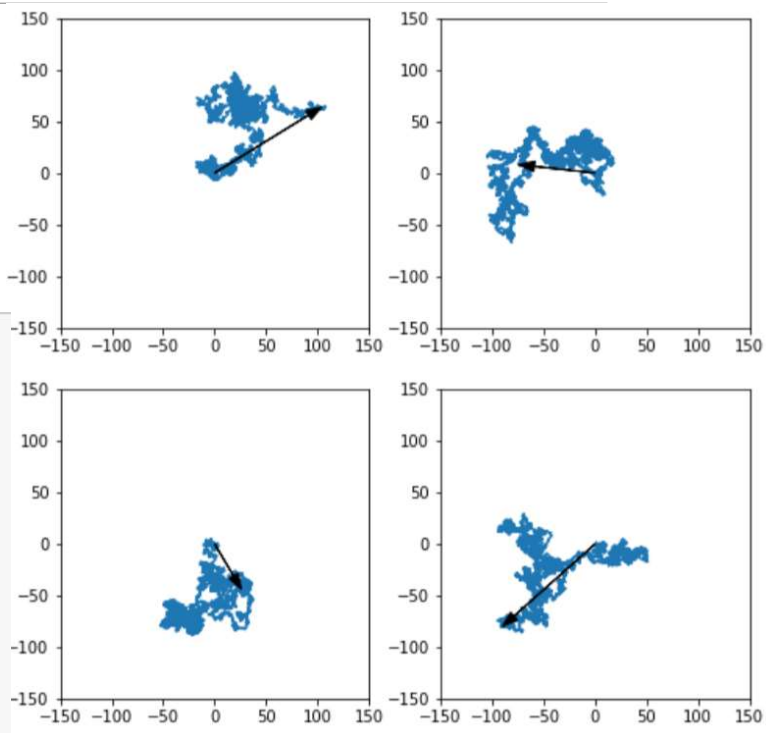
subplots

```
fig, ax = plt.subplots(nrows=2, ncols=2)
```

creates a 2D-array ax of subplots. ax[0,0], ax[0,1], ax[1,0], and ax[1,1]

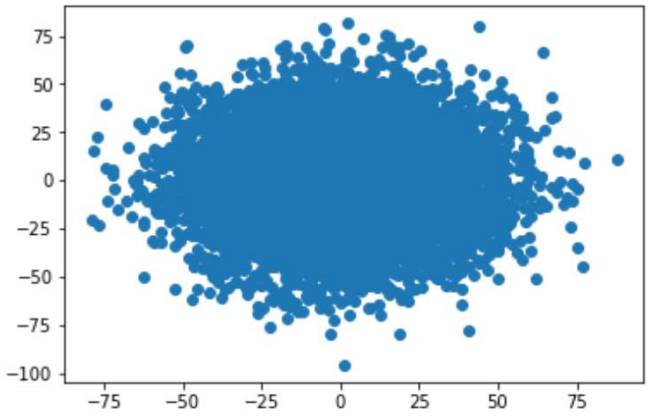
To show 4 instances of a 2D random walk, we could use the following

```
n=2;
m=2;
fig=plt.figure(figsize=(4*m,4*n))
ax=[fig.add_subplot(n,m,i+1) for i in range(n*m)]

for a in ax:
    stepsx=(-1)**np.round(np.random.rand(5000))
    stepsy=(-1)**np.round(np.random.rand(5000))
    posx=stepsx.cumsum()
    posy=stepsy.cumsum()
    a.plot(posx,posy)
    a.arrow(0,0,posx[-1],posy[-1],color='k',head_width=10,length_includes_head='true',zorder=2)
    a.set_aspect('equal')
    a.set_xlim((-150,150))
    a.set_ylim((-150,150))
fig.subplots_adjust(wspace=.2,hspace=.2)

plt.show()
```

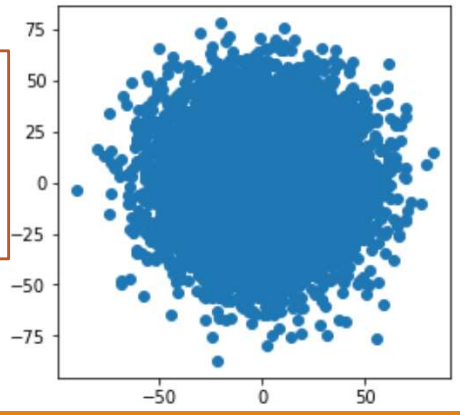


Need to do many walks for decent statistics

```
fpos=[]
for i in range(10000):
    thetas=np.random.rand(1000)*2*np.pi
    dx=np.cos(thetas)
    dy=np.sin(thetas)
    xpos=dx.cumsum()
    ypos=dy.cumsum()
    fpos.append([xpos[-1],ypos[-1]])
fpos=np.array(fpos)
plt.plot(fpos[:,0],fpos[:,1],'o')
plt.show()
```




```
fpos=[]
for i in range(10000):
    thetas=np.random.rand(1000)*2*np.pi
    dx=np.cos(thetas)
    dy=np.sin(thetas)
    xpos=dx.cumsum()
    ypos=dy.cumsum()
    fpos.append([xpos[-1],ypos[-1]])
fpos=np.array(fpos)
plt.axes().set_aspect(1)
plt.plot(fpos[:,0],fpos[:,1],'o')
plt.show()
```



proper aspect ratio is important, otherwise results can be misleading

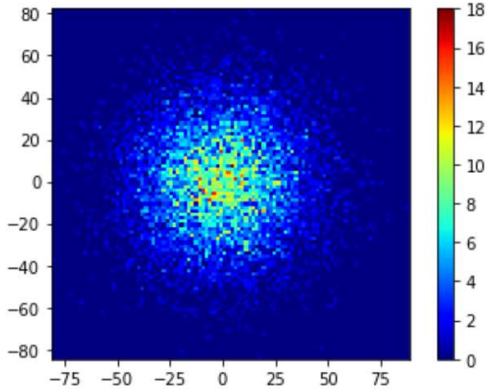
final positions of 10,000 walks (each of 1000 steps)

2D histogram

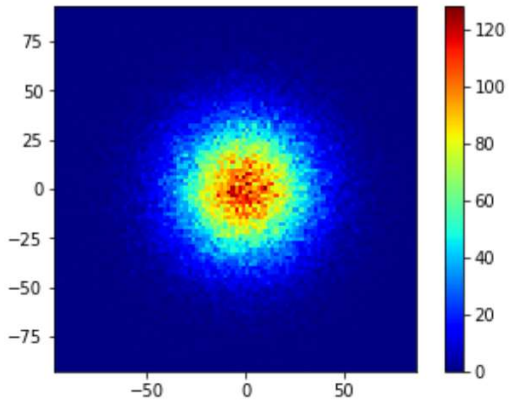
```
def walk2D(N):   #returns final position after
                 # random 2D walk with N steps
    thetas=np.random.rand(N)*2*np.pi
    dx=np.cos(thetas)
    dy=np.sin(thetas)
    return [dx.sum(),dy.sum()]
```

```
fpos=np.array([walk2D(1000) for _ in range(Nwalks)])
plt.hist2d(fpos[:,0],fpos[:,1],100,cmap=plt.cm.jet)
plt.axes().set_aspect(1)
plt.colorbar()
plt.show()
```

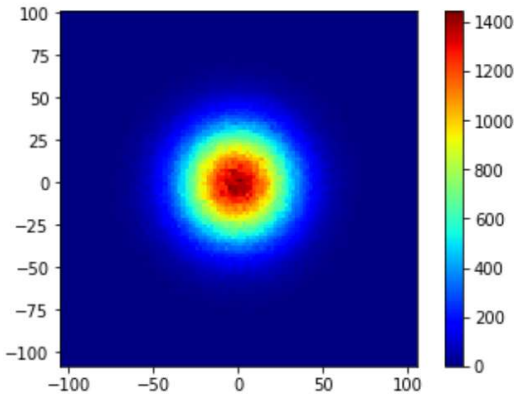
10^4 walks



10^5 walks

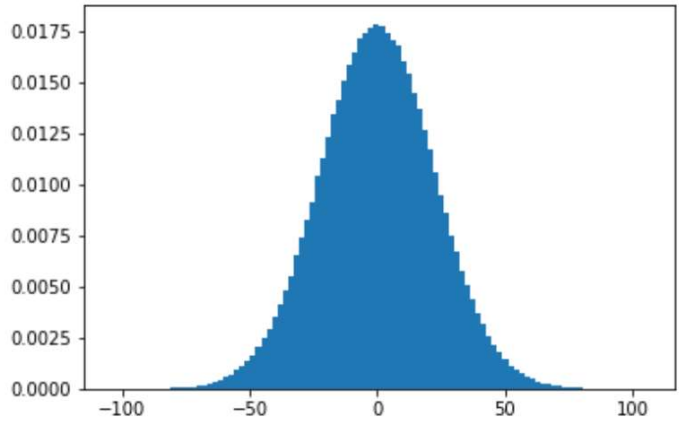


10^6 walks

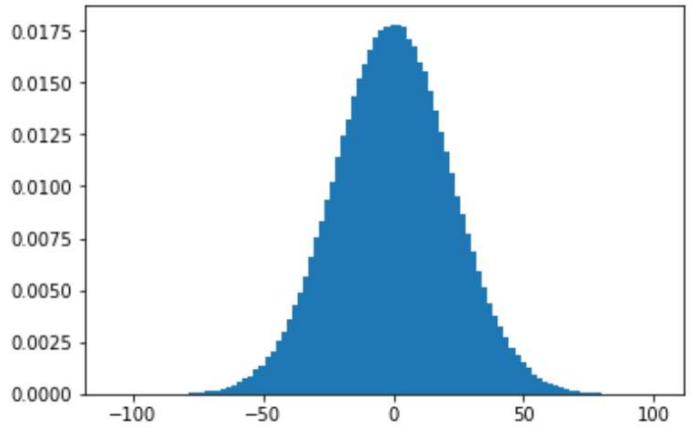


Looking at x and y values

```
plt.hist(fpos[:,0],100,density='true')
plt.show()
```




```
plt.hist(fpos[:,1],100,density='true')
plt.show()
```



statistics:

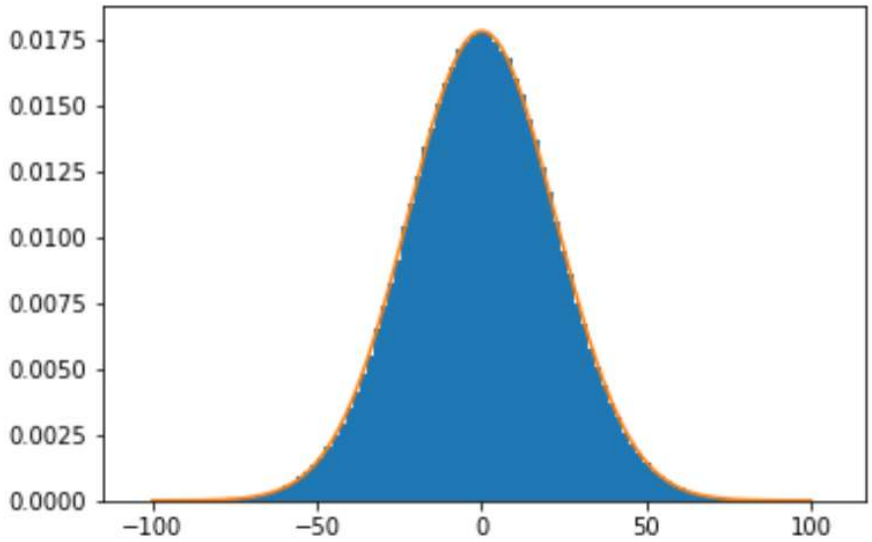
```
print(fpos[:,0].std())
print(fpos[:,1].std())
```

```
22.365966837166575
22.356927140751047
```

```
print(fpos[:,0].mean())
print(fpos[:,1].mean())
```

```
-0.02719996683452895
-0.041976258326789806
```

```
plt.hist(fpos[:,0],100,density='true')
mu=fpos[:,0].mean()
sigma=fpos[:,0].std()
xvals=np.linspace(-100,100,1000)
yvals= (1/(sigma * np.sqrt(2 * np.pi))) * np.exp( - (xvals-mu)**2 / (2 * sigma**2))
plt.plot(xvals,yvals)
plt.show()
```



68% of trajectories fall within $\pm\sigma$
95% within $\pm 2\sigma$
$\sigma \approx 22.3$

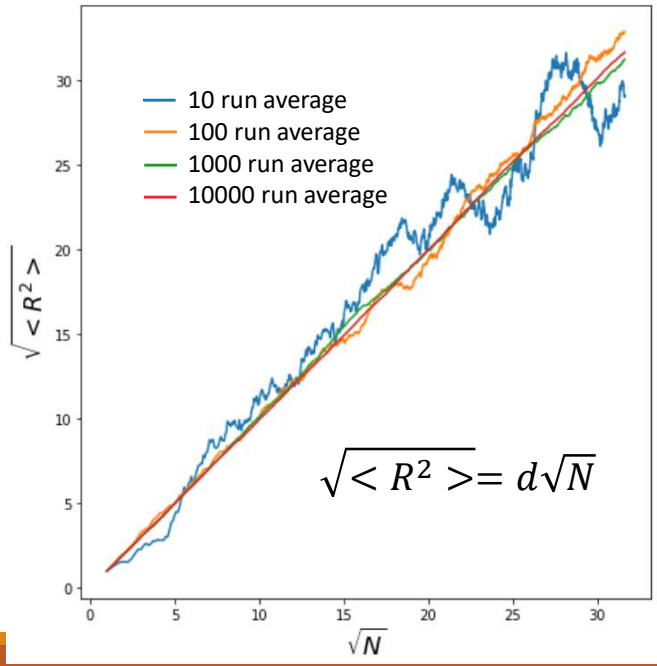
The final x and y positions after a random walk follow a Gaussian distribution.

How does number of steps effect result?

root mean square distance:

$$\sqrt{< R^2 >} = \sqrt{< x^2 + y^2 >}$$

(brackets indicate average)




```
np.sqrt((fpos[:,0]**2+fpos[:,1]**2).mean())
```

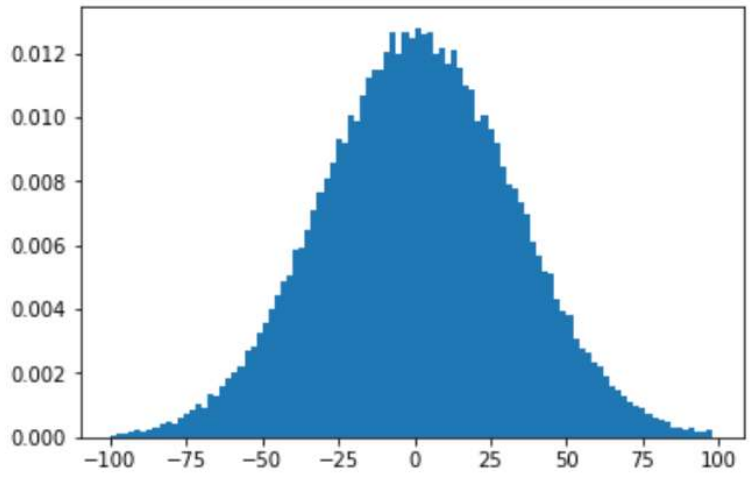
```
31.623901808324423
```

```
runs=100
sqpos=np.zeros(1000)
for i in range(runs):
    thetas=np.random.rand(1000)*2*np.pi
    dx=np.cos(thetas)
    dy=np.sin(thetas)
    xpos=dx.cumsum()
    ypos=dy.cumsum()
    sqpos = sqpos + xpos*xpos+ypos*ypos
sqpos=np.sqrt(sqpos/runs)
sq100=sqpos
```

Note on 1D walks

can avoid effects from even/odd number of steps by choosing bins of width 2.

```
n, bins,_ = plt.hist(fpos, np.arange(-100,100)[::2],density='true')
```



Activity (lecture 5)

1D random walk (equal probability of going left or right)

1) Construct the probability distribution (histogram) associated with the final position after a 5000 step walk.

2) Generate a plot of the RMS position $\sqrt{< R^2 >}$ as a function of $\sqrt{N}$.