PY541 PS2

Raymond Kil

September 29,2022

Problem 1: Generating Random Walks (Sethna 2.5)

(a)

Attached below.

(b)

Attached below.

(c)

The RMS step-size for 1D steps uniformly distributed in $(-1/2, 1/2)$ is given by:

$$a = \sqrt{\langle x_{\text{step}}^2 \rangle} = \sqrt{\int_{-1/2}^{1/2} x^2 dx} = \frac{1}{\sqrt{12}} \approx 0.289 \tag{1}$$

Rest of part (c) is attached below.

Problem 2: Stocks, Volatility, and Diversification (Sethna 2.11)

(a)

Attached below.

(b)

Attached below.

(c)

The Gaussian is given by:

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp - \frac{(x-\mu)^2}{2\sigma^2} \tag{2}$$

Taking the logarithm gives precisely the equation for an inverted parabola:

$$\log g(x) = -\log\left[\frac{1}{\sigma\sqrt{2\pi}}\right] \frac{(x-\mu)^2}{2\sigma^2} = -\alpha x^2 \tag{3}$$

where $\alpha = \log\left[\frac{1}{\sigma\sqrt{2\pi}}\right]\frac{1}{2\sigma^2}$ and $\mu = 0$. The rest is attached below.

(d)

Attached below.

(e)

Let's understand the question first. When we buy stocks, we get returns. The problem gives two ways of buying the stocks. First, buying the index, and second, buying the individual stocks. Buying the index means that the stocks are weighted equally. Let's assign variables to these quantities.

- ξ_i = percentage return of ith stock, where $1 \leq i \leq 500 = N$
- m_i = mean annual percentage return = $\langle \xi_i \rangle$
- σ_i = mean volatility=$\sqrt{\langle (\xi_i - m_i)^2 \rangle}$
- Ω_i = Fraction of ith stock that one has out of total number of stocks. That is, $\sum_{i=1}^{500} \Omega_i = 1$
- ξ_Ω = Total percentage return when one buys stocks with distribution $\{\Omega_i\}$. That is, $\xi_\Omega = \sum_{i=1}^{N} \Omega_i \xi_i$

The question asks the followings:

1. Return and volatility if one buys all stock as index. That is, m_i when $\Omega_i = \frac{1}{N}$ for all i
2. Higher return between {index} and {individual stocks}
3. Higher volatility between {index} and {individual stocks}

For question 1, we are interested in $\langle \xi_\Omega \rangle$. We have:

$$\langle \xi_\Omega \rangle = \left\langle \sum_{i=1}^{N} \Omega_i \xi_i \right\rangle = \sum_{i=1}^{N} \frac{1}{N} \langle \xi_i \rangle = \frac{1}{N} \sum_{i=1}^{N} m_i \tag{4}$$

For question 2, we are comparing the percentage return obtained in question 1 with the percentage return we get if we buy the stocks with random fraction (since we have no inside information). Note that there is equal probability to choose the fraction of each stock. Thus, choosing random fractions will yield a mean of equal weighting. That is,

$$\langle \xi_{random} \rangle = \langle \xi_\Omega \rangle \tag{5}$$

Therefore, the expected return for the index is equal to the average return for buying individual stocks.

For the last question, let's derive the expressions for the volatility when $\Omega_{index} = \frac{1}{N}$ and when Ω_{random}.

$$\sigma_{index} = \sqrt{\sum_{i=1}^{N} \Omega_{index}^2 \sigma_i^2} = \frac{1}{N} \sqrt{\sum_{i=1}^{N} \sigma_i^2} \tag{6}$$

On the other hand, we have for Ω_{random}:

$$\sigma_{random} = \left\langle \sqrt{\sum_{i=1}^{N} \Omega_{random}^2 \sigma_i^2} \right\rangle \tag{7}$$

Comparing 6 and 7 gives the intuition that, if we have a certain volatility distribution, σ_{random} has a chance that it will give more weight to the stock that has more volatility, and less weight to those with less volatility, whereas σ_{index} gives equal weight to every stock regardless of the volatility distribution. Therefore, generally, $\sigma_{random} > \sigma_{index}$. That is, there is higher volatility when buying individual stocks, compared to that when buying the index.

Problem 3: Run and Tumble (Sethna 2.19)

(a)

Generally, the mean-square distance is given by:

$$\left\langle \mathbf{r}^2(t) \right\rangle = N l^2 \tag{8}$$

where $l = VT$ is the step size, and $N = \frac{t}{T+\tau}$. Thus, we have the expression:

$$\left\langle \mathbf{r}^2(t) \right\rangle = \frac{t}{T+\tau} (VT)^2 \tag{9}$$

Also, we have the following relationship:

$$\langle \mathbf{r}^2(t) \rangle = q_i D t \tag{10}$$

where $q_i = 6$ in 3-dimensions, D is the diffusion coefficient, and t is time. Thus, we have:

$$\sqrt{\langle \mathbf{r}^2(t) \rangle} = \sqrt{6Dt} = \sqrt{\frac{t}{T+\tau}(VT)^2} \Rightarrow \boxed{D = \frac{V^2T^2}{6(T+\tau)}} \tag{11}$$

(b)

The diffusion equation is given by:

$$\frac{\partial \rho}{\partial t} = D\nabla^2 \rho = 0 \tag{12}$$

Let's find the expression for $\rho(r)$. We know that the bacterium eats all food at range $r = a$, so $\rho(0) = \rho(a) = 0$. On the other hand, at $r = \infty$, the food concentration is N, so $\rho(\infty) = N$. Based on these boundary conditions, we have:

$$\rho(r) = N - \frac{Na}{r} \tag{13}$$

Sanity check: let's see if $\nabla^2\rho$ indeed gives 0:

$$\nabla^2 \rho = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{\partial \rho}{\partial r}\right) = \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{Na}{r^2}\right) = 0 \tag{14}$$

On the other hand, we can obtain an expression for food "current" from the continuity equation:

$$\frac{\partial \rho}{\partial t} = D\nabla^2\rho = -\nabla \cdot \mathbf{J} \Rightarrow \mathbf{J} = -D\nabla\rho = -D\frac{\partial \rho}{\partial r} = \frac{aND}{r^2}\hat{r} \tag{15}$$

Since the bacterium eats the food with perfect efficiency, the space within radius a from the bacterium will be empty. That means that the food consumption rate of the bacterium will be equal to the food diffusion rate $4\pi aND$. We get this value by integrating the food current $\mathbf{J}$ over the surface area of the sphere with radius a.

$$\int_\Sigma \mathbf{J} \cdot d\mathbf{A} = aND \int_{\varphi=0}^{2\pi} \int_{\theta=0}^{\pi} \frac{1}{r^2} r^2 \sin(\theta) d\theta d\varphi = \boxed{4\pi aND} \tag{16}$$

(c)

Let's first recall the variables.

- V = velocity of the bacterium (the direction is chosen at random for each random walk)
- T_+ = Time duration that the bacterium travels in $+x$ direction (direction for a better life)
- T_- = Time duration that the bacterium travels in $-x$ direction (Sethna denotes T, but I added the subscript to be clear)
- τ = Duration of tumble

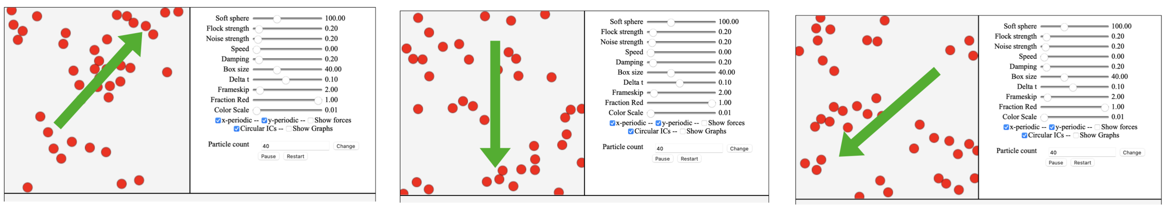
We are interested in the average velocity, i.e., $\left\langle \frac{dx}{dt} \right\rangle$. This is obtained by:

$$\left\langle \frac{dx}{dt} \right\rangle = \left\langle \frac{\text{total distance}}{\text{total time}} \right\rangle = \left\langle \frac{VT_+ - VT_-}{T_+ + T_- + 2\tau} \right\rangle = \boxed{V\left\langle \frac{T_+ - T_-}{T_+ + T_- + 2\tau} \right\rangle} \tag{17}$$

Problem 4: Flocking (Sethna 2.20)

(a)

The boid captures the general shape and trend of the starling murmuration. However, it fails to capture the specific behaviors of the murmuration.



```
Soft sphere      100.00
Flock strength   0.20
Noise strength   0.20
Speed            0.00
Damping          0.20
Box size         40.00
Delta t          0.10
Frameskip        2.00
Fraction Red     1.00
Color Scale      0.01
☑x-periodic -- ☑y-periodic -- ☐Show forces
☑Circular ICs -- ☐Show Graphs
Particle count   40   Change
Pause  Restart
```

```
Soft sphere      100.00
Flock strength   0.20
Noise strength   0.20
Speed            0.00
Damping          0.20
Box size         40.00
Delta t          0.10
Frameskip        2.00
Fraction Red     1.00
Color Scale      0.01
☑x-periodic -- ☑y-periodic -- ☐Show forces
☑Circular ICs -- ☐Show Graphs
Particle count   40   Change
Pause  Restart
```

```
Soft sphere      100.00
Flock strength   0.20
Noise strength   0.20
Speed            0.00
Damping          0.20
Box size         40.00
Delta t          0.10
Frameskip        2.00
Fraction Red     1.00
Color Scale      0.01
☑x-periodic -- ☑y-periodic -- ☐Show forces
☑Circular ICs -- ☐Show Graphs
Particle count   40   Change
Pause  Restart
```

Figure 1: Three trials of mosh pit simulator.

(b)

Surprisingly, the flocks always end up going in a single direction! However, the directions are different. I included some screenshots of the animation with the rough direction that the flocks end up going.

The slow particles are the ones that are located far away from other particles. The slow particles are almost stationary, until they are hit by fast-moving particles. Slow particles abruptly start moving fast, whereas fast particles are moving in a more continuous manner.

As I increase the noise level, the particles gradually start moving individually and not collectively. There is no certain point where the particles start moving individually. However, I would say that the particles move collectively until noise level ≈ 1.4, and from noise level ≈ 1.6, the particles move individually.

(c)

The forces on θ_j is given by:

$$F = -\frac{\partial \mathcal{H}}{\partial \theta_j}\text{ , where } \mathcal{H} = \sum_i \frac{1}{2}K(\theta_i - \theta_{i-1})^2 \tag{18}$$

In a given situation where a physicist has two neighboring physicists in a one-dimensional lattice, we have $i = j \pm 1$. Therefore, the expression in Eq. 18 becomes:

$$\begin{aligned} F &= -\frac{\partial \mathcal{H}}{\partial \theta_j} = -\frac{\partial}{\partial \theta_j}\left[\sum_i \frac{1}{2}K(\theta_i - \theta_{i-1})^2\right] = -\frac{1}{2}K\frac{\partial}{\partial \theta_j}\left[(\theta_j - \theta_{j-1})^2 + (\theta_{j+1} - \theta_j)^2\right] \\ &= -\frac{1}{2}K\frac{\partial}{\partial \theta_j}\left[2\theta_j^2 - 2\theta_j(\theta_{j+1} + \theta_{j-1}) + \theta_{j+1}^2 + \theta_{j-1}^2\right] = -\frac{1}{2}K\left[4\theta_j - 2(\theta_{j+1} + \theta_{j-1})\right] \\ &= -K(2\theta_j - \theta_{j+1} - \theta_{j-1}) \end{aligned} \tag{19}$$

Energy is minimum when $\frac{\partial \mathcal{H}}{\partial \theta_j} = 0$, i.e., when $F = 0$. Let's plug in $\theta_j = \frac{\theta_{j+1} + \theta_{j-1}}{2}$ and see if it returns 0.

$$F = -K(2\theta_j - \theta_{j+1} - \theta_{j-1}) = -K\left(2 \times \frac{\theta_{j+1} + \theta_{j-1}}{2} - \theta_{j+1} - \theta_{j-1}\right) = 0 \tag{20}$$

Therefore, energy is indeed at minimum when θ_j points along the average angle of its two near neighbors.

We have the following expression: $\frac{1}{2}K\delta^2 = \frac{1}{2}k_BT$. Rearranging it gives:

$$\langle \delta_j^2 \rangle = \frac{k_BT}{K} \tag{21}$$

(d)

Continuing from Eq. 21, we have:

$$\boxed{\sqrt{\langle \delta^2 \rangle} = \sqrt{\frac{k_BT}{K}}} \tag{22}$$

Since K has unit of energy per radian squared, we need to convert 1 degree into radians: $1^\circ = \frac{\pi}{180}$ rad. Now, rearranging Eq. 22 and plugging in values for the angle, we get:

$$\left(\frac{\pi}{180}\right)^{\circ 2} = \frac{k_BT}{K} \Rightarrow \boxed{T = \frac{K}{k_B}\left(\frac{\pi}{180}\right)^2} \tag{23}$$

$\theta_n - \theta_0$ is the difference in angle between nth physicist and 0th physicist. That is, $(\theta_n - \theta_0)^2 = n(\theta_j - \theta_{j-1})^2 = n\delta^2$. Now, we have:

$$\sqrt{\langle(\theta_n - \theta_0)^2\rangle} = \sqrt{\langle n(\theta_j - \theta_{j-1})^2\rangle} = \sqrt{n}\sqrt{\langle \delta^2 \rangle} = \sqrt{n}(1^\circ) = 180^\circ \Rightarrow \boxed{n = 180^2 = 32400} \tag{24}$$

PY541_PS2 (final)

September 28, 2022

1 PY541 Problem Set 2

1.1 Raymond Kil

1.1.1 September 29, 2022

[1]:
```
%pylab inline
```

```
Populating the interactive namespace from numpy and matplotlib
```

1.2 Problem 1: Generating Random Walks (Sethna 2.5)

1.2.1 Part (a)

Let's define a function that generates a N step random walk in d-dimensions (I am building a function for only $d = 1, 2$). Each step increases or decreases by 1/2 in both x or y coordinate. Thus, each step will have a DIFFERENT STEP SIZE. Maximum step size will be the step where magnitude of x and y coordinates are both 1/2.

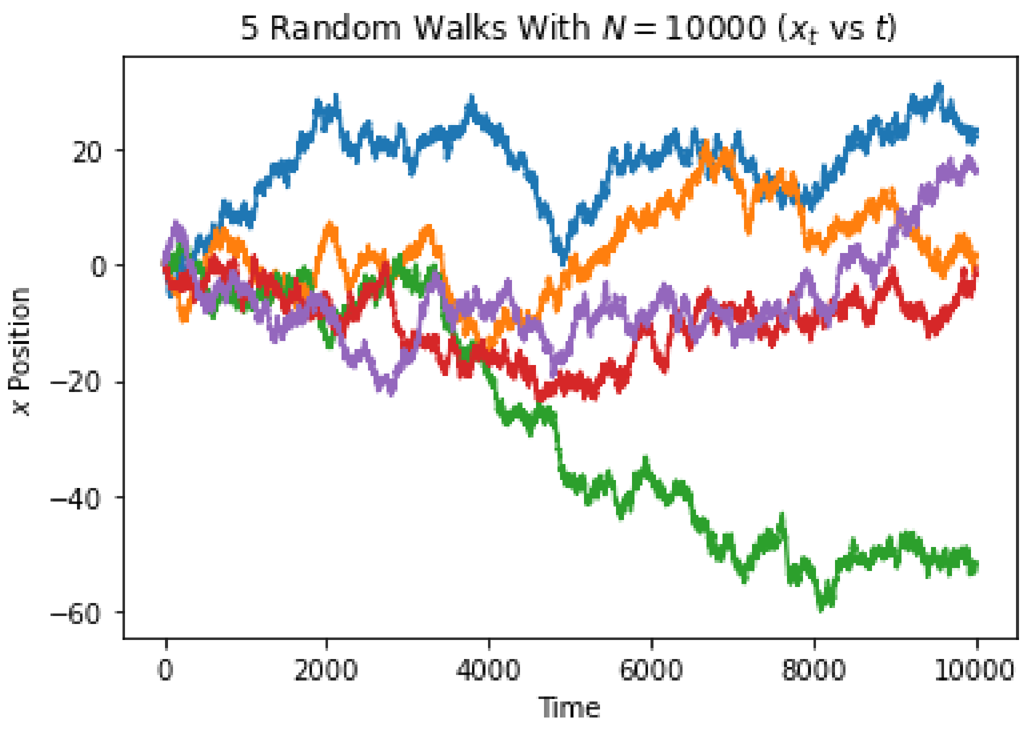
[2]:
```
# The function returns x,y, and t. If d=1, y=zero array.

def NstepRandomWalk(N,d):
    t = arange(N)
    x = zeros(N)
    y = zeros(N)

    for i in range(1,N):
        x[i] = x[i - 1] + uniform(-0.5,0.5)
        if d==2:
            y[i] = y[i - 1] + uniform(-0.5,0.5)

    return x,y,t
```

[3]:
```
# Here, I am generating 1D random walks.

N = 10000
d = 1
nWalks = 5
walks = [0]*nWalks
```

```
times = [0]*nWalks

for walk in range(nWalks):
    walks[walk] = NstepRandomWalk(N,d)[0]
    times[walk] = NstepRandomWalk(N,d)[2]
```

[4]:
```
# Let's plot 1D random walks as a function of time.

figure()
for walk in range(nWalks):
    plot(times[walk],walks[walk])

title(r"{0} Random Walks With $N = {1}$ ($x_t$ vs $t$)".format(nWalks,N))
xlabel(r"Time")
ylabel(r"$x$ Position")
```

[4]:
```
Text(0, 0.5, '$x$ Position')
```

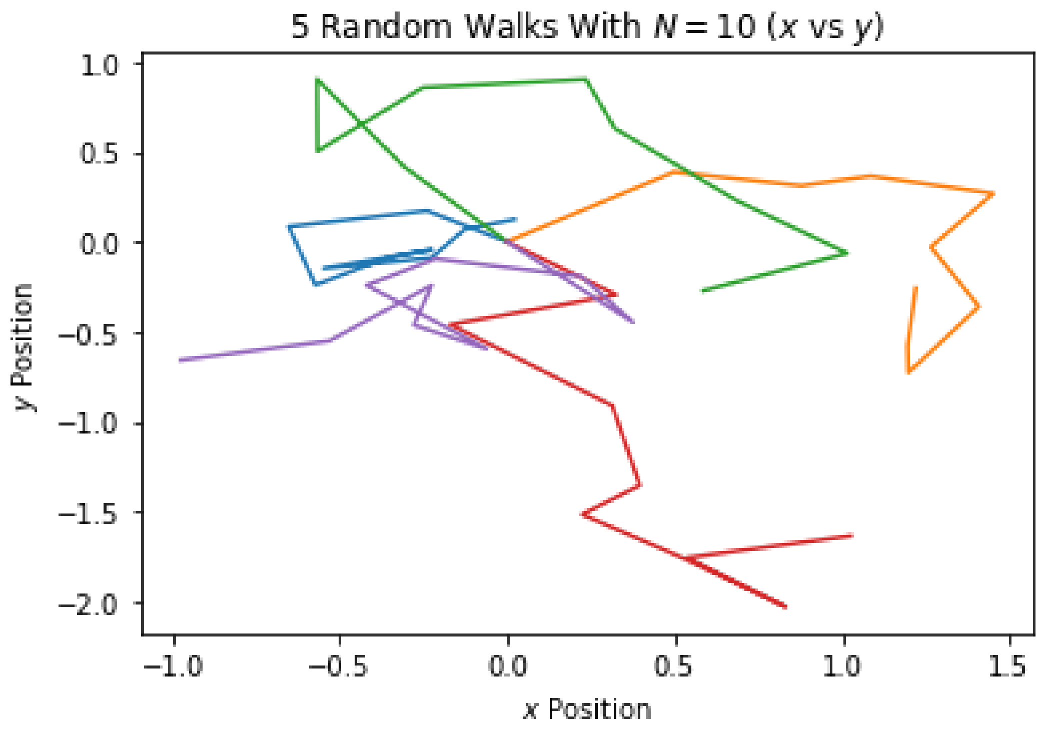


[5]:
```
# Here I am generating 2D random walks.
# I appended S,M,L (small, medium, large) to the variables that correspond to N␣
↪= 10, 1000, and 100000, respectively.

N_S = 10
```

```
N_M = 1000
N_L = 100000

d = 2
nWalks = 5

walksX_S = [0]*nWalks
walksY_S = [0]*nWalks

walksX_M = [0]*nWalks
walksY_M = [0]*nWalks

walksX_L = [0]*nWalks
walksY_L = [0]*nWalks

for walk in range(nWalks):
    walksX_S[walk], walksY_S[walk] = NstepRandomWalk(N_S,d)[:2]
    walksX_M[walk], walksY_M[walk] = NstepRandomWalk(N_M,d)[:2]
    walksX_L[walk], walksY_L[walk] = NstepRandomWalk(N_L,d)[:2]
```

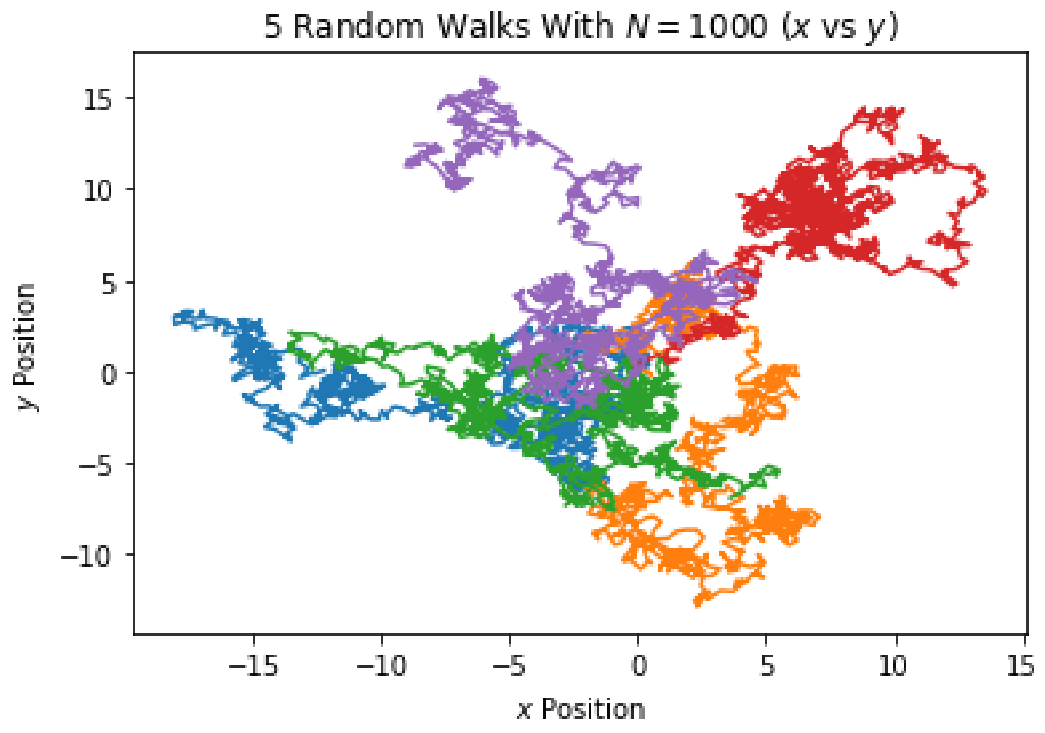
[6]:
```
# Now let's plot 2D random walks in position space.

figure()
for walk in range(nWalks):
    plot(walksX_S[walk],walksY_S[walk])

title(r"{0} Random Walks With $N = {1}$ ($x$ vs $y$)".format(nWalks,N_S))
xlabel(r"$x$ Position")
ylabel(r"$y$ Position")
```

[6]:
```
Text(0, 0.5, '$y$ Position')
```



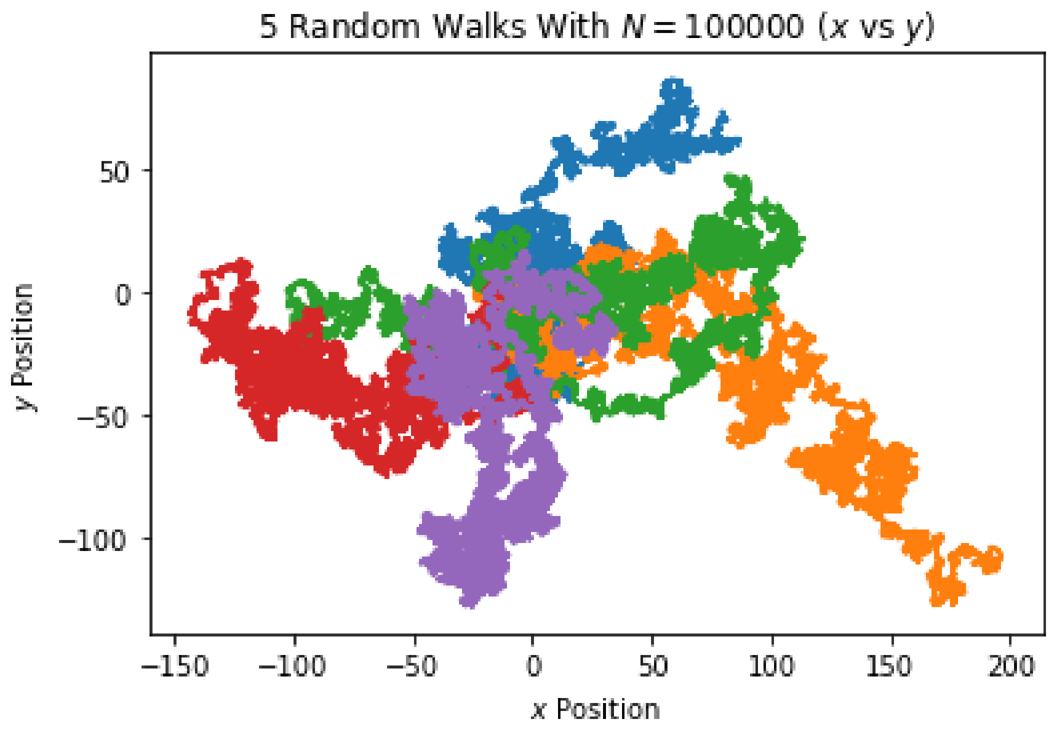
```
[7]: figure()
     for walk in range(nWalks):
         plot(walksX_M[walk],walksY_M[walk])

     title(r"{0} Random Walks With $N = {1}$ ($x$ vs $y$)".format(nWalks,N_M))
     xlabel(r"$x$ Position")
     ylabel(r"$y$ Position")
```

```
[7]: Text(0, 0.5, '$y$ Position')
```



```
[8]: figure()
     for walk in range(nWalks):
         plot(walksX_L[walk],walksY_L[walk])

     title(r"{0} Random Walks With $N = {1}$ ($x$ vs $y$)".format(nWalks,N_L))
     xlabel(r"$x$ Position")
     ylabel(r"$y$ Position")
```

```
[8]: Text(0, 0.5, '$y$ Position')
```

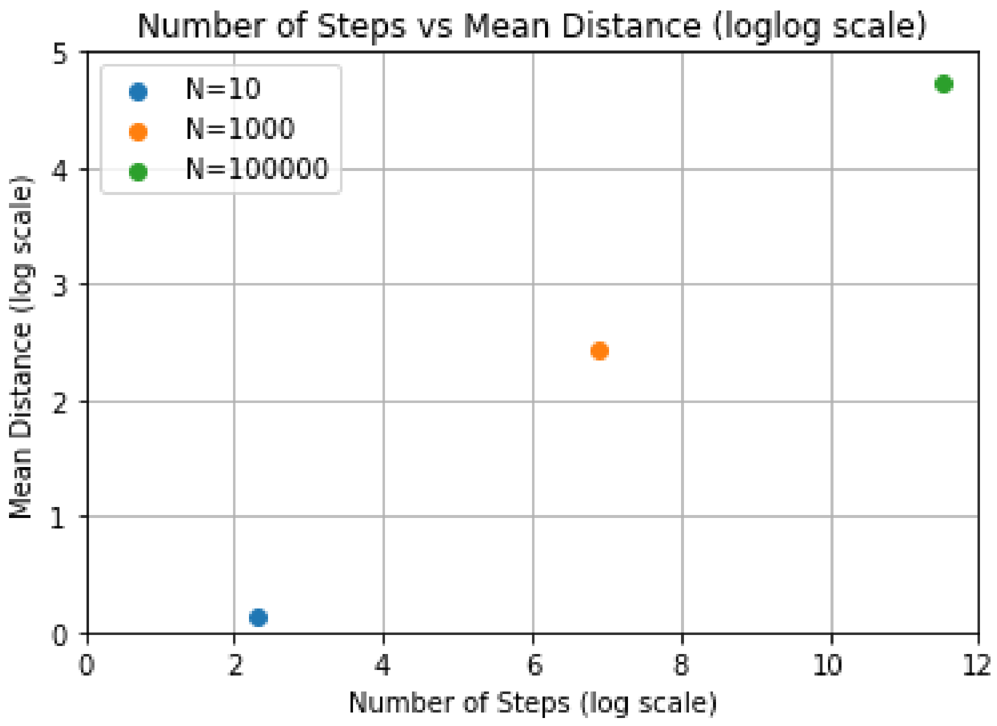


Let's now see if multiplying the number of steps by 100 roughly increases the net distance by ten. In theory, the RMS distance of random walk of N steps of step size l is given by $\sqrt{\langle d^2 \rangle} = l\sqrt{N}$. I will be looking at the endpoints of an ensemble of random walks with $N = 10, 1000, 100000$ steps and calculate the distances of the endpoints from the origin. I will make use of the "endpoint" function defined in part (b).

[9]:
```
# Below is a function that takes a number of steps (N) and dimension (d), and␣
 ↪returns the net distance traveled.

def distance(N,d):
    endX,endY = endpoint(N,d)
    dist = sqrt(endX**2+endY**2)
    return dist
```

[12]:
```
nWalks = 100
distances_S = [0]*nWalks
distances_M = [0]*nWalks
distances_L = [0]*nWalks

for i in range(nWalks):
    distances_S[i] = distance(10,2)
    distances_M[i] = distance(1000,2)
    distances_L[i] = distance(100000,2)
```

[13]:
```
figure()
scatter(log(10),log(mean(distances_S)))
scatter(log(1000),log(mean(distances_M)))
scatter(log(100000),log(mean(distances_L)))
title("Number of Steps vs Mean Distance (loglog scale)")
xlabel("Number of Steps (log scale)")
ylabel("Mean Distance (log scale)")
xlim(0,12)
ylim(0,5)
legend(["N=10","N=1000","N=100000"])
grid()
```



Thus, we can see that multiplying the number of steps by one hundred roughly increase the net distance by ten.

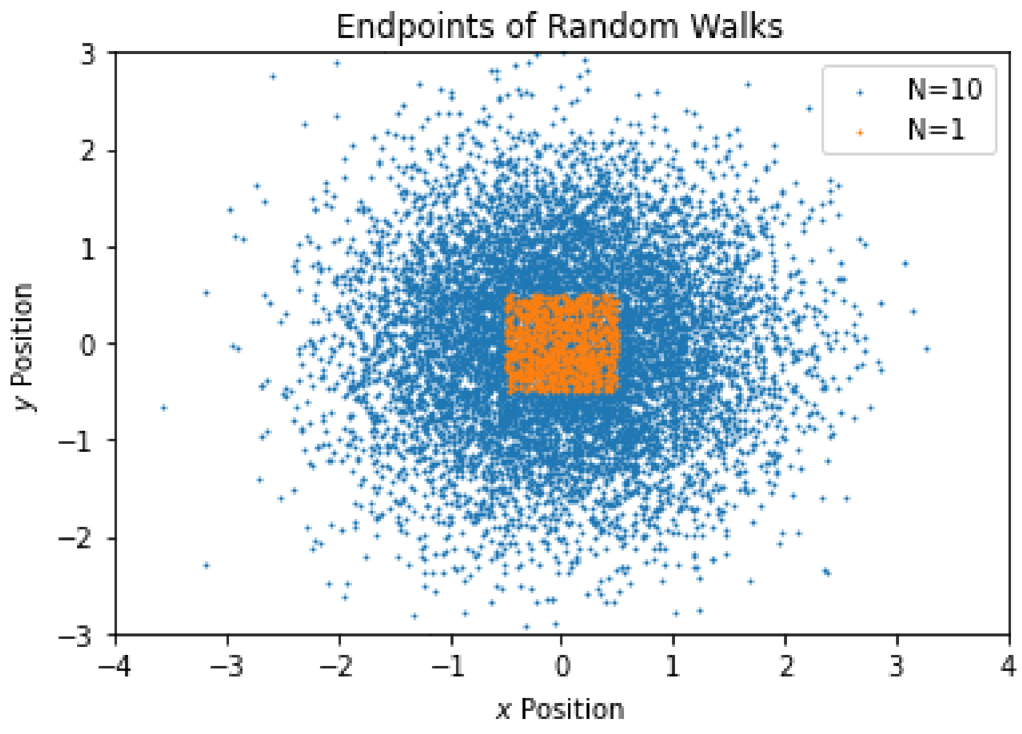
1.2.2 Part (b)

[14]:
```
# Let's define a function that returns the location in d-dimension after N␣
↪steps of random walk.
# If d=1, then endY=0.

def endpoint(N,d):
    x,y = NstepRandomWalk(N+1,d)[:2]
    endX = x[-1]
    endY = y[-1]
    return endX, endY
```

[15]:
```
# Let's generate 10000 2D random walks of step 1.
# As above, I appended S and L (small, large) to denote N=1 walks and N=10␣
↪walks, respectively.

nWalks_S = 1000
N_S = 1
d_S = 2

nWalks_L = 10000
N_L = 10
d_L = 2

endXs_S = zeros(nWalks_S)
endYs_S = zeros(nWalks_S)

endXs_L = zeros(nWalks_L)
endYs_L = zeros(nWalks_L)

for walk in range(nWalks_S):
    endXs_S[walk], endYs_S[walk] = endpoint(N_S,d)[:2]
for walk in range(nWalks_L):
    endXs_L[walk], endYs_L[walk] = endpoint(N_L,d)[:2]
```

[16]:
```
# Let's plot the endpoints.

figure()
scatter(endXs_L, endYs_L, s=0.8)
scatter(endXs_S, endYs_S, s=0.8)
title(r"Endpoints of Random Walks")
xlabel(r"$x$ Position")
ylabel(r"$y$ Position")
xlim(-4,4)
ylim(-3,3)
legend(["N=10","N=1"])
```

[16]:
```
<matplotlib.legend.Legend at 0x7f42265ca410>
```



Appreciate the emergent spherical symmetry! Notice that the longer random walks are distributed in a circularly symmetric pattern, even though the single step random walk has a square probability distribution.

1.2.3 Part (c)

[17]:
```
# Generating 10000 1D random walks with N = 1,2,3, and 5 steps.

W = 10000

endpoints1 = [0]*W
endpoints2 = [0]*W
endpoints3 = [0]*W
endpoints5 = [0]*W

for i in range(W):
    endpoints1[i] = endpoint(1,1)[0]
    endpoints2[i] = endpoint(2,1)[0]
    endpoints3[i] = endpoint(3,1)[0]
    endpoints5[i] = endpoint(5,1)[0]
```

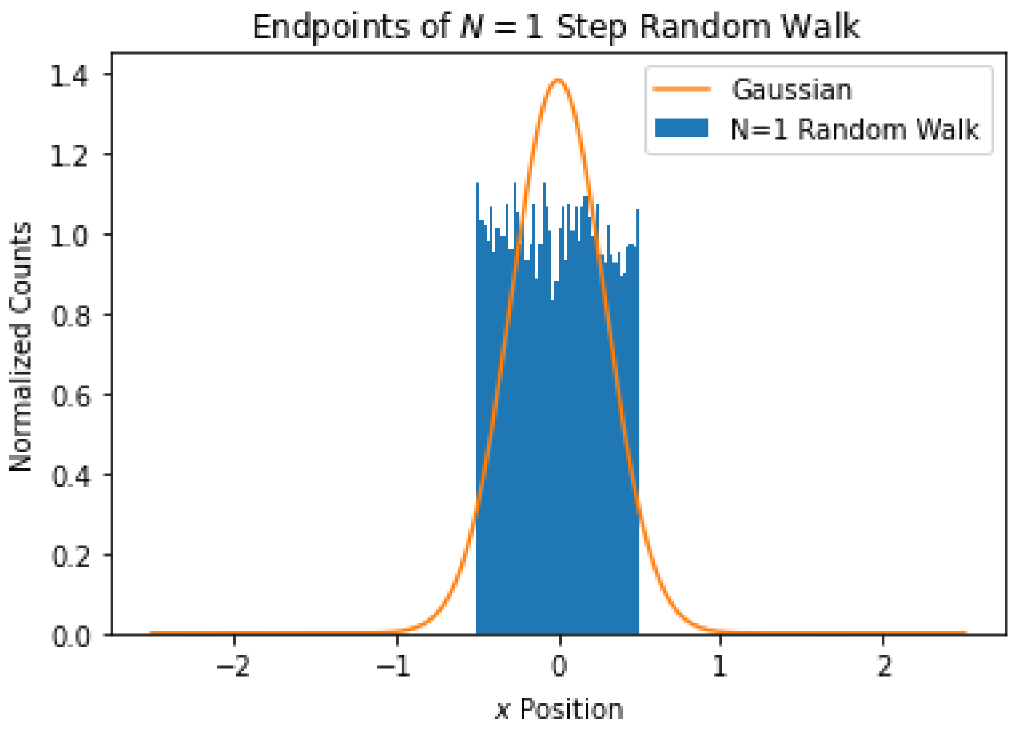
[18]:
```
# Below is a function for Gaussian distribution.
```

```
def Gaussian(x,a,N):
    sigma = a*sqrt(N)
    rho = 1/(sqrt(2*pi)*sigma)*exp(-x**2/(2*sigma**2))
    return rho
```

[19]:
```
# Let's generate Gaussians for N = 1,2,3, and 5.

a = 1/sqrt(12)
x = linspace(-2.5,2.5,300)

Gaussian1 = Gaussian(x,a,1)
Gaussian2 = Gaussian(x,a,2)
Gaussian3 = Gaussian(x,a,3)
Gaussian5 = Gaussian(x,a,5)
```

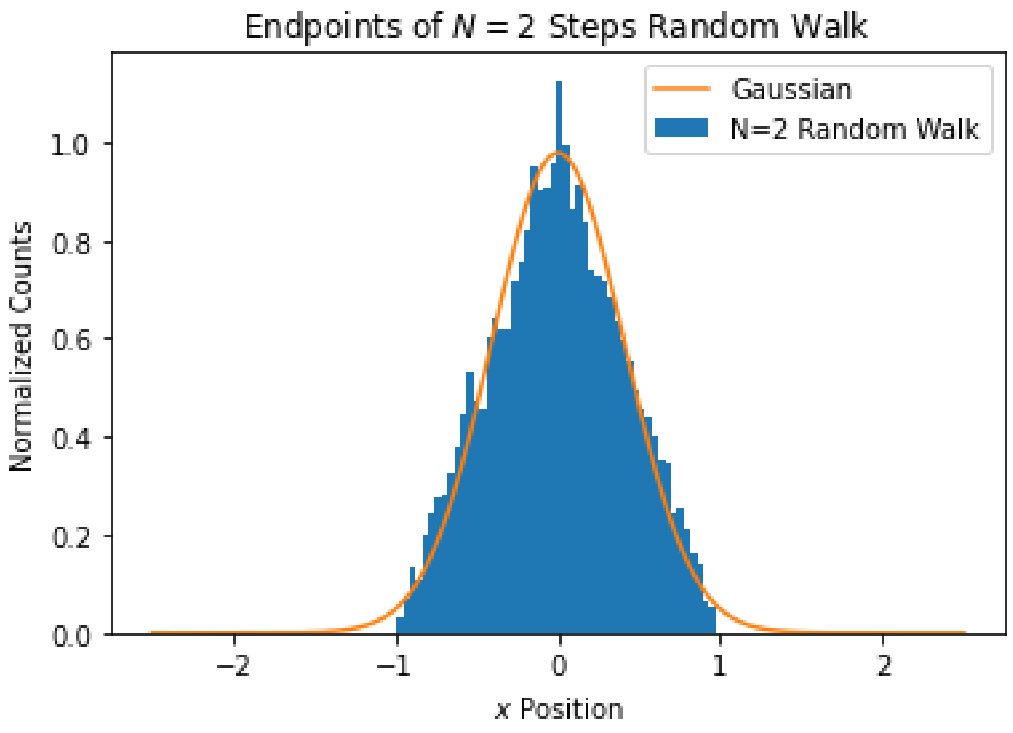
[20]:
```
# Plotting the histogram of endpoints with normalized counts, along with␣
 ↪Gaussian approximation.

figure()
hist(endpoints1, bins=50,density=True)
plot(x,Gaussian1)
title(r"Endpoints of $N=1$ Step Random Walk")
xlabel(r"$x$ Position")
ylabel("Normalized Counts")
legend(["Gaussian","N=1 Random Walk"])
```

[20]:
```
<matplotlib.legend.Legend at 0x7f4224553390>
```



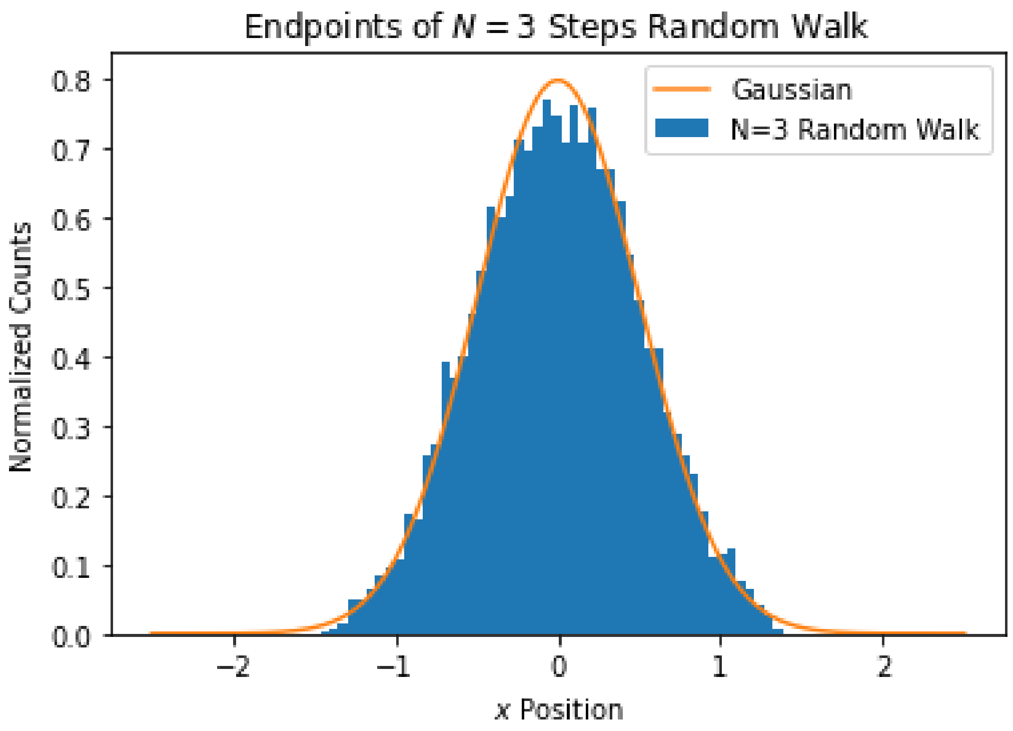
```
[21]: figure()
      hist(endpoints2, bins=50,density=True)
      plot(x,Gaussian2)
      title(r"Endpoints of $N=2$ Steps Random Walk")
      xlabel(r"$x$ Position")
      ylabel("Normalized Counts")
      legend(["Gaussian","N=2 Random Walk"])
```

```
[21]: <matplotlib.legend.Legend at 0x7f422445a7d0>
```



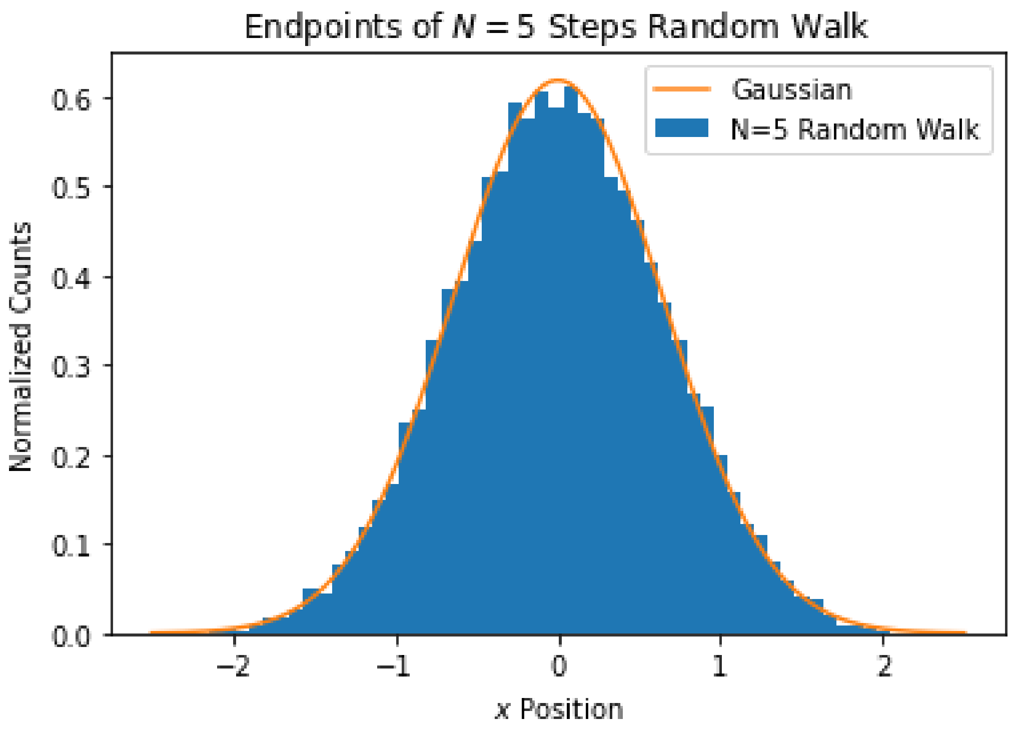
```
[22]: figure()
      hist(endpoints3, bins=50,density=True)
      plot(x,Gaussian3)
      title(r"Endpoints of $N=3$ Steps Random Walk")
      xlabel(r"$x$ Position")
      ylabel("Normalized Counts")
      legend(["Gaussian","N=3 Random Walk"])
```

```
[22]: <matplotlib.legend.Legend at 0x7f4224355fd0>
```



[23]:
```
figure()
hist(endpoints5, bins=50,density=True)
plot(x,Gaussian5)
title(r"Endpoints of $N=5$ Steps Random Walk")
xlabel(r"$x$ Position")
ylabel("Normalized Counts")
legend(["Gaussian","N=5 Random Walk"])
```

[23]:
```
<matplotlib.legend.Legend at 0x7f4224217c90>
```



We can see that the Gaussian becomes a fairly good approximation for $N = 2$, but from $N = 3$, Gaussian becomes very good approximation to the random walk.

1.3 Problem 2: Stocks, Volatility, and Diversification (Sethna 2.11)

```
[24]: t, SP = array(loadtxt("SandPConstantDollars.dat").transpose())
```

Some notes about what I loaded above.

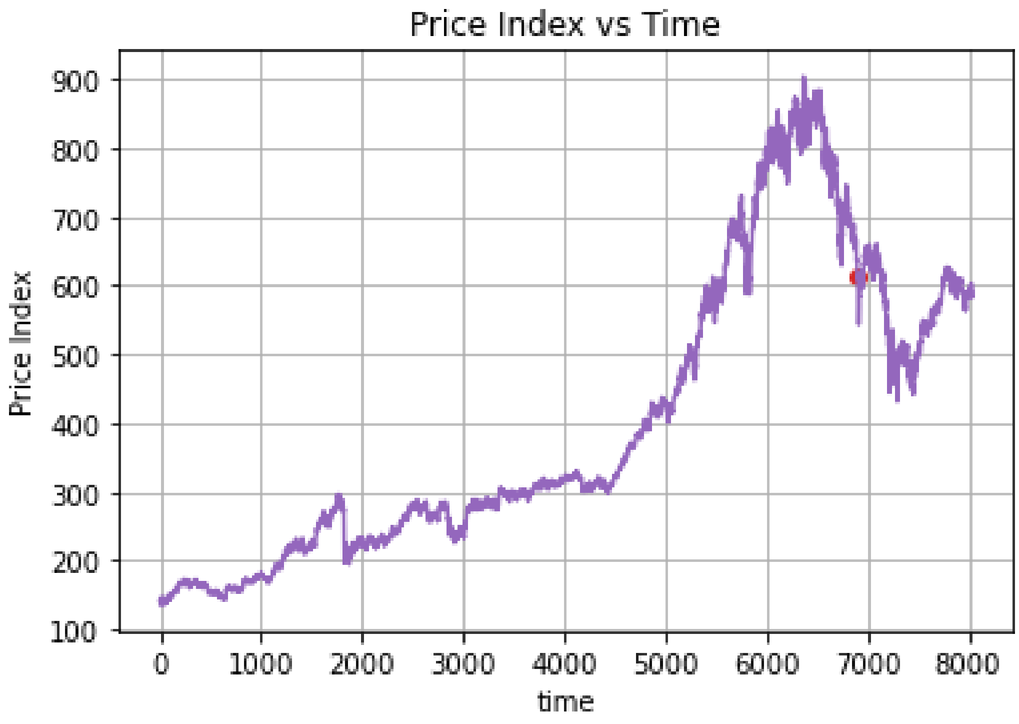
- Each line represents a weekday (Monday - Friday)
- t = time. It has the format $5, 6, 7, 8, 9, 12, 13, 14, 15, 16 \cdots$. Notice that number is consecutive 5 times, then skips two numbers. This corresponds to five weekdays and two weekend days.
- SP = Standard and Poor's index for the corresponding day.

1.3.1 Part (a)

Let's plot the "price index" vs "time".

```
[25]: figure()
      scatter([6903],[614],color="tab:red")
      plot(t,SP,color="tab:purple")
      title("Price Index vs Time")
      xlabel("time")
      ylabel("Price Index")
```

```
grid()
```



I marked the day the World Trade Center was attacked (day 6903) by a red dot. We can see that the price index has been falling before the day of attack. Therefore, most of the fluctuation seem to be due to internal reasons, not because of the external event.

1.3.2 Part (b)

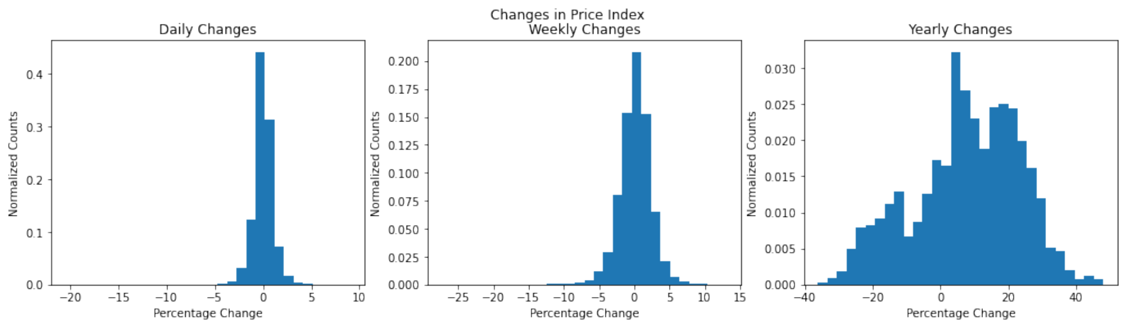
I am writing a function P_l that finds all pairs of time points separated by time interval $\Delta t = l$ and returns a percent changes over that time interval.

[26]:
```
# t= array of time, SP = array of SP index, l = integer that represents the␣
↪time interval
# inputs for l: 1=daily changes, 5=weekly changes, 252=yearly changes

def Pl(t,SP,lag):
    ratios = SP[lag:]/SP[:-lag]
    P = 100*(ratios-1)
    return P
```

[27]:
```
daily_changes  = Pl(t,SP,1)
weekly_changes = Pl(t,SP,5)
yearly_changes = Pl(t,SP,252)
```

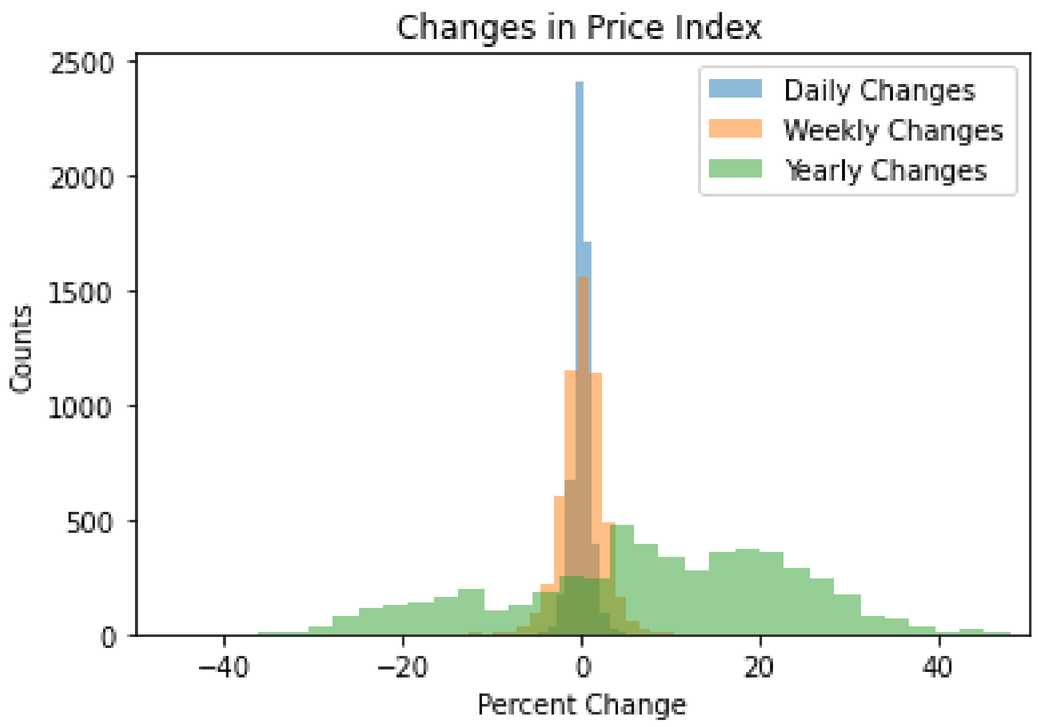
[28]:
```python
# Plotting the histograms for percent changes

fig, (hist1,hist2,hist3) = subplots(1, 3)
suptitle("Changes in Price Index")

fig.set_figheight(4)
fig.set_figwidth(17)

hist1.hist(daily_changes,bins=30,density=True)
hist1.set_title("Daily Changes")
hist1.set_xlabel("Percentage Change")
hist1.set_ylabel("Normalized Counts")

hist2.hist(weekly_changes,bins=30,density=True)
hist2.set_title("Weekly Changes")
hist2.set_xlabel("Percentage Change")
hist2.set_ylabel("Normalized Counts")

hist3.hist(yearly_changes,bins=30,density=True)
hist3.set_title("Yearly Changes")
hist3.set_xlabel("Percentage Change")
hist3.set_ylabel("Normalized Counts")
```

[28]:
```
Text(0, 0.5, 'Normalized Counts')
```



[29]:
```python
# Let's draw the historgrams in one figure for comparison.
figure()
hist(daily_changes,bins=30,alpha=0.5)
hist(weekly_changes,bins=30,alpha=0.5)
hist(yearly_changes,bins=30,alpha=0.5)
title("Changes in Price Index")
xlabel("Percent Change")
ylabel("Counts")
xlim(-50,50)
legend(["Daily Changes","Weekly Changes","Yearly Changes"])
```

[29]: <matplotlib.legend.Legend at 0x7f421fefa210>

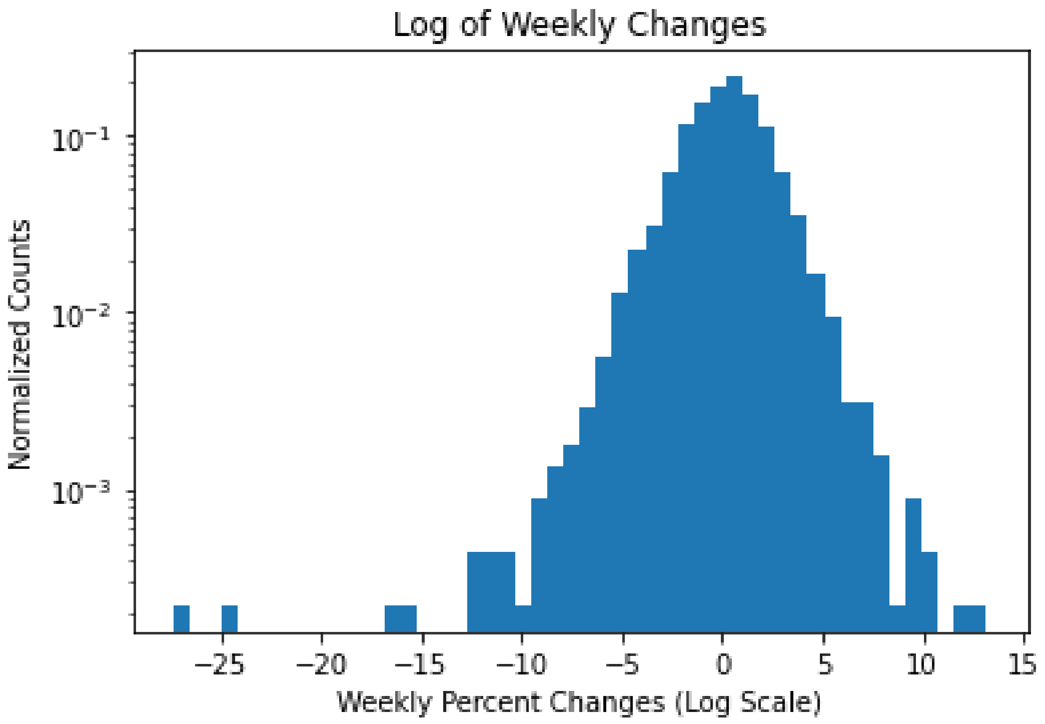


We can observe that the histogram of yearly percent change has mean percentage growth larger than a tiny fraction of the fluctuations. Therefore, the reasonable time for me to stay invested is in the scale of years. Yearly changes consider many overlapping data points. For example, in daily changes, there are no overlapping days when calculating the percent change (except for immediate neigbors, of course). On the other hand, pretty much the same data points are used to calculate the percent change from, say, t_1 to $t_1 + 252$ and from $t_2 = t_1 + 1$ to $t_2 + 252$. Therefore, the percent changes are not independent from each other, which makes the distribution not Gaussian.

1.3.3 Part (c)

[30]:
```
# I am plotting the weekly percentage changes in log scale.

figure()
hist(weekly_changes,bins=50,log=True,density=True)
title("Log of Weekly Changes")
ylabel("Normalized Counts")
xlabel("Weekly Percent Changes (Log Scale)")
```

[30]: Text(0.5, 0, 'Weekly Percent Changes (Log Scale)')



We can see that there are more counts on the sides for the log plot. That is, there are more large percentage changes than expected from a Gaussian distribution.

1.3.4 Part (d)

[31]:
```
# Function that takes in an array of percent changes and returns the volatility.

def volatility(Pl):
    v = sqrt(mean((Pl-mean(Pl))**2))
    return v
```

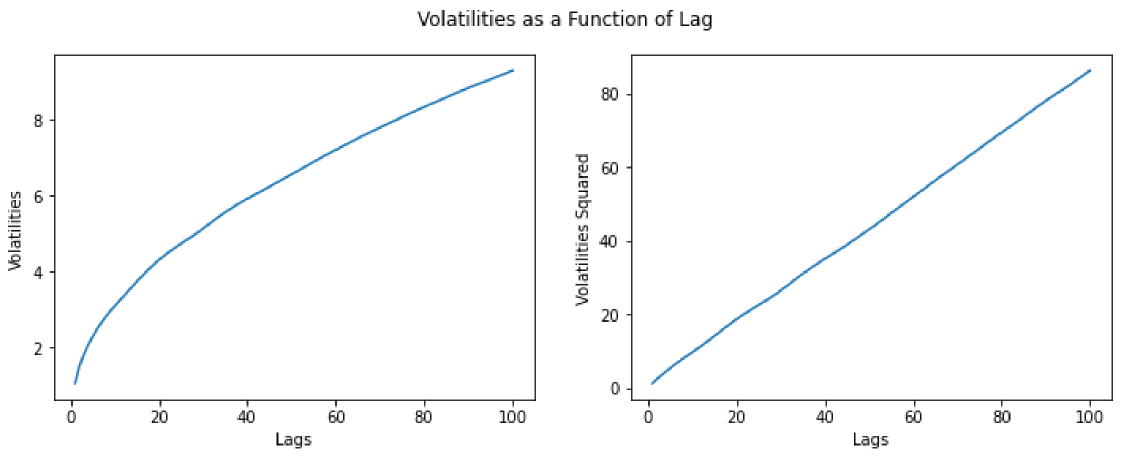
[32]:
```
daily_volatility = volatility(daily_changes)
weekly_volatility = volatility(weekly_changes)
yearly_volatility = volatility(yearly_changes)

print("Daily volativity: {0}".format(daily_volatility))
print("Weekly volativity: {0}".format(weekly_volatility))
print("Yearly volativity: {0}".format(yearly_volatility))
```

```
Daily volativity: 1.0606483448236002
Weekly volativity: 2.291749421353568
Yearly volativity: 15.723439026075111
```

```
[33]: lags = arange(1,101)
      lag_changes = [0]*len(lags) # Each array element has array of percent changes␣
       ↪in "lag" days. 1 <= lag <= 100

      for lag in lags:
          lag_changes[lag-1] = Pl(t,SP,lag)

      volatilities = [0]*len(lag_changes)

      for vol in range(len(volatilities)):
          volatilities[vol] = volatility(lag_changes[vol])
```

```
[34]: # Plotting volatilities and volatilities squared as a function of lag.
      fig, (plot1,plot2) = subplots(1,2)
      fig.set_figheight(4)
      fig.set_figwidth(12)
      fig.suptitle("Volatilities as a Function of Lag")


      plot1.plot(lags,volatilities)
      plot2.plot(lags,(array(volatilities))**2)
      plot1.set_xlabel("Lags")
      plot2.set_xlabel("Lags")
      plot1.set_ylabel("Volatilities")
      plot2.set_ylabel("Volatilities Squared")
```

```
[34]: Text(0, 0.5, 'Volatilities Squared')
```



Volatility can be seen as a "root-mean-squared distance" in time space. Recalling that the behavior of random walk is that the RMS distance of a random walk increases like $\sqrt{N}$. In terms of volatility and lag, $v_l \propto \sqrt{lags}$. This is what is precisely shown in the graphs above. Therefore, the volatility indeed behaves as a random walk should.