ELEC2600 HW4 Question 6 Spring 2019

In this problem, we will look at simulating a random walk by a sum process.

Import useful libraries

In [1]:

```
import numpy as np
import scipy.stats
import matplotlib.pyplot as plt
import scipy.stats
```

The function below defines a random walk as a sum process. The initial position is zero and the steps are generated by an i.i.d. random process where each step is uniformly distributed over the set [-1,0,1]. In the returned array, each column corresponds to one walk and each row corresponds to one timestep.

In [2]:

```
def random_walk(nwalk,nstep):
    '''Generate nwalk random walks that last for nstep steps'''
    steps = np.random.randint(-1,2,size=[nstep,nwalk])
    steps[0] = np.zeros(nwalk)
    walk = np.cumsum(steps,axis=0)
    return walk
```

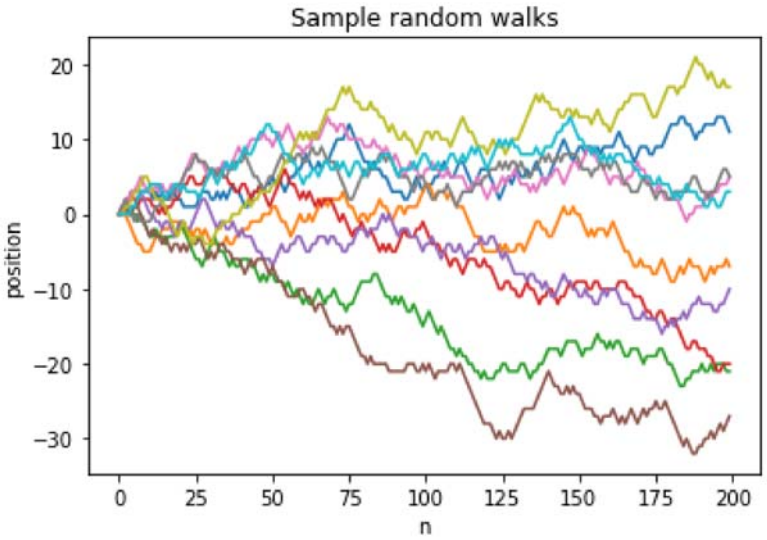
In the following, we generate and plot 10 random walks that go over 200 time steps (from 0 to 199).

In [3]:

```
# generate 10 sample random walks
nwalk = 10
nstep = 200
sample_walks = random_walk(nwalk,nstep)

# generate plot of random walks
n = range(0,nstep)
plt.plot(n,sample_walks)
plt.xlabel('n')
plt.ylabel('position')
plt.title('Sample random walks')

plt.show()
```



In the following, we generate many random walks.

In [4]:

```
# generate many random walks
nwalk = 2000
walk = random_walk(nwalk,nstep)
bins = range(-50,51,2)
```

Part (a)

Since the position at time n is obtained by taking the sum of all the steps from time 0 up to time n, by the central limit theorem, the position is given by a Gaussian random variable. Compute the theoretical mean (tmean) and standard deviation (tstd) of the position at time n1 = 20, and compare the theoretically estimated distribution with the empirical histogram. Note that the initial values provided below are incorrect dummy variables provided to make sure the code runs.

In [8]:

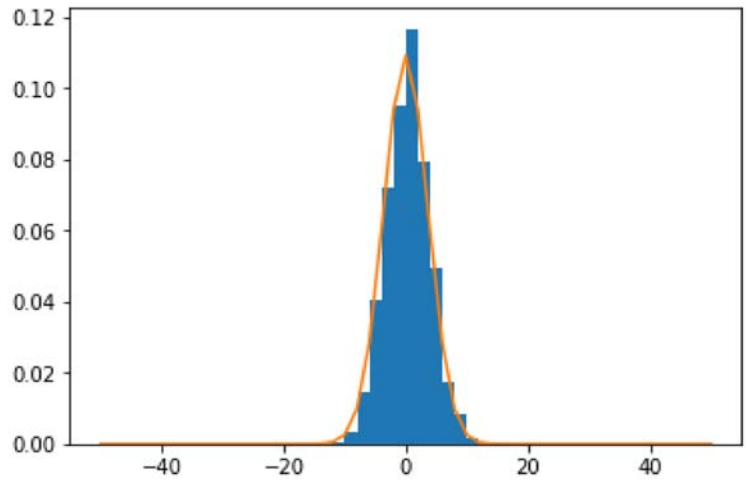
```
# pick out samples from the random walk at a particular timestep n1
n1 = 20
samples = walk[n1]

# YOUR WORK HERE
# put your calculations for the mean and variance of the distribution herr
tmean = 0 * n1
var_single = 1 * 1/3 + 1 * 1/3 - 0
tstd = np.sqrt(var_single * n1)

# compute the empirical mean and standard deviation
emean = np.mean(samples)
estd = np.std(samples)

# generate plot of empirical histogram and the theoretical prediction
fig, ax = plt.subplots()
ax.hist(samples,bins=bins,density=True)
f = scipy.stats.norm.pdf(bins,loc=tmean,scale=tstd)
ax.plot(bins,f)

plt.show()
print('Mean: '+str(emean)+' (empirical), '+str(tmean)+' (theoretical)')
print('Standard deviation: '+str(estd)+' (empirical), '+str(tstd)+' (theoretical)')
```




```
Mean: -0.1795 (empirical), 0 (theoretical)
Standard deviation: 3.627434320563227 (empirical), 3.651483716701107 (theoretical)
```

Part (b)

In a random process, the mean and standard deviation (or variance) can change over time. Repeat Part (a) above for the time n1=199.

In [10]:

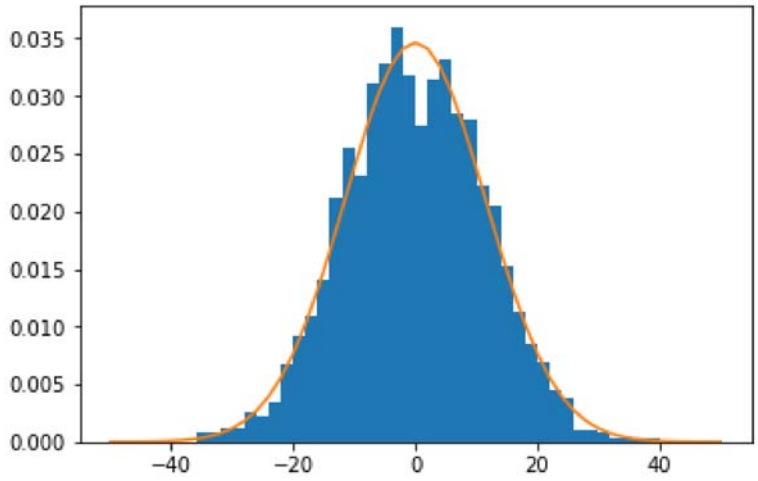
```
# pick out samples from the random walk at a particular timestep n1
n1 = 199
samples = walk[n1]

# YOUR WORK HERE
# put your calculations for the mean and variance of the distribution herr
tmean = 0 * n1
var_single = 1 * 1/3 + 1 * 1/3 - 0
tstd = np.sqrt(var_single * n1)

# compute the empirical mean and standard deviation
emean = np.mean(samples)
estd = np.std(samples)

# generate plot of empirical histogram and the theoretical prediction
fig, ax = plt.subplots()
ax.hist(samples,bins=bins,density=True)
f = scipy.stats.norm.pdf(bins,loc=tmean,scale=tstd)
ax.plot(bins,f)

plt.show()
print('Mean: '+str(emean)+' (empirical), '+str(tmean)+' (theoretical)')
print('Standard deviation: '+str(estd)+' (empirical), '+str(tstd)+' (theoretical)')
```




```
Mean: -0.5835 (empirical), 0 (theoretical)
Standard deviation: 11.402193988439242 (empirical), 11.51810169544733 (theoretica
l)
```

Part (c)

Describe the difference between the distributions at times 20 and 199. What do you expect to see for the distribution at time 100?

Your answer here

- the mean of the distributions remains the same, that is for the mean of a single step is zero
- but the standard deviation increases, std at time n should be std_n, which is shown below
  - $std_n = (\sum Var[X_i] + \sum Cov(X_i, X_j))^{1/2} = std_0 * n^{1/2}$
- at time 100, the mean should still be zero, the std will fall in between std of time step 20 and 199, which is shown below

In [11]:

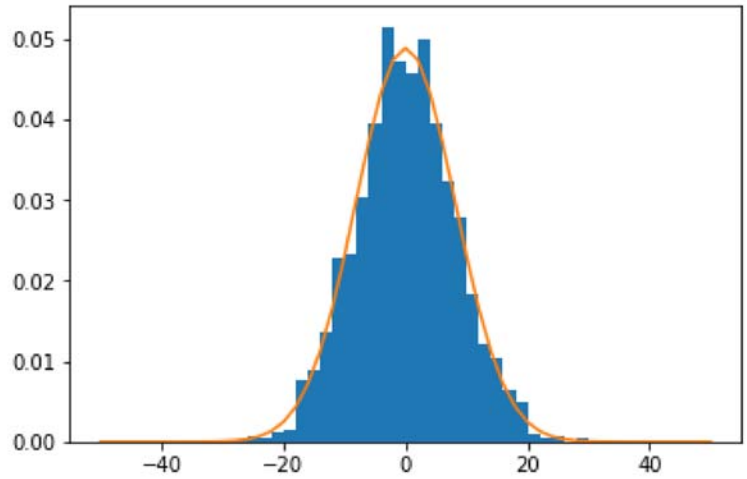
```
# pick out samples from the random walk at a particular timestep n1
n1 = 100
samples = walk[n1]

# YOUR WORK HERE
# put your calculations for the mean and variance of the distribution herr
tmean = 0 * n1
var_single = 1 * 1/3 + 1 * 1/3 - 0
tstd = np.sqrt(var_single * n1)

# compute the empirical mean and standard deviation
emean = np.mean(samples)
estd = np.std(samples)

# generate plot of empirical histogram and the theoretical prediction
fig, ax = plt.subplots()
ax.hist(samples,bins=bins,density=True)
f = scipy.stats.norm.pdf(bins,loc=tmean,scale=tstd)
ax.plot(bins,f)

plt.show()
print('Mean: '+str(emean)+' (empirical), '+str(tmean)+' (theoretical)')
print('Standard deviation: '+str(estd)+' (empirical), '+str(tstd)+' (theoretical)')
```




```
Mean: -0.4065 (empirical), 0 (theoretical)
Standard deviation: 8.17002189409551 (empirical), 8.164965809277259 (theoretical)
```

Part (d)

The correlation between values of the random process at different points in time differs according to the time samples chosen. Visualize this by plotting

- A 2D scatter plot of the values of walk at time 20 and 25
- A 2D scatter plot of the values of walk at time 20 and 199

The function plt.scatter(x,y) may be useful here. Set the limits of the axis in both plots using the axis() command to axlim defined below to enable easy comparison.

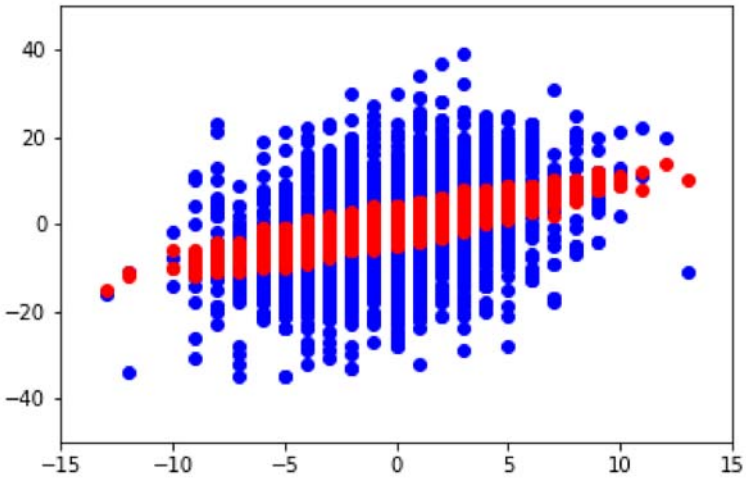
In [18]:

```
axlim = [-15,15,-50,50]

plt.scatter(walk[20], walk[199], color='blue')
plt.scatter(walk[20], walk[25], color='red')
plt.axis(axlim)
```

Out[18]:

```
[-15, 15, -50, 50]
```



Part (d)

Compute the theoretical value of the correlation coefficient between walk[20] and walk[25], as well the correlation coefficient between walk[20] and walk[199]. Compare this with the empirical estimate from the data. If you use the function np.corrcoeff(x,y) to estimate the correlation coefficent between vectors x and y, the correlation coefficient will be in the off diagonal entries of the 2D arrays returned.

In [28]:

```
def get_std(n_step):
    var_single = 1 * 1/3 + 1 * 1/3 - 0
    return np.sqrt(var_single * n_step)

cov_20_25 = 20 * (get_std(1) * get_std(1))
cov_20_199 = 20 * (get_std(1) * get_std(1))

corr_20_25 = cov_20_25 / (get_std(20) * get_std(25))
corr_20_199 = cov_20_199 / get_std(20) / get_std(199)

print("Theoretical estimate of corr is ", corr_20_25, "while the empirical estimate is", np.corr
coef(walk[20], walk[25])[0][1])
print("Theoretical estimate of corr is ", corr_20_199, "while the empirical estimate is", np.cor
rcoef(walk[20], walk[199])[0][1])
```

```
Theoretical estimate of corr is  0.8944271909999161 while the empirical estimate i
s 0.8860996222290938
Theoretical estimate of corr is  0.3170213124741207 while the empirical estimate i
s 0.3168770993607953
```

Part (e)

Compare the correlation coefficients calculated for times (20,25) and times (20,199). How are the differences in the correlation coefficients reflected in the differences in the scatter plots?

Your answer here:

- for larger correlation coefficients of X and Y, it is highly possible that (x, y) scatters around some line in the 2D plane, i.e. given a fixed x, y is constrained around the line, this is shown in the above plot where red points are densely distributed around the y = x line.
- and vise versa, (x, y) will be less consentrated around the y = x line if their correlation coefficient is lower. It is shown that in the above plot, given a fixed x, the corresponding y value is still quite flexible, resulting in a longer blue line(which is made up of blue points).

In [ ]:

```
```