Problem 1 Got expected values of Dn of 1/2 for all three dimensions. Present error decreases significantly from N=5000 to N=10000 runs.

Problem 2 Had an issue with getting tensorflow probability to work, fixed it by uninstalling tf-nighty, tfp-nightly and tfp, and then reinstalling tfp stable version. The issue was the the clash of versions. Got the program to run as expected.

Problem1-random_walkers

April 9, 2021

1 Random walks

Now we will look at Random walks in n dimensions. This will be the first Markov Chain Monte Carlo (MCMC) that we will utilize.

We will keep track of the paths of random walkers and use it to derive the conditions for diffusion in Brownian motion.

The "choice" and "cumsum" strategy here is adapted from here

[1]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

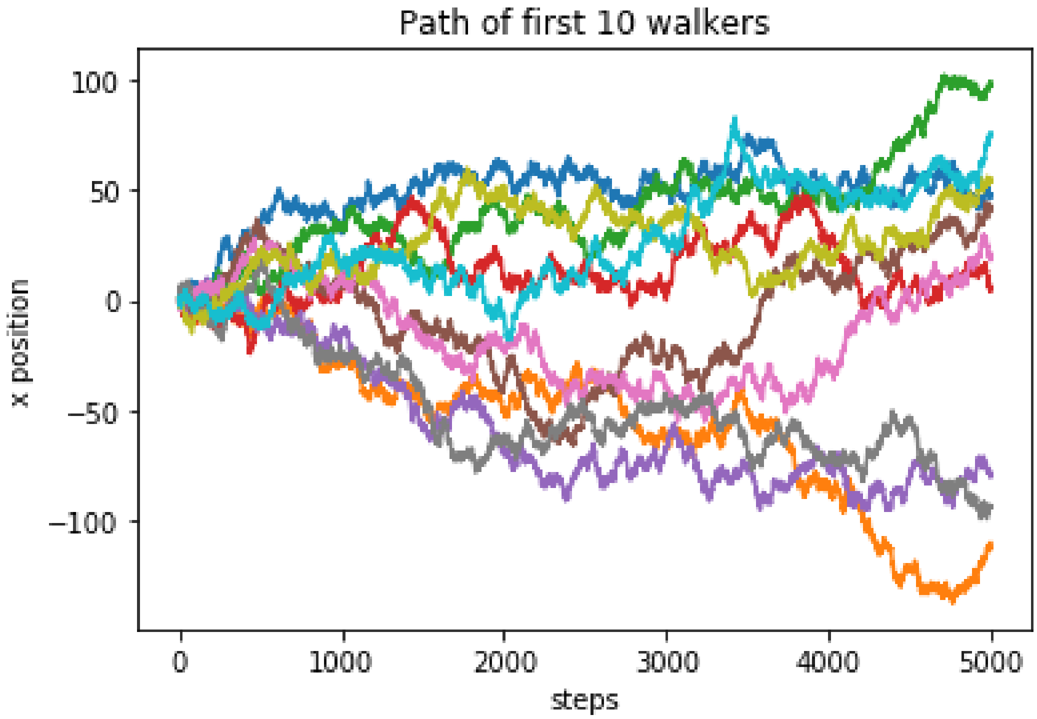
1.0.1 Run the walkers

[2]:
```python
dims = 3
n_walkers = 1000
n_steps = 5000
t = np.arange(n_steps)
# Walkers can go in + direction, - direction, or stay still
step_set = [-1, 0, 1]
# The shape is for "n_walkers" taking "n_steps" in "dims" dimensions.
# So, in 1d if there are 10 walkers making 100 steps each,
# it will be (10, 100, 1)
step_shape = (n_walkers,n_steps,dims)
# These are the steps at each stage
steps = np.random.choice(a=step_set, size=step_shape)
# Now we add up the steps for each walker to get the x positions
x = steps.cumsum(axis=1)
```

1.0.2 Plot the x position of the first 10 walkers

[3]:
```python
for i in range( min(10,n_walkers) ):
    plt.plot( x[i,:,0] )
plt.title("Path of first 10 walkers")
plt.xlabel("steps")
plt.ylabel("x position")
```

```
[3]: Text(0, 0.5, 'x position')
```



1.0.3 Accumulate statistics

Here, we now want to determine the relationship between diffusion and walks.

We know from lecture that after the nth step, each walker will have position

$$x_n = \sum_{i=1}^{n} s_i$$

where s_i is each walkers' step from the `steps` construct above. The average of s_i is zero because they are uniformly chosen from $(-1, 0, 1)$. However, the standard deviation for each walker is

$$\langle x_n^2 \rangle = \left\langle \sum_{i=1}^{n} \sum_{j=1}^{n} s_i s_j \right\rangle$$

$$\langle x_n^2 \rangle = \left\langle \sum_{i} s_i^2 \right\rangle + \left\langle \sum_{i} \sum_{j \neq i} s_i s_j \right\rangle$$

If there are m walkers each walking n steps, and the index k iterates over the walkers, then at each step n we have ensemble averages (in 1 dimension):

$$\langle x_n^4 \rangle = \sum_{k=1}^{m} \frac{x_{k,n}^4}{m}$$

$$\langle x_n^2 \rangle = \sum_{k=1}^{m} \frac{x_{k,n}^2}{m}$$

The overall diffusion width at the nth step, taking these ensemble averages, is therefore

$$\sigma_n^2 = \sqrt{\langle x_n^4 \rangle - \langle x_n^2 \rangle^2}$$

1.0.4 Homework assignment will go here:

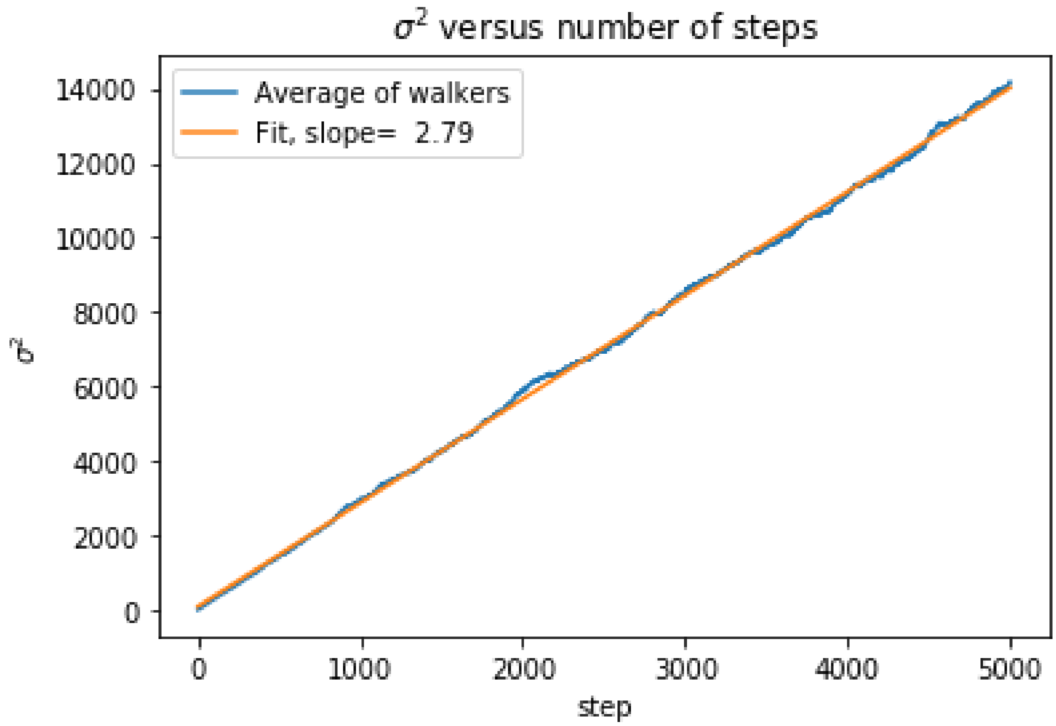
For 1d, 2d, and 3d:

- Calculate and plot σ^2 as a function of n.
- Compute a linear fit of σ^2 as a function of n, and also plot that.
- Compute the diffusion constant D in each of 1d,2d,3d

[4]:
```
# Now get the averages over the walkers
x2 = np.average( x**2, axis=0 )
x4 = np.average( x**4, axis=0 )
sigma2_nd = np.sqrt( x4 - x2**2 )
sigma2 = np.sum( sigma2_nd, axis=1 )
```

[5]:
```
plt.plot( sigma2,  label='Average of walkers' )
res = np.polyfit(t, sigma2,1 )
plt.plot( t, res[0]*t + res[1], label='Fit, slope=%6.2f' % res[0] )
plt.title(r"$\sigma^2$ versus number of steps")
plt.xlabel("step")
plt.ylabel(r"$\sigma^2$")
plt.legend()
```

[5]:
```
<matplotlib.legend.Legend at 0x7f79732438>
```



2 Problem 1

[6]:
```
import tensorflow as tf
```

2.1 1 Dimensions

2.1.1 Run the walkers

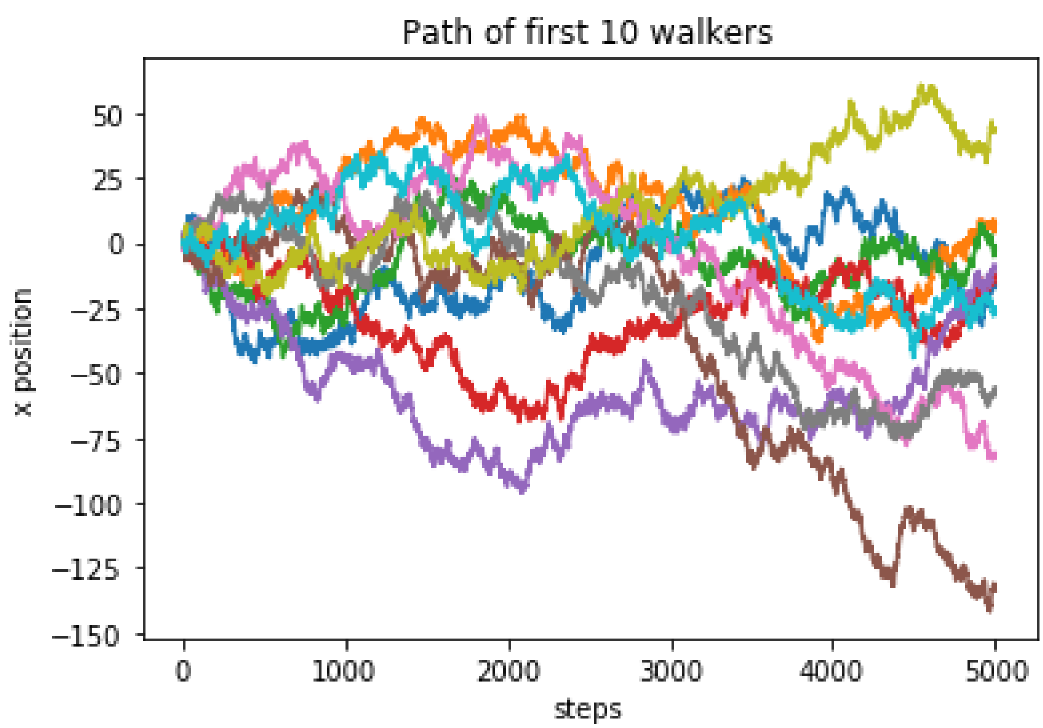
[7]:
```
g = tf.random.Generator.from_seed(1)

dims = 1
n_walkers = 1000
n_steps = 5000
t = np.arange(n_steps)
# Walkers can go in + direction, - direction, or stay still
step_set = [-1, 0, 1]
# The shape is for "n_walkers" taking "n_steps" in "dims" dimensions.
# So, in 1d if there are 10 walkers making 100 steps each,
# it will be (10, 100, 1)
step_shape = (n_walkers,n_steps,dims)
# These are the steps at each stage
steps = np.random.choice(a=step_set, size=step_shape)
steps = tf.random.stateless_uniform(
```

```
    step_shape, [0,1], minval=-1, maxval=2, dtype=tf.int32, name=None
)
# Now we add up the steps for each walker to get the x positions
steps = np.array(steps)
x = steps.cumsum(axis=1)
```

2.1.2 Plot the x position of the first 10 walkers

[8]:
```
for i in range( min(10,n_walkers) ):
    plt.plot( x[i,:,0] )
plt.title("Path of first 10 walkers")
plt.xlabel("steps")
plt.ylabel("x position")
```

[8]:
```
Text(0, 0.5, 'x position')
```



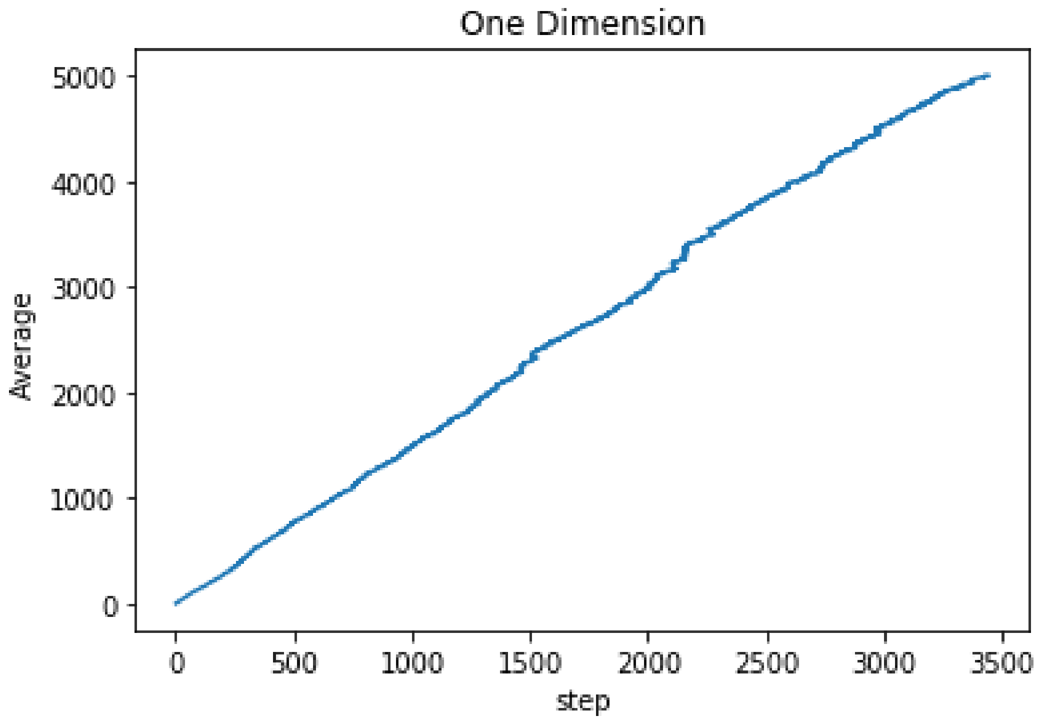
2.1.3 Compute the Averages

[9]:
```
# Now get the averages over the walkers
x2 = np.average( x**2, axis=0 )
x4 = np.average( x**4, axis=0 )
sigma2_nd = np.sqrt( x4 - x2**2 )
sigma2 = np.sum( sigma2_nd, axis=1 )
```

2.1.4 Plot the quantity $\langle |x_n|^2 \rangle$ vs n

[10]:
```
plt.plot( x2,t)
plt.title(r"One Dimension")
plt.xlabel("step")
plt.ylabel(r"Average")
```

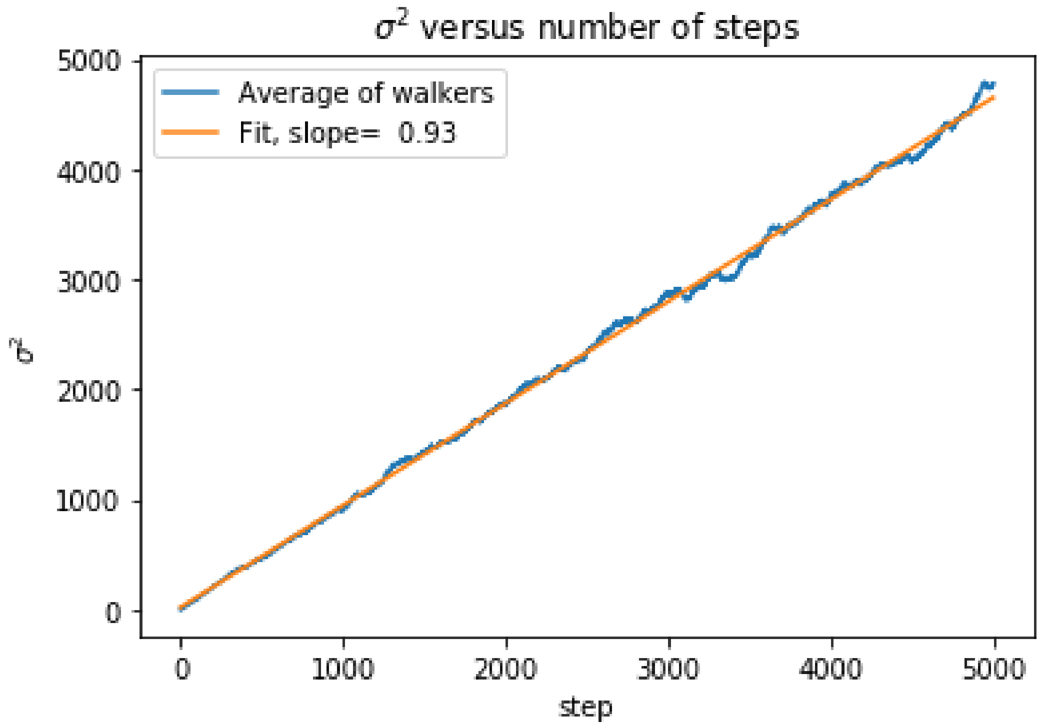
[10]: Text(0, 0.5, 'Average')



2.1.5 Calculate the Diffusion Constant and Compare to Theory

[11]:
```
plt.plot( sigma2,  label='Average of walkers' )
res = np.polyfit(t, sigma2,1 )
plt.plot( t, res[0]*t + res[1], label='Fit, slope=%6.2f' % res[0] )
plt.title(r"$\sigma^2$ versus number of steps")
plt.xlabel("step")
plt.ylabel(r"$\sigma^2$")
plt.legend()
```

[11]: <matplotlib.legend.Legend at 0x7f828f8f98>



[12]:
```
D = res[0]/(dims*2)
theoryD = 1/2
percentError = (abs(D-theoryD)/theoryD)*100
print("The value of the diffusion constant for " + str(dims) + " Dimensions is␣
↪" + str(D))
print("The theoretical value is " + str(theoryD))
print("The percent Error is: " +str(percentError) + " percent")
```

```
The value of the diffusion constant for 1 Dimensions is 0.46279260928789795
The theoretical value is 0.5
The percent Error is: 7.44147814242041 percent
```

2.2 2 Dimensions

2.2.1 Run the walkers

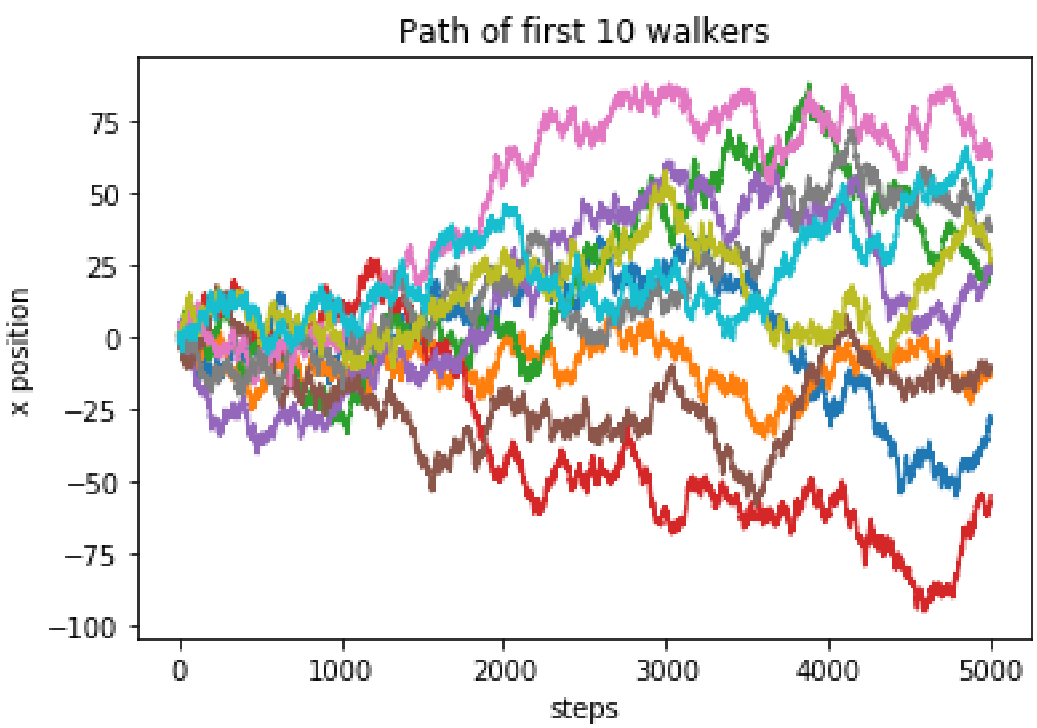
[13]:
```
g = tf.random.Generator.from_seed(1)

dims = 2
n_walkers = 1000
n_steps = 5000
t = np.arange(n_steps)
# Walkers can go in + direction, - direction, or stay still
step_set = [-1, 0, 1]
```

```
# The shape is for "n_walkers" taking "n_steps" in "dims" dimensions.
# So, in 1d if there are 10 walkers making 100 steps each,
# it will be (10, 100, 1)
step_shape = (n_walkers,n_steps,dims)
# These are the steps at each stage
steps = np.random.choice(a=step_set, size=step_shape)
steps = tf.random.stateless_uniform(
    step_shape, [0,1], minval=-1, maxval=2, dtype=tf.int32, name=None
)
# Now we add up the steps for each walker to get the x positions
steps = np.array(steps)
x = steps.cumsum(axis=1)
```

2.2.2 Plot the x position of the first 10 walkers

[14]:
```
for i in range( min(10,n_walkers) ):
    plt.plot( x[i,:,0] )
plt.title("Path of first 10 walkers")
plt.xlabel("steps")
plt.ylabel("x position")
```

[14]: Text(0, 0.5, 'x position')



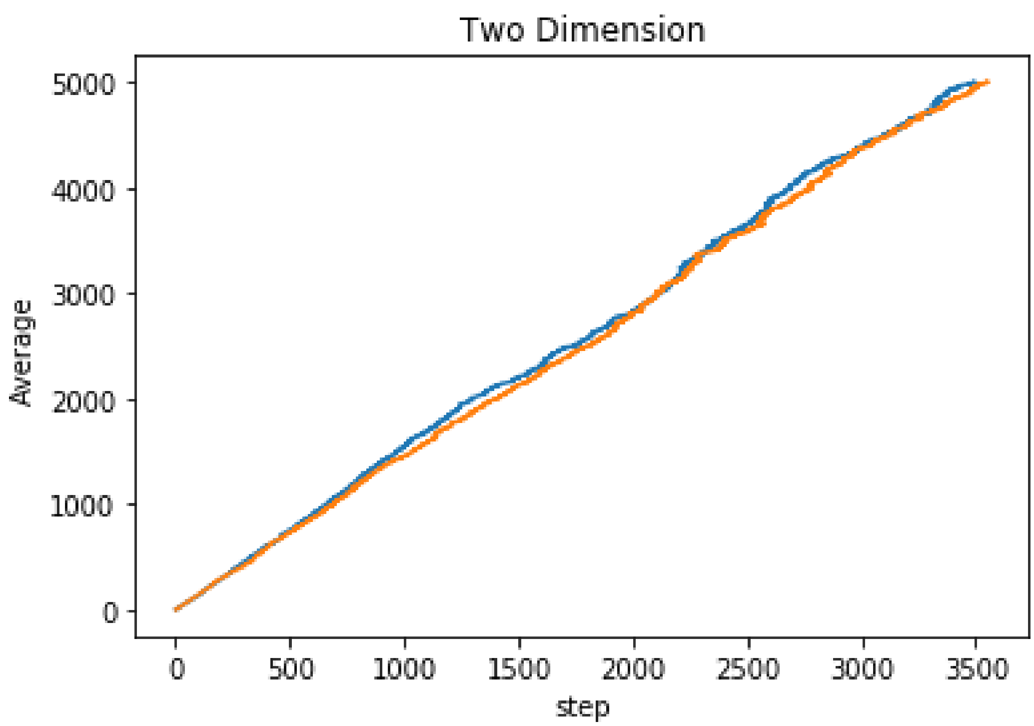
2.2.3 Compute the Averages

[15]:
```
# Now get the averages over the walkers
x2 = np.average( x**2, axis=0 )
x4 = np.average( x**4, axis=0 )
sigma2_nd = np.sqrt( x4 - x2**2 )
sigma2 = np.sum( sigma2_nd, axis=1 )
```

2.2.4 Plot the quantity $\langle |x_n|^2 \rangle$ vs n

[16]:
```
plt.plot( x2,t)
plt.title(r"Two Dimension")
plt.xlabel("step")
plt.ylabel(r"Average")
```

[16]: Text(0, 0.5, 'Average')

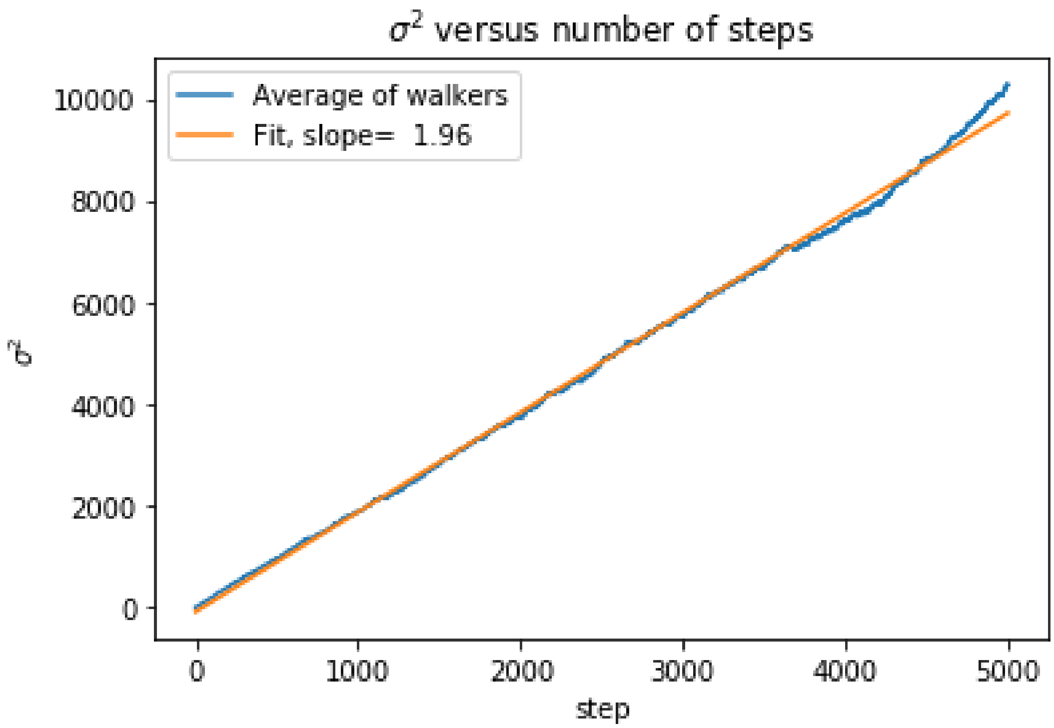


2.2.5 Calculate the Diffusion Constant and Compare to Theory

[17]:
```
plt.plot( sigma2,  label='Average of walkers' )
res = np.polyfit(t, sigma2,1 )
plt.plot( t, res[0]*t + res[1], label='Fit, slope=%6.2f' % res[0] )
plt.title(r"$\sigma^2$ versus number of steps")
```

```
plt.xlabel("step")
plt.ylabel(r"$\sigma^2$")
plt.legend()
```

[17]: <matplotlib.legend.Legend at 0x7f827c7198>



[18]:
```
D = res[0]/(dims*2)
theoryD = 1/2
percentError = (abs(D-theoryD)/theoryD)*100
print("The value of the diffusion constant for " + str(dims) + " Dimensions is␣
↪" + str(D))
print("The theoretical value is " + str(theoryD))
print("The percent Error is: " +str(percentError) + " percent")
```

```
The value of the diffusion constant for 2 Dimensions is 0.4904570667290827
The theoretical value is 0.5
The percent Error is: 1.908586654183464 percent
```

2.3 3 Dimensions

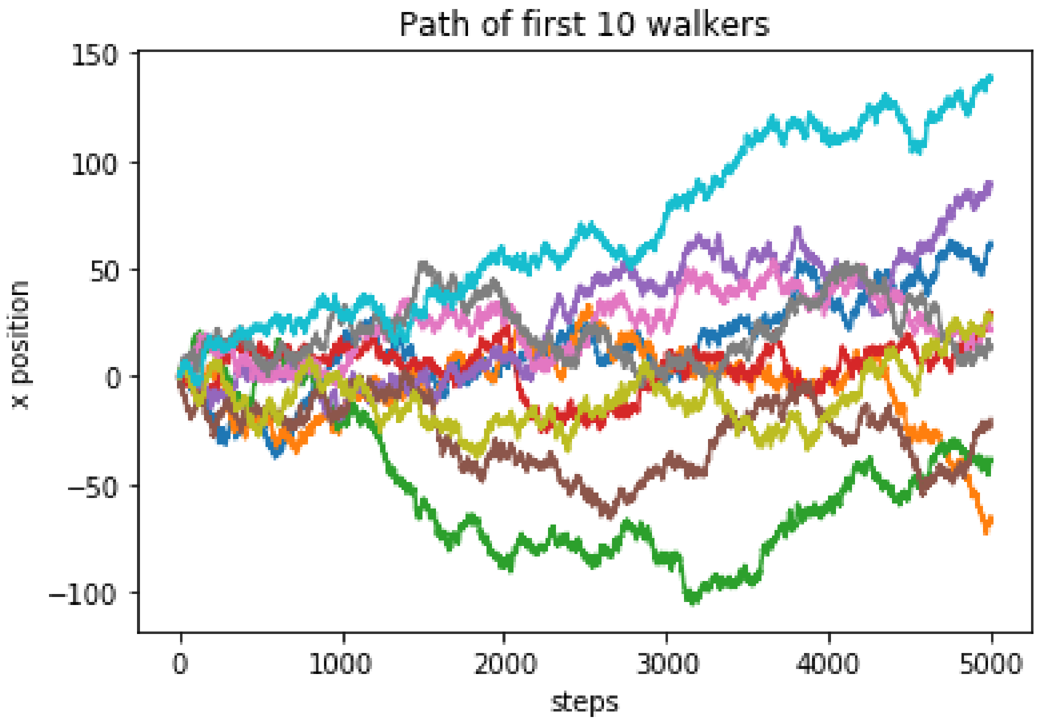
2.3.1 Run the walkers

[19]:
```
g = tf.random.Generator.from_seed(1)

dims = 3
n_walkers = 1000
n_steps = 5000
t = np.arange(n_steps)
# Walkers can go in + direction, - direction, or stay still
step_set = [-1, 0, 1]
# The shape is for "n_walkers" taking "n_steps" in "dims" dimensions.
# So, in 1d if there are 10 walkers making 100 steps each,
# it will be (10, 100, 1)
step_shape = (n_walkers,n_steps,dims)
# These are the steps at each stage
steps = np.random.choice(a=step_set, size=step_shape)
steps = tf.random.stateless_uniform(
    step_shape, [0,1], minval=-1, maxval=2, dtype=tf.int32, name=None
)
# Now we add up the steps for each walker to get the x positions
steps = np.array(steps)
x = steps.cumsum(axis=1)
```

2.3.2 Plot the x position of the first 10 walkers

[20]:
```
for i in range( min(10,n_walkers) ):
    plt.plot( x[i,:,0] )
plt.title("Path of first 10 walkers")
plt.xlabel("steps")
plt.ylabel("x position")
```

[20]:
```
Text(0, 0.5, 'x position')
```



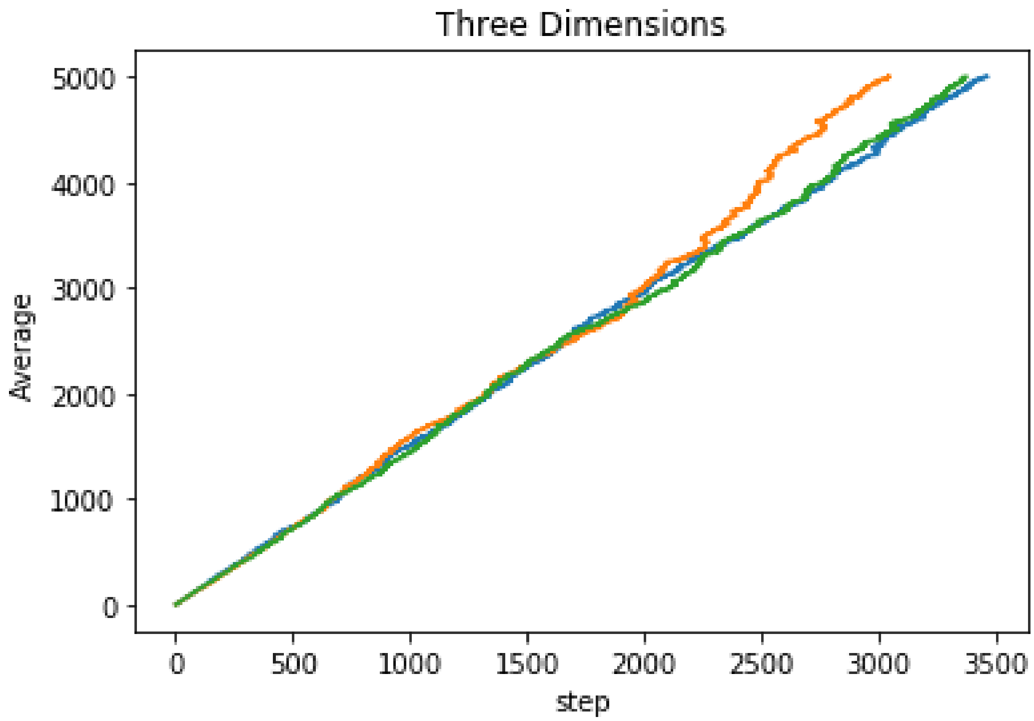
2.3.3 Compute the Averages

[21]:
```
# Now get the averages over the walkers
x2 = np.average( x**2, axis=0 )
x4 = np.average( x**4, axis=0 )
sigma2_nd = np.sqrt( x4 - x2**2 )
sigma2 = np.sum( sigma2_nd, axis=1 )
```

2.3.4 Plot the quantity $\langle |x_n|^2 \rangle$ vs n

[22]:
```
plt.plot( x2,t)
plt.title(r"Three Dimensions")
plt.xlabel("step")
plt.ylabel(r"Average")
```

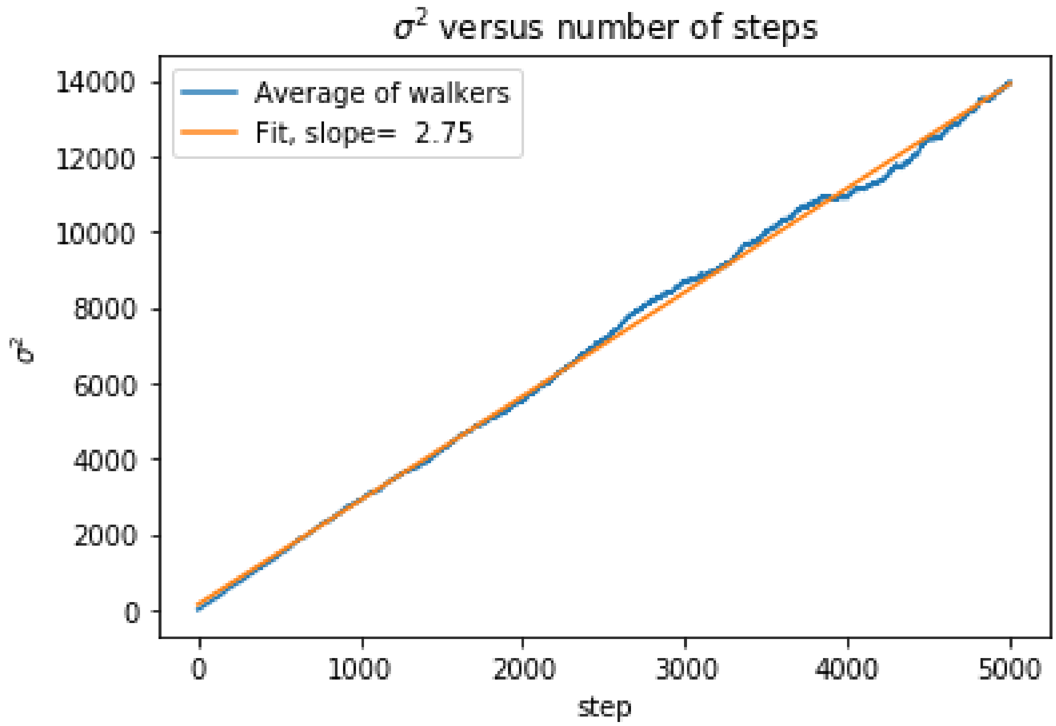
[22]: Text(0, 0.5, 'Average')



2.3.5 Calculate the Diffusion Constant and Compare to Theory

[23]:
```
plt.plot( sigma2,  label='Average of walkers' )
res = np.polyfit(t, sigma2,1 )
plt.plot( t, res[0]*t + res[1], label='Fit, slope=%6.2f' % res[0] )
plt.title(r"$\sigma^2$ versus number of steps")
plt.xlabel("step")
plt.ylabel(r"$\sigma^2$")
plt.legend()
```

[23]:
```
<matplotlib.legend.Legend at 0x7f8268fda0>
```



[24]:
```
D = res[0]/(dims*2)
theoryD = 1/2
percentError = (abs(D-theoryD)/theoryD)*100
print("The value of the diffusion constant for " + str(dims) + " Dimensions is␣
 ↪" + str(D))
print("The theoretical value is " + str(theoryD))
print("The percent Error is: " +str(percentError) + " percent")
```

```
The value of the diffusion constant for 3 Dimensions is 0.4591364281910324
The theoretical value is 0.5
The percent Error is: 8.17271436179352 percent
```

[ ]:

Problem2-metropolis

April 9, 2021

1 Metropolis algorithm example

Here we look at the Metropolis-Hastings algorithm, which is a Markov-Chain Monte Carlo (MCMC) technique.

1.0.1 Swig it, compile it, add it to the path

[1]:
```
! swig -c++ -python swig/metropolis.i
! python swig/setup_metropolis.py build_ext --inplace
```

```
running build_ext
building '_metropolis' extension
aarch64-linux-gnu-gcc -pthread -DNDEBUG -g -fwrapv -O2 -Wall -g -fstack-
protector-strong -Wformat -Werror=format-security -Wdate-time
-D_FORTIFY_SOURCE=2 -fPIC -I/usr/include/python3.7m -c swig/metropolis_wrap.cxx
-o build/temp.linux-aarch64-3.7/swig/metropolis_wrap.o -I./ -std=c++11 -O3
aarch64-linux-gnu-g++ -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions
-Wl,-z,relro -Wl,-z,relro -g -fstack-protector-strong -Wformat -Werror=format-
security -Wdate-time -D_FORTIFY_SOURCE=2 build/temp.linux-
aarch64-3.7/swig/metropolis_wrap.o -o /home/pi/tensorflow-probability-
ChanceStarr/RandomNumbers/_metropolis.cpython-37m-aarch64-linux-gnu.so
```

[2]:
```
import sys
import os
sys.path.append( os.path.abspath("swig") )
```

[3]:
```
import metropolis
import numpy as np
import matplotlib.pyplot as plt
import tensorflow as tf
```

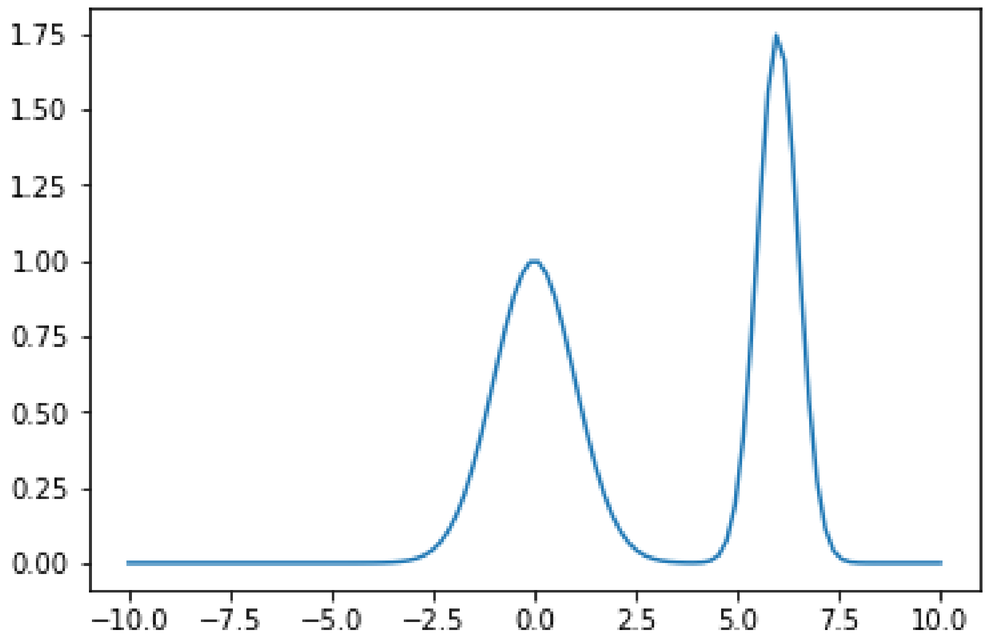
1.1 Probability distribution

Make the probability distribution equal to a sum of Gaussians.

[4]:
```
A = [1., 1.75]
sigma = [1.0, 0.5]
center = [0.0, 6.0]
```

```
g = metropolis.gaussianD( A, sigma, center )

gvalsi = []
gxvals = np.linspace(-10,10,100)
for x in gxvals:
    gvalsi.append( g(x) )
gvals = np.array(gvalsi)
```

[5]:
```
plt.plot(gxvals, gvals)
plt.show()
```



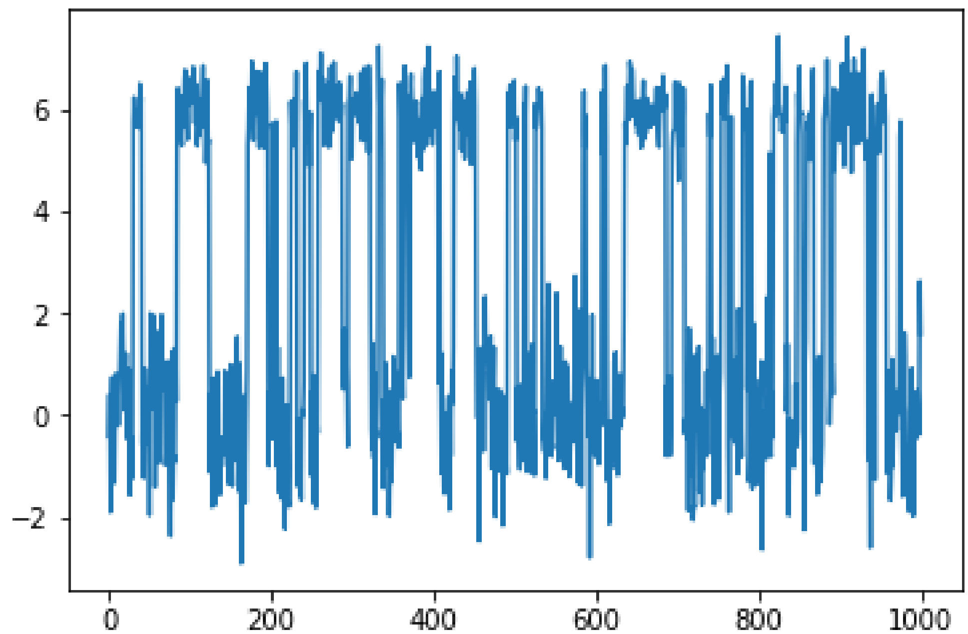
1.2 Run Metropolis-Hastings

[6]:
```
x0 = 0.0
delta = 1.0
nskip = 1000

m = metropolis.metropolisD( g, x0, delta, nskip, False )
xvals = []

nmcsteps = 1000
for i in range(nmcsteps):
    m.monte_carlo_step()
    xvals.append( m.get() )
```

1.3 Plot the time series of the "walker"

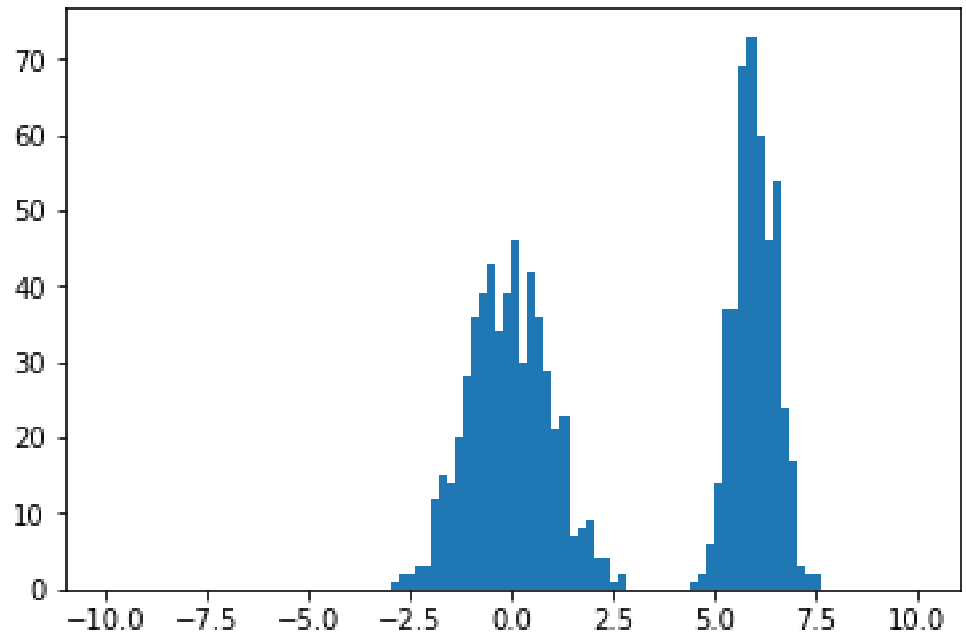
[7]:
```
plt.plot(xvals)
```

[7]: [<matplotlib.lines.Line2D at 0x7f4be05f28>]



1.4 Plot the distribution that MH arrives at

[8]:
```
res = plt.hist( xvals, bins=100, range=(-10,10) )
plt.show()
```



2 Problem 2 - TensorFlow

Below tensorflow and tensorflow-probability are used to calculate the Metropolis algorithm.

[9]:
```
import numpy as np
import tensorflow.compat.v2 as tf
import tensorflow_probability as tfp
tf.enable_v2_behavior()

tfd = tfp.distributions

dtype = np.float32

target = tfd.Normal(loc=dtype(0), scale=dtype(1))

samples = tfp.mcmc.sample_chain(
  num_results=1000,
  current_state=dtype(1),
  kernel=tfp.mcmc.RandomWalkMetropolis(target.log_prob),
  num_burnin_steps=500,
  trace_fn=None,
  seed=42)

sample_mean = tf.math.reduce_mean(samples, axis=0)
sample_std = tf.sqrt(
```

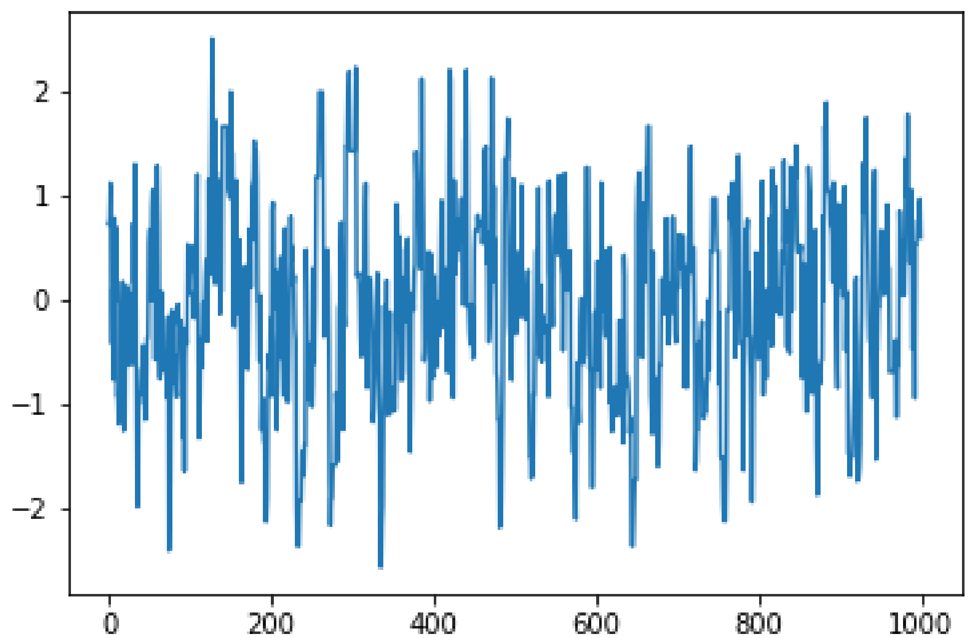
```
    tf.math.reduce_mean(
        tf.math.squared_difference(samples, sample_mean),
        axis=0))

print('Estimated mean: {}'.format(sample_mean))
print('Estimated standard deviation: {}'.format(sample_std))
```

```
Estimated mean: -0.06041654199361801
Estimated standard deviation: 0.9357634782791138
```

2.1 Plot the time series of the "walker"

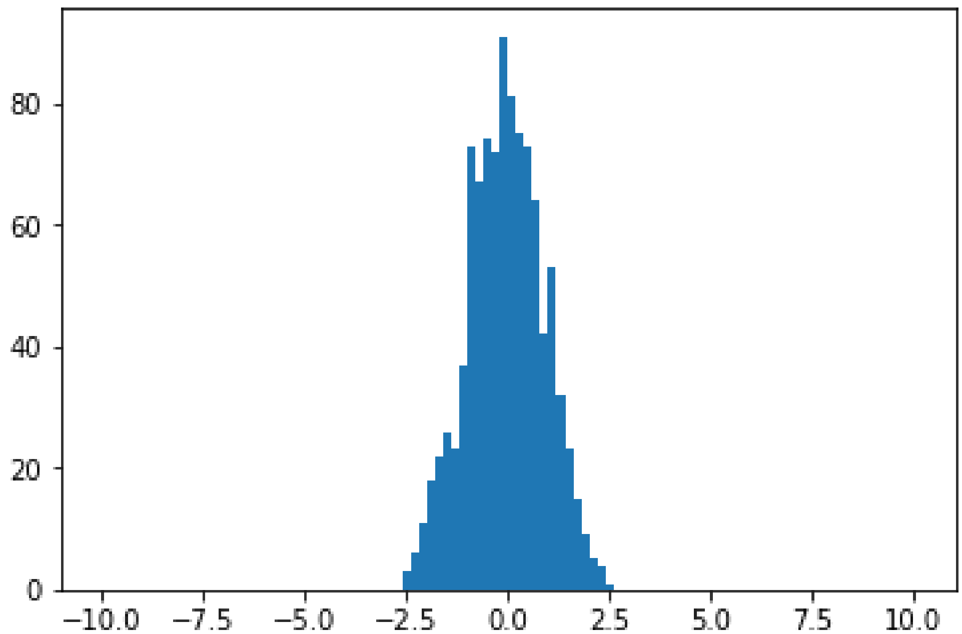
[10]:
```
plt.plot(samples)
```

[10]:
```
[<matplotlib.lines.Line2D at 0x7f32410da0>]
```



2.2 Plot the distribution that MH arrives at

[11]:
```
res = plt.hist( samples, bins=100, range=(-10,10) )
plt.show()
```



0
20
40
60
80
−10.0
−7.5
−5.0
−2.5
0.0
2.5
5.0
7.5
10.0


```
[ ]:
```