Random Walk: Jessica Murphy

13/02/2023

Homework 15

This python programme involves an Initial Vlue Porblem where it uses the Monte Carlo technique to determine the probabilty of how much metres were walked in a one dimensional random walk and displaying 1000 runs of this calculation on a histogram.

A random walker starts at the origin of the x-axis. They have a probability p = 0.66 of stepping right (positive x-direction) and (1−p) of stepping left (negative x-direction). They take N = 50 steps and then stop. Each step has a length of s = 0.75 m.

A python function is written which takes N, p and s as parameters, where N is the number of steps the walker takes, p is the probability of stepping right, and s is the size of the step they make.

Then a main python function is written which calls the random walker function M = 1000 times and a histogram is plotted of the result of the 1000 resulting final x-locations. The mean and standard deviation of the 1000 x-locations is also displayed.

A comparison is then given at the end of the notebook comparing the mean μ and standard deviation σ with the expected values, which for sufficently large M is given as follows:

$$\mu = (2p-1)Ns$$

$$\sigma = 2s\sqrt{p(1-p)N}$$

An additional discussion is made, commenting whether the computed *Monte Carlo* mean and expected mean agree within an explicit uncertainty - the statsitical error , the equation for which is given as follows:

$$\Delta\mu = \frac{\sigma}{\sqrt{M}}$$

1. Define a function to simulate N random walks
    - Import python library NumPy
2. Initialise the origin, the walker starts at x = 0, the walker will remain on the x-axis as it is a one dimensional walk
3. Use a for loop to set a probability of stepping along the x-axis, using if-else statements to establish the probabilty of stepping right along the axis, and left if not.
4. Define a function that defines the process of the walker taking 50 steps and gives a final x result
5. Define a main function calling the above function of the walker taking 50 steps with a final x value
    - Import python libraries NumPy and Matplotlib
    - Set the initial values again
    - Set the amount of walks
    - Create array to hold walk distances
    - Assign a for loop to analyse the final x value in every sequence
    - Display the computed mean and standard deviation
6. Plot a histogram of the 1000 runs of the monte carlo simulations of the random walk
7. Discuss the acquired result and the calculated result and how they compare within the statistical error of the mea

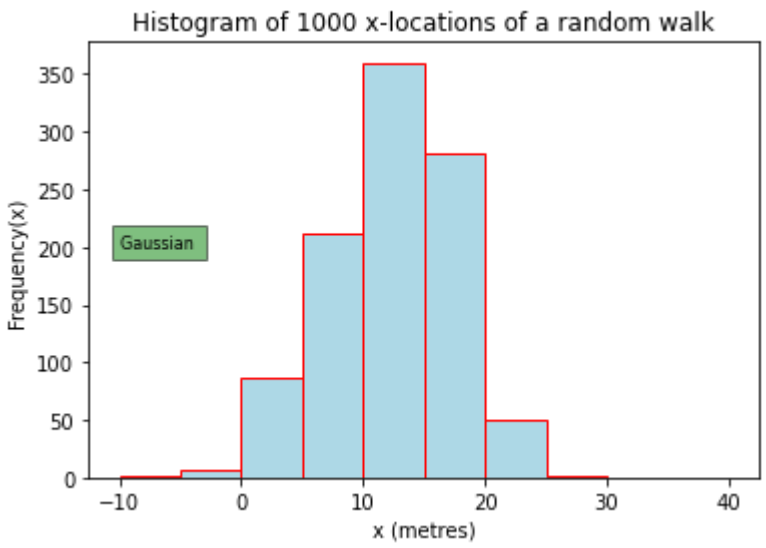
In [1]:
```
# Procedure to simulate N random walks
# N is the number of steps , p is the probability of stepping right, s step size
# Returns x teh distance walked (gained)
# define function setting the conditions of the programme
def walk(N, p, s):

    # import python libary needed
    import numpy as np

    x = 0 # the walker starts at x = 0, origin of axis, the walker will remain on the x-axis as it is a one dimensional walk

    # using a for loop containing if-else statements to establish the probabilty of stepping right along the axis, and left if not.
    for i in range(N):
        if(np.random.rand() < p):
            x = x + s
        else:
            x = x - s

    return(x)

# define a driver code - this defines the process of the walker taking 50 steps and gives a final x result
def main():

    N = 50 # set the number of steps
    p = 0.66 # set the proability stepping right
    s = 0.75 # set the step length

    # Call the function to determine the random walk
```

```
    x = walk(N, p, s) # assign a variable x which is the net distance travelled aftrer 50 steps
    print('final x value =', x) # the final distance travelled after 50 steps

main() # bound the function
```

```
final x value = 15.0
```

In [2]:
```
def main2(): # the main function calling the process of the walker taking 50 steps, with a final x value, 1000 times

    # import python libraries needed
    import numpy as np
    import matplotlib.pyplot as plt

    # set the same initial values again
    N = 50
    p = 0.66
    s = 0.75

    # set the number of events where 50 one dimensional random walks take place
    M = 1000

    X = np.zeros(M) # create array to hold walk distances
    for i in range(M): # assign a for loop to analyse the final x value in every sequence
        X[i] = walk(N, p, s)

    print('Mean ={0:8.3f}'.format(np.mean(X))) # print the mean of the 1000 x-locations
    print('Standard deviation ={0:8.3f} '.format(np.std(X))) # print the standard deviation of the 1000 x-locations

    # plot a histogram of the 1000 runs of the monte carlo simulations of the random walk
    bins = np.linspace(-10, 40, 11) #specify the bin edges using a linearly spaced array, and set the bin size as 5 metres
    plt.hist(X, bins, color='lightblue', ec='red', rwidth = 1.75, lw =1)
    plt.title('Histogram of 1000 x-locations of a random walk')
    plt.xlabel('x (metres)')
    plt.ylabel('Frequency(x)')
    plt.text(-10, 200, 'Gaussian ',fontsize =8, bbox = dict(facecolor = 'green', alpha = 0.5))
    plt.show()

main2() # bound the function
```

```
Mean =  12.040
Standard deviation =   5.098
```



Comparing the acquired mean and standard deviation of the mean

---

The graph displays a vague Gaussian distribution. It is shown form the programme that the computed mean μ = 12.040 , and the standard deviation σ = 5.098. Comparing this to the expected values, of μ = 12.0 , σ = 5.0 , calculated below, we can see that the computed values differ from the expected values as 0.040 and 0.098, respectively.

The calculated values for the mean and standard deviation were determined from the following equations, which can be used for sufficently large M:

$$\mu = (2p-1)Ns$$

$$\sigma = 2s\sqrt{p(1-p)N}$$

In [3]:
```
import numpy as np
mu = (2*(0.66)-1)*50*0.75
sigma = 2 * 0.75 * np.sqrt((0.66)*(1-0.66)*50)

print('Mean ={0:5.1f}'.format(mu))
print('Standard deviation ={0:5.1f}'.format(sigma))
```

```
Mean = 12.0
Standard deviation =  5.0
```

In [4]:
```
del_mu = sigma/np.sqrt(1000)

print('Stastical error = {0:5.1f}'.format(del_mu))
```

```
Stastical error =   0.2
```

The **statistical error** on the mean $\Delta\mu$ is given by

$$\Delta\mu = \frac{\sigma}{\sqrt{M}}$$

which is equal to 0.158113883 which is approximately 0.2. As shown in the above comment, the difference is 0.04 , and does agree within this error. Repeated runs of this programme were made and the range of an acceptable statistical error, $\Delta\mu$, stayed consistent.

It should be noted that with an increase on the number of samples M, the estimate should converge to the true value of x, which can be realised from the law of large numbers. Knowing this, if the programme is re-run many times we would see the programme is centred on the true value of x. This shows the Monte Carlo technique yields close approximations to the true value that depends on the number of values used for an average.