

Introduction to Monte Carlo Methods

Bruce Mvubele - MVBBRU001

13 May 2019

Contents

1	Introduction	2
2	Monte Carlo Data Generation	2
3	Monte Carlo Determination of π	4
4	Appendix	5

1 Introduction

Monte Carlo Methods are a large class of computational algorithms that depend on the use of random numbers to obtain numerical results. And are largely useful when looking to solve problems that might be deterministic. In principle these methods can be used to solve any problem having a probabilistic interpretation.

2 Monte Carlo Data Generation

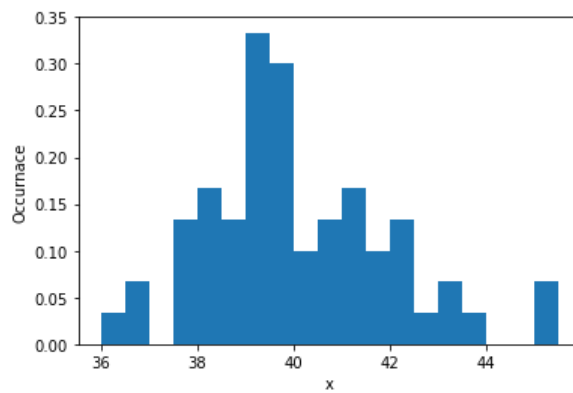


Figure 1: Histogram of Data in Activity1Data.txt

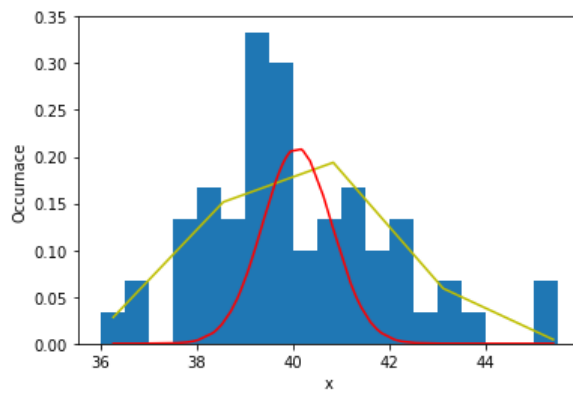


Figure 2: Histogram Activity1Data.txt with Gaussian

In including the Gaussian for The Activity 1 data shown in figure 2, I have include two such curves. The reason being I could not decide on which curve is more suitable, the red curve has the characteristic bell shape curve of a Gaussian however it seems that the curve does not seem to represent the data at the extreme left and extreme right where the red curve approaches zero. In trying to address this issue I came across a method which allowed me plot the yellow curve, now this yellow curve does address the issue of not having data represented at the extreme edges of the plot however it does not have the charestic bell shape curve and hence they are both included in this plot.

A Gaussian (normal) distribution of mean μ and standard deviation σ given by

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

has been normalized by the term in denominator such that the area under the graph is equal to one, we therefore need to normalize the histogram. By my understanding we multiply the histogram by the number on entries plotted on the histogram in order to normalize it.

```
Random numbers drawn from Gaussian with mu = 40.0 and sigma = 2.0
1      43.6740972208
2      41.0100478688
3      38.4128377831
4      37.5505958905
5      38.180136269
6      37.6486166404
7      39.4475335592
8      39.3908710757
9      41.8687322386
10     40.7154610394
11     41.7689206482
12     43.6541058042
13     43.2977132462
14     41.3713156424
15     33.7393000787
16     41.0056634566
17     44.0612782028
18     42.3267152249
19     42.0999239308
20     36.9549477747
21     43.2668724936
22     38.4725547792
23     41.0880958511
24     39.7381211722
25     38.1351180105
26     42.7090568757
27     39.4932814568
28     42.4054810489
29     40.6711568363
30     40.2420229328
31     40.4801777508
32     42.6939169879
33     38.9877644909
34     39.4967852649
35     38.0929799646
36     41.5159955424
37     35.603047031
```

Figure 3: Text file with 60 random entries generated with random.normal

For the data produced in the text file above the mean is 39.73 and standard deviation is 2.02 which are good approximations of the intended values of having

mean 40 and standard deviation 2 which confirms that if the data is drawn from a Gaussian distribution the sample mean and sample standard deviations can be used to approximate the underlying Gaussian.

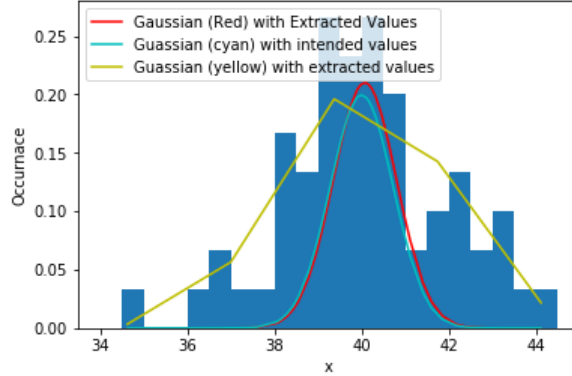


Figure 4: Histogram of random data with intended Gaussian and Gaussian from extracted parameters

Figure 4 shows the histogram plot of the data in figure 3.. On the histogram plot I have included 3 curves. The red curve shows the Gaussian with the extracted values. The cyan curve is the Gaussian with the intended values. Furthermore I once again include an alternate method for plotting the Gaussian with the extracted values, however when this method is used with the intended values, the Gaussian then upsets the plot and show a scaling problem I have therefore not included it as it does not give any useful insight to the Carlo Monte Method. Additionally we note that the Gaussian from extracted values peaks at higher height then that which is constructed from the intended values. However it still presents a good approximation of it.

3 Monte Carlo Determination of π

In order to estimate π the hit and miss method was employed where we consider a 2×2 square with an inscribed circle of radius 1, we then simulate throwing darts at the square and we have the following relations. fraction of points in circle is equal to

$$\frac{A_{circle}}{A_{square}} \approx \pi$$

The hit and miss method was carried out five to acquire five π estimates, the average of the values was taken to be the best estimate. This leads to type A uncertainty, the uncertainty was then determined by taking the standard deviation. Hence the final result for the estimate of π is $\pi = 3.922 \pm 0.004$ which does not agree the accepted the value of π possible reasons for this could be

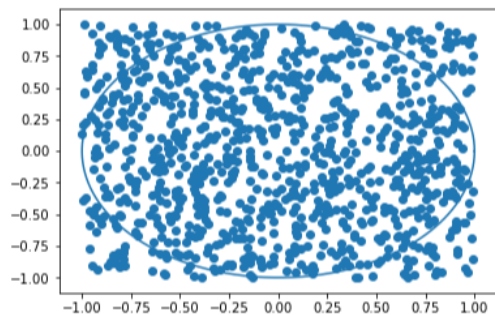


Figure 5: Illustration of the hit and miss method used to estimate π using 1000 throws

number of points used in our code, as introspection I realized that using less points would have yielded a accurate estimate of π which I initially did not think of as I thought the more values I used the more accurate the estimate would be.

4 Appendix

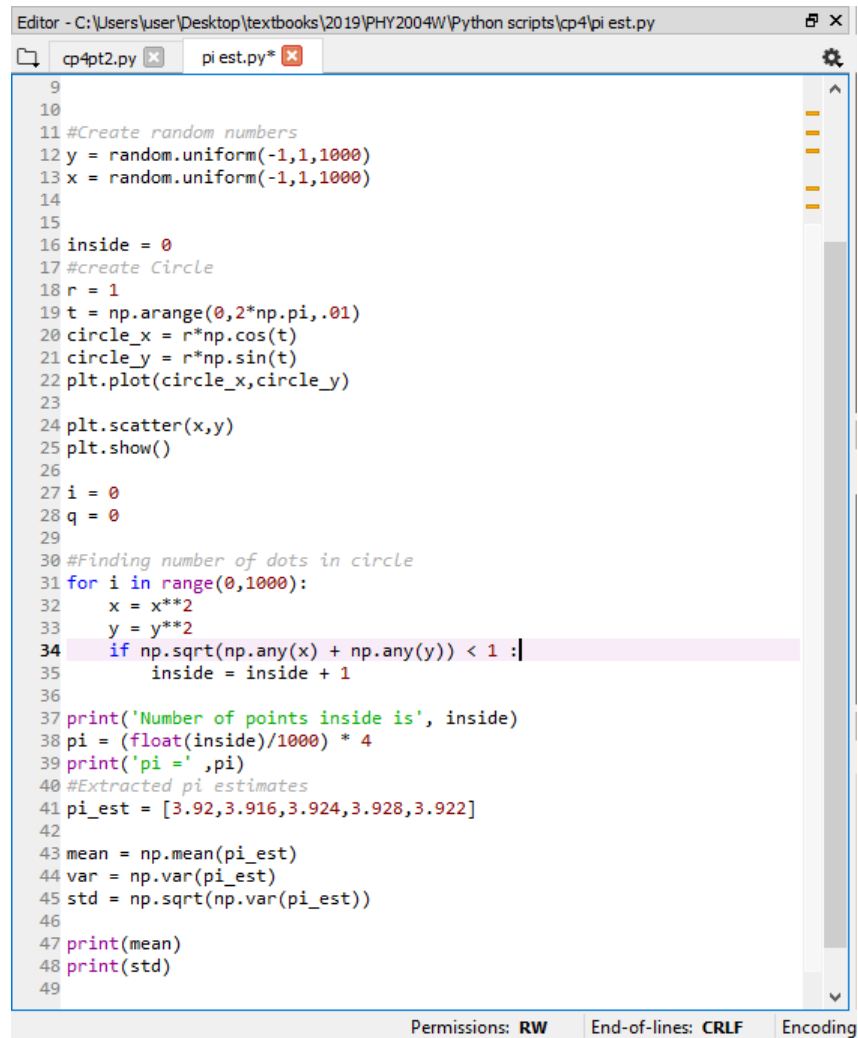
```

Editor - C:\Users\user\Desktop\textbooks\2019\PHY2004W\Python scripts\cp4\cp4pt2.py
cp4pt2.py* x pi est.py* x

53 #Gaussian with bell shape and extracted values
54 f = np.exp(-np.square(n_sort-mean)/2*std)/(std*(np.sqrt(2*np.pi)))
55 #print(len(f))
56 sumv = sum(f)
57 #print(sumv)
58 plt.plot(n_sort,f,'r-', label = 'Gaussian (Red) with Extracted Values')
59 minv = min(y)
60 maxv = max(y)
61 sumv = sum(y)
62
63 #secondary gaussian with bell shape and intended values
64 f = np.exp(-np.square(n_sort-40)/2*std)/(2*(np.sqrt(2*np.pi)))
65 sumv = sum(f)
66 plt.plot(n_sort,f,'c-', label = 'Guassian (cyan) with intended values')
67
68 #print(minv)
69 #print(maxv)
70 print(sumv)
71
72 binwidth = 0.5
73 bins = np.arange(np.floor(min(y)),np.floor(max(y))+1 , binwidth)
74 result = plt.hist(y,bins, normed = True)
75 plt.xlabel('x')
76 plt.ylabel('Occurnace')
77
78
79 #Gaussian with no bell shape and extracred values
80 q = np.linspace(minv, maxv,5)
81 dq = result[1][1] - result[1][0]
82 scale = len(y) * dq
83 plt.plot(q,mlab.normpdf(q, mean, std), 'y-',label = 'Guassian (yellow)')
84 plt.legend(loc='upper left')
85 plt.show()
86
87 #secondary gaussian not bell shaped and with inted values
88 w = np.linspace(minv, maxv,5)
89 dw = result[1][1] - result[1][0]
90 scale = len(y) * dw
91 #plt.plot(w,mlab.normpdf(w, 40, 2),'g-')
92

```

Figure 6: Code showing the alternative methods used to plot the Gaussian Curve



```
Editor - C:\Users\user\Desktop\textbooks\2019\PHY2004W\Python scripts\cp4\pi est.py
cp4pt2.py x pi est.py* x
9
10
11 #Create random numbers
12 y = random.uniform(-1,1,1000)
13 x = random.uniform(-1,1,1000)
14
15
16 inside = 0
17 #create Circle
18 r = 1
19 t = np.arange(0,2*np.pi,.01)
20 circle_x = r*np.cos(t)
21 circle_y = r*np.sin(t)
22 plt.plot(circle_x,circle_y)
23
24 plt.scatter(x,y)
25 plt.show()
26
27 i = 0
28 q = 0
29
30 #Finding number of dots in circle
31 for i in range(0,1000):
32     x = x**2
33     y = y**2
34     if np.sqrt(np.any(x) + np.any(y)) < 1 :|
35         inside = inside + 1
36
37 print('Number of points inside is', inside)
38 pi = (float(inside)/1000) * 4
39 print('pi =' ,pi)
40 #Extracted pi estimates
41 pi_est = [3.92,3.916,3.924,3.928,3.922]
42
43 mean = np.mean(pi_est)
44 var = np.var(pi_est)
45 std = np.sqrt(np.var(pi_est))
46
47 print(mean)
48 print(std)
49
Permissions: RW End-of-lines: CRLF Encoding
```

Figure 7: Code for estimating π

```
Editor - C:\Users\user\Desktop\textbooks\2019\PHY2004W\Python scripts\cp4\cp4pt2.py
cp4pt2.py
8 import numpy.random as random
9
10 numbers = random.normal(40, 2, 61) #60 random numbers
11
12 axis = np.arange(0,61,1) # Labeling of the numbers
13
14 #creating a text file called random60 with two columns one for the numbering system
15 #and the other with the the random numbers created above
16 f = open('random60.txt', 'w')
17 f.write('Random numbers drawn from Gaussian with mu = 40.0 and sigma = 2.0 \n')
18 i = 0
19
20 for i in range(1,61):
21     val = str(axis[i])+'\t'+str(numbers[i])+'\n'
22     f.write(val)
23     i += 1
24 f.close()
25
26 #create histogram
27 g = open('random60.txt', 'r')
28 header = g.readline()
29 data = np.zeros(60)
30 i = 0
31
32 x,y = [],[]
33 for line in g:
34     values = [float(s) for s in line.split()]
35     x.append(values[0])
36     y.append(values[1])
37
38 x = np.array(x)
39 y = np.array(y)
40
41 mu = 40
42 sigma = 2
43
44 mean = np.mean(y)
45 var = np.var(y)
46 std = np.sqrt(np.var(y))
47
```

Figure 8: Code for 60 random number