Introduction to Monte Carlo Methods

5 May 2020

PHY2004W KDSMIL001

Contents

1 Introduction and Aim

In this assignment we investigated the Monte Carlo method, a way of simulating systems by use of random numbers. We revisited some data from a previous assignment and created our own version of the data to analyse.

2 Activity

- **Histogramming Data**
  Firstly, we had a look back at some data from CP1, namely Activity1Data.txt, a list of 60 values generated with a certain mean ($\mu = 40$) and standard deviation ($\sigma = 2$) using a Monte Carlo method. We plotted these values on a histogram (Appendix 1) and plotted the gaussian based on a μ and σ calculated from the data itself. We also plotted the expected gaussian given the μ and σ used to generate the data.

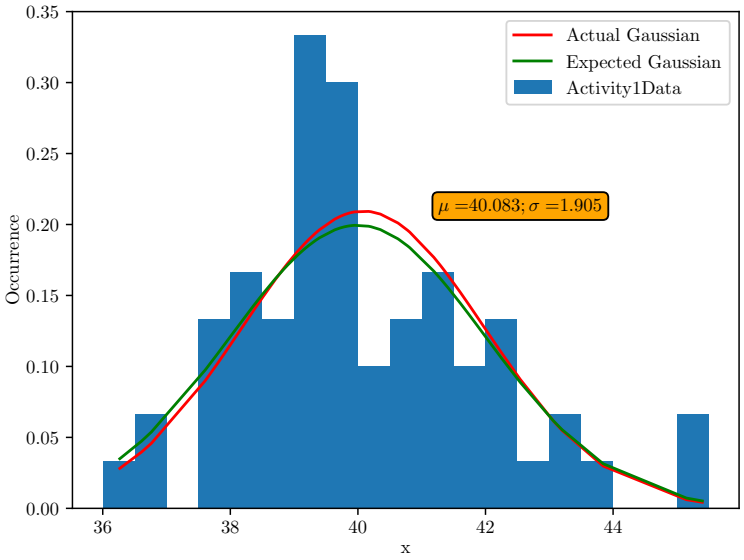

Figure 1: Activity1Data Histogram and Gaussians

Before scaling, the histogram was much larger than the gaussian as the area under a gaussian distribution is always 1, whereas the histogram doesn't necessarily have an area of 1. To resolve this and make the graph more readable, we set the density argument to True when creating the histogram, which normalises it, giving us the graph above.
Now, looking at the Actual Gaussian compared to the Expected Gaussian in

Figure 1, we see that they're a bit different, in fact the actual gaussian peaks slightly higher than the expected one. This is probably because the method of generating data isn't perfect so we can expect slight variations from dataset to dataset. We will investigate this further in the next section.

- **Generating Random Numbers**
  Next, we created some of our own data to analyse in the same way (Appendix 2). Below is an example of the file we created to analyse.

```
1 Random numbers drawn from Gaussian with mu = 40.0 and sigma = 2.0
2 1 43.19
3 2 42.839
4 3 42.441
```

Randomised Data

Doing exactly the same analysis as before, we have the plots in Figure 2, an example of 2 different datasets created from the same initial conditions.

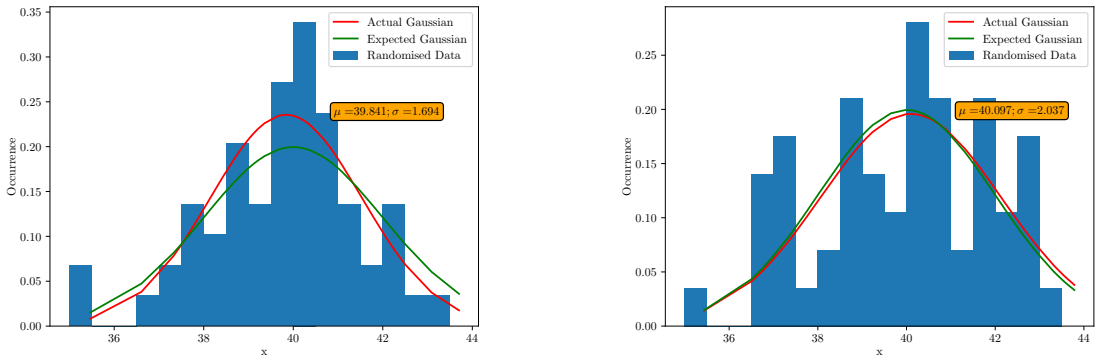

Figure 2: 60 Random points with specific μ and σ

As can be seen, the mean still sits at around 40 but due to the different distributions of the random numbers used to create the data it's never exactly 40. It is likely that if we were to increase the number of random numbers, we would get a more accurate mean, as well as a consistently more accurate gaussian plot. Below in Figure 3 is the exact same program but with 10 times as many data points. Clearly this gives more accurate results.

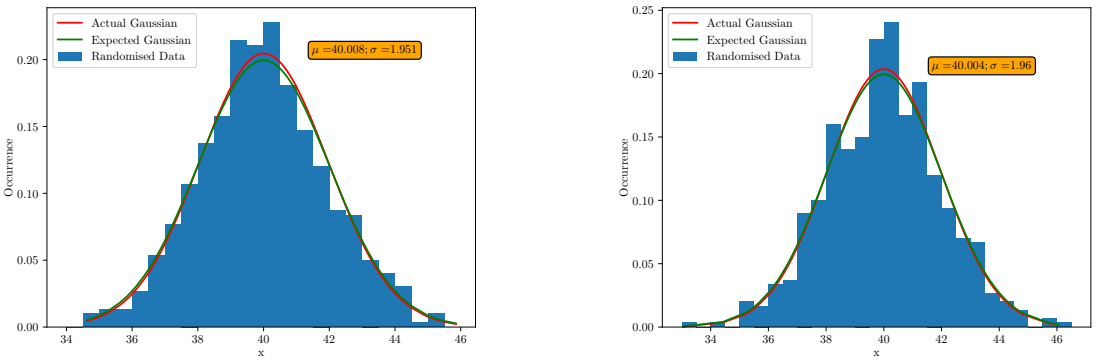

Figure 3: 600 Random points with specific μ and σ

- **Monte Carlo Determination of π**
  Finally, to see what else Monte Carlo simulations can do, we tried to determine an approximation of π. To do this, we generated a number of random coordinates within a 2×2 square around the origin. Statistically, these points should distribute evenly around the square, but if we check which ones fall within a circle of radius 1 centred on the origin and divide it by the number that fall within the square, which is all of them, we should have $\frac{\pi}{4}$, the ratio of the area of the circle to that of the square. Finally we multiply that number by 4 to find and approximation of π. All of this code is in Appendix 3 which, after 1 run with 100000 random coordinates, gave a value of $\pi = 3.13924$.

  We can improve the accuracy of this number by running the program many times and taking the mean. We can also increase the number of random coordinates as this will increase the "resolution" of the circle that we're measuring the area of, increasing the accuracy of the approximation. We will also find a standard deviation which, when divided by the square root of the number of times the program ran, would be our uncertainty for this value. So we ran the program from before 10000 times, producing the plot in Figure 4 with a value of $\pi = 3.141615 \pm 0.000052$.

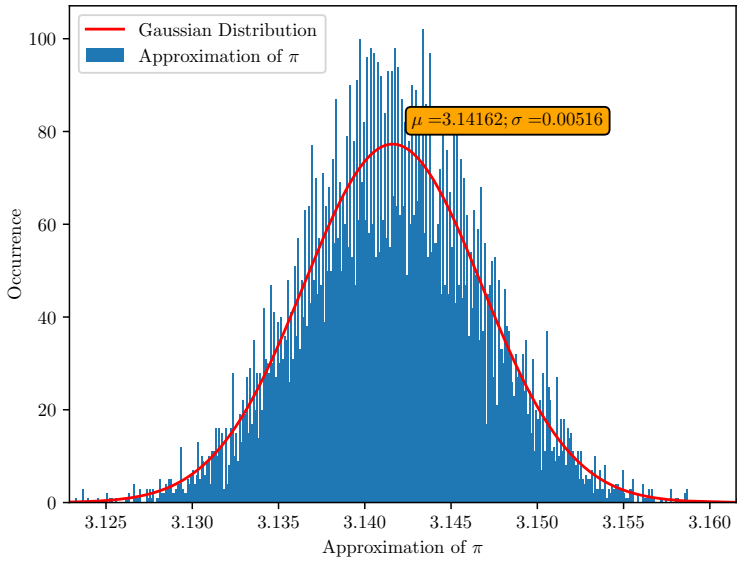

Figure 4: Monte Carlo Approximation of π

To compare the effect of multiple runs to the effect of more data points, we ran it 100 times with 10000000 coordinates, giving us a value of $\pi = 3.141591 \pm 0.000054$. The two runs effectively had the same number of evaluations and, as it turns out, resulted in roughly the same uncertainty. When comparing our value to the actual value of $\pi = 3.14159265$ it's clear that both values agree with the actual value of pi, but the value determined with "higher resolution", in this case at least, seemed to be more accurate when not considering the uncertainty. This makes sense as having more points will effectively give a more accurate measurement of the circle, and thus a more accurate approximation.

3 Appendix

Appendix 1: CP4_2a

```
from matplotlib import pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
from numpy import cos, pi, sin, sqrt, exp, random
import matplotlib
matplotlib.use('pgf')
matplotlib.rcParams.update({
    'pgf.texsystem': 'pdflatex',
    'font.family': 'serif',
    'text.usetex': True,
    'pgf.rcfonts': False,
})
# Creates a numpy array to hold the data
data = np.zeros(60)
i = 0
# Reads the data and puts it into the array
f = open(r'PHY2004W Computational\CP4\Activity1Data.txt', 'r')
header = f.readline()
for line in f:
    line = line.strip()
    columns = line.split()
    data[i] = float(columns[1])
    i += 1
f.close()
data.sort()
# A function to compute a Gaussian
def gaussian(x, mu, sigma):
    return (1/(sigma*sqrt(2*pi)))*exp(-((x-mu)**2)/(2*(sigma**2)))
# Computes values for the gaussian plot
dataMu = np.mean(data)
dataSigma = sqrt(np.var(data))
xplot = data
yplot = gaussian(xplot, dataMu, dataSigma)
gaussianMax = gaussian(dataMu, dataMu, dataSigma)
trueGaussian = gaussian(xplot, 40, 2)
# Defines some things for the histogram plotting
binwidth = 0.5
bins = np.arange(np.floor(min(data)), np.floor(max(data))+1, binwidth
    )
# Creates and plots the histogram
hist = plt.hist(data, bins, density=True, label='Activity1Data')
plt.xlabel('x')
plt.ylabel('Occurrence')
plt.draw()
# Plots the gaussian
gaussian = plt.plot(xplot, yplot, 'r-', label='Actual Gaussian')
```

```
trueGaussian = plt.plot(xplot, trueGaussian, 'g-', label='Expected Gaussian')
# Plots the graph
plt.legend()
plt.annotate(r'$\mu=$'+str(round(dataMu, 3))+'$; \sigma=$'+str(round(dataSigma, 3)), (dataMu, gaussianMax), textcoords='offset points', bbox=dict(facecolor='orange', edgecolor='black', boxstyle='round'))
# plt.show()
plt.savefig('PHY2004W Computational\CP4\Activity1DataHist.pgf')
```

Appendix 2: CP4_2b

```
from matplotlib import pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
from numpy import cos, pi, sin, sqrt, exp, random
import matplotlib
# matplotlib.use('pgf')
matplotlib.rcParams.update({
    'pgf.texsystem': 'pdflatex',
    'font.family': 'serif',
    'text.usetex': True,
    'pgf.rcfonts': False,
})
# Creates random data to use, with specific mean and standard deviation and writes it to a file
N = 600
data = random.normal(40, 2, N)
writeFile = open(r"PHY2004W Computational\CP4\RandomisedData.txt", 'w+')
writeFile.write("Random numbers drawn from Gaussian with mu = 40.0 and sigma = 2.0\n")
c = 0
for i in data:
    c+=1
    writeFile.write(str(c)+" "+str(round(i, 3))+"\n")
writeFile.close()
# Creates a numpy array to hold the data
data2 = np.zeros(N)
i = 0
# Reads the data and puts it into the array
f = open(r"PHY2004W Computational\CP4\RandomisedData.txt", 'r')
header = f.readline()
for line in f:
    line = line.strip()
    columns = line.split()
    data2[i] = float(columns[1])
    i += 1
f.close()
```

```
data2.sort()
# A function to compute a Gaussian
def gaussian(x, mu, sigma):
    return (1/(sigma*sqrt(2*pi)))*exp(-((x-mu)**2)/(2*sigma**2))
# Computes values for the gaussian plot
dataMu = np.mean(data2)
dataSigma = sqrt(np.var(data2))
xplot = data2
yplot = gaussian(xplot, dataMu, dataSigma)
gaussianMax = gaussian(dataMu, dataMu, dataSigma)
trueGaussian = gaussian(xplot, 40, 2)
# Defines some things for the histogram plotting
binwidth = 0.5
bins = np.arange(np.floor(min(data2)), np.floor(max(data2))+1,
    binwidth)
# Creates and plots the histogram
hist = plt.hist(data2, bins, density=True, label='Randomised Data')
plt.xlabel('x')
plt.ylabel('Occurrence')
plt.draw()
# Plots the gaussian and the expected gaussian
gaussian = plt.plot(xplot, yplot, 'r-', label='Actual Gaussian')
trueGaussian = plt.plot(xplot, trueGaussian, 'g-', label='Expected
    Gaussian')
# plt.show()
plt.legend(loc='upper left')
plt.annotate(r'$\mu=$'+str(round(dataMu, 3))+'$; \sigma=$'+str(round(
    dataSigma, 3)), (dataMu, gaussianMax), textcoords='offset points',
     bbox=dict(facecolor='orange', edgecolor='black', boxstyle='round'
    ))
plt.savefig('PHY2004W Computational\CP4\RandomisedDataHist600_1.pgf')
```

Appendix 3: CP4_3

```
from matplotlib import pyplot as plt
import numpy as np
from scipy.optimize import curve_fit
from numpy import cos, pi, sin, sqrt, exp, random
import matplotlib
matplotlib.use('pgf')
matplotlib.rcParams.update({
    'pgf.texsystem': 'pdflatex',
    'font.family': 'serif',
    'text.usetex': True,
    'pgf.rcfonts': False,
})
# Quick note, this will take quite a while to run, for me it took
    around 2 hours, so watch out
# Creates the random coordinates
numRuns = 100
```

```
N = 10000000
approxPi = np.zeros(numRuns)
for c in range(numRuns):
    xs = random.uniform(-1, 1, N)
    ys = random.uniform(-1, 1, N)
    inCircle = 0
    for i in range(N):
        if sqrt((xs[i]**2) + (ys[i]**2)) <= 1:
            inCircle += 1
    approxPi[c] = ((inCircle/N)*4)
    print((c/numRuns)*100, "%")
# A function to find the gaussian distribution
def gaussian(x, mu, sigma):
    return (1/(sigma*sqrt(2*pi)))*exp(-((x-mu)**2)/(2*sigma**2))
# Sorts things out for analysis, getting values we need
approxPi.sort()
dataMu = np.mean(approxPi)
dataSigma = sqrt(np.var(approxPi))
xplot = approxPi
yplot = gaussian(xplot, dataMu, dataSigma)
gaussianMax = gaussian(dataMu, dataMu, dataSigma)
trueGaussian = gaussian(xplot, 40, 2)
print("pi = ", dataMu, " +/- ", dataSigma/sqrt(numRuns))
# Defines some constants needed to make the histogram
binwidth = 0.0001
bins = np.arange(np.floor(min(approxPi)), np.floor(max(approxPi))+1,
    binwidth)
# Creates and plots the histogram
plt.hist(approxPi, bins, density=True, label=r'Approximation of $\pi$
    ')
plt.xlabel(r'Approximation of $\pi$')
plt.ylabel('Occurrence')
plt.draw()
plt.xlim(min(approxPi), max(approxPi))
# Plots the graph, along with the legend and an annotation of the
    mean and std dev
plt.plot(xplot, yplot, 'r-', label='Gaussian Distribution')
plt.legend(loc='upper left')
plt.annotate(r'$\mu=$'+str(round(dataMu, 5))+'$; \sigma=$'+str(round(
    dataSigma, 5)), (dataMu, gaussianMax), textcoords='offset points',
     xytext=(10, 10), bbox=dict(facecolor='orange', edgecolor='black',
     boxstyle='round'))
plt.savefig('ApproxPi3a.pgf')
# plt.show()
```