Exercise 3: Random Numbers and Monte Carlo

Oliver Sellers

University of Bristol

April 26, 2020

1 Introduction

This computational exercise focuses on the use of random numbers for Monte Carlo techniques. The following report documents analytical inversion and reject-accept methods to generate random numbers (x) proportional to $\sin(x)$; the simulation of experimental data for radioactive decay and gamma emission; and the modelling of statistical uncertainties for particle physics experiments.

2 Random Numbers

To implement the analytical inversion and reject-accept methods, random numbers had to be generated in a uniform distribution to be passed through each routine. For this, the numpy.random random number generator was chosen which utilises the Mersenne Twister algorithm. A random number generator produces a long string of numbers which aim to pass statistical tests for randomness. The length of the string for this generator, known as the period, is a Mersenne prime number $2^{19937} - 1$. The Mersenne Twister algorithm is one of the most heavily tested random number generators available today. Numpy.random automatically seeds this generator to choose a different starting position each time it is called. To do this, the module either uses the computers inbuilt random number generator or the time read from the computer's clock. The computer's clock is a constantly changing large number but, in some situations, it may not be suitable for seeding the Mersenne Twister. If the seeds are chosen systematically, the output sequences will be correlated and the randomness will be compromised [1]. This is significant for large scale simulations where many independent random number sequences are required.

Figure 1 shows the results from running the analytical inversion and reject-accept routines and analysing with the pyplot.hist histogram function to confirm the resultant distribution is proportional to $\sin(x)$. It can be seen that both routines give the required distribution with R^2 values of 0.993 and 0.994. These R^2 values were calculated using the sklearn.metrics r2_score function. The slight difference could indicate that the reject-accept routine does not follow the $\sin(x)$ distribution as well as the analytical inversion method. To explore this further, the fitting parameter was found for a range of generated number quantities and number of bins in the histogram. This is shown in figure 2, where the error bars are calculated by averaging over five runs of the algorithm.

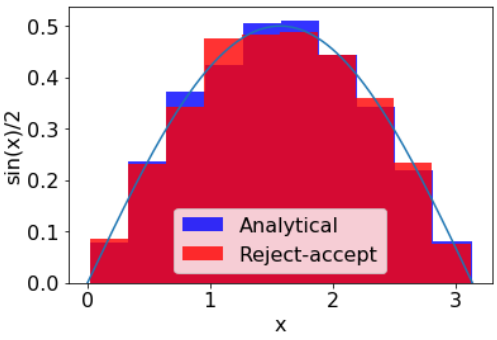

Figure 1: A histogram with 10 bins to show the distribution of 10,000 numbers generated with the analytical inversion and reject-accept methods. Both histograms have been fitted with a sine curve with R^2 values of 0.993 and 0.994 respectively.

It can be seen in figure 2 that for both routines the R^2 parameter approaches unity as the ratio of the quantity of generated number to the number of bins for the histogram increases. Both plots show large error bars for points with a small ratio suggesting a large variance in the fit quality. As the ratio increases, the error bars become smaller until they are no longer visible. In comparing the two routines, it appears that the analytical inversion method is more accurate for lower ratio values. Yet, both routines rapidly

approach unity showing that the required distribution can be accurately produced with both routines.

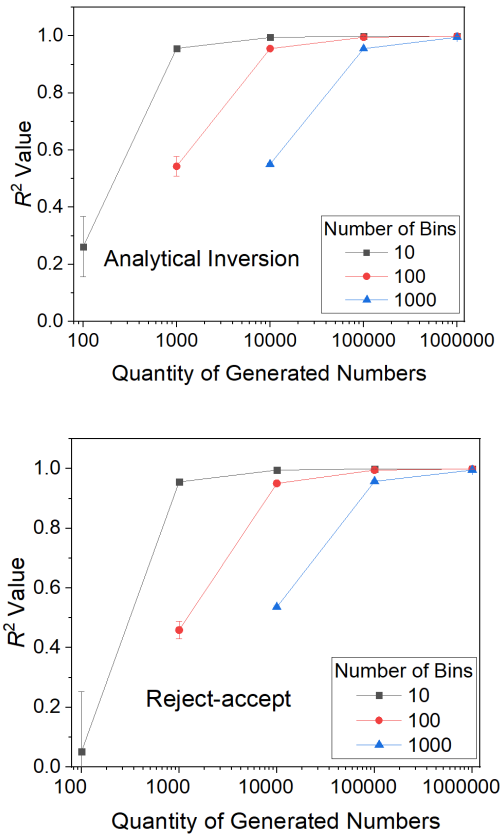

Figure 2: Plots showing the R^2 values for sine curve fitting to histograms with a range of bins and quantities of numbers generated.

Another way to compare the is efficiency. It was found that the reject-accept algorithm rejected 36.56 ± 0.6 % of the total numbers fed through the algorithm to achieve the quantities in figure 2. This is expected when the problem is treated as throwing random numbers at a sine curve in a box that is $\pi \times 1$ in area. The ratio of the area outside the curve to the total area of the box is 0.3634. If the random numbers are thrown uniformly, the ratio of numbers outside to total numbers will be this, agreeing with the rejection rate to within the specified error. Additionally, two arrays of random numbers need to be produced which adds significant time when the array become long. It is clear that the analytical inversion method is preferred but it is not always possible. In many Monte Carlo simulations, the functions being dealt with are complicated and the rejection rates can be high. Various approaches can be taken to minimise rejection and optimise efficiency in adaptive rejection sampling. However, in some situations, for example with higher dimensions, alternative methods must be used [2].

3 Simulations

The second task from the problem sheet was implemented again using the numpy.random module. In this case, the module was used to generate random number arrays in exponential and uniform distributions as well as using the the analytical inversion routine to generate sine distributed numbers. The exponential distribution was required to model at which point the nuclei decays and produces a gamma ray. This distribution was checked in the same way as the sine distributions from the previous section. A plot of this and the R^2 value is generated when the code is run. To model the isotropic distribution of gamma rays emitted, spherical polar coordinates were chosen. A random ϕ value from 0 to 2π was generated in a uniform distribution and a θ value from 0 to π was generated in a sine distribution to avoid clumping at the poles. This clumping is caused by the choice of coordinates being spherical polar. An area element in spherical coordinates is smaller at the poles as the lines of latitude and longitude are closer together. Therefore, choosing a uniform distribution for angles will evenly generate over area elements, leading to concentration at the poles. To normalise this area element, the θ values must be generated in a sine distribution [3]. Figure 3 shows the distribution of gamma rays from 1000 generated emissions. It can be seen that this follows an isotropic distribution over the surface of the unit sphere and there is no clumping at the poles.

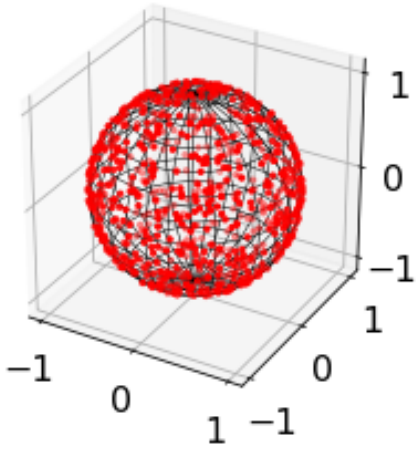

Figure 3: A unit sphere plot showing the distribution of gamma ray for 1000 generated emissions.

Code was then written to determine which emitted gamma rays would be detected on a certain array size using the spherical polar coordinate angles of emission and the point at which

the nuclei decayed. The emissions in the negative direction had to be disregarded by calculating a z component. The detector resolution was modelled by applying a smearing to the hit values generated with a numpy.random Gaussian distribution.

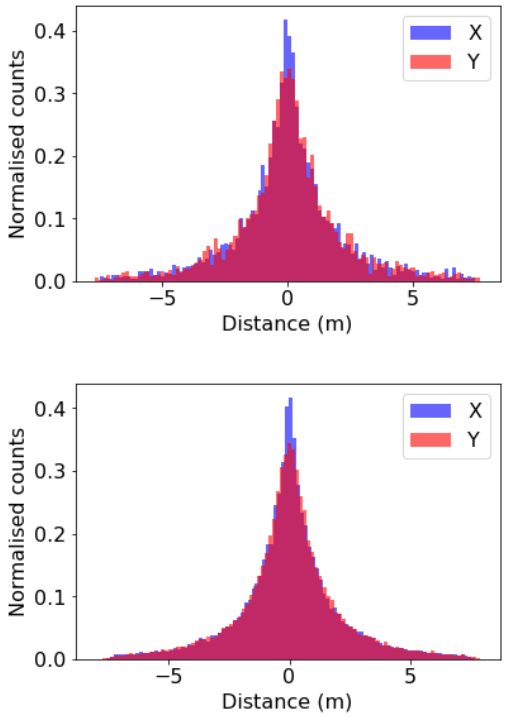

Figure 4: Plots showing the distributions of counts across a 15×15 detector array in the x and y direction. Top: 10,000 nuclei. Bottom: 100,000 nuclei.

Figure 4 shows the resulting distributions in the x and y directions for 10,000 and 100,000 nuclei. The first point to note is the difference in noise between the two plots. In the top plot, the general idea of the results can be taken but the bottom plot gives greater accuracy. This shows the number of simulated experiments must be sufficient in order to get the desired results. Although, in both plots a clear difference can be seen between the x and y distributions. The x distribution is more localised in the centre of the plot. This is the difference in resolution in the two directions leading to more uncertainty of position in y. To confirm this, the model was run without smearing and the distributions were identical along both directions. This final distribution was not fitted with a function due to the complexity from its constituent parts. The combination of an exponential decay, an isotropic spherical distribution, and Gaussian smearing leads to a complex final distribution.

The plot in figure 5 shows that a significant percentage of nuclei did not decay before the detector array. The code was such that these nuclei were ignored. It can be seen that as the quantity of generated numbers increases, the percentage not decayed converges just above 16.2 %. This reinforces the importance of running a sufficient number of experiments to get accurate results. The plot also shows that for each detector array size, the percentage of gamma rays to miss the array converges on a certain point. Initially the error bars are large, showing a range in the data produced over ten runs of the model. Following this, the error bars are no longer visible. As the detector array size increases, the percentage missed decreases. It is anticipated that this value will approach 50 % as the detector array tends to infinity. This is due to the gamma rays being emitted isotropically in all directions so the largest quantity that can conceivably be detected by an array on one side of the emission is 50 % of the total emissions for an infinitely large array.

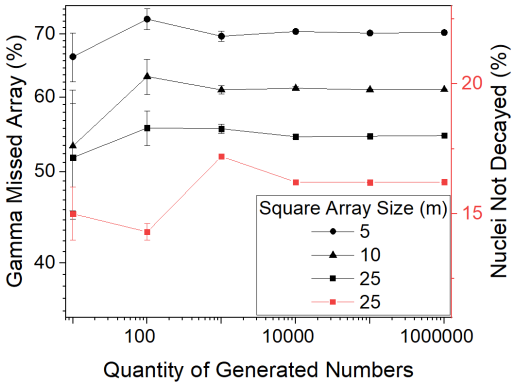

Figure 5: A plot showing the percentage of gamma rays that missed the array and the percentage of nuclei that did not decay before the array for varying quantities of generated numbers for a range of array sizes.

4 Statistical Solutions

The final task involved using numpy.random again to produce various distributions to simulate a particle physics experiment. The confidence level then had to be calculated by finding the percentage of the final distribution that was six events or more for an input cross section. Various cross sections were input to find the point at which the confidence level was 95 %, giving an upper bound on the cross section to that confidence level. To model the uncertainty in the luminosity, a random Gaussian distribution was used with the average luminosity and the spread. Modern particle physics experiments can determine the luminosity with high accuracy, a recent paper from Cern determines a luminosity value to 1.16 % [4].

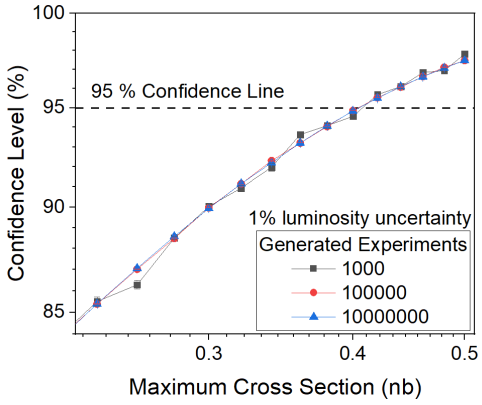

Figure 6: A plot showing the maximum cross section plotted against confidence level for a number of different quantities of generated experiments.

Figure 6 shows the confidence level on the upper bound of the cross section for different numbers of generated experiments. The error bars are calculated by averaging over five runs of the program, they are not visible for any points on the plot. It is clear that for 1000 generated experiments, there is more noise in the signal. Some points appear to be outliers from the general trend. As the number of generated experiments increases, the noise in the signal seemingly disappears. Very little, if any, difference can be seen between 100,000 and 10,000,000 generated experiments. To find the value of the cross section, the 10,000,000 generated experiments data was fitted with a 3rd order polynomial to find the point at which the 95 % confidence level is reached. The result of this is a cross section of 0.405, where the error was calculated from the errors on the fit parameters. The R^2 for this is 0.999995827 giving strong evidence for a good fit. Adding a 1 % luminosity uncertainty did not have a significant effect on the result, the effect that was found was a slight decrease in confidence level. This makes sense as the addition of another uncertainty reduces the confidence in the final result. The change in confidence was found to be an order of magnitude smaller than the uncertainty value. It is expected that the change will be small as the uncertainty in the luminosity is Gaussian, so it smooths the Poisson function that it is input to. This means the distribution spread has increased and the confidence will decrease but the effect is small compared to the other uncertainty and statistics of the system.

In figure 7 can be seen the distributions that result from a cross section of 0.4 for 10,000,000 generated experiments. The total distribution is a combination of the signal and the background with an average value of 10 events. One benefit that the Monte Carlo modelling method has for simulating experiment statistics is that the signal and background distributions can be seen; whereas, in the real experiment, only the total can be seen. This gives a better insight into the statistics and distributions involved in the experiment.

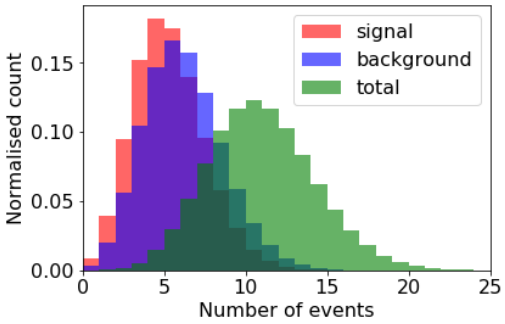

Figure 7: A histogram showing the distributions of signal, background, and total events for 10,000,000 generated experiments.

The use of random numbers in Monte Carlo methods has proven to be very important in modern science, from modelling particle physics to the spread of infectious diseases. Truly random numbers allow the simulation of very complicated real life systems that can not be solved from first principles. The models in this assignment could be improved by extending to more complex and larger simulations. In this case, a new random number generator method may need to be considered. Also, the models could adapted to include the detector counting efficiency. An additional improvement would be to include relativistic effects for the nuclei; however, their speed is of the order $10^{-}5$ c so these effects are not significant.

References

[1] A. Jagannatam, *Mersenne Twister – A Pseudo Random Number Generator and its Variants*, available online: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.175.9735, (2010).

[2] M.I. Jordan, *Monte Carlo Sampling*, Stat260: Bayesian Modeling and Inference, University of California, Berkeley, (2010).

[3] M.K. Arthur, *Point Picking and Distributing on the Disc and Sphere*, Army Research Laboratory ARL-TR-7333, (2015).

[4] The LHCb collaboration, *Precision luminosity measurements at LHCb*, European Organization for Nuclear Research (CERN), (2014).