Assignment 3: Random Numbers and Monte Carlo

INTRODUCTION

This assignment will look at implementing the use of pseudo-random number generators (PRNG's) into computational models. Many physical phenomena are random i.e radioactive decay, and being able to model different distributions is key.This investigation focused on two methods, an analytical method, and the accept/reject method. The analytical method converts the uniform random distribution generated using numpy.uniform into the desired distribution by the following method. [1]

$$\int_{x_0}^{x_{gen}} P(x)dx = \int_{x_0'}^{x_{req}'} P'(x')\,dx' \tag{1}$$

By equating the integrals of the generated distribution $P(x)$ and the desired distribution $P'(x)$, where, in our case, the generated distribution is uniform and $x_0 = 0$ such that

$$x_{gen} = \int_{x_0}^{x_{gen}} P(x)dx \tag{2}$$

We can then define the integral such that

$$Q\left(x_{req}'\right) = \int_{x_0'}^{x_{req}'} P'(x')\,dx' \tag{3}$$

and by using the inverse of the function Q we can find x_{req}' such that [1]

$$x_{req}' = Q^{-1}\left(x_{gen}\right) \tag{4}$$

The PRNG's used in the numpy module are based on the Mersenne Twister MT19937 [2], so called as it has a repeating period of $2^{19937}-1$ and generally regarded as a good PRNG [3]. Computers are deterministic, hence generating a truly random number comes with problems, unless based on a truly random external process. Computer algorithms can produce pseudo-random sequences of numbers, but they are based on an initial number or 'seed' and any sequences generated with that same seed will be identical, and at some point repeat, such as the Mersenne Twister. For the purposes of this assignment and most models a PRNG such as MT19937 will be sufficient. [3]

The second method we will test is the accept/reject method, which is particularly useful when the integral of the required distribution cannot be determined. In this method a number x is generated, a second number y is generated, if $y < P'(x)$ then we accept x, if it is above then it is rejected. This should be less efficient as more numbers are needed to be generated to produce the same distribution, this will be looked at in Task 1.

TASK 1

In this task a sinusoidal distribution is generated using the two methods described above. Applying the analytical method to $sin(x)$ between $0 < x < \pi$ we get

$$x_{\text{req}}' = Q^{-1}\left(x_{\text{gen}}\right) = \arccos\left(1 - x_{\text{gen}}\right) \tag{5}$$

Comparing both methods we obtain the following graphs,

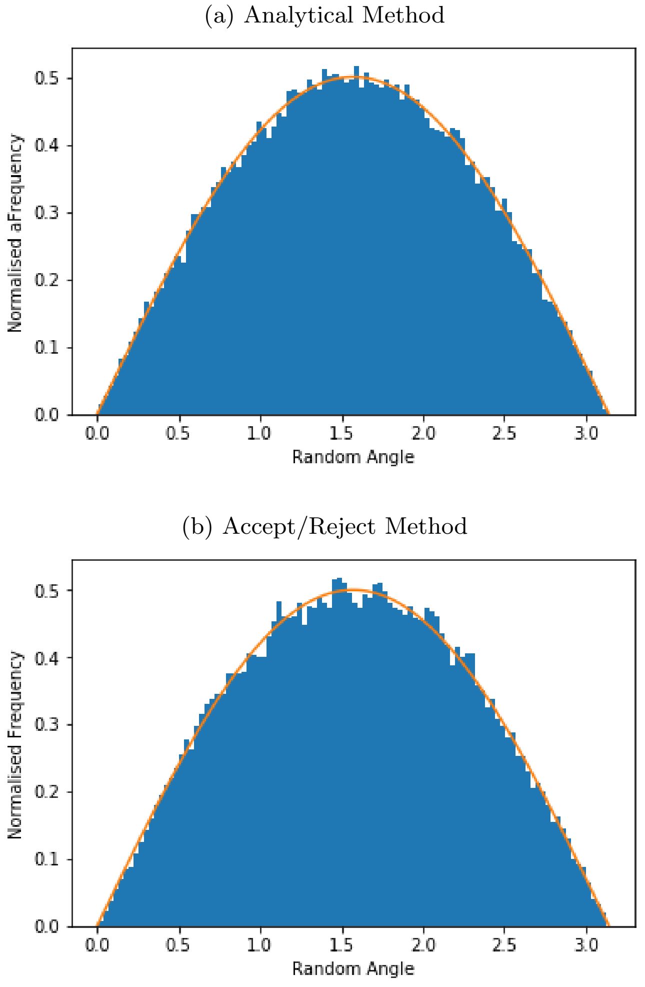


FIG. 1. The sinusoidal random number distributions for both methods, taken using 10^5 iterations. The orange line represents the desired sin(x) distribution. (a) had a chi-squared value of 0.059 and (b) had a value of 0.064.

We can see from the chi-squared values that the Analytical method is a closer representation of the distribution, and this appears to be true for all values of iterations as we would expect due to the accept/reject method rejecting a proportion of generated numbers, therefore the produced distribution would have an effective number of iterations similar to a distribution of fewer iterations analytically.

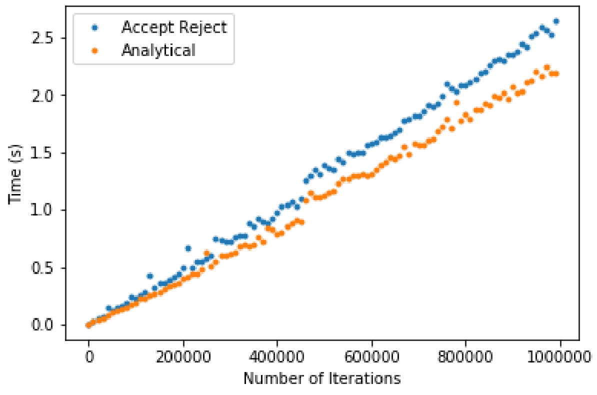


FIG. 2. Graph showing time difference between analytical and accept/reject methods for various total iterations.

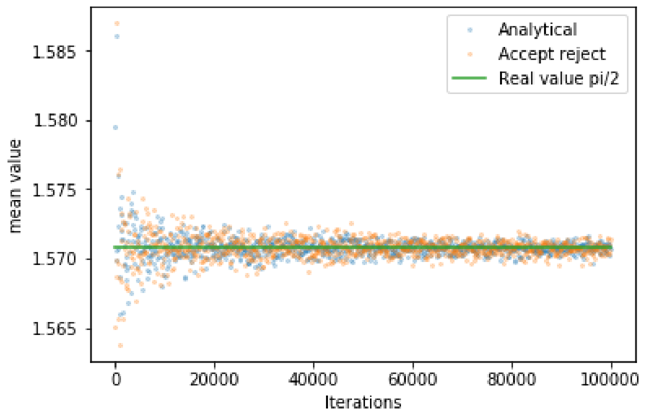
Looking at the computational time for each method at different number of iterations we see that with with increasing iterations, the time difference increases linearly, with the analytical method always being faster, as we would expect due to the accept reject method having to generate two numbers and to check each iteration, whereas the analytical method can generate one set of values in a numpy array and apply the inverse equation to the whole thing, reducing computation time. The linear relationship is also expected, as the number of calculations scales linearly with number of iterations, due to the simplistic nature of the sin(x) and inverse function.

Operations on numpy arrays are much more efficient than looping through lists in python as they are implemented in C. However the difference in run time is not too disparate and only reaches around 0.4s difference at 10^6 iterations, which only becomes an issue for large repeats. [5]

The comparatively low time difference is mainly due to the simplicity of the inverse function,complex inverse functions may even cause the accept/reject method to become the more efficient method, this would be a good extension to this task, to see if the latter can sometimes be a better method to implement even when the inverse can be calculated.

The expected value of the distribution should be equal to $\frac{\pi}{2}$, to see how well our routines converge to this value, data was taken over 50 repeats from a range of iterations up to 10^5, the distance from the expected mean was then plotted, these two graphs can be seen below in FIG.3.

(a) Distribution of mean values dots.png dots.png



(b) Difference from mean value graph.png graph.png

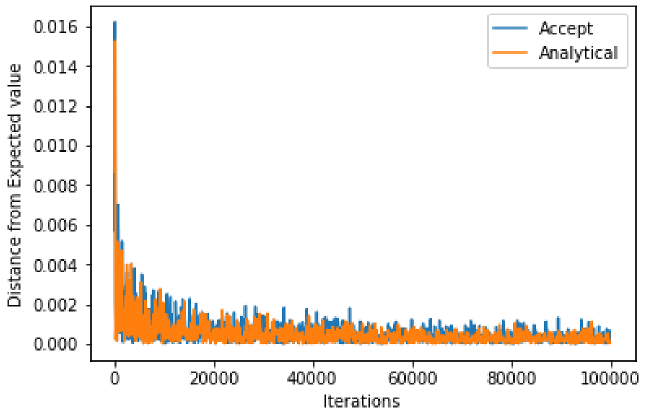


FIG. 3. (a) Plot of distribution of mean values for both methods, showing the convergence to the mean value. (b) Plot of distance to mean value, showing more quantitatively the convergence towards the mean.

From FIG.3a we can clearly see that the values do converge towards the mean as the number of iterations increases, however there is too much noise to see if either method does so at a faster rate than the other. In FIG.3b We clearly see that the analytical does, as we would expect, converge to the mean faster, being consistently closer to the expected value of $\frac{\pi}{2}$. This is due to the fact described above, where due to a certain number of generated numbers being rejected in the second method, the effective accuracy of the distribution is lowered when compared to the analytical method, which FIG.3b shows empirically.

TASK 2

In this task, the accept/reject method was implemented to model the radioactive decay from a beam of unstable nuclei using the Monte-Carlo method, which decay to produce a gamma ray in a random direction, and model the detections at a detector 2m away perpendicular to the axis of the beam, with a resolution of 0.1×0.3m in x and y respectively. First the decay position along the beam needs to be modelled. Radioactive decay follows the form,

$$N = N_0 e^{-d/v\tau}, \tag{6}$$

Where N is the number of decays, N_0 is the number of initial particles, d is the distance from source, v is the velocity (given to be 2000 m/s), and τ is the mean lifetime given to be 550μs. From this the distribution of decay distances can be found from the source as shown below in FIG.4, giving an exponential decay.

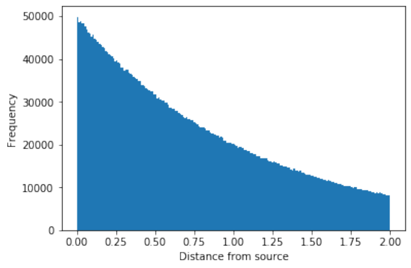


FIG. 4. Histogram showing the decay distance from the source for 10^5 generated numbers.

The isotropic direction of gamma decay was then modelled. To get an even distribution of angles in a sphere you cannot simply generate random angles, this is due to the fact that the differential area element is a function of the elevation angle ϕ

$$dA = r^2 \sin(\phi) d\phi d\theta \tag{7}$$

in spherical coordinates. Uniform random angles will give us a distribution bunched at the poles as can be seen in FIG.5.

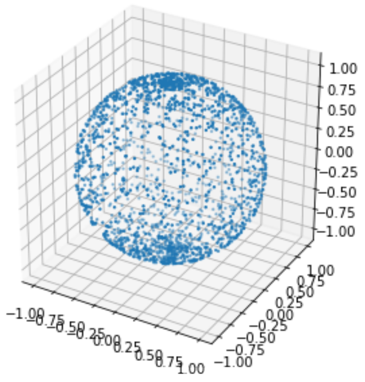


FIG. 5. Plot showing non-isotropic distribution of decay directions

To solve this the ϕ angles follow a non-uniform distribution of $\arccos(1-2\phi)$. This solves the problem seen in FIG.5.

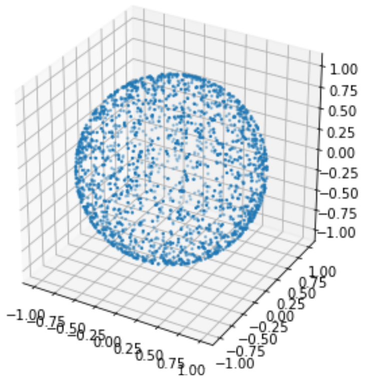


FIG. 6. Plot showing isotropic distribution of decay directions.

Once this is done the experiment can be simulated. A particle is produced and decays at some point along the beam with a gamma ray being given off in a random direction, the program then checks if it would hit the detector using trigonometry, and if it would hit the position is recorded. The graph below shows the distribution we obtain over 10^6 generated numbers. LogNorm() is used as with so many instances, every point gets a few hits and this makes it easier to visualise.

The resolution of the screen is also simulated as can be seen in the right graph in FIG.7. This is simulated by adding a small Gaussian distributed term to every point generated, modelling the error in readings due to resolution. This produces the effect we would expect, elongating the data in the y direction due to the unequal resolution in x and y direction.

TASK 3

In this task another implementation of Monte-Carlo simulation is used. In this case a collider experiment is looking for a new particle by counting the number of events observed to meet certain criteria.

The number of expected particle's to be produced is $\mathcal{L}\sigma$ where, $\mathcal{L} = 12$ /nb is the integrated luminosity and σ is the unknown cross-section. With a background of 5.7 ± 0.4, the limit on σ at 95% is to be found if the total number of candidate events is 5. To do this numpy.random.normal is used to generate a number in the Gaussian distribution around 5.7 with a standard deviation of 0.4, then a number is generated using a Poisson distribution with the expected value of the initial number, this is now the background count. Then a number is generated using Poisson again with the expected value of $\mathcal{L}\sigma$ for the σ being tested, this is the counts from the experiment. Summing the background and experimental counts, the total is obtained which if is more than 5, is a significant result. Repeating for many pseudo-experiments and scanning across a range of σ, a limit on σ that gives 95% significant events can be found, and therefore can claim is the candidate with 95% confidence.

To find the error on the cross section, a number of runs were taken at the value of the cross section found, with the standard deviation in the confidence being found, then scanning the array of σ values for the range that this standard deviation covers, giving the upper and lower error on the cross-section limit for 95% confidence. For constant luminosity, the limit on σ was found to be 0.404 ± 0.001 with 95% certainty and for luminosity with error ± 5 nb the limit on the σ was found to be 0.50 ± 0.08 nb with 95% certainty. FIG.8a shows the added error on the luminosity adds some random error on the confidence, which is reflected in our calculated error on the σ limit. By calculating the expectation value for the number of counts $\mathcal{L}\sigma$ from the cross-sections obtained, expectation values of number of events are found to be 4.84 ± 0.01 and 6 ± 2 respectively, which are reasonable estimates from the information given.

CONCLUSION

In conclusion the usefulness of PRNG's has been demonstrated, in task 1 two different methods of acquiring a probability distribution were shown, and while comparable, the analytical method is shown to be empirically better than the accept/reject method, in terms of efficiency and accuracy, though the accept/reject method still has its place where the integral of the PDF is not known. It is also simple to implement as no calculation of the inverse function is required, which can be complex.

In the second task, the Monte-Carlo method is used. Here, the known phenomena of radioactive decay is simulated, resulting in data we would expect with circular contours radiating outwards from the centre, for an incident beam on a perpendicular detector. The resolution of the detector array was also successfully implemented, showing the effect the resolution has on the data collected.

The third task demonstrates the simulation of a set of pseudo experiments (Toy Monte-Carlo) that are used to find a limit on the cross-section ,σ, of a particle experiment. It successfully created distributions where the limit on σ (cross-section) was found to be 0.404 ± 0.001 nb with 95% certainty and for luminosity with error ± 5 nb the limit on the cross section was found to be 0.50 ± 0.08 nb with 95% certainty. These are reasonable values, with decent sized error, the large error in the latter coming from the Luminosity error not from the routine.

The use of random numbers in computer modelling, through Monte-Carlo simulation was demonstrated, with two main methods used to obtain a probability distribution compared. Some physics experiments were then implemented successfully using routines based on these methods.

REFERENCES

[1] Devroye, Luc. "Nonuniform random variate generation." (1986), Springer-Verlag New York Inc.

[2] Matsumoto, Makoto, and Takuji Nishimura. "Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator." (1998), ACM Transactions on Modeling and Computer Simulation, TOMACS

[3] Jagannatam, Archana. "Mersenne TwisterA Pseudo Random Number Generator and its Variants.",(2008), George Mason University, Department of Electrical and Computer Engineering

[4] Casella, George, Christian P. Robert, and Martin T. Wells. "Generalized accept-reject sampling schemes.", (2004), A Festschrift for Herman Rubin. Institute of Mathematical Statistics.

[5] Van Der Walt, Stefan, S. Chris Colbert, and Gael Varoquaux. "The NumPy array: a structure for efficient numerical computation.", (2011), Computing in Science Engineering

APPENDIX

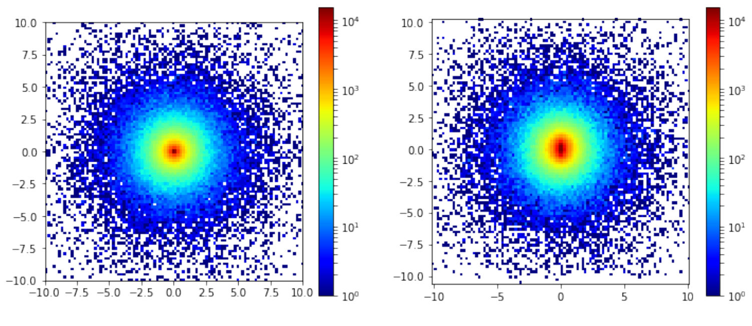
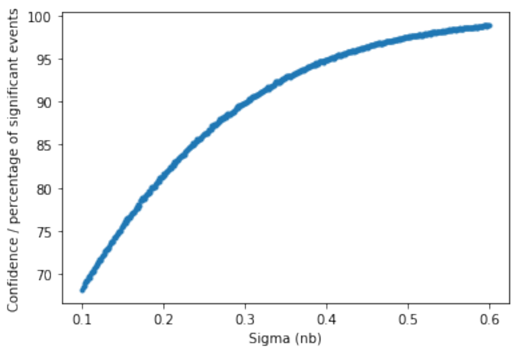
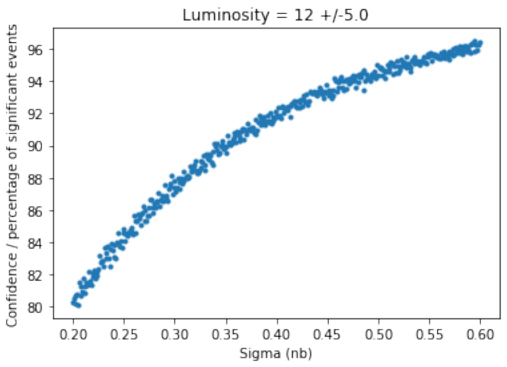


FIG. 7. Plot showing simulation of gamma ray detection from radioactive decays, with the plot on the right having additional error added to simulate resolution of detector array. Both taken with 10^6 iterations, and 100x100 bins.



(a)Graph of cross-section vs percentage of significant events simulated$\mathcal{L} = 12$ /nb



(b)Graph of cross-section vs percentage of significant events simulated with $\mathcal{L} = 12 \pm 5$ /nb

FIG. 8. Two graphs used to find cross-section limit at 95% confidence