Random Numbers and Monte Carlo

Ben Crabbe
Exercise 3, Computational Physics 301, University of Bristol.
(Dated: March 30, 2014)

Here we present and analyse the Inverse transform and reject-accept methods of converting uniformly distributed random variables to those of a specific distribution. These methods are then applied to an example of a monte carlo simulation of a simple particle physics experiment. We find them to both be highly effective, with only a slight performance increase for the inverse transform method.

RANDOM NUMBERS IN A DISTRIBUTION

Stock random number generators (RNGs) are generally designed to produce uniform deviates, numbers which behave as though each is selected independently from a uniform distribution. However, many applications of random numbers require them to be produced from a different underlying distribution. Here we present and analyse two methods for transforming from uniform deviates to those of a required distribution. Motivated by our purposes in the second part of this paper, we will use as an example the problem of producing angles between 0 and π in a sine distribution to demonstrate these methods.

For the production of our uniform deviates we have employed GNU Scientific Library (GSL)[1] implementation of the "Mersenne Twister" generator[2] with the recommended seed of 5489. This algortimn has been shown to pass the DIEHARD statistical tests[3] and has a period of around 10^{6000} making it suitable for our purposes.

Inverse Transformation Method

The Inverse Transformation takes a uniform deviate, x, between 0 and 1, and produces the deviate, x', as if sampled from the probability density function $P(x')$. To do this it uses the one to one mapping provided by the cumulative distribution function[4], $F(x')$, as shown in fig 1. Specifically,

$$x' = F^{-1}(x) = Q(x) \tag{1}$$

where $Q(x)$ is the inverse of the cumulative distribution, known as the quantile function.

In the case of $P(x') = sin(x')$ between 0 and π, after normalisation we find

$$F(x') = \frac{1}{2}\int_0^{x'} sin(x')dx' = \frac{1 - cos(x')}{2} \tag{2}$$

giving

$$Q(x) = cos^{-1}(1 - 2x) \tag{3}$$

In this case the method may be achieved analytically since $P(x)$ is integrable and $F(x)$ is invertible, however for most distributions this is not possible, then requiring some numerical approximations which often compromise the performance this method. [6]

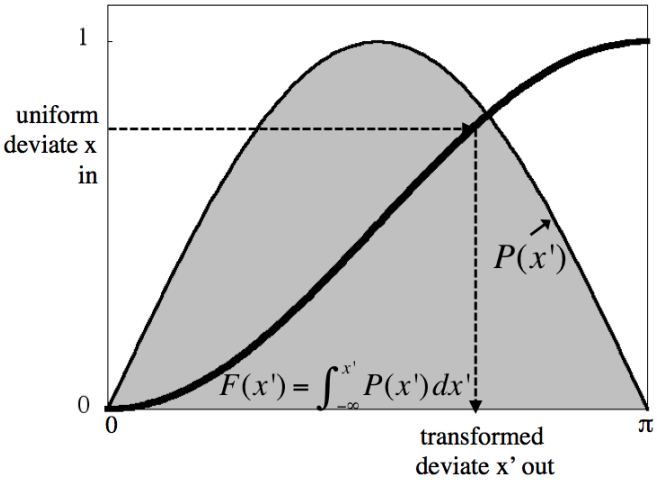


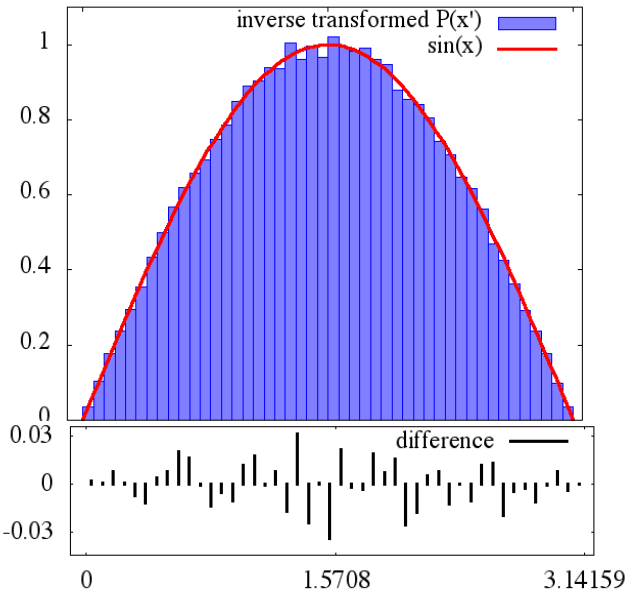
FIG. 1: An illustration of the inverse transform. A uniform deviate is transformed to one between 0 and π with a probability density function of sin(x), using the cumulative distribution function $F(x')$. Figure inspired by [5].

In figure 2a), 1×10^5 samples have been used to produce the probability density function, normalised to an area of 2. Compared to sin(x) it is clear that the differences are only statistical, the sizes of which depend on the number of bins, which we have chosen to be $\sim \sqrt[3]{1 \times 10^5} \simeq 46$

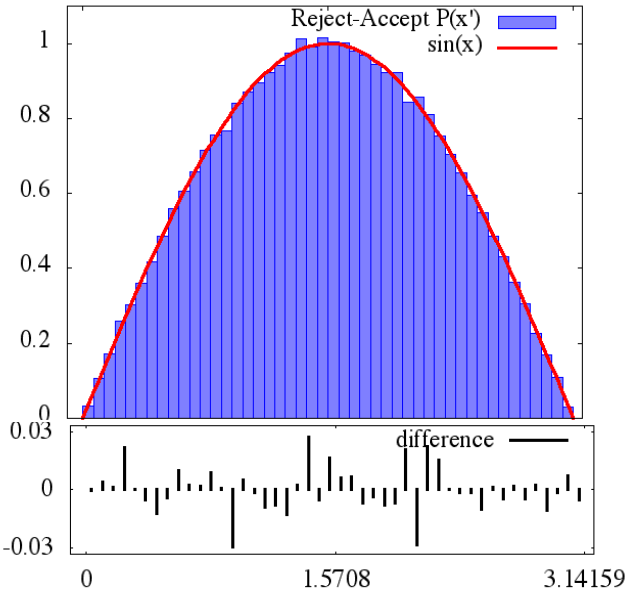
Reject-Accept Method

Rejection sampling is a versatile method which may be used to sample the complicated distributions where the Inverse Transform method fails. Its works by taking an area which encloses the desired distribution across its range and generating a random coordinate somewhere in this area using two uniform deviates. Then, if this coordinate falls with the desired $P(x)$ we keep it, if not, it is discarded. For our example of $P(x') = sin(x)$ we take the enclosing volume to be the rectangular area $x : 0 \rightarrow \pi, y : 0 \rightarrow 1$. However, this area does not necessarily have to be rectangular, often the performance of the algorithm can be improved by choosing an enclosing area which is closer to the desired distribution, one then uses the inverse transform method to generate the coordinate in this area as shown in figure 3.

The probability density function of the deviate produced us-



(a) For the inverse transform method.



(b) For the reject-accept method.

FIG. 2: The probability density functions of the produced deviates compared with the sine function, and below, P(x') - sin(x) for each bin. We observe that the distribution is well replicated by both methods with the differences appearing to be only statistical in nature. 1×10^5 deviates and 46 bins have been used in each sample.

ing this method is shown in fig 2 b). As before, the normalised distribution agrees with sin(x) to within statistical errors. We find that we must generate on average 1.57229 samples before one is accepted, which is approximately equal to ratio of the area of the enclosing square to that under the curve i.e. $\pi/2$

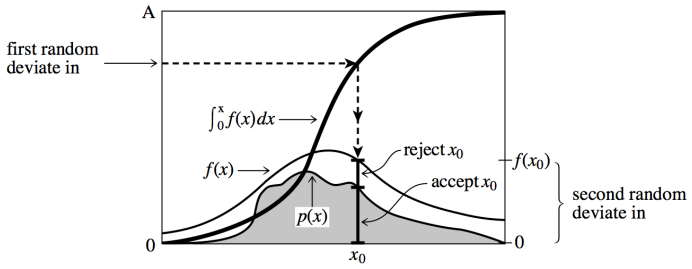


FIG. 3: An illustration of the reject-accept method for a desired distribution $p(x)$, first using the inverse transform method to generate a point from the enclosing distribution $f(x)$, we then accept this point if a second uniform deviate is less than or equal to the ratio $p(x)/f(x)$. Figure from [7].

For further comparison of the statistical performance of the two methods, the first six central moments of these distributions have been calculated and are compared with those of sin(x) in table I. Both methods appear to perform to roughly the same accuracy. It is noted that the performance in these tests is effected by the choice of seed, a non optimal choice of seed can lead to a factor of ten increase in the combined difference in table I.

Moment	sin(x)	Reject-Accept	Inverse Transform
1st (mean)	$\pi/2$	0.000444	0.002296
2nd	0.46779	7.6E-05	7.3E-05
3rd	0	0.000139	0.000529
4th	0.480343	0.000181	0.000277
5th	0	0.000895	0.001665
6th	0.646836	0.000259	0.00105
Combined Difference:		0.001994	0.00589

TABLE I: Displays the first six statistical moments of a sine distribution between 0 and π and the absolute difference between those calculated from the data displayed in figures 1 a) and b). Again the acuracy is limited by the bin size.

On the execution time of each method; we expect the reject-accept method to take additional time due to the number of events thrown away, however, as can be seen in figure 4, we find the difference in execution time to be negligible. This may suggest that the time taken to compute each quantile function is large compared to the time taken to generate each random deviate (since we generate two for each in the reject-accept method), however it is difficult to draw strong conclusions due to the inconsistency of CPU process scheduling in typical computers[8].

SIMULATING GAMMA DECAYS

Monte Carlo methods are used frequently in particle physics throughout the experimental process, from detector

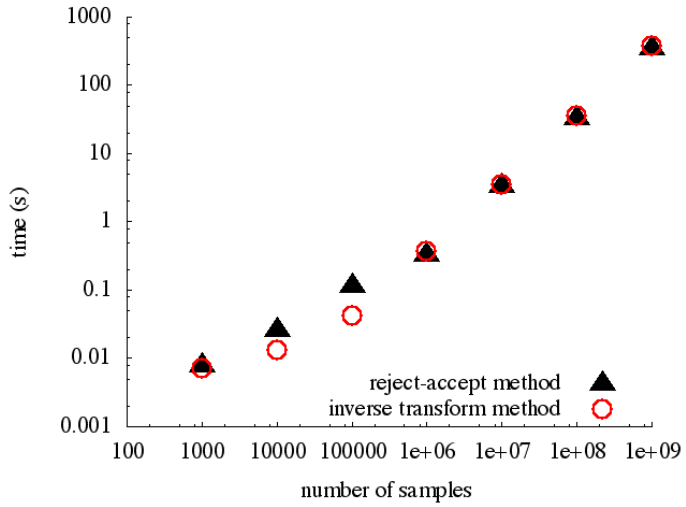


FIG. 4: A comparison of the execution of each method. We find there to be little discernable difference in performance.

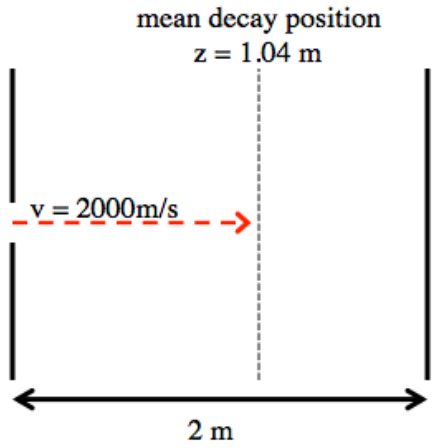


FIG. 5: Unstable nuclei are produced at z=0 where they enter a 2 m decay volume travelling at 2000ms^{-1}. The nuclei decay via gamma decay with a mean lifetime of 520μs, at this speed the Lorentz factor is negligible leading to a average decay position of 1.04 m. The gamma rays are produced isotropically. At z = 2 m there is a 10m^2 detector array which measures the position of incident gamma rays with an uncertainty of 10 cm in the x direction and 30 cm in the y direction.

design to data analysis. For example, simulations of a data sample of pure signal can be made and then compared to actual measured data to give an estimation of the background present in an analysis. These simulations are produced using random variables to model all the possible variations e.g. decay vertex, particle momenta, energy, decay products, which are then "measured" as if they were real particles, using additional random variables to simulate the imprecision of the detector.

For this second task a Monte Carlo generator for a simple experiment, illustrated and described in figure 5, has been produced using the methods already described for random variable generation.

To start with the lifetime of each particle is generated (in μs) randomly from a poisson distribution with a mean of 520, using the GSL gsl_ran_poisson function. This is multiplied by the velocity of the particle to give the decay position, which we then require to be between 0 and 2 meters since the decay must happen within the acceptance of the detector. We then generate a random trajectory for the emitted gamma ray, in the program task2RejectAccept.c this is done by generating points uniformly within a cube of side 2, then rejecting those that are at a radial distance greater than 1, leaving only those within a unit sphere which are then normalised. In the program task2InverseTransform.c we directly generate a point on the surface of the unit sphere using two uniform deviates z and ϕ, where $-1 \geq z \geq 1$ and $0 \geq \phi \geq 2\pi$, the z value is then transformed to a polar angle $\theta = sin^{-1}(z)$ where $-\frac{\pi}{2} \geq \theta \geq \frac{\pi}{2}$ with a $P(x)$ proportional to $cos(x)$. Then the rectilinear coordinates of this point are given in the usual manner.

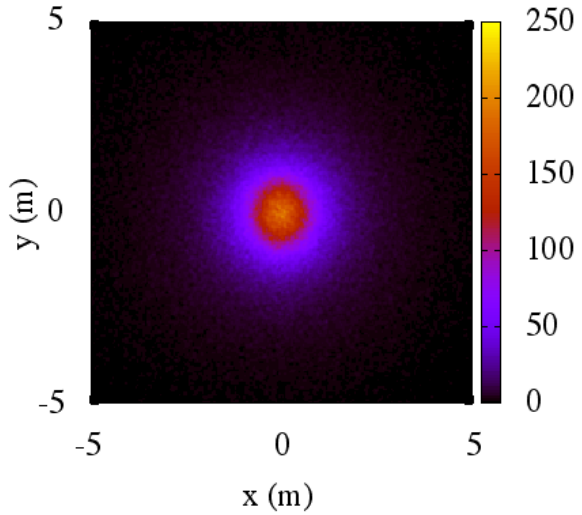
If the gamma ray travels in the positive z direction we then extrapolate its trajectory to the plane of the detector to give the true hit position. We then add to each of the coordinates a gaussian deviate, generated using the GSL implementation of Marsaglia's ziggurat algorithm [9], corresponding to the detector resolution in each direction. This particular method of generating normally distributed random variables was chosen for performance considerations, since it requires fewer operations per call than alternative methods.[10]

Hit maps of the detected gamma rays are shown for each method in figure 6, as expected we find the hits clustered at the origin in a roughly gaussian distribution, although with non gaussian tails as can be seen in fig 7. The measured standard deviations differ by $\sim$ 0.02 m for x and y which corresponds exactly to the additional 20cm of smearing in the y direction caused by the detector resolution. To further understand the effect of the detector resolution the true hit position values have been plotted in figure 8, we then find that the x-y standard deviations differ by only 0.0001 which can be attributed to statistical fluctuations. It can be seen by eye that the distributions in figure 6 are slightly "peanutted" in comparison, with a slightly higher density of hits at the origin in figure 8.

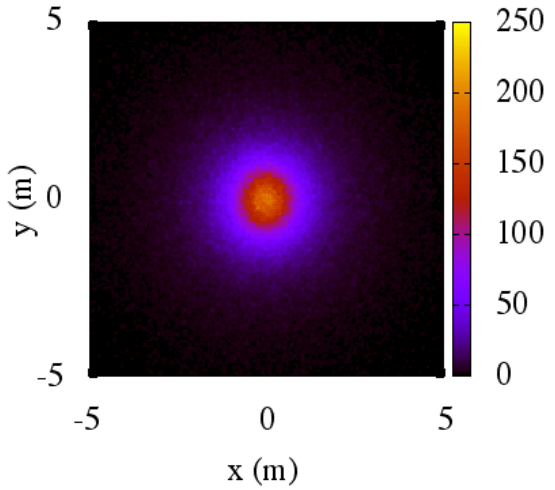
As before we find little difference in performance between the two methods, the execution time for each can be seen in figure 9.

[1] *GNU Scientific Library Reference Manual*, Edition 1.0, for GSL Version 1.0 (2001)

[2] *Mersenne Twister: A 623-dimensionally equidistributed uniform pseudorandom number generator*, M. Matsumoto and T. Nishimura, ACM Transactions on Modeling and Computer Simulations, Vol. 8, (1998), **3-30**.



(a) Using the inverse transform method we measured 414556 hits with an average FWHM of 3.584406.



(b) Using the reject accept method we measured 415332 hits with a average FWHM of 3.58362.

FIG. 6: Hitmaps showing the measured position of the gamma rays for 1×10^6 events histogrammed using 200×200 equally sized bins.

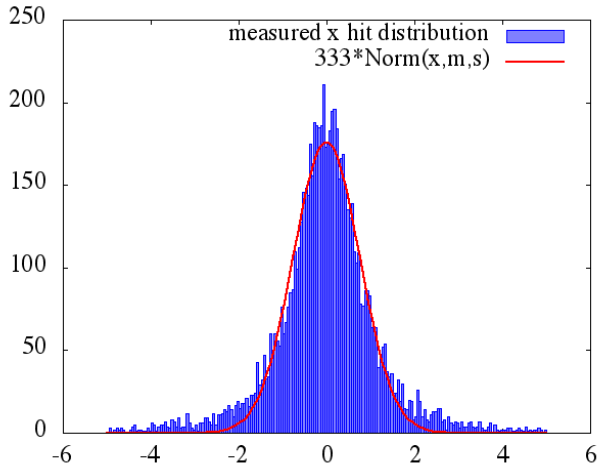


FIG. 7: Shows the measure hit distribution in x at $y = 0$. We see that the distribution is approximately gaussian in shape, with some non gaussian behavior in the tails.

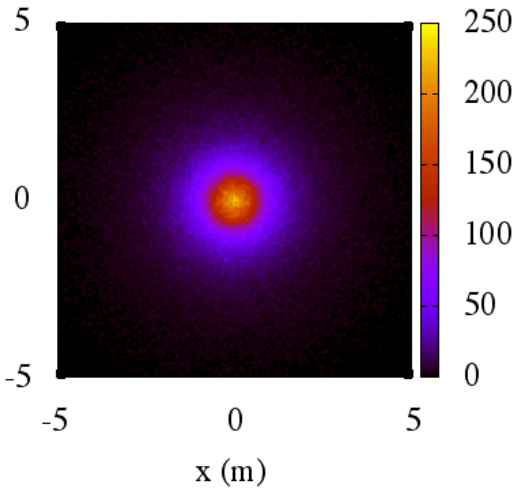


FIG. 8: Shows the hit map for true gamma position at the plane $z = 2$ m. Taken under the same conditions as figure 6, we find the FWHM decreased to 3.555 and roughly equal for x and y.

[3] *DIEHARD Statistical Tests*, G. Marsaglia, http://www.stat.fsu.edu/pub/diehard/ Accessed March 30, 2014.

[4] *The Art of Computer Programming, Volume 2: Seminumerical Algorithms*, D. Knuth, Addison-Wesley, third edition (1998), **2.2**.

[5] *Numerical Recipes*,W. H. Press et al, Cambridge University Press, second edition (1973), FIgure **7.2.1**.

[6] *Non-Uniform Random Variate Generation*, L. Devroye, Springer-Verlag, first edition (1986) **2.2**.

[7] *Numerical Recipes*,W. H. Press et al, Cambridge University Press, second edition (1973), **7.3**.

[8] *Modern Operating Systems*, A. Tanenbaum, Pearson International, third edition (2008) **753-4**.

[9] *The Ziggurat Method for Generating Random Variables*, G. Marsaglia, W. W. Tang, Journal of Statistical Software, 5 (2000) **8**.

[10] *Gaussian Random Number Generators*, D. B. Thomas, P. Leong, W. Luk, J. D. Villasenor, ACM Computing Surveys, 39, 4 (2007) **11**.

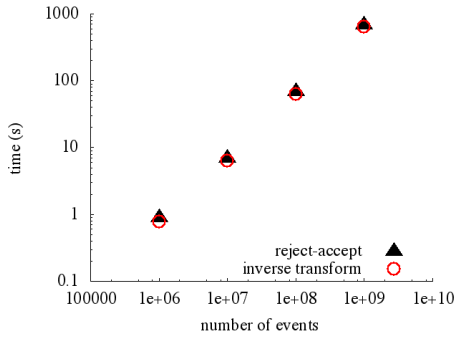


FIG. 9: Shows the excution time taken to simulate progressively larger numbers of events for the two methods of generating the gamma trajectory. Again we find only a small difference in performance between the two methods.