# SECTION 8
# Random Numbers and Monte Carlo Methods

## Outline of Section

- Pseudo-Random Numbers
- Transformation Method
- Rejection Method
- Monte Carlo Minimisation
- Monte Carlo Integration

We first look at how to generate a uniform distribution of pseudo-random numbers in the range 0 to 1. We then see how to use this to generate arbitrary non-uniform distributions (e.g. triangular, Gaussian, etc.). Such distributions can directly be used in physical systems where randomness occurs, e.g., to determine the random direction of a particle emitted by a decay process. Another example is a ***stochastic force*** $\vec{F}_{\text{coll}}$ such as that producing Brownian motion of a small grain in a fluid, due to collisions with the fluid molecules.

Monte Carlo Methods are a broad class of methods where random numbers are used to statistically solve a problem. The 'problem' might be:

(a) A deterministic system where an enormous number of things are coupled together, and exact solution is unfeasible. Classical Brownian motion system is an example. Here it is impractical to actually follow the classical motion of the all the individual molecules. However, the statistics of all the collisions in a small time yield a distribution of net force on the grain. This can be randomly sampled.

(b) A statistical physics problem, such as calculating macroscopic thermodynamic quantities given the energy levels of the system.

(c) Minimising a function of many variables.

(d) Performing an integral over many variables/dimensions.

Generally, Monte Carlo Methods are useful for systems of many variables or dimensions.

Here we concentrate on the last two types of problem. They actually get used in solving statistical physics problems anyway. (The first problem type is straight forward to tackle if the appropriate statistical distribution is known, simply requiring knowing how to generate a random distribution.)

## 8.1. Random Numbers

There are countless uses for random numbers in computational physics. Generally we need random numbers when we are simulating stochastic systems, e.g., Brownian motion, thermodynamical systems, state realisations and when we use Monte Carlo methods to calculate the variability of stochastic systems (more on this later).

A **uniform deviate** is a random number lying within a range where any number has the same probability as any other. The range is usually $0 \leq x \leq 1$. Here $x$ is a *random variable* and the deviates are the values that $x$ can randomly take.

Most `C` implementations come with a uniform random number generator that returns **pseudo-random numbers**. This means that the same sequence of (seemingly) random numbers can be regenerated by using the same **seed** number to start the sequence. This is a useful feature for debugging codes and reproducing results.

A simple implementation of a uniform deviate generator is the **linear congruential generator**

$$\boxed{I_{n+1} = (a\, I_n + c)\%m \ ,} \tag{8.1}$$

where $I_n$ are the random numbers, $a$ is a multiplier, $c$ is the increment, and $m$ is the modulus. ($a\%b$ denotes modulo division; i.e. the remainder of dividing $a$ by $b$.) The parameters $a$, $c$ and $m$ are all integers. This will iteratively generate a sequence of pseudo-random **integers** in the (closed) interval between 0 and $m-1$ (usually stored as `RAND_MAX` in `C` implementations). We can turn the results into a real number by dividing by $m$

$$x_n = \frac{I_n}{m} = \frac{I_n}{\texttt{RAND\_MAX}+1} \qquad \text{where} \qquad 0 \leq x_n < 1 \ . \tag{8.2}$$

One problem with this method is that the sequence of numbers will repeat itself with periodicity which is at most $m$, and for unfortunate choices of $a$, $c$ & $m$ the sequence length is much less than $m$ (with a significant portion of integers in the range $0 \leq I < m$ being skipped). So generally we want a large $m$ and a choice of $a$ and $c$ that ensures that the period is equal to $m$. You should be wary of the `rand()` function supplied with a compiler, especially `C` and `C++` compilers. The ANSI C standard specifies that `rand()` return type `int`, which can be as small as 2-bytes on some systems (i.e. `RAND_MAX`=32767). This is is not a very long period for Monte Carlo applications! These typically require millions of random numbers.

Another problem with linear congruential generators is that there is correlation between successive numbers; if $k$ successive random numbers are used to plot points in $k$-dimensional space, then the points do not fill up the space, but lie on distinct planes (of dimension $k-1$).

It is best to use random number generators other than the standard `C` one. There are good ones in the GSL libraries. Of note is the "Mersenne Twister" generator with an exceptionally long period of $m = 2^{19937} - 1 \sim 10^{6000}$, and the "RANLUX" generators which provides the most reliable source of uncorrelated numbers. In `Python` the basic `random()` function uses the Mersenne Twister generator.

## 8.2. Non-uniform distributions – Transformation Method

Often we need random deviates with a distribution other than uniform. The most common requirement is for a Gaussian distribution which is ubiquitous due to the Central Limit Theorem.

Given a generator that returns a uniform deviate $x$ in the range 0 to 1, we can use the **transformation method** to obtain a new deviate (random variable) $y$ which is distributed according to a *probability density function* (PDF) $P(y)$. Recall the basic properties of a PDF;

- A PDF must be positive, i.e., $P(y) \geq 0$.
- A PDF must be normalised, i.e., $\int_{-\infty}^{\infty} P(y)\,dy = 1$.

We are given $x$ with PDF

$$U(x)\,dx = \begin{cases} dx & 0 \leq x \leq 1 \\ 0 & \text{otherwise} \end{cases} \quad . \tag{8.3}$$

which is also normalised; $\int_{-\infty}^{\infty} U(x)\,dx = 1$. We want our new random variable $y$ to be a function of $x$, i.e., $y(x)$. The fundamental transformation law of probabilities then relates the PDFs of each variable via

$$|P(y)\,dy| = |P(x)\,dx| \quad , \tag{8.4}$$

where $P(x) = U(x)$ in our case. This satisfies the requirement that both PDFs integrate up to unity. Assuming $dy/dx \geq 0$ (so that the $|\ldots|$ can be discarded) we have

$$\frac{dx}{dy} = P(y) \quad ,$$

since $P(x) = U(x) = 1$. We can then integrate up and define the function

$$\boxed{F(y) = \int_{-\infty}^{y} P(\tilde{y})\,d\tilde{y} \quad ,} \tag{8.5}$$

and then using the fundamental transformation law of probabilities it follows that

$$F(y) = \int_{-\infty}^{y} \frac{d\tilde{x}}{d\tilde{y}}\,d\tilde{y} = \int_{0}^{x} d\tilde{x} = x \quad . \tag{8.6}$$

(Note that $\sim$ is used to denote dummy integration variables here.) So if $F(y)$ is an invertible function, i.e.,

$$\boxed{y = F^{-1}(x)} \tag{8.7}$$

can be found, we can map each $x$ given by a uniform deviate generator into a new variable $y$ which will have the desired PDF $P(y)$. Note that $F(y)$ is just the **Cumulative Distribution Function** (CDF) of $y$. Since the PDF is normalised, the range of $F(y)$ is $0 \leq F(y) \leq 1$. The transformation method is illustrated in fig. 8.1.

**Example** – consider the *triangular* distribution $P(y) = 2y$ with $0 < y < 1$. The CDF is $F(y) = y^2 = x$ such that the transformation $y = \sqrt{x}$ gives deviates with the PDF $P(y)$.

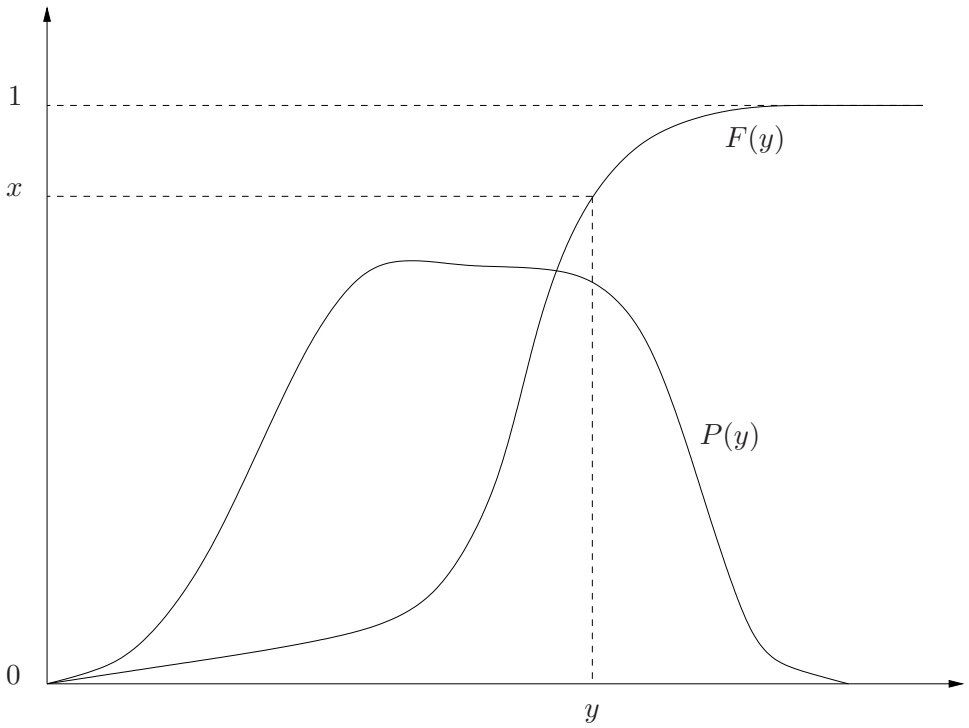


**Figure 8.1.** The transformation method. The Cumulative Distribution function $F(y) = \int_{-\infty}^{y} P(\tilde{y})d\tilde{y}$ is inverted to find a deviate $y$ for every uniform random deviate $x$.

## 8.3. Non-uniform distributions – Rejection Method

If the CDF is not a computable or invertible function then we can use the **rejection method** to obtain deviates with the required distribution.

We first draw a new function $f(y)$ called the ***comparison function*** which lies above $P(y)$ for all $y$. For simplicity let's define

$$f(y) = C \ , \tag{8.8}$$

where $C > P(y)$ everywhere. The procedure is then as follows:

(a) Pick a random number $y_i$, uniformly distributed in the range between $y_{\min}$ and $y_{\max}$.
(b) Pick a second random number $p_i$, uniformly distributed in the range between 0 and $C$.
(c) If $P(y_i) < p_i$ then reject $y_i$ and go back to step (a). Otherwise accept.

**Accepted $y_i$ will have the desired distribution $P(y)$.** To prove this, consider the probability that the algorithm above creates an acceptable $y_i$ in the range between $y$ and $y + dy$; this is

$$\text{Prob} = \frac{dy}{y_{\max} - y_{\min}} \times \frac{P(y)}{C} = \text{const} \times P(y)\, dy \, . \tag{8.9}$$

**Efficiency** – The efficiency of the rejection method is obtained by integrating the probability of acceptance which, since $P(y)$ is normalised, gives

$$\text{Eff} = \frac{1}{C} \frac{1}{y_{\max} - y_{\min}} \ . \tag{8.10}$$

1/Eff is the average number of times the steps (a)–(c) have to be carried out to get a $y_i$ value. A way to interpret (8.10) is that it is the ratio of areas under the desired PDF $P(y)$ (perhaps a peaked curve) and the cover function $f(y)$ (a rectangle that fits over the PDF). By definition the PDF's area is $A_P = 1$. For the cover function $A_f = C \times (y_{\max} - y_{\min})$. Thus

$$\text{Eff} = \frac{A_P}{A_f} \ .$$

The rejection algorithm above effectively randomly picks a pair of coordinates $(y_i, p_i)$ uniformly over the area $A_f$. A fraction of these 'throws' fall outside the area $A_P$ under the PDF, in which case the coordinates are rejected and another throw is taken.

The efficiency can be improved by using a comparison function $f(y)$ that conforms more closely to the desired $P(y)$ (but still lies above it) and is suitable for use in the transformation method (i.e. it has invertible definite integral $\int_{y_{min}}^{y} f(\tilde{y}) d\tilde{y}$). Step (a) then becomes;

(a) Using the transformation method, pick a random deviate $y_i$ in the range between $y_{\min}$ and $y_{\max}$, distributed according to the PDF obtained by normalising $f(y)$.

Note that since $\int_{y_{min}}^{y_{max}} P(y) dy = 1$ and $f(y) \geq P(y)$ then the area under $f(y)$ is bigger than unity, so $f(y)$ needs to be normalised to be a PDF.

## 8.4. Monte Carlo Minimisation – Simulated Annealing

An application of the Monte Carlo approach is to the problem of minimisation (optimisation) of functions. In section 7 we saw so called *direct search* methods which did not involve random numbers. Simulated annealing and related methods for optimisation introduce some randomness in the search for the minimum. They are particularly useful in cases where;

- The degrees of freedom in the system are so many that a direct search for an optimal configuration is too time consuming.
- Many local minima exist in which direct search methods can get stuck instead of finding the global minimum.

The simulated annealing approach is motivated by thermodynamics where random fluctuations can temporarily move a system to a less optimal (i.e. higher energy) configuration. The analogy is quantified in the use of the **Boltzmann Probability Distribution**

$$P(E)\,dE \sim \exp\left(-\frac{E}{k_{_B}T}\right)\,dE \quad , \tag{8.11}$$

where 'energy' $E$ encodes a **cost function** and 'temperature' $T$ is a variable which determines the probability of changes in the energy of the system. In particular, even for low temperatures, it allows for the system to move to higher energies with very low probability. This is what can allow the system to get 'unstuck' from local minima and continue the search for a global minimum.

In practice the method involves the use of a **Metropolis Algorithm** where at each step in the iteration the configuration of the system is changed randomly. The energy of the system before ($E_1$) and after ($E_2$) the change are calculated to get the change in the system's energy $\Delta E = E_2 - E_1$. The step is **accepted** with a probability $p_{acc}$ given by the simple algorithm

$$\boxed{p_{acc} = \begin{cases} 1 & \text{if} \quad \Delta E \leq 0 \\ \exp(-\Delta E/k_{_B}T) & \text{if} \quad \Delta E > 0 \end{cases}} \tag{8.12}$$

In functional minimisation the energy is simply the value of the function $E = f(\vec{x})$ and the temperature $T$ is slowly lowered to 0, hence the analogy with annealing of metals.

## 8.5. Monte Carlo Integration

Consider calculating an integral of a $d$-dimensional function $f(\vec{x})$ over some volume $V$ defined by boundaries $a_i$ and $b_i$

$$I = \int_{a_1}^{b_1} dx_1 \int_{a_2}^{b_2} dx_2 \ldots f(\vec{x}) = \int_V f(\vec{x})\, d\vec{x} \quad . \tag{8.13}$$

The integral $I$ is related to the mean value of the function in the volume. This can be written as

$$\langle f \rangle = \frac{1}{V} \int_V f(\vec{x})\, d\vec{x} \quad .$$

Then $I$ can be obtained from the mean of the function as

$$I \equiv V \langle f \rangle \quad . \tag{8.14}$$

This means we can *estimate* $I$ by taking $N$ random samples of the function in the volume

$$\hat{I} = \frac{V}{N} \sum_{i=1}^{N} f(\vec{x}_i) \quad . \tag{8.15}$$

We can also derive an estimate of the error in $\hat{I}$ by considering the estimate of the parent variance of the samples $f(\vec{x}_i)$, i.e.,

$$\sigma_{f_i}^2 = \frac{1}{N-1} \sum_{i=1}^{N} (f(\vec{x}_i) - \langle f \rangle)^2 \quad . \tag{8.16}$$

Given this, the variance of the mean $\langle f \rangle$ is $\sigma_{\langle f \rangle}^2 = \sigma_{f_i}^2 / N$ and given (8.14), we can write the variance in the estimate of $I$ as

$$\sigma_{\hat{I}}^2 = V^2 \sigma_{\langle f \rangle}^2 = \frac{V^2}{N} \sigma_{f_i}^2 \quad . \tag{8.17}$$

Therefore we can state the estimate of $I$ (including its error) as

$$\boxed{\hat{I} \;=\; \frac{V}{N} \sum_{i=1}^{N} f(\vec{x}_i) \;\pm\; \frac{V}{\sqrt{N}} \sigma_{f_i} \quad .} \tag{8.18}$$

A key point is that the error in MC integration scales as $\mathcal{O}(h^{1/2})$.

**Example** – Consider the integral of the function shown in Fig. 8.2, a uniform circle of radius 1/2;

$$f(x,y) = \begin{cases} 1 & \text{if } (x - \frac{1}{2})^2 + (y - \frac{1}{2})^2 < (\frac{1}{2})^2 \\ 0 & \text{otherwise} \end{cases} \quad . \tag{8.19}$$

In this example, we have the luxury of knowing that

$$I = \int f(x,y)\, dx\, dy = \text{circle area} = \pi r^2 = \frac{1}{4}\pi \quad . \tag{8.20}$$

To calculate $I$ using MC integration, we sample $N$ random, uniformly distributed points in the range $0 \le x < 1$ and $0 \le y < 1$. The value of the sampled integrand $f(x_i, y_i)$ will either be 1 (*success*) or 0 (*fail* to hit the circle). The ***probability of success***, $p$, for

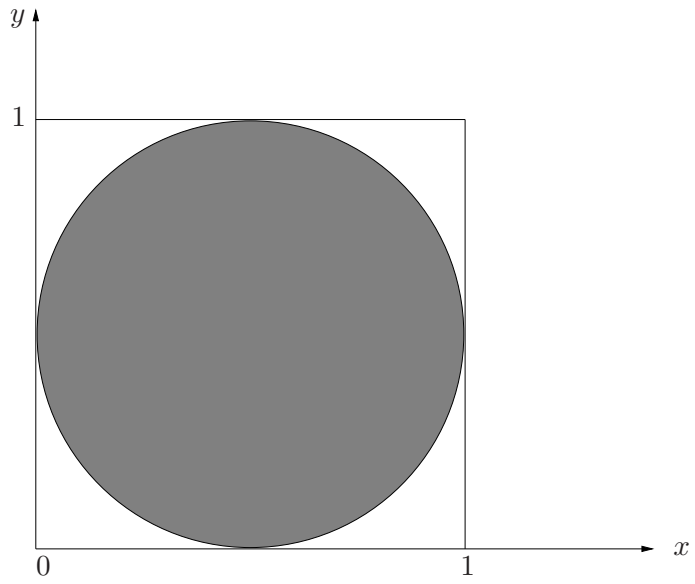


**Figure 8.2.** Circle of radius 1/2. The function $f(x, y) = 1$ inside the circle and $f(x, y) = 0$ outside

any given sample is the fraction of the square (area $A = \int_0^1 \int_0^1 dx\, dy = 1$) covered by the circle

$$p \equiv \frac{I}{A} \quad . \tag{8.21}$$

The successes should follow a binomial distribution. Let $N_1$ be the number of successes. We can define an estimator for $p$ as

$$\hat{p} = \frac{N_1}{N} \quad .$$

We know that for a binomial distribution the variance of $N_1$ is

$$\sigma_{N_1}^2 = N\, p\, (1 - p) \quad ,$$

such that

$$\sigma_{\hat{p}}^2 = \frac{\sigma_{N_1}^2}{N^2} = \frac{p\,(1-p)}{N} \quad .$$

We can then have our estimate for the value of the integral $I$ and its error

$$\boxed{\hat{I} = \hat{p}\, A = \hat{p} = \frac{N_1}{N} \pm \sqrt{\frac{p\,(1-p)}{N}} \quad .} \tag{8.22}$$

Again we see that the error on the integral scales as the square root of the number of samples.

**Comparison with the trapezium method** – The trapezium method for calculating the integral of a function is equivalent to a linear expansion (interpolation) of the function in each interval $x_i \to x_i + h$. Being a linear expansion, we know that the truncation error

goes as $\mathcal{O}(h^2)$. In general for $d$-dimensions we will evaluate the function at $N \sim h^{-d}$ points to calculate the integral. (Exactly $N = h^{-d}$ if the integration range in each dimension has unit length.) Therefore

$$h \sim N^{-1/d} \ , \tag{8.23}$$

such that the error in the trapezium method calculation of the integral is

$$\epsilon \sim \mathcal{O}(h^2) \sim \mathcal{O}(N^{-2/d}) \ . \tag{8.24}$$

Thus for 4 or more dimensions the Monte Carlo method is as accurate or more accurate than the trapezium method for the same number of samples.

(8.25) Error scalings $\longrightarrow$

| $d$ | Trapezium | MC |
|---|---|---|
| 1 | $N^{-2}$ | $N^{-1/2}$ |
| 2 | $N^{-1}$ | $N^{-1/2}$ |
| 3 | $N^{-2/3}$ | $N^{-1/2}$ |
| 4 | $N^{-1/2}$ | $N^{-1/2}$ |
| 5 | $N^{-2/5}$ | $N^{-1/2}$ |
| 6 | $N^{-1/3}$ | $N^{-1/2}$ |

**Using the Metropolis Algorithm** – For some integrals/summations the integrand is highly weighted to particular regions of $\vec{x}$. Then it is more efficient to bias random samples to where the integrand has more 'weight' and less to the larger volume of parameter space where the integrand is vanishingly small. For an integral of the form

$$I = \int_V f(\vec{x}) d\vec{x} = \int_V Q(\vec{x}) P(\vec{x}) d\vec{x} = \langle Q \rangle \tag{8.26}$$

where $P(\vec{x})$ behaves as (or is) a PDF (i.e. normalised and $\geq 0$) the Metropolis algorithm can be used to pick samples that concentrate where the integrand is largest. Essentially, it guides the 'trajectory' of $\vec{x}$ (taken to pick the samples) to seek the maximum of the $P(\vec{x})$ and wander about there.

Statistical Physics is a prime example of this. One wants to calculate the average of a quantity $Q$ over all states, with a particular state being specified by $\vec{x}$. For example, for a classical system (or a hot quantum one) where the probability is governed by the Boltzmann energy distribution $P(\vec{x}) \propto \exp\left(-E(\vec{x})/k_{_B}T\right)$, the average over states is

$$\langle Q \rangle = \frac{1}{Z} \int_{\vec{x}} Q(\vec{x}) \exp\left(-\frac{E(\vec{x})}{k_{_B}T}\right) d\vec{x} \ , \tag{8.27}$$

where $Z = \int_{\vec{x}} \exp\left(-E(\vec{x})/k_{_B}T\right) d\vec{x}$ is the partition function.

The Metropolis algorithm for calculating $\langle Q \rangle$ is

- Randomly pick a starting point $\vec{x}$.
- Repeatedly use the Metropolis algorithm to move to a new sample point.
  - Make a small trial step/change in the parameters to get $\vec{x}\,'$.

– Calculate $P(\vec{x}')$. Accept or reject step using

$$p_{acc} = \begin{cases} 1 & \text{if} \quad P(\vec{x}') \geq P(\vec{x}) \\ P(\vec{x}')/P(\vec{x}) & \text{if} \quad P(\vec{x}') < P(\vec{x}) \end{cases} \tag{8.28}$$

(Acceptance means $\vec{x} = \vec{x}'$.)

– Evaluate $Q(\vec{x})$ and add to running total $\sum Q$ (and $\sum Q^2$ if necessary).

- (Repeat whole process for a series of different starting points.)
- Divide through by the total number of samples to get $\langle Q \rangle$, etc.

We are essentially doing

$$\langle \hat{Q} \rangle \;=\; \frac{1}{N} \sum_{i=1}^{N} Q(\vec{x}_i) \;\pm\; \frac{1}{\sqrt{N}} \sigma_{Q_i} \quad , \tag{8.29}$$

as before (c.f. (8.18) ), but with a different strategy for picking the samples. The variation in $Q(\vec{x})$ is relatively flat, compared to if we had sampled/averaged $f = QP$ directly. This reduces $\sigma_{Q_i}$ compared to $\sigma_{f_i}$ and therefore *makes the error smaller* (for a given number of samples $N$).

In the context of statistical physics the maximum of $P$ corresponds to *equilibrium states* and the wandering corresponds to *thermal fluctuations*. With the Boltzmann PDF, equation (8.28) above is equivalent to (8.12) seen before, i.e., $P(\vec{x}')/P(\vec{x}) = \exp(-\Delta E/k_{_B}T)$. Note that if started far from equilibrium, the samples obtained during the process of finding the equilibrium skew the result somewhat. They should be omitted unless they are a negligibly small proportion of the total samples. Repeating the process for different random starting points is optional but ensures good, even coverage of the integrand. Otherwise with a single starting point, more samples will need to be taken.