PHAS0030: Computational Physics
Session 8 Preview: Random numbers and Monte Carlo methods

David Bowler

January 31, 2020

Abstract

In this summary, we will touch on the generation of *pseudo*-random numbers by computers. We will then examine the Monte Carlo approach to integration, and see how this can be used to study problems in thermodynamics such as phase transitions, and be adapted to optimise a function.

1 Pseudo-random numbers

There are many areas of physics where events occur in truly random fashion, such as radioactive decay or the Brownian motion of a particle. In order to model such processes, we need to have a way to simulate random events on a computer, which is the same as saying that we need to generate numbers randomly. It turns out that there are certain classes of problem where replacing the original problem with a stochastic one (i.e. a random one) can give us a useful answer. Of course, in these cases we need some statistical measure of the accuracy of the simulation.

There is no way to generate a truly random number on a computer; instead, there are algorithms that allow us to generate a long sequence of *uncorrelated* numbers, which we treat as *pseudo*-random numbers. These sequences are set by a seed number[1] which means that it is possible to reproduce what should be a "random" process: this is important for testing.

We will use the Numpy `np.random` module, which uses an extremely good pseudo-random number generator (the Mersenne Twister). To generate a single random number from 0 to 1 (with the but not including 1 itself) you should use `np.random.random()`. Other commands that can be used include:

- `(b-a)*np.random.random() + a` to generate numbers from `a` to `b`
- `np.random.rand()` to generate a set of random numbers in a given shape (e.g. a (2×5) array would use `np.random.rand(2,5)`[2])
- `np.random.randint(low,high)` to generate integers between `low` and `high`
- `np.random.seed()` to set the seed, if needed

If we are modelling an event which has a particular probability of occurring, then in general we will choose a random number between zero and one, and if that number is *less than* the probability, we say that the event has occurred.

2 Monte Carlo integration

Monte Carlo methods[3] use points selected from a probability distribution function to perform integrals of one kind or another. A simple way to understand this is to imagine a shape that cannot be integrated whose area you want to know; place a square (say) around it and select points at random from within the square. The fraction of points that lie within the shape give the area.

[1]This is often taken to be the number of seconds since a particular date, or can use environmental noise from computer hardware.
[2]Note that we do *not* need to pass a tuple here, unlike for `np.zeros()` etc.
[3]These are indeed named after the famous part of Monaco known for its casino.

More formally and generally, we choose points at random from within a multi-dimensional integration domain, sum over the value of the integrand and then scale by the average volume[4] per sampling point. We can write:

$$I = \int d\mathbf{r} f(\mathbf{r}) \simeq \frac{V}{N}\sum_{i=1}^{N} f(\mathbf{r}_i) \tag{1}$$

with $\mathbf{r}_i$ chosen *uniformly* from within the domain V. It can be shown that the error in the integral scales as $1/\sqrt{N}$ *regardless* of the dimension of the problem.

Why would we want to take this approach, rather than use one of the highly-optimised numerical approaches we saw in Session 3? The answer lies in the dimensionality that we mentioned above: numerical integration methods scale poorly with dimension, while Monte Carlo methods scale extremely well. One area where they have found many applications is statistical mechanics, where the calculation of a partition function requires the integral over all coordinates of N particles: a $3N$-dimensional integral. With only five points in each dimension and ten particles, this would be 5^{30} points, which is not a practical calculation.

However, uniform sampling is often a very bad choice. If we consider the partition function then we are sampling the function $\exp(-E/k_BT)$, where $E = E(\mathbf{r}_1, \mathbf{r}_2, \ldots, \mathbf{r}_N)$. The important part of this function[5] forms only a small part of its space, so uniform sampling will waste a lot of time and computational resources. We turn to a method called *importance sampling* which shares similarities with the methods we used above to generate non-uniform distributions. We write:

$$\int_a^b f(x)dx = \int_a^b \frac{f(x)}{p(x)} p(x)dx \tag{2}$$

and note that $p(x)$ is a normalised probability distribution. Then if we discretise, we find:

$$I \simeq \frac{1}{N}\sum_{i=1}^{N} \frac{f(x_i)}{p(x_i)} \tag{3}$$

where the points x_i are drawn from the normalised distribution $p(x)$. The trick, of course, is to find a good distribution to use that samples more heavily in the areas where the function has a large value and less heavily in the areas where it has small values. This can significantly improve the convergence of the integral with number of samples.

3 Monte Carlo simulation

A standard result in thermal physics or statistical mechanics is that for a set of distinguishable particles at temperature T, the probability that a particle will occupy the energy level E_i is given by:

$$p(E_i) = \frac{e^{-E_i/k_BT}}{\sum_i e^{-E_i/k_BT}} \tag{4}$$

As we saw with the work on molecular dynamics in Session 7, it turns out that the properties of a system can be modelled with relatively few particles. With the molecular dynamics method, we found time averages of quantities by integrating[6] Newton's equations of motion. With the Monte Carlo method, we can take *ensemble* averages of quantities by sampling from distributions.

What we will model here is the approach to equilibrium (which is exactly what we did with molecular dynamics: after creating an initial set of positions and velocities, we then needed to equilibrate), by randomly exchanging energy between particles. This method dates back over fifty years, and is often known as the Metropolis method[7].

The basic idea is to start with a random distribution of particles over energy levels (which are often determined by particle positions, or spins, or other physical properties). We then select a particle at random and alter its energy (again, generally at random, unless there are only two energy levels). If the energy goes down, then we accept the move; the key point of the Metropolis algorithm is that there is a finite chance to accept a move if the energy goes up. We use the probability $p = e^{-\Delta E/k_BT}$ where the energy change is ΔE, so that the probability decreases rapidly as the energy change increases. This process means that the system can climb out of local minima (but not the global minimum), with the probability of escaping from a minimum increasing as the temperature increases (just as is seen in nature).

[4]In however many dimensions are appropriate

[5]i.e. where $E \sim k_BT$, before the exponential becomes essentially zero

[6]We also say that we *evolve* the system forward in time; the two terms describe the same process, as we integrate to move forward in time.

[7]Metropolis et al., J. Chem. Phys. **21**, 1087 (1953)

We will use this relatively simple procedure to model the behaviour of the Ising model: a model system for interacting spins on a lattice which is widely studied. We assume a square lattice with spin-1/2 particles that interact only with their nearest neighbours, and potentially an external magnetic field. The energy in this case is given by:

$$E = -\sum_{i,j} s_i s_j J - \sum_i B m s_i \tag{5}$$

where the spins s_i take on values of 1 or -1, $m = 1/2$ and J is the quantum mechanical exchange coupling between spins. We will characterise the system in terms of the ratios $J/k_B T$ and $mB/k_B T$.

To set up a simulation, we start with a random array of spins on a lattice (which should then have a net spin of close to zero). We then iterate the following algorithm:

1. Choose a particle at random
2. Calculate the change in energy when its spin is flipped (note that this is only for this spin, and only requires its neighbours) using Eq. (5)
3. If the energy goes down, accept the move
4. If the energy goes up, accept the move with probability $p = e^{-\Delta E/k_B T}$ (in other words, choose a random number from 0 to 1 and accept the move if this number is *less than* p)

The system will have periodic boundary conditions (easily implemented with modulo, as we saw last week), and we will need to monitor the total energy and the average spin (this is a measure of long-range order, which is important to monitor in this kind of simulation). We will test two cases: first, equilibration given an external field but zero coupling between spins (i.e. $J = 0$), equivalent to a paramagnet; second, equilibration with no external field but varying coupling between spins, which will display ferromagnetic behaviour given strong enough coupling.

3.1 Simulated annealing

We can apply the Monte Carlo method to the minimisation of functions: if we start at a high temperature, and equilibrate, and then gradually reduce the temperature, equilibrating at each stage, it is quite likely[8] that we will find the minimum of the function. This is a direct analogy of the process used to remove defects from metals and glasses[9]. The process is simple to implement, but finding the details of how to reduce the temperature, and how long each run at a fixed temperature should be, is extremely difficult.

[8] Note that, as with all optimisation procedures, we cannot guarantee to find the global minimum.
[9] And, bizarrely, chocolate: the process is known as tempering as well as annealing.