Report: Monte Carlo Simulations

Thibeau Wouters

December 8, 2020

Contents

1 Non-uniform random numbers

Central to any Monte Carlo method or Monte Carlo algorithm are random numbers. A first step towards creating such algorithms to solve problems in physics is understanding how exactly we can generate random numbers. In this section, we investigate how one can get random numbers from any continuous probability distribution f while only generating random numbers from the simplest possible distribution, namely the uniform distribution in the interval $[0, 1]$, also called the standard uniform distribution.

We present and investigate two methods to obtain non-uniform random numbers while only making use of Python's `random.uniform` function to generate random numbers.

1.1 Inverse transform random sampling

Proposition 1 (Probability integral transform)**.** *Suppose the random variable X satisfies $X \sim \mathcal{U}(0, 1)$, f is a continuous probability distribution with corresponding cumulative distribution F and $Y = F^{-1}(X)$. Then Y has the cumulative distribution function F.*

Proof. We will show that $P(Y \leq y) = F(y)$. For this, note that

$$\begin{aligned} P(Y \leq y) &= P(F^{-1}(X) \leq y) \\ &= P\left(F(F^{-1}(X)) \leq F(y)\right) \\ &= P(X \leq F(y)) = F(y) \end{aligned}$$

since F is by definition a non-decreasing function, and for all $x \in [0,1]$, we have $P(X \leq x) = x$, since $X \sim \mathcal{U}(0,1)$. $\square$

We now illustrate this method by applying it to three different functions f.

First function: Consider the probability density function f_1 of a $\mathcal{U}(-2,1)$ distribution, i.e.:

$$f_1 : \mathbb{R} \to \mathbb{R} : x \mapsto \begin{cases} \frac{1}{3} & x \in [-2,1] \\ 0 & \text{elsewhere} \end{cases} . \tag{1.1}$$

Then the inverse of the cumulative distribution function F_1 is

$$F_1^{-1} : [0,1] \to [-2,1] : x \mapsto \frac{x+2}{3} . \tag{1.2}$$

Graphs and discussion

Second function: Consider the probability density function f_2 of an exponentially distributed random variable:

$$f_2 : \mathbb{R} \to \mathbb{R}^+ : x \mapsto \begin{cases} e^{-x} & x \geq 0 \\ 0 & x < 0 \end{cases} . \tag{1.3}$$

The inverse of the cumulative distribution function F_2 is given by

$$F_2^{-1} : [0,1] \to \mathbb{R}^+ : x \mapsto -\log(1-x) . \tag{1.4}$$

Graphs and discussion

Third function: Consider the linear probability function

$$f_3 : \mathbb{R} \to \mathbb{R} : x \mapsto \begin{cases} \frac{x+1}{2} & x \in [-1,1] \\ 0 & \text{elsewhere} \end{cases} . \tag{1.5}$$

The inverse of the cumulative distribution function F_3 is given by

$$F_3^{-1} : [0,1] \to [-1,1] : x \mapsto 2\sqrt{x} - 1 . \tag{1.6}$$

Graphs and discussion

1.2 Hit-and-miss method

The hit-and-miss method works for distributions f that are only non-zero in a finite interval $[a, b]$. Let M be a number that is greater than $f(x)$ for all $x \in [a, b]$. We generate t and s out of a $\mathcal{U}(a, b)$ and a $\mathcal{U}(0, M)$ distribution, respectively. If $s \leq f(t)$, then this is called a 'hit' and the value t is stored. If $s > f(t)$, then this is called a 'miss', and the value is not returned. It is immediately clear that the hit-and-miss method therefore wastes computer resources whenever a miss is generated, which is a drawback of the algorithm.

We illustrate the hit-and-miss method for two distributions: the first is the function f_3 defined above, and so we could in principle compare the two methods described for generating random numbers described in this section. Note that the hit-and-miss method cannot be used for the exponential distribution (f_2 defined above), since this distribution is non-zero on an infinite interval.

Graphs and discussion

The second example for the hit-and-miss method is the function f_4 defined as

$$f_4 : [0, 1] \to -x(1-x)e^{-x^2 \log(1-x)} \,. \tag{1.7}$$

Even though the hit-and-miss method wastes computer resources, as mentioned above, for more complicated functions such as f_4 it can be more convenient, since the user does not have to solve and invert an integral equation.

Graphs and discussion

1.3 Uniform random points in a circle

The goal is to generate uniform random points in a circle of radius R in two ways: the first one makes use of the hit-and-miss method, while the second one uses an appropriate distribution function $f(r, \theta)$, where r, θ are polar coordinates.

TO DO: Hit and miss!

Now we generate the random points using an appropriate distribution function $f(r, \theta)$. We can in fact seperate the variables for this function, and look for appropriate distribution functions $\mathcal{R}(r), \mathcal{T}(\theta)$ to generate the random points. For T, we use a uniform distribution in the interval $[0, 2\pi]$. One could naively think that we could also generate r out of a uniform distribution on $[0, R]$, but this is in fact not true. Indeed, Figure X shows the result in that case: these points are clearly not uniformly distributed: we generate more points closer towards the origin.

Graphs and discussion

To understand the problem, we note that if we look at a certain interval for the angle θ, say $[\theta, \theta + \mathrm{d}\theta]$, then the length of the arc between (r, θ) and $(r, \theta + \mathrm{d}\theta)$ is equal to $L = r\,\mathrm{d}\theta$. Hence for the same range of the angle, more points should be generated further away from the origin, in a linear manner.

This thought experiment indicates we should generate the radius r from the distribution which satisfies

$$\mathcal{R} : [0, R] \to \mathbb{R} : r \mapsto \frac{2}{R^2} r \,, \tag{1.8}$$

where the factor $2/R^2$ is used for normalisation. By applying Proposition 1, this can be done by generating a random number from the standard uniform distribution, and transforming it via the function

$$\mathcal{R}^{-1} : [0, 1] \to [0, R] : x \mapsto \mathcal{R}^{-1}(x) = R\sqrt{x} \,. \tag{1.9}$$

This seems to be the correct distribution function for the radial coordinate: Figure X shows the points generated by $f(r, \theta) = \mathcal{R}\mathcal{T}(\theta)$.

Graphs and discussions.

2 Gaussian RNG

The method in the previous section of the probability integral transformation cannot be applied to one of the most common distributions in all of physics: the Gaussian distribution. Therefore, we try to look for another way to generate random numbers from a Gaussian distribution in this section, which is known as the Box-Muller transformation.

2.1 Standard Gaussian distribution

If we draw x and y from a standard Gaussian distribution, the joint density function is

$$f(x, y)\,\mathrm{d}x\,\mathrm{d}y = \frac{1}{2\pi} \exp\left(-\frac{x^2 + y^2}{2}\right) \mathrm{d}x\,\mathrm{d}y \,, \tag{2.1}$$

which can be transformed into polar coordinates:

$$f(r, \theta)\,\mathrm{d}r\,\mathrm{d}\theta = \frac{1}{2\pi} \exp\left(-\frac{r^2}{2}\right) r\,\mathrm{d}r\,\mathrm{d}\theta \,. \tag{2.2}$$

So the idea behind the Box-Muller transformation is to generate two random numbers r, θ from the distribution (2.2), such that $x = r\cos\theta$, $y = r\sin\theta$ are two random numbers from a standard Gaussian. The number θ is drawn from a $\mathcal{U}(0, 2\pi)$ distribution, while r can be obtained via a probability integral transformation. For this, we have to invert

$$F(r) = \int_0^r \exp\left(-\frac{z^2}{2}\right) z \, \mathrm{d}z \,, \tag{2.3}$$

which, contrary to the density function of a standard Gaussian in one dimension, can more easily be inverted. The trick behind Box-Muller is essentially to go to two dimensions, and benefit from the Jacobian of polar coordinates: this gives a factor r, which makes that we can solve the integral above. Indeed, if we make the change of variables $u = z^2/2$, then $z\,\mathrm{d}z = \mathrm{d}u$ and the integral evaluates to

$$F(r) = 1 - \exp\left(-\frac{r^2}{2}\right) \,. \tag{2.4}$$

The inverse function is therefore

$$F^{-1} : [0,1] \to \mathbb{R}^+ : x \mapsto \sqrt{-2\log(1-x)} \,, \tag{2.5}$$

and we are now able to generate random numbers r from the distribution in (2.2).

The Python function `generate_gaussian_numbers` generates one couple of random numbers x and y from a standard Gaussian distribution using the algorithm described above. The parameter npts represents the number of times we repeat this algorithm. In Figure 1, we show the histograms that we obtain after npts $= 10^5$ repetitions of the function. The joint probability distribution indeed resembles the density plot of a two-dimensional Gaussian distribution, and the one-dimensional histograms clearly follow a Gaussian distribution, of which the graph is shown in blue.

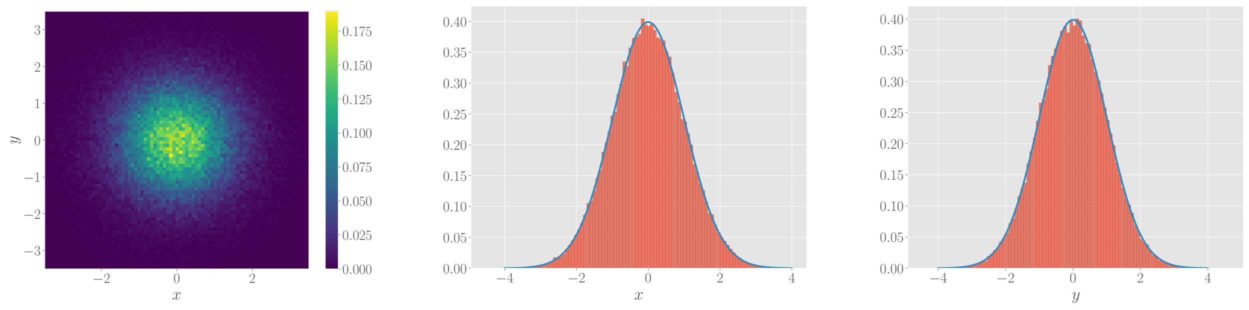

Figure 1: From left to right: a two-dimensional density plot of (x, y) pairs, and histograms for x and y after applying the algorithm npts $= 10^5$ times. The blue curve on top of the 1D histograms is a plot of the standard Gaussian distribution.

2.2 Arbitrary Gaussian distributions

As we saw, we are now able to generate random numbers from a Gaussian distribution with $\mu = 0$ and $\sigma = 1$. However, the above algorithm can easily be extended such that random numbers from *any* normal distribution $\mathcal{N}(\mu, \sigma)$ can be generated. This is done via a change of variables: if $Z \sim \mathcal{N}(0,1)$, then the variable $X = \sigma Z + \mu$ is normal distributed with mean μ and standard deviation σ. This is implemented in the Python function `generate_gaussian_numbers2`, and we illustrate the function for $\mu = 2$ and $\sigma = 1.5$ in Figure 2, which is similar to Figure 1 for the standard Gaussians. For the one-dimensional histograms, we have also plotted the probability distributions of $\mathcal{N}(\mu, \sigma)$.

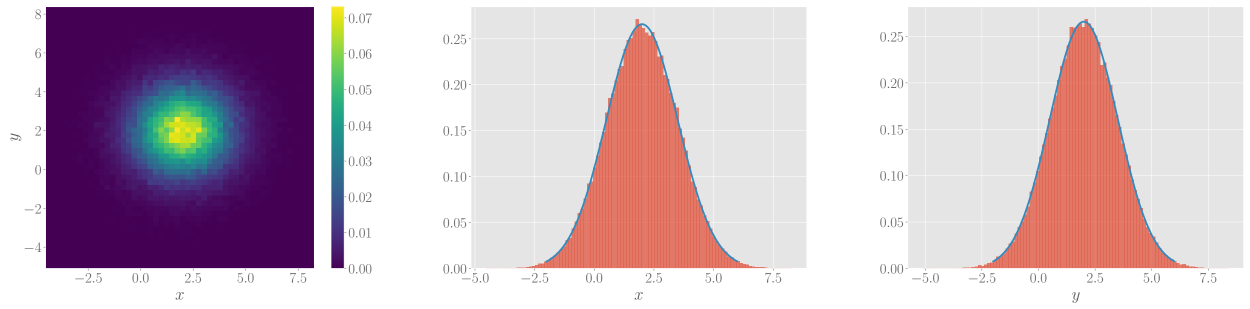

Figure 2: From left to right: a two-dimensional density plot of (x, y) pairs, and histograms for x and y after applying the algorithm npts $= 10^5$ times. The blue curve on top of the 1D histograms is a plot of the Gaussian distribution $\mathcal{N}(\mu, \sigma)$.

3 Monte Carlo integration (I)

Now that we understand the methods behind generating random numbers from continuous distributions, we will go to a first and one of the most common applications of the Monte Carlo method, namely numerical approximation of integrals. We will estimate the following integral by relying on the Monte Carlo methods seen in the course notes:

$$\mathcal{I} = \int_{\mathbb{R}^3} \exp\left(-\frac{r^2}{2}\right) (x + y + z)^2 \,\mathrm{d}x \,\mathrm{d}y \,\mathrm{d}z \,, \tag{3.1}$$

where $r^2 = x^2 + y^2 + z^2$. We first compute this result analytically, before approximating it using the Monte Carlo method.

3.1 Analytical method

We expand the square:

$$\mathcal{I} = \int_{\mathbb{R}^3} \exp\left(-\frac{r^2}{2}\right) (x^2 + y^2 + z^2 + 2xy + 2yz + 2xz) \,\mathrm{d}x \,\mathrm{d}y \,\mathrm{d}z \,. \tag{3.2}$$

We see that there are only two different sorts of integrandi. Indeed, it is sufficient to calculate the following two integrals analytically:

$$I_1 = \int_{\mathbb{R}^3} x^2 \exp\left(\frac{x^2+y^2+z^2}{2}\right) \mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z, \quad I_2 = 2\int_{\mathbb{R}^3} xy \exp\left(\frac{x^2+y^2+z^2}{2}\right) \mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z\,, \tag{3.3}$$

since then the integral that we wish to compute is simply $\mathcal{I} = 3I_1 + 3I_2$. The integral I_2 is equal to zero: we can factorise the exponential, and then note that we need to compute an integral over x of an odd function in x with domain $\mathbb{R}$, which necessarily vanishes. The value of I_1 can be obtained from the well-known Gaussian integral

$$\mathcal{G}_\alpha = \int_{\mathbb{R}} \exp\bigl(-\alpha x^2\bigr)\,\mathrm{d}x = \sqrt{\frac{\pi}{\alpha}}\,, \tag{3.4}$$

and, by viewing it as a function of α, we also obtain that

$$\int x^2 \exp\left(-\alpha x^2\right)\mathrm{d}x = -\frac{\mathrm{d}}{\mathrm{d}\alpha}\mathcal{G}_\alpha = \frac{\sqrt{\pi}}{2}\alpha^{-3/2}\,. \tag{3.5}$$

All together, we find the value of I_1 to be

$$\begin{aligned} I_1 &= \int_{\mathbb{R}} x^2 \exp\left(-\frac{x^2}{2}\right)\mathrm{d}x \int_{\mathbb{R}} \exp\left(-\frac{y^2}{2}\right)\mathrm{d}y \int_{R} \exp\left(-\frac{z^2}{2}\right)\mathrm{d}z \\ &= \left(\mathcal{G}_{1/2}\right)^2 \left(-\frac{\mathrm{d}\mathcal{G}_\alpha}{\mathrm{d}\alpha}\right)\Big|_{\alpha=\frac{1}{2}} = (2\pi)^{3/2}\,. \end{aligned}$$

So we conclude that

$$\mathcal{I} = 3(2\pi)^{3/2}\,. \tag{3.6}$$

3.2 Numerical method using the Monte Carlo method

We now approximate the integral via the Monte Carlo methods discussed in the lecture notes, using the factor $\exp\left(-\frac{r^2}{2}\right)$ as a density function. We have first to impose a normalisation to this function, i.e. we have to choose N such that the function $W(x,y,z) = N\exp\left(-\frac{x^2+y^2+z^2}{2}\right)$ is normalised to one. We calculate, using the results mentioned above, that

$$N\int_{R} \exp\left(-\frac{x^2+y^2+z^2}{2}\right)\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z = N\left(\int_{\mathbb{R}} \exp\left(-\frac{x^2}{2}\right)\mathrm{d}x\right)^3 = N(2\pi)^{3/2}\,, \tag{3.7}$$

and hence the normalised density function is

$$W(x,y,z) = \frac{1}{(2\pi)^{3/2}}\exp\left(-\frac{x^2+y^2+z^2}{2}\right)\,, \tag{3.8}$$

which is the density function for a normal distribution for three variables, with mean zero and variance one, and no correlations between the variables.

To approximate the integral, let f denote the integrand, i.e. $\mathcal{I} = \int_{\mathbb{R}^3} f(x, y, z)\, \mathrm{d}x\, \mathrm{d}y\, \mathrm{d}z$ and let $g(x, y, z) := (x + y + z)^2$. Since f can be factorised in a function proportional to W multiplied with the function g, we have that

$$\mathcal{I} = \int_{\mathbb{R}^3} f(\mathbf{x})\, \mathrm{d}^3\mathbf{x} \approx \frac{1}{N} \sum_{n=1}^{N} \frac{f(\mathbf{x}_n)}{W(\mathbf{x}_n)} \approx \frac{1}{N} (2\pi)^{3/2} \sum_{n=1}^{N} g(\mathbf{x}_n)\,, \tag{3.9}$$

where we used the notation $\mathbf{x} := (x, y, z)$ for notational simplicity. The points $\mathbf{x}_n, n = 1, \ldots, N$ are generated according to the distribution $W(\mathbf{x}_n)$. Comparing this with our result for $\mathcal{I}$ in equation (3.6), we see that in order to have a good approximation, we would like $\sum_{n=1}^{N} g(\mathbf{x}_n) \approx 3$.

We now implement the above algorithm to compute the integral numerically. We choose $N = 8000$ (this is called npts in the Python program) and we repeat this calculation nrep $=$ 100 times. We then take the average of those values as the approximation of the integral. In Figure 3 we show a plot of the obtained approximations along with the exact value that we computed analytically.

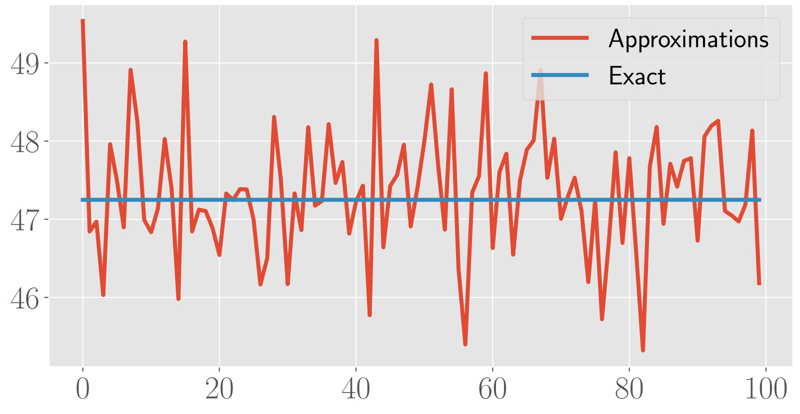

Figure 3: Plot of the obtained approximations of $\mathcal{I}$, along with the exact value.

The average of our approximations of the integral is equal to 47.3497, while the exact value is 47.2488. The standard deviation in our sampled data of approximations is 0.0171, which is quite low compared to the difference between our approximation and the exact result. The method can likely be improved. One assumption made above is that the function g is slowly varying, which is indeed the case for $g = (x + y + z)^2$ around the origin...

To do: why is the approximation so bad, even for high npts and nrep?

We now look at the variances: we are interested in the behaviour of σ^2 of the sampled values for different numbers of sampled points npts. We let the algorithm run with npts ranging from 50 to 5950, calculating σ^2 for each value of npts. Figure 4 below shows the resulting values of σ^2. We can recognise the behaviour $\sigma^2 \sim 1/\mathrm{npts}$ in the sampled data points. In order to firmly establish this relation, we fit a linear curve through the couples $(\mathrm{npts}, 1/\sigma^2)$ with the Scipy package from Python. The result is

$$1/\sigma^2 = m \cdot \mathrm{npts} + c, \quad m = 0.0109; c = -0.0644, \quad R^2 = 0.9171\,. \tag{3.10}$$

Figure 4 also contains a plot showing the $1/\sigma^2$ data points along with the linear fit. Since the coefficient of determination is fairly close to one for this linear fit, we can indeed conclude that $\sigma^2 \sim 1/\mathrm{npts}$.

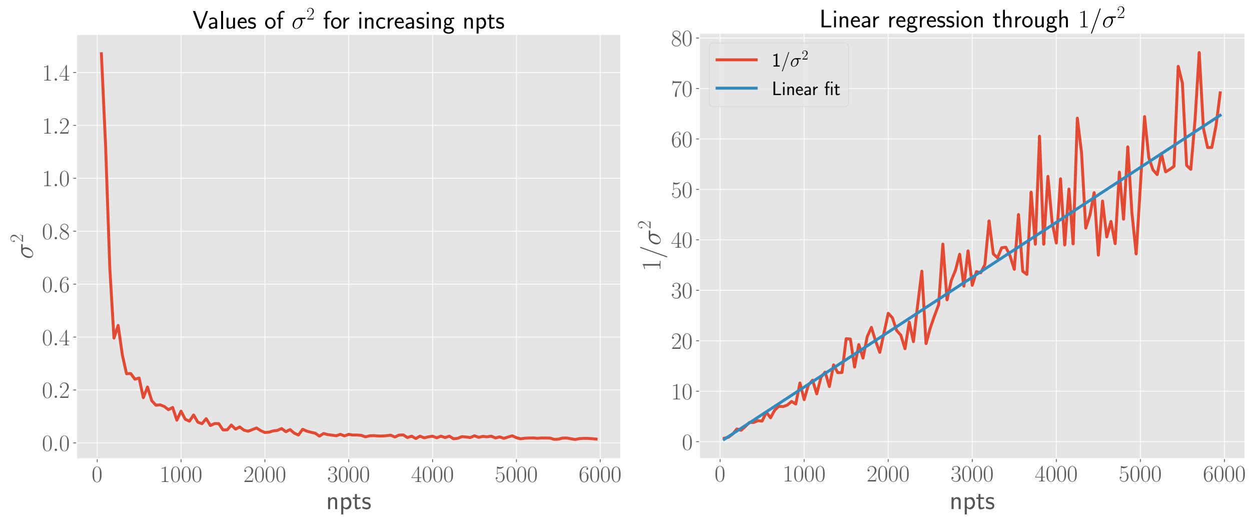

Figure 4: *Left*: Plot of the variances σ^2 for increasing values of npts. *Right*: Linear regression through the $1/\sigma^2$ values, with coefficient of determination $R^2 = 0.9171$.

As a final remark, we note that we cannot change the roles of the factors of the integrand f. With this, we mean that we cannot use the factor $(x+y+z)^2$ as density, and $\exp\left(-\frac{r^2}{2}\right)$ as the integrand (which we called g above). This is because the factor $(x+y+z)^2$ cannot be normalised in the domain of integration, which we have to require of a density function. Indeed, it is clear that we cannot choose a $N \in \mathbb{R}$ such that

$$N \int_{\mathbb{R}^3} (x+y+z)^2 \,\mathrm{d}x\,\mathrm{d}y\,\mathrm{d}z = 1\,, \tag{3.11}$$

and hence this factor cannot be used to define a density function.

4 Monte Carlo integration (II)

We again try to numermically approximate the integral from the previous section

$$\mathcal{I} = \int_{\mathbb{R}^3} \exp\left(-\frac{r^2}{2}\right)(x+y+z)^2 \, \mathrm{d}x \, \mathrm{d}y \, \mathrm{d}z \,, \tag{4.1}$$

but now we use the Metropolis algorithm. Therefore, we need to define a Markov chain, characterised by the transition probabilities $P(\mathbf{r} \to \mathbf{r}')$, of which the dynamics converge to the distribution $w(\mathbf{r}) = \frac{1}{(2\pi)^{3/2}} \exp\left(-r^2/2\right)$. This will be the case if we require detailed balance in the following way:

$$\frac{P(\mathbf{r} \to \mathbf{r}')}{P(\mathbf{r}' \to \mathbf{r})} = \exp\left(-\frac{r'^2 - r^2}{2}\right) . \tag{4.2}$$

We split the transition probability in two parts: the selection probability g and acceptance ratio A:

$$P(\mathbf{r} \to \mathbf{r}') = g(\mathbf{r} \to \mathbf{r}') A(\mathbf{r} \to \mathbf{r}') \,. \tag{4.3}$$

In our case, g is generated by going from $\mathbf{r}$ to $\mathbf{r}' = \mathbf{r} + \boldsymbol{\delta}$, where $\boldsymbol{\delta} = (\delta_x, \delta_y, \delta_z)$ and the components δ_i are chosen independently from a uniform distribution centered at the origin and of width h, called $W_h(\delta)$. The acceptance ratio is given by

$$A(\delta) = \begin{cases} 1 & \text{if } \ r'^2 < r^2, \\ \exp\left(-\frac{r'^2 - r^2}{2}\right) & \text{otherwise} \,. \end{cases} \tag{4.4}$$

Using the Metropolis algorithm, we plot the trajectory followed by the x, y coordinates of the points, where we use $N = 10^4$ steps, starting from $\mathbf{r} = (0,0,0)$ and $\mathbf{r} = (10,10,10)$. We do this for both $h = 1$ and $h = 0.1$. ...

We also verify, by plotting a histogram of the positions for a run with $N = 10^6$ that the dynamics generates an equilibruim distribution of points proportional to $\exp(-r^2/2)$.

5 Stochastic Matrices

In this problem, we will look at stochastic matrices. More intro...

Suppose we are given a non-negative initial vector w. Then the stochastic matrix generates non-negative vectors at all later times. Indeed, the stochastic matrix lets the system evolve according to the prescription that if the system is at time t described by $w(t)$, then at a later time step $t + \Delta t$, it is described by the vector

$$w(t + \Delta t) = P w(t) \,, \tag{5.1}$$

where

$$P : (P)_{\mu\nu} = P(\nu \to \mu) \tag{5.2}$$

is the stochastic matrix. By definition, all the entries of the stochastic matrix are non-negative. Therefore, if we assume that the vector $w(t)$ has only non-negative entries and we expand the matrix product in equation (5.1), we end up with a sum of products of non-negative quantities, hence these are again non-negative. So we conclude that $w(t + \Delta t)$ is again a non-negative vector.

This immediately implies by an inductive argument that if we start with an initial non-negative vector w, the stochastic matrix generates non-negative vectors at all later times.

Assume that P and Q are two $N \times N$ stochastic matrices. We now claim that PQ is again a stochastic matrix. To prove this, we have to show that *i)* all the elements of PQ are non-negative, and *ii)* that the sum of the columns of PQ are equal one.

To prove *i)*, we note that the an element of the matrix PQ is simply

$$(PQ)_{ij} = \sum_{k=1}^{N} P_{ik} Q_{kj} \geq 0\,, \tag{5.3}$$

since both P_{ik}, Q_{kj} are non-negative for all $i \in \{1, \ldots, N\}$ by the assumption that P and Q are stochastic matrices.

To show *ii)*, we have to show that for all $j \in \{1, \ldots, N\}$

$$\sum_{i=1}^{N} (PQ)_{ij} = 1\,. \tag{5.4}$$

We verify this by expanding the matrix product: take $j \in \{1, \ldots, N\}$ arbitrarily, then

$$\begin{aligned} \sum_{i=1}^{N} (PQ)_{ij} &= \sum_{i=1}^{N} \left(\sum_{k=1}^{N} P_{ik} Q_{kj} \right) = \sum_{i,k=1}^{N} P_{ik} Q_{kj} \\ &= \sum_{k=1}^{N} \left(Q_{kj} \sum_{i=1}^{N} P_{ik} \right) = 1\,, \end{aligned}$$

since for all $k \in \{1, \ldots, N\}$, we have that $\sum_i P_{ik} = 1$ since P is a stochastic matrix, and also since for all $j \in \{1, \ldots, N\}$, we have $\sum_k Q_{kj} = 1$ by the same assumption on Q.

We now introduce the norm $||w||_1 = \sum_j |w_j|$. We consider an arbitrary vector y, not necessarily non-negative, and show that

$$||Py||_1 \leq ||y||_1\,. \tag{5.5}$$

We can verify this directly by substituting the expansion of the matrix product AND WHAT ELSE?

$$\begin{aligned}||Py||_1 &= \sum_{j=1}^{N} |(Py)_j| = \sum_{j=1}^{N} \left| \sum_{i=1}^{N} P_{ji} y_i \right| \leq \sum_{j,i=1}^{N} P_{ji} |y_i| \\ &= \sum_{i=1}^{N} \left(|y_i| \sum_{j=1}^{N} P_{ji} \right) = \sum_{i=1}^{N} |y_i| = ||y||_1 \,,\end{aligned}$$

where we used both properties of the definition of a stochastic matrix.

An immediate consequence from equation (5.5) is that an eigenvector y of a stochastic matrix P, then its eigenvalue λ satisfies $|\lambda| \leq 1$. Indeed, we find that

$$||Py||_1 = ||\lambda y||_1 = |\lambda| \sum_{j=1}^{n} |y_j| \leq \sum_{j=1}^{n} |y_j| \,,$$

from which we immediately find $|\lambda| \leq 1$.

We now show that a stochastic matrix P has at least one eigenvalue $\lambda = 1$. Take any stochastic $N \times N$ matrix P and define the $1 \times N$ row-vector $z = (1, 1, \ldots, 1)$. We claim that z is a left eigenvector of P with eigenvalue $\lambda = 1$. Indeed, we note that for all $i \in \{1, \ldots, N\}$, we have

$$(zP)_i = \sum_{j=1} z_i P_{ij} = \sum_{j=1} P_{ij} = 1 = z_i \,,$$

such that we indeed have that $zP = z$. Note that we only had to make use of the property that the sum of columns of a stochastic matrix is equal to one. Since the stochastic matrix P was arbitrary, this shows that any stochastic matrix P has at least one eigenvalue equal to one.

6 Detailed balance

Consider an system with three states with energies $E_1 < E_2 < E_3$. The dynamics are such that the system can make the transitions $1 \to 2$ with probability a, $2 \to 3$ with probability b, $3 \to 1$ with probability c or stay in the current state, where a, b, c are arbitrary rates for these transitions. Note that we require $0 < a, b, c < 1$.

The stochastic matrix governing the evolution of this system is

$$P = \begin{pmatrix} 1-a & 0 & c \\ a & 1-b & 0 \\ 0 & b & 1-c \end{pmatrix} \,. \tag{6.1}$$

The dynamics of this system is ergodic: starting from 1, we have a non-zero probability to directly go to 2, and there exists a path from 1 to 3, which is $1 \to 2 \to 3$. From 2, there is a non-zero probabilty to go to 3, and there exists a path from 2 to 1, which is $2 \to 3 \to 1$. Starting from 3, there is a non-zero probability to go to 1, and there exists a path from 3 to 2, which is $3 \to 1 \to 2$. Hence any state in the system can be reached from any other state.

To verify if detailed balance is satisfied or not, we first compute the stationary state ω corresponding to this dynamics. This is the state that satisfied $P\omega = \omega$, and hence is an eigenvector of the matrix P with eigenvalue equal to one. In the previous section, we have proven that for any stochastic matrix P, such a vector always exists. Hence we can directly look for its components by solving the eigenvalue equation, which yields the system of equations

$$\begin{cases} (1-a)\omega_1 + c\omega_3 & = \omega_1 \\ a\omega_1 + (1-b)\omega_2 & = \omega_2 \\ b\omega_2 + (1-c)\omega_3 & = \omega_3 \end{cases} \quad \Longleftrightarrow \quad \begin{cases} -a\omega_1 + c\omega_3 & = 0 \\ a\omega_1 - b\omega_2 & = 0 \\ b\omega_2 - c\omega_3 & = 0 \end{cases} . \tag{6.2}$$

It can be directly verified that $\omega = (c, \frac{ac}{b}, a)$ is a solution to this set of equations (remember that $b > 0$). The actual form of the components is irrelevant; the crucial point is that all components of the stationary state vector are different from zero, by the assumptions made on a, b, c. This information is sufficient to show that detailed balance is **not** satisfied for this dynamics. The problem here is that if we look again at the stochastic matrix responsible for the development of the dymanics, given in equation (6.1), we see that the zeroes in the matrix are not present in a symmetric manner. Indeed, take $\mu = 2$ and $\nu = 1$; then $P(2 \to 1) = 0$, while $P(1 \to 2) \neq 0$, and hence the condition for detailed balance

$$\omega_\nu P(\nu \to \mu) = \omega_\mu P(\mu \to \nu) , \tag{6.3}$$

is not satisfied for this choice of μ and ν, since the left hand side is different from zero, while the right hand side is equal to zero.

In the lecture notes, we saw that detailed balance is a sufficient condition to get a Boltzmann distribution for the equilibrium of the dynamics. Here we show that even though detailed balance is not satisfied, we can still choose the rates a, b, c in such a manner that the equilibrium distribution is a Boltzmann distribution, proving that detailed balance is in fact not a necessary condition. In order to have a Boltzmann distribution, we require that the probability of finding the system in state i is given by

$$p_i = \frac{e^{-\beta E_i}}{Z} , \tag{6.4}$$

where $Z = \sum_{i=1}^{3} e^{-\beta E_i}$ is the partition function. Looking back at our result for the stationary state ω, we find that we can achieve this by setting

$$a = \frac{e^{-\beta E_3}}{Z}, \quad b = \frac{e^{-\beta(E_1 + E_3 - E_2)}}{Z}, \quad c = \frac{e^{-\beta E_1}}{Z} . \tag{6.5}$$

Since then we indeed have

$$\begin{aligned}\omega_1 &= c = \frac{e^{-\beta E_1}}{Z} \\ \omega_2 &= \frac{ac}{b} = \frac{e^{-\beta E_2}}{Z} \\ \omega_3 &= a = \frac{e^{-\beta E_3}}{Z}\,,\end{aligned}$$

such that the equilibrium distribution of the dynamics is a Boltzmann distribution.

7 Ising Model: Uniform sampling

In the 2D Ising model, spins $s_i = \pm 1$ are located at the vertices of a $N \times N$ square lattice. We will often refer to a spin value $+1$ as spin up, while spin value -1 is called spin down. The energy of such a configuration of spins is given by

$$E\left(\{s_k\}\right) = -J \sum_{\langle ij \rangle} s_i s_j\,, \tag{7.1}$$

where the summation is over nearest neighboring spins on the lattice (for a 2D lattice, there are 4 such nearest neighboring spins).The coupling constant J is set to 1 throughout this and the next section.

In this section, we approach the Ising model with a Monte Carlo algorithm that uses uniform sampling as a demonstration that this is an inefficient method. We generate a lattice using the helical boundary conditions, as described in full detail in the lecture notes. The method of uniform sampling to generate configurations of spins is also described in the lecture notes. Each run of the algorithm yields a vector ω, which is of length N^2, representing the 2D lattice of spins. Next, we discuss the method of Boltzmann sampling with a hit-and-miss method, which will also prove to be inefficient for most ranges of the temperature. Finally, we investigate the technique of reweighting, which will then be compared with the output of the Metropolis algorithm in the next section.

7.1 Comments

First of all, we make some comments about the Ising model which will be used later on. First of all, to compute the energy of a configuration of spins, we can iterate over all spins s_i in the lattice, and compute $s_i s_j$, for nearest neighbours s_j in equation (7.1) for each spin. However, it is clear that this method will count all pairs of spins that contribute to the energy exactly twice, and hence we have to divide the end result by two to obtain the correct energy.

This way of computing the energy is very simple to implement in an algorithm, but it is immediately clear that this can be done more efficiently: we now waste a lot of computer resources to compute terms which are redundant for the calculation.

It is also useful to note that the ground state of the Ising model is $E_0 := -2N^2$. Indeed, it is clear from equation (7.1) that the ground state corresponds to a lattice where all the spins are aligned with each other ($s_i = +1$ or $s_i = -1$ for all i). The energy of this configuration can be computed by the method described above, and since all terms equal $s_i s_j = 1$, this yields $E_0 = -4N^2/2 = -2N^2$.

7.2 Uniform sampling

The method of uniform sampling is easy to implement: for each lattice site, we 'flip a coin', determining whether the spin at that lattice site is up or down. The result, as mentioned before, is stored in a vector of length N^2.

For a fixed value of N, e.g. $N = 16$, we generate the histogram of energy values that we obtain after running the uniform sampling algorithm a large number of times (10^4 or 10^5 iterations). Instead of looking at the energy E, we look at the intensive variable $e = E/2N^2$, called the energies per bond.

7.3 Hit-and-miss method

The hit-and-miss method depends on the temperature T: as before, a random configuration of spins is generated, and the energy of the system E is computed. Next, we generate a uniform random number r in the interval $[0, 1]$. If we have $r \leq \exp(-\beta(E - E_0))$, where $\beta = 1/T$ (we set $k_b = 1$), then this is a 'hit' and the configuration is accepted.

We are interested in the efficiency of the hit-and-miss algorithm in function of the temperature T. For this, we calculate for various temperatures the fraction of hits over total generated configurations.

Discussion

Hence we can conclude that he hit-and-miss algorithm performs very poorly.

7.4 Reweighting technique

Lastly, we discuss the reweighting technique. From the histogram we obtained by the uniform sampling method, $P(E)$, we generate for a given temperature T another histogram $P_T(E) =$

$\exp(-E/T)P(E)$. We plot these histograms for various temperatures

8 Metropolis algorithm for the Ising model

We continue the analysis of the Ising model from the previous section, where we now implement the Metropolis algorithm, where we sample spin configurations through a Markov chain. Let μ and ν represent two configurations of spins. The transition probability $P(\mu \to \nu)$ is non-zero if and only if the two configurations differ only by a single spin.

Given an initial configuration of the spins, randomly generated, we select one of the spins at random, and then suggest to flip it. We compute the change in energy $\Delta E = E_f - E_i$ that would arise to flipping that spin. If $\Delta E \leq 0$, i.e. if the system would evolve towards a lower energy configuration, we accept the flip. If $\Delta E > 0$, we still accept the suggested flip if $r \leq \exp(-\Delta E/T)$, where r is a uniform random number generated in the interval $[0, 1]$. This cycle is repeated a large number of times.

We use the Metropolis algorithm to investigate *i*) the evolution of the magnetisation per spin of a spin configuration, defined as

$$m = \frac{1}{N^2} \sum_i s_i \, , \tag{8.1}$$

and *ii*) the autocorrelation function, defined as

$$\chi(t) = \frac{1}{t_f} \sum_{s=\tau_{eq}}^{\tau_{eq}+t_f} (m(s) - \langle m \rangle)(m(s+t) - \langle m \rangle) \, , \tag{8.2}$$

which is valid for $t > \tau_{eq}$.

8.1 Efficiency of the Monte Carlo algorithm

We would like to make our code for the Monte Carlo algorithm as efficient as possible, since this code will be repeated a large number of times. In the Metropolis algorithm described above, we need the difference in energy ΔE between two different configurations. However, since only one spin differs between these configurations, it is sufficient to compute the change in energy that arises because of this flipped spin. Indeed, all other spins remain unchanged, and will give no difference in energy. Therefore, we only need to compute the contributions toward the energy due to this spin and its four nearest neighbours, instead of computing the total energy in the lattice.

If $\Delta E > 0$, we need to compute an exponential to determine whether or not we accept the changed spin. This requires a certain amount of polynomial time, but this can be sped up if we compute the possible outcomes of this exponential function beforehand, which can be done since there are only finitely many $\Delta E > 0$. Indeed, we claim that the only possible values of $\Delta E > 0$ are $\Delta E = 4$ and $\Delta E = 8$.

To prove this claim, recall that ΔE is only determined by the selected spin and its neighbours. Before the spin is flipped, the energy configuration is

$$E_i = (-1) \cdot \chi_a + (+1) \cdot \chi_u \,, \tag{8.3}$$

where χ_a, respectively chi_u, is the number of neighbouring spins that are aligned, respectively unaligned with the selected spin. By flipping the spin, the energy of the final configuration is

$$E_f = (-1) \cdot \chi_u + (+1) \cdot \chi_a = -E_i \,, \tag{8.4}$$

and so we have $\Delta E = E_f - E_i = -2E_i > 0$. Hence we have $E_i < 0$, such that $\chi_a > \chi_u$. Since $\chi_a + \chi_u = 4$, the only possibilities for the initial configuration are $\chi_a = 3, \chi_u = 1$ or $\chi_a = 4$, $\chi_u = 0$. The first case has an energy $E_i = -2$, the second one an energy $E_i = -4$, and since $\Delta E = -2E_i$, this proves that if $\Delta E > 0$, then we have $\Delta E = 4$ or $\Delta E = 8$. This proves our claim.

Hence given a temperature for the system T, we compute and store the values of $\exp(-4/T)$ and $\exp(-8/T)$ beforehand to make the algorithm more efficient.

9 Lotka-Volterra Model

The Lotka-Volterra model describes the evolution of two populations Y_1, Y_2, which are also called the prey and the predator. The relevant reactions are

$$\begin{aligned} X_1 &\xrightarrow{k_1} 2X_1 \\ X_1 + Y_2 &\xrightarrow{k_2} 2X_2 \\ X_2 &\xrightarrow{k_3} \emptyset \,, \end{aligned}$$

where $k_i, i = 1, 2, 3$ are the rates. We set $k_1 = 1$, $k_2 = 0.005$ and $k_3 = 0.6$ and start from the intial state with 100 preys and 20 predators. As mentioned in the lecture notes, it is shown that the rate equations for the Lotka-Volterra system are

$$\frac{\mathrm{d}}{\mathrm{d}t} X_1 = k_1 X_1 - k_2 X_1 X_2 \tag{9.1}$$

$$\frac{\mathrm{d}}{\mathrm{d}t} X_2 = k_2 X_1 X_2 - k_3 X_2 \,. \tag{9.2}$$

Hence the reaction rates are

$$a_1 = k_1 X_1, \quad a_2 = k_2 X_1 X_2, \quad a_3 = k_3 X_2\,. \tag{9.3}$$

Now that we know these rates, we are able to implement the Gillespie algorithm for the

9.1 The Gillespie algorithm for Lotka-Volterra

Discussion and lots of pictures

9.2 Absorbing states of the Lotka-Volterra system

Looking back at equations (9.1) and (9.2), we see that there are two so-called *absorbing states* for the Lotka-Volterra system. The first one occurs when $X_1 = 0$, since then $a_1 = 0$ and $a_2 = 0$, and the only reaction that can happen is the third reaction. Since this reaction causes the population X_2 to decrease, this will eventually lead to $X_2 = 0$ after a finite time. Our predator-prey analogy is then that when all the prey is extinct, the predators will eventually also end up being extinct, since they have no food. This scenario can therefore be called the 'extinction scenario', since after both populations will end up going extinct.

The second absorbing state is when $X_2 = 0$. In that case, $a_2 = 0$ and $a_3 = 0$, such that only the first reaction can take place. This will cause the population X_1 to increase indefinitely from that moment onwards. This scenario can be dubbed the 'victory of the prey'.

We now investigate these absorbing states. First of all, we estimate the average number of oscillations performed in the system before one of the absorbing states is reached.

Discussion

In one of those runs, where we let the algorithm run a 1000 thousand times, 97.90% of the runs ended in the extinction of X_1, and 2.10% ended in the extinction of X_2. Running the code block several times, we see that at least 95% of the runs end with the extinction of X_1. Hence we conclude that only very rarely, the predators go extinct.