MONTE CARLO METHODS

In this chapter, we discuss Monte Carlo methods.

Color code for the chapter:

- Definitions and important results are given in pink boxes.
- Skeletons of algorithms are in purple boxes.
- Examples are given in green boxes.
- Remarks and notions that can be skipped for the first read are in **black** boxes.
- Links are emphasized in blue.

Definition
Monte Carlo methods are non-deterministic methods which use **random numbers** for numerical computations.

In this chapter, we discuss three applications of Monte Carlo methods.

1. **Sampling** of complicated probability distributions.
2. **Integration** of functions in high-dimensional spaces.
3. **Optimization** of rugged functions (finding an approximation of the global minimum of a function with several local minima).

The mathematical concepts at the root of the algorithms presented in this chapter are discussed at the end of the chapter. They can be skipped for the first read, but the interested reader can come back to them for a second read.

Before discussing the Monte Carlo methods, we recall basic notions on probabilities.

I. Reminders about probabilities

The objective of this section is to give some basic reminders about probability theory useful for Physicists. The presentation is not rigorous mathematically. For a more rigorous and throrough presentation, see W. Appel, *Mathematics for Physics and Physicists*.

1. Random variable

Definition
A **random variable** X is a quantity/observable (scalar, vector, matrix, tensor, etc.) whose value is not deterministic (certain).

In Physics, there are at least two reasons which explain why some quantities are random.

- In quantum mechanics, there is an intrinsic uncertainty about observables (*e.g.*, position of a quantum particle in a box).
- In statistical physics, quantities are not intrinsically random but fluctuate because of external factors (*e.g.*, temperature of a room).

We distinguish between discrete random variables and continuous random variables.

Definition
A **discrete random variable** is a random variable that can take a discrete/countable number of values.

A **continuous random variable** is a random variable that can take a continuous/uncountable number of values.

For both discrete and continuous random variables, we can define their **support** as the set I of all possible values they can take.

Consider X the result of the rolling of a die: X is a discrete random variable and $I = \{1, 2, 3, 4, 5, 6\}$.

Consider X the number of photons counted by a photodetector: X is a discrete random variable and $I = \mathbb{N} = \{0, 1, \dots\}$.

Consider X the temperature of a room in kelvins: X is a continuous random variable and $I = [0, +\infty[$.

Consider X the velocity of a particle in one dimension: X is a continuous random variable and $I =]-c, c[$ according to special relativity (with c the speed of light in vacuum).

2. Probability distribution

a. Discrete random variable

We consider a discrete random variable X of support I. We denote x_n the values X can take. In other words $I = \{x_i\}$.

Consider X the result of the rolling of a die, then $x_1 = 1$, $x_2 = 2$, $\dots$, $x_6 = 6$.

We imagine that we measure N times the value of X. Each possible x_i then appears N_i times.

Definition
We define the **probability** $\mathcal{P}(X = x_i)$ that the random variable X equals x_i as

$$\mathcal{P}(X = x_i) = \lim_{N \to +\infty} \frac{N_i}{N}. \tag{1}$$

By construction, the probability distribution is **positive** $\mathcal{P}(X = x_i) \geq 0$ and **normalized**, namely,

$$\sum_i \mathcal{P}(X = x_i) = 1. \tag{2}$$

Consider X the result of the rolling of a die. The probability distribution is uniform, namely, $\mathcal{P}(X = x_i) = 1/6$ for all x_i's.

Consider X the number of photons measured by a photodetector during an experiment. Classically, the distribution is a Poisson distribution, namely, $\mathcal{P}(X = n) = \lambda^n e^{-\lambda}/n!$, with $\lambda > 0$ a parameter.

b. Continuous random variable

We now consider a continuous random variable X which can take a continuous number of values $x \in I$ (with I a subset of $\mathbb{R}$).
We again imagine that we measure N times the value of X. Unlike discrete random variables, we cannot count how many times each value of $x \in I$ appears, because the random variable is continuous. Instead, we can say that among the N measurements $\mathrm{d}N(x)$ are in the range $[x, x + \mathrm{d}x[$: we have binned the support into infinitesimal intervals.

Definition
We define the **elementary (or infinitesimal) probability** $\mathrm{d}\mathcal{P}(X = x)$ that the random variable X is in the range $[x, x + \mathrm{d}x[$ as

$$\mathrm{d}\mathcal{P}(X = x) = \lim_{N \to +\infty} \frac{\mathrm{d}N(x)}{N}. \tag{3}$$

We also define the **probability density** $p(x)$ such that

$$\mathrm{d}\mathcal{P}(X = x) = p(x)\mathrm{d}x. \tag{4}$$

In other words, $p(x)\mathrm{d}x$ is the probability that X lies in the range $[x, x + \mathrm{d}x[$.

By construction, the probability density is positive $p(x) \geq 0$ and **normalized**, namely,

$$\int_{x \in I} p(x)\mathrm{d}x = 1. \tag{5}$$

Consider X one cartesian component of the velocity of a gas molecule in the atmosphere at thermal equilibrium. Statistical Physics tells us that the distribution is a Gaussian distribution, namely $p(x) = e^{-x^2/(2\varsigma^2)}/\sqrt{2\pi\varsigma^2}$, with $\varsigma > 0$ a parameter (proportional to temperature).

3. Averages

From the probability distribution of a random variable X, you can compute the statistical properties of X.

Mean
For a **discrete random variable** X, its **mean** μ reads

$$\mu = \langle X \rangle = \sum_i x_i \mathcal{P}(X = x_i). \tag{6}$$

For a **continuous random variable** X, its **mean** μ reads

$$\mu = \langle X \rangle = \int_{x \in I} x p(x)\mathrm{d}x. \tag{7}$$

Consider X the result of the rolling of a die, then:

$$\mu = \frac{1}{6} \times 1 + \frac{1}{6} \times 2 + \frac{1}{6} \times 3 + \frac{1}{6} \times 4 + \frac{1}{6} \times 5 + \frac{1}{6} \times 6 = \frac{7}{2}.$$

Consider X the number of photons measured by a photodetector during an experiment, then:

$$\mu = \sum_{n=0}^{+\infty} n \frac{\lambda^n}{n!} e^{-\lambda} = \lambda.$$

Consider X one cartesian component of the velocity of a gas molecule in the atmosphere at thermal equilibrium, then:

$$\mu = \int_{-\infty}^{+\infty} x \frac{e^{-x^2/(2\varsigma)^2}}{\sqrt{2\pi\varsigma^2}} \mathrm{d}x = 0.$$

You can also access the statistical properties of any observable $g(X)$ related to X (*i.e.*, of any function g of the random variable).

Average of observables
For a **discrete random variable** X, the average of the observable $g(X)$ reads

$$\langle X \rangle = \sum_i g(x_i) \mathcal{P}(X = x_i). \tag{8}$$

For a **continuous random variable** X, the average of the observable $g(X)$ reads

$$\langle X \rangle = \int_{x \in I} g(x) p(x) \mathrm{d}x. \tag{9}$$

This definition generalizes the definition of the mean given above [with $g(x) = x$]. Averages are denoted with brackets $\langle \cdot \rangle$. In particular, one can define the variance.

Variance
The **variance** σ^2 of a random variable X is defined as

$$\sigma^2 = \langle X^2 \rangle - \langle X \rangle^2. \tag{10}$$

The square root of the variance is called the **standard deviation** σ.

Consider X the result of the rolling of a die, then:

$$< X^2 >= \frac{1}{6} \times 1^2 + \frac{1}{6} \times 2^2 + \frac{1}{6} \times 3^2 + \frac{1}{6} \times 4^2 + \frac{1}{6} \times 5^2 + \frac{1}{6} \times 6^2 = \frac{91}{6},$$

so that

$$\sigma^2 = \langle X^2 \rangle - \langle X \rangle^2 = \frac{91}{6} - \left(\frac{7}{2}\right)^2 = \frac{35}{12}.$$

Consider X one cartesian component of the velocity of a gas molecule in the atmosphere at thermal equilibrium, then:

$$\sigma^2 = \langle X^2 \rangle - \langle X \rangle^2 = \langle X^2 \rangle = \int_{-\infty}^{+\infty} x^2 \frac{e^{-x^2/(2\varsigma)^2}}{\sqrt{2\pi\varsigma^2}} \mathrm{d}x = \varsigma^2.$$

4. Correlation between random variables

We now consider two random variables X and Y. The two random variables X and Y may be **correlated** or **uncorrelated**.

Correlation between random variables.
Two random variables X and Y are **correlated** if they both deviate from their mean not independently. Mathematically, X and Y are correlated if

$$\langle XY \rangle \neq \langle X \rangle \times \langle Y \rangle. \tag{11}$$

Otherwise, the two random variables are said to be **uncorrelated**. Mathematically, X and Y are uncorrelated if

$$\langle XY \rangle = \langle X \rangle \times \langle Y \rangle. \tag{12}$$

We now explain the above statement, and we provide an illustration Fig. 1. It is first important to note that

$$\langle XY \rangle - \langle X \rangle \langle Y \rangle = \langle (X - \langle X \rangle)(Y - \langle Y \rangle) \rangle.$$

For one random variable X fluctuates around $\langle X \rangle$ and can take positive or negative values. For two random variables X and Y, if they are uncorrelated, it means that X fluctuates around $\langle X \rangle$ and Y around $\langle Y \rangle$ independently. As a result $(X - \langle X \rangle)(Y - \langle Y \rangle)$ takes positive or negative values and its average vanishes: $\langle XY \rangle - \langle X \rangle \times \langle Y \rangle = 0$. Instead, if the two random variables X and Y are correlated, it means that the fluctuations of X around $\langle X \rangle$ and of Y around $\langle Y \rangle$ are correlated. In other words, when X is slightly higher than $\langle X \rangle$, it is more likely that Y is also higher than $\langle Y \rangle$ (when X and Y are positively correlated) or smaller than $\langle Y \rangle$ (when X and Y are negatively correlated). As a result, $(X - \langle X \rangle)(Y - \langle Y \rangle)$ takes more values which are positive (when X and Y are positively correlated) or negative (when X and Y are positively correlated) and its average does not vanish: $\langle XY \rangle - \langle X \rangle \times \langle Y \rangle \neq 0$.

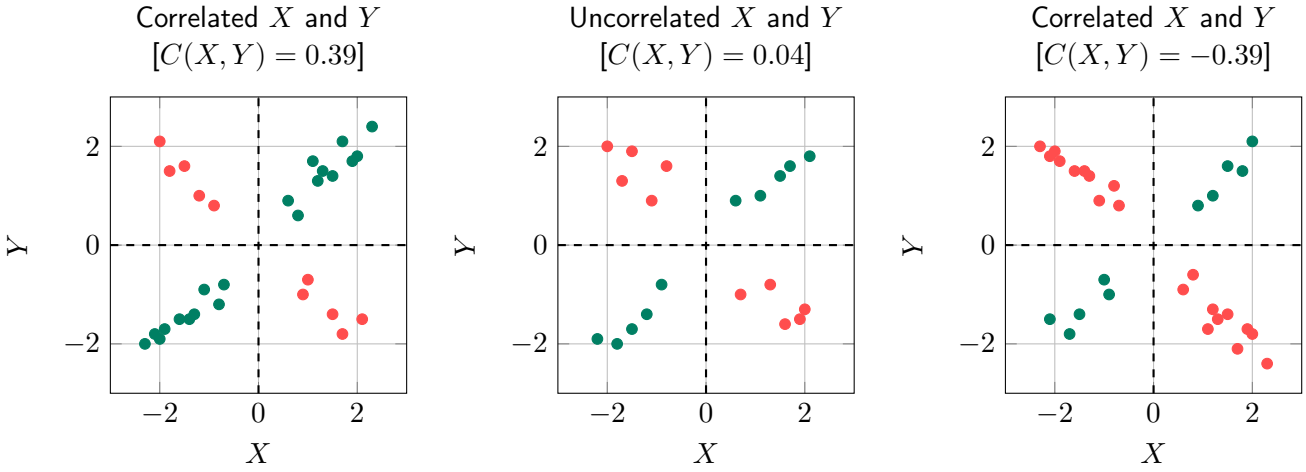


Figure 1: **Illustration of the correlation between two random variables.** We represent points corresponding to different values of two random variables X and Y both of zero mean. Points in the top-right and bottom-left corners of each plot (in green) correspond to similar deviations of X and Y from their mean. Points in the bottom-right and top-left corners of each plot (in pink) correspond to opposite deviations of X and Y from their mean. In the left panel, there are more data points in the top-right and bottom-left corners than in the bottom-right and top-left corners, so that X and Y are positively correlated. In the right panel, there is the same number of data points in the top-right and bottom-left corners and in the bottom-right and top-left corners, so that X and Y are uncorrelated. In the bottom panel, there are more data points in the bottom-right and top-left corners than in the top-right and bottom-left corners, so that X and Y are negatively correlated.

Consider X and Y the results of the rolling of two different dice: X and Y are uncorrelated, and you can check it mathematically. Each pair (x, y) of values of (X, Y) has the same probability $1/36$ to be obtained. Therefore,

$$\langle XY \rangle = \frac{1}{36} \times 1 \times 1 + \frac{1}{36} \times 1 \times 2 + \cdots + \frac{1}{36} \times 1 \times 6 + \frac{1}{36} \times 2 \times 1 + \cdots + \frac{1}{36} \times 6 \times 6 = \frac{49}{4} = \langle X \rangle \times \langle Y \rangle,$$

as $\langle X \rangle = \langle Y \rangle = 7/2$.

Consider again the rolling of two different dice. You take X as the result of the first die, and Y the maximum value of the two dice: X and Y are correlated, and you can again check that mathematically. The different values of (x, y) are

$$\{(1,1),(1,2),(1,3),(1,4),(1,5),(1,6),(2,2),(2,3),(2,4),(2,5),(2,6),$$
$$(3,3),(3,4),(3,5),(3,6),(4,4),(4,5),(4,6),(5,5),(5,6),(6,6)\}$$

of respective probabilities (make a tree)

$$\{1/36, 1/36, 1/36, 1/36, 1/36, 1/36, 1/18, 1/36, 1/36, 1/36, 1/36,$$
$$1/12, 1/36, 1/36, 1/36, 2/18, 1/36, 1/36, 5/36, 1/36, 1/6\}.$$

From the above probabilities, the average of XY reads $\langle XY\rangle = 154/9$. Besides, we still have $\langle X\rangle = 7/2$ while $\langle Y\rangle = 161/36$, such that $\langle X\rangle \times \langle Y\rangle = 1127/72 \neq \langle XY\rangle$.

Pearson correlation coefficient
The degree of correlation of two random variables X and Y is quantified by the Pearson correlation coefficient

$$C(X,Y) = \frac{\langle XY\rangle - \langle X\rangle\langle Y\rangle}{\sigma_X \sigma_Y}, \tag{13}$$

where $\sigma_X = \sqrt{\langle X^2\rangle - \langle X\rangle^2}$ and $\sigma_Y = \sqrt{\langle Y^2\rangle - \langle Y\rangle^2}$ are the standard deviations of X and Y respectively.

The Pearson correlation coefficient verifies $-1 \leq C(X,Y) \leq 1$, and vanishes when X and Y are uncorrelated.

Consider X and Y the results of the rolling of two different dice: X and Y are uncorrelated, and $C(X,Y) = 0$.

Consider again the rolling of two different dice. You take X as the result of the first die, and Y the maximum value of the two dice (see the previous example). To compute the correlation coefficient, we need to estimate the variances of X and Y. We have $\sigma_X = \sqrt{35/12}$ while the above probabilities lead to $\sigma_Y = \sqrt{2555}/36$, such that $C(X,Y) = 3\sqrt{3}/\sqrt{73} \simeq 0.61$.

5. Central Limit Theorem (CLT)

The Central Limit Theorem is an important theorem of probability theory and justifies the overwhelming importance of the Gaussian distribution in Physics.

Central Limit Theorem
Let $X_1, X_2, \ldots, X_n$ be **independent and identically distributed** (i.i.d.) random variables of mean μ and standard deviation σ. In the limit $n \to +\infty$, the empirical mean

$$\overline{X} = \frac{1}{n}\sum_{i=1}^{n} X_i \tag{14}$$

is a random variable which follows a Gaussian distribution of mean μ and of variance σ^2/n. The theorem is illustrated in Fig. 2

Two important comments are in order:

- The random variables must be **independent**: knowing the value of one random variable does not give any information on the values of the other random variables. It implies that they are **uncorrelated** from each other.
- The random variables must be **identically distributed**: they all follow the same probability distribution. In particular, they all have the same mean μ and variance σ^2.

We can provide an intuitive interpretation of the CLT. In the limit $n \to +\infty$, the probability distribution of $\overline{X}$ is a very narrow peak centered around μ, and its typical width is the standard deviation $\sigma/\sqrt{n}$ (the square root of the variance).

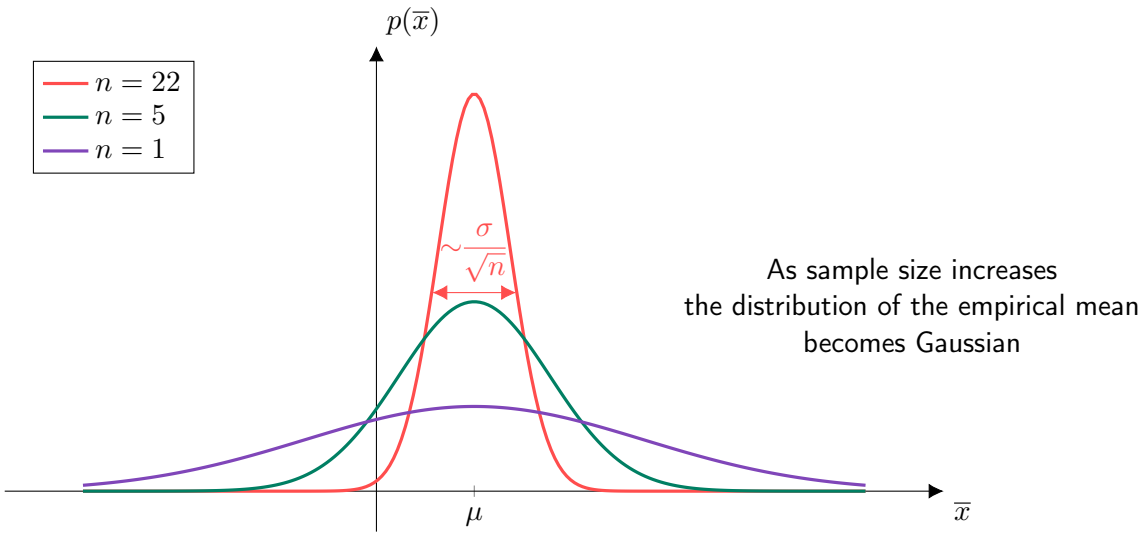


Figure 2: **Illustration of the Central Limit Theorem.** Probability density $p(\overline{x})$ of the empirical mean $\overline{X} = n^{-1}\sum_{i=1}^{n} X_i$ of i.i.d random variables X_i for several values of n.

Physical interpretation of the Central Limit Theorem
When $n \gg 1$, we have

$$\begin{cases} \mu \simeq \dfrac{1}{n}\displaystyle\sum_{i=1}^{n} X_i \quad \text{(law of large numbers)}, \\ \left|\mu - \dfrac{1}{n}\displaystyle\sum_{i=1}^{n} X_i\right| \sim \dfrac{\sigma}{\sqrt{n}}. \end{cases} \tag{15}$$

Said differently, the measure of the empirical mean $\overline{X}$ approaches the mean μ and the typical difference between a measure of $\overline{X}$ and μ is of the order of $\sigma/\sqrt{n}$ (and goes to zero when $n \to +\infty$).

II. Sampling

We now come back to the first objective of Monte Carlo methods: **sampling** probability distributions.

Objective
Introduce different methods to generate N samples $x_1, \ldots, x_N$ of a random variable X with a target probability density $p(x)$.

In this section, we mostly deal with continuous random variables but everything we will discuss also applies to discrete random variables.

1. Motivations

The sampling of probability distributions is of paramount interest for two main reasons.

In the realm of **quantum mechanics**, observables have an intrinsic uncertainty (Heisenberg's indeterminacy principle). For instance, the position of a quantum particle in $1D$ is no longer deterministic but is a random variable of probability density $|\psi(x)|^2$, with $\psi(x)$ the wavefunction of the particle.

In the realm of **statistical physics**, thermodynamic systems comprise a very large number $\mathcal{N}$ of particles (of the order of 10^{23}). It is impossible to compute the position $\vec{r_k}$ and the momentum $\vec{p_k}$ of all particles $k = 1, \ldots, \mathcal{N}$ (this is what is called a micro-state). Instead, statistical physics uses ensembles, *i.e.*, copies of the same system and computes statistical properties. For instance, in the canonical ensemble, the probability distribution of a

micro-state is given by the Boltzmann weight $\rho(\vec{r_1}, \dots, \vec{r_\mathcal{N}}, \vec{p_1}, \dots, \vec{p_\mathcal{N}}) \propto \exp[-\beta\mathcal{H}(\vec{r_1}, \dots, \vec{r_\mathcal{N}}, \vec{p_1}, \dots, \vec{p_\mathcal{N}})]/\mathcal{Z}$, with $\mathcal{H}$ the Hamiltonian of the system, $\mathcal{Z}$ its partition function, and $\beta = 1/(k_\mathrm{B}T)$.

2. Reflex: use Python librairies

When you have to generate N samples of a probability distribution $p(x)$, the first reflex is to use Python. There is a full library of different **pseudo-random number generators** with different probability distributions, look at the documentation page of `numpy.random`.

> **Pseudo-random number generators**
> The random number generators implemented in Python are not true random number generators. They instead generate deterministic sequences of numbers whose statistical properties are close to the target distribution.
>
> Be careful that the sequence repeats itself when reaching the end, which may lead to spurious correlations in your sequence of random numbers due to this periodic behavior.
>
> The sequence of pseudo-random numbers can be controlled or changed *via* the seed, *i.e.*, the initiator of the algorithm which generates pseudo-random numbers.

The most used distribution, when doing Monte Carlo calculations, is the **uniform distribution** in the range $[0, 1[$ of probability density $p(x) = 1$. It generates random numbers uniformly distributed between 0 (included) and 1 (excluded). In the following, we denote it **rand**. In Python, you can use the command `numpy.random.random_sample()`.

How to sample a distribution which is not already implemented in Python?

3. The inverse transform sampling

In simple cases, you can relate the random variable X of target probability density $p(x)$ and of support $I = [a, b[$ to another random variable Y whose probability density is the uniform distribution on $[0, 1[$. It is easy to prove (see below) that this new random variable is $Y = F(X)$ with

$$F(x) = \int_a^x p(x')\mathrm{d}x'.$$

The function $F(x)$ is called the cumulative probability distribution. You finally just have to invert the relation to get X from Y: $X = F^{-1}(Y)$.

> **The inverse transform sampling algorithm**
> If X is a random variable of probability distribution density $p(x)$ and of support $I = [a, b[$, then $Y = F(X)$ is a random variable of uniform probability distribution density on $[0, 1[$, with
>
> $$F(x) = \int_a^x p(x')\mathrm{d}x' \tag{16}$$
>
> the cumulative probability distribution.

Let X be a random variable of probability distribution density $p(x) = \alpha e^{-\alpha x}$ in the range $[0, +\infty[$. Then, we have:

$$F(x) = \int_0^x \alpha e^{-\alpha x'}\mathrm{d}x' = 1 - e^{-\alpha x},$$

which can be inverted:

$$Y = F(X) \iff Y = 1 - e^{-\alpha X} \iff X = -\frac{\ln(1-Y)}{\alpha}.$$

As a consequence, if you draw N values of the random variable Y uniformly in the range $[0, 1[$, then the corresponding

values of X will follow the exponential distribution $p(x)$.

This algorithm can be easily implemented on a computer.

The inverse transform sampling algorithm

for $i = 1 \dots N$ **do**
- $y \leftarrow \text{RAND}(0, 1)$ ▷ *Draw a random number y between 0 and 1 with a uniform probability density.*
- $x \leftarrow F^{-1}(y)$ ▷ *Apply the inverse transformation.*
- $\text{STORE}(x)$ ▷ *Store the current value of x.*

This algorithm works well, but only for random variables for which you can compute analytically the function F^{-1}. In the following sections, we discuss how to proceed when F^{-1} cannot be computed by hand.

4. The acceptance-rejection method

The simplest method which can be used for any probability distribution *a priori* is called the acceptance-rejection method (see below for limitations).

The acceptance-rejection method
To generate N samples of a random variable X of target probability distribution $p(x)$ of support $I = [a, b[$, first **draw values of X uniformly** in the support I and then **accept or reject these values with probability** $\propto p(x)$.

To implement such a procedure, we propose the following algorithm which is illustrated Fig. 3.

The acceptance-rejection method algorithm

$n = 0$ ▷ *n: number of samples of x*
while $n < N$ **do**
- $x \leftarrow \text{RAND}(a, b)$ ▷ *Draw a random number x uniformly in the support $I = [a, b[$ of the target probability distribution density $p(x)$.*
- $y \leftarrow \text{RAND}(0, p_{\max})$ ▷ *Draw a random number y uniformly in the range $I = [0, p_{\max}[$ with $p_{\max}$ the maximum value of the target probability distribution density $p(x)$.*
- **if** $y < p(x)$ **then**
  - $\text{STORE}(x)$ ▷ *Accept the current value of x.*
  - $n \leftarrow n + 1$ ▷ *Increase by one the number of sampled values of x.*

We can give another intuitive interpretation of the acceptance-rejection method when the random variable x is a scalar. At each step of the loop, you draw a point (x, y) in the rectangle $[a, b[\times[0, p_{\max}[$, and you store x if the point is below the curve $p(x)$.

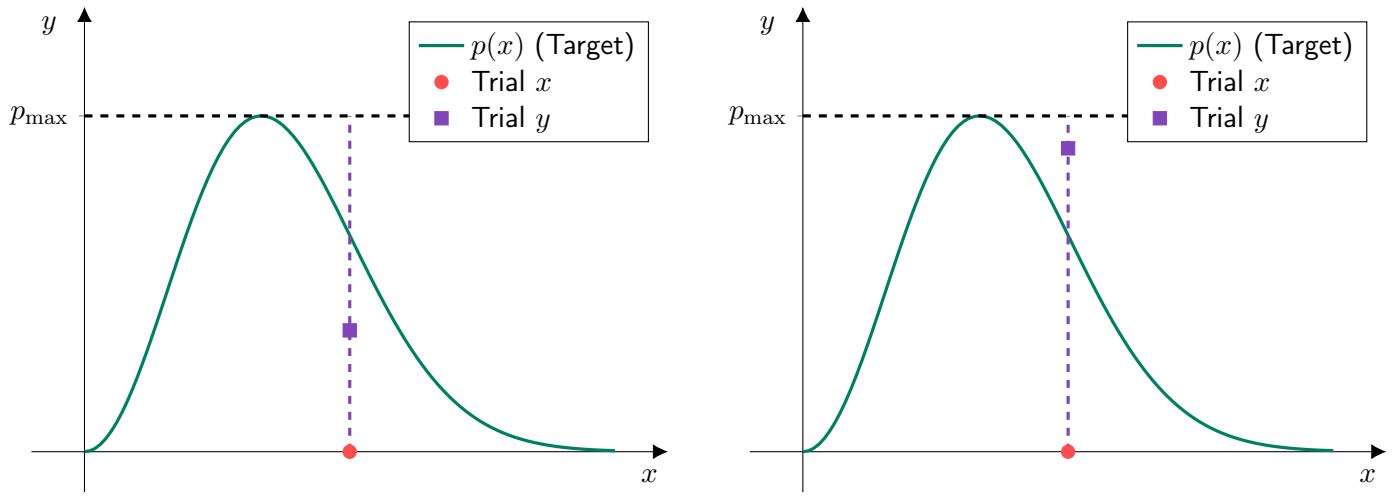
We illustrate the above statement and the algorithm with the example of the beta distribution $p(x) = 30x(1-x)^4$ in the range $[0, 1[$. The different steps of the algorithm are:

1. draw a random number x in $[0, 1[$, for instance $x = 0.4$;
2. draw a random number y in $[0, 2.4576[$ (with $2.4576 = p_{\max}$, do the calculation!), for instance $y = 1.2$;
3. compare $p(x) = 1.5552$ with 1.2: here $y < p(x)$ so the value of x is accepted.

You should also note that the above algorithm requires to sample numbers uniformly in the range $[a, b[$.

How to generate random numbers in an arbitrary range?
To generate random numbers in an arbitrary range $[a, b[$, there are two different strategies.

1. You can use the command `numpy.random.uniform(a,b)`.
2. You can also use the fact that **if X is uniformly distributed in the range $[0, 1[$, then $(b-a)X+a$ is uniformly distributed in the range $[a, b[$.**



1. Sample x from a uniform distribution on the support of $p(x)$.
2. Generate y from a uniform distribution in $[0, p_{\max}[$.
3. Accept x if $y \leq p(x)$.

Figure 3: **Illustration of the acceptance-rejection method.** In the first case (top figure), the trial value of x is accepted since $y \leq p(x)$. In the second case (bottom figure), the trial value of x is rejected since $y > p(x)$.

Two importants remarks to end this section.

- When the support of $p(x)$ has infinite bounds ($a, b = \pm\infty$), one needs to truncate the support. This can alter the accuracy of the sampling process if $p(x)$ decays slowly.
- This method is inefficient when $p(x)$ has a narrow peak, because most trial values of x will be rejected.

5. Markov Chain Monte Carlo (MCMC)

a. The method

The idea of the MCMC method is to explore the support of the probability density $p(x)$ of the random variable X by random walks. Doing this, you then construct a sequence (Markov Chain) of samples of X. There are several implementations of the method, we here only present the most famous one, the **Metropolis algorithm**. For other MCMC algorithms, see M.P. Allen and D.J. Tildesley, *Computer Simulation of Liquids*.

The Metropolis algorithm
The Metropolis algorithm can be summarized as follows.

1. Start from x_1 drawn randomly in the support of $p(x)$.
2. Given x_1, choose a random y close by.
3. Take $x_2 = y$ with probability $\min(p(y)/p(x_1), 1)$ (accept). Otherwise, set $x_2 = x_1$ (reject).
4. Eventually repeat the entire procedure starting from x_2.

We note that only ratios of $p(x)$ appear in the Metropolis algorithm. It means that the algorithm can be applied even when the normalization constant of the distribution cannot be computed analytically (this is often the case in statistical mechanics). We illustrate the Metropolis algorithm with an example below.

Imagine that you want to sample the beta distribution $p(x) = 30x(1-x)^4$ in the range $[0, 1[$. The Markov chain is initially at $x_1 = 0.3$. The different steps are:

1. draw a random number y close to x_1, for instance $y = 0.4$;

2. compute the probability $\min(p(y)/p(x_1), 1) \simeq 0.7197$ for the trial move to be accepted;
3. take $x_2 = y$ with probability 0.7197 and $x_2 = x_1$ with probability 0.2803.

We can do another example with the same probability distribution and the same value of $x_1 = 0.3$:

1. draw a random number y close to x_1, for instance $y = 0.25$;
2. compute the probability $\min(p(y)/p(x_1), 1) = 1$ for the trial move to be accepted;
3. take $x_2 = y$.

We propose a code implementation of the Metropolis algorithm below, which clarifies some parts of the definition, in particular, how to generate y randomly close by and how to accept or reject the move with the computed probability.

The Metropolis algorithm

$x \leftarrow \text{RAND}(a, b)$ ▷ *Draw a random number x uniformly in the support $I = [a, b[$ of the target probability distribution density $p(x)$.*
for $i = 1 \dots N$ **do**
  $\delta \leftarrow \text{RAND}(-\epsilon, \epsilon)$ ▷ *Draw a random number δ uniformly in the range $I = [-\epsilon, \epsilon[$ with ϵ a parameter which quantifies how far x is randomly kicked.*
  $y \leftarrow x + \delta$ ▷ *Kick x of δ to generate y close to x.*
  if $p(y) > p(x)$ **then** ▷ *y is more probable than x and is therefore automatically accepted.*
    $x \leftarrow y$
  else
    $r \leftarrow \text{RAND}(0, 1)$ ▷ *Draw a random number r uniformly in the range $I = [0, 1[$.*
    if $p(y)/p(x) > r$ **then** ▷ *Accept y with probability $p(y)/p(x)$.*
      $x \leftarrow y$
  $\text{STORE}(x)$ ▷ *Store the current value of x.*

It is important to note that **you must always store the value of x, whether it has changed or not**! A schematic representation of the Markov chain is shown Fig. 4.

We can give an intuitive interpretation of the algorithm. Starting from a value x, we perturb it to get another value y. If this value is more probable, *i.e.*, if $p(y) \geq p(x)$, then this new value is automatically accepted. Otherwise, the new value y is accepted with probability $p(y)/p(x)$ (which is indeed between 0 and 1). In practice, this is done by drawing a random number r uniformly between 0 and 1 and accepting y when $p(y)/p(x) > r$. Indeed, by definition, the probability that r is smaller than $p(y)/p(x)$ is

$$\mathcal{P}(r \leq p(y)/p(x)) = \int_0^{p(y)/p(x)} 1 \times \mathrm{d}r' = \frac{p(y)}{p(x)}.$$

b. Practical implementation of the Metropolis algorithm

We now discuss practical aspects regarding the Metropolis algorithm. The latter has parameters (*e.g.*, ϵ, N) that you need to choose wisely. Besides, we usually distinguish two phases in a MCMC run, the **burn-in** phase and the **equilibrium** phase.

Burn-in phase. In many applications, the random variable is not a scalar but a vector living in a high-dimensional space $\vec{X}$ (phase space), and you do not know *a priori* the most probable values of $\vec{X}$. As a consequence, you may start in a low-probability region of phase space and the MCMC will first converge to more probable regions of phase space (see Fig. 4). This first phase of the algorithm gives too much weight to regions of phase space of small probability and must not be considered in the statistical analysis of the data. In other words, they must be thrown away.

Length of the burn-in phase. The number of steps $N_{\mathrm{b-i}}$ of the burn-in phase depends on the initial condition and can be estimated by looking at the time series of the random variable itself or an associated quantity, for

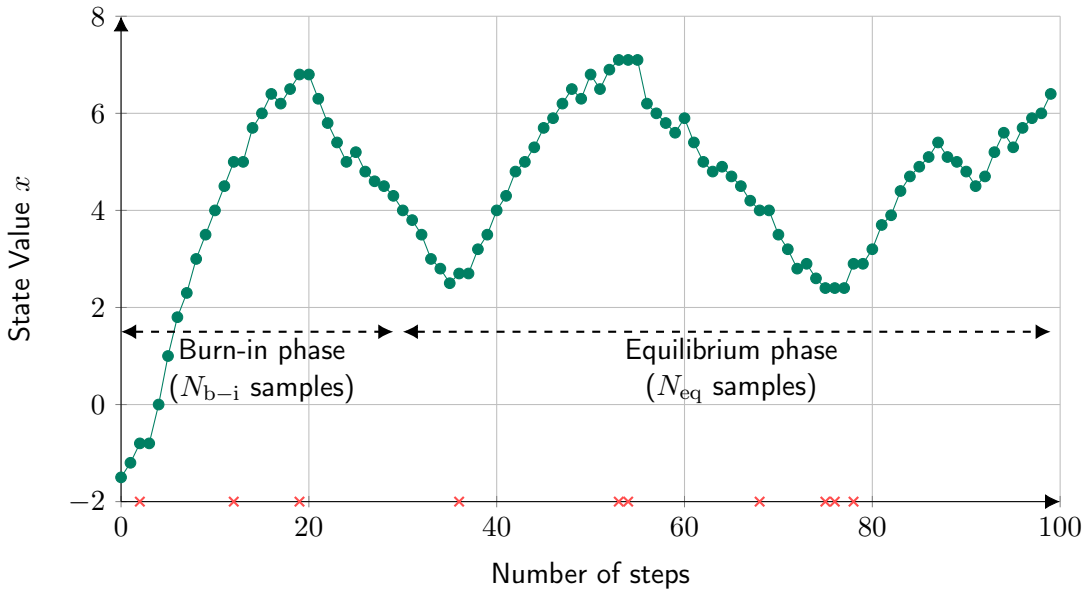


Figure 4: **Schematic representation of a Markov chain from the Metropolis algorithm.** Pink crosses mark the steps where the trial moves have been rejected, and where the Markov chain keeps the same value at the step after. For all other steps, the trial moves have been accepted. The first $N_{\mathrm{b-i}}$ samples of the random variable X are thrown away (burn-in phase) because they may be sampled in a region of low probability of X and are not representative of $p(x)$. Only the remaining N_{eq} samples (equilibrium phase) are kept for statistical analysis.

instance the instantaneous energy of the system in statistical mechanics if the random variable is a vector. You have to choose $N_{\mathrm{b-i}}$ large enough so that the time series of X (or the associated quantity) reaches a stationary state where it fluctuates around a fixed value.

Equilibrium phase. Once you have reached regions of high probability, you start the equilibrium phase where you gather samples of X to compute its statistical properties afterwards (see Fig. 4).

Length of the equilibrium phase. The number of steps N_{eq} ($N = N_{\mathrm{b-i}} + N_{\mathrm{eq}}$) of the equilibrium phase must be large enough. How large? Of course, the larger, the better... But the longer the equilibrium phase, the slower the algorithm. Therefore, when to stop the chain? In the following, we present two tests to decide whether the equilibrium phase is long enough.

First test: running averages. To check that N_{eq} is large enough, you can first plot the running empirical average of the random variable X as a function of the step n in the equilibrium phase (see Fig. 5).

> **Definition of the running empirical average**
> The running empirical average of a Markov chain is defined as
>
> $$\frac{1}{n - N_{\mathrm{b-i}}} \sum_{i=N_{\mathrm{b-i}}+1}^{n} x_i. \tag{17}$$

If the equilibrium phase of the Markov chain is long enough then this quantity converges to a constant value (which is the estimated mean according to the central limit theorem). In the case where the random variable is a vector (for instance in statistical mechanics), you can consider the running empirical average of the energy. However, this is not a strict-enough test. Below, we present a second and more strict test.

Second test: correlation functions. To check that N_{eq} is large enough, and that your sampling of the probability distribution $p(x)$ is sufficient, you need to verify that you have enough uncorrelated samples. For that, you can compute the correlation function $C(n)$ of the Markov chain in the equilibrium phase.

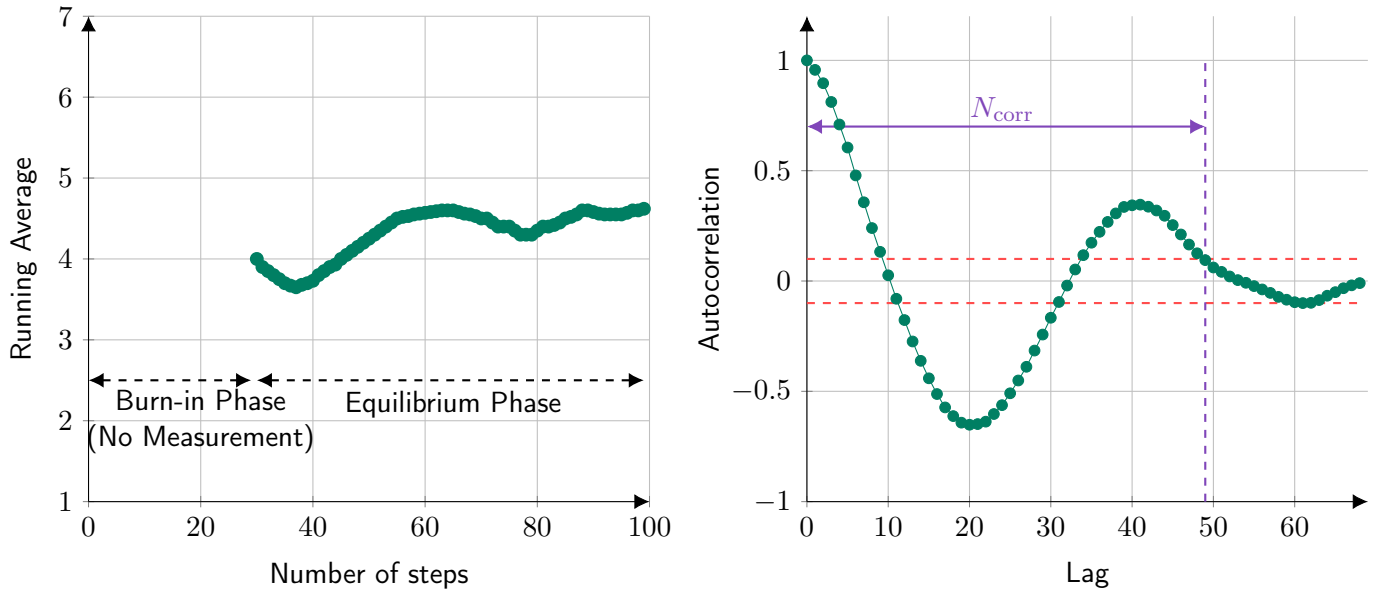


Figure 5: **Choosing the length of the equilibrium phase during a Markov Chain Monte Carlo algorithm.** You can monitor the running average of the sampled random variable (or related quantities) and check its convergence (left panel). You can also compute the correlation function of the sampled random variable (right panel), estimate the typical number of steps N_{corr} over which samples are correlated (below which the correlation function remains under a defined threshold, *e.g.*, 0.1), and check *a posteriori* that the length of the equilibrium phase N_{eq} verifies $N_{\text{eq}} \gg N_{\text{corr}}$. On this example, the running average seems to plateau. However, $N_{\text{corr}} \simeq N_{\text{eq}}$, therefore the sampling is not sufficient.

Definition of the correlation function

The correlation function is defined as the Pearson correlation coefficient of the Markov chain X and of the same Markov chain but shifted by a number of steps n called the "lag". Mathematically,

$$C(n) = \frac{\langle x_i x_{i+n} \rangle - \mu^2}{\sigma^2}, \tag{18}$$

where the averages and variances are computed empirically from the Markov chain:

$$\langle x_i x_{i+n} \rangle = \frac{1}{N_{\text{eq}} - n} \sum_{i=N_{\text{b-i}}+1}^{N-n} x_i x_{i+n}, \quad \mu = \frac{1}{N_{\text{eq}}} \sum_{i=N_{\text{b-i}}+1}^{N} x_i, \quad \sigma^2 = \frac{1}{N_{\text{eq}}} \sum_{i=N_{\text{b-i}}+1}^{N} x_i^2 - \mu^2. \tag{19}$$

In the case where the random variable is a vector (for instance in statistical mechanics), you just replace the products in the above definition by scalar products.

A typical shape of $C(n)$ is shown Fig. 5. It decays from 1 (by definition) to 0 as a function of the lag n, meaning that samples become uncorrelated after many kicks. The typical number of steps N_{corr} to perform between two uncorrelated samples can be estimated from the shape of $C(n)$. This is typically done by choosing a cutoff value (for instance 0.1, its precise value is not important). Then, N_{corr} is defined as the smallest value of the lag n such that $C(n)$ remains smaller than the cutoff for $n \geq N_{\text{corr}}$. The sampling is correct when $N_{\text{eq}} \gg N_{\text{corr}}$ (at least a hundred times). This is usually more strict than the first test based on running averages.

A last parameter: the amplitude of the random kicks. The parameter ϵ represents the typical amplitude of kicks in the Markov chain to go from one sample to the consecutive one. Of course, if you take ϵ is very small, you almost keep x unchanged so trial moves are often accepted. However, you do not explore the support of $p(x)$ efficiently and you increase the number of steps N_{corr} to perform before generating an uncorrelated sample. This forces you to increase N_{eq} and therefore the total duration of your simulation. Instead, if you take ϵ too large, you change x a lot and trial moves are more likely to be rejected. But when they are accepted, it is more

likely that you generate an uncorrelated sample. As a result, you do not explore the support of $p(x)$ efficiently again. Therefore, there is a compromise to make. Typically, ϵ is set such that the acceptance rate of the trial moves, *i.e.*, the percentage of moves which are accepted, lies between 20 % and 50 %.

We can summarize the content of this paragraph in the advice given below.

Criteria for an accurate Monte Carlo sampling
Choose the amplitude of random kicks to have an acceptance rate between $20\,\%$ and $50\,\%$.

Throw away the burn-in phase when the Markov chain converges to a stationary state.

Make the equilibrium phase long enough to accumulate uncorrelated samples (compute the correlation function).

III. Integration

In this section, we discuss an application of Monte Carlo methods, namely, the **integration** of functions.

1. The method

Imagine that you want to compute the integral of a function f on a domain Ω embedded in a D-dimensional space. Then you can rewrite the integral I as

$$I = \int_{\vec{x}\in\Omega} f(\vec{x})\mathrm{d}^D\vec{x} = \int_{\vec{x}\in\Omega} \frac{f(\vec{x})}{p(\vec{x})}p(\vec{x})\mathrm{d}^D\vec{x} = \int_{\vec{x}\in\mathbb{R}^D} \frac{f(\vec{x})}{p(\vec{x})}w(\vec{x})p(\vec{x})\mathrm{d}^D\vec{x},$$

with $w(\vec{x})$ the function which equals 1 on Ω and 0 otherwise. You then recognize the average of fw/p for a random variable $\vec{X}$ of probability distribution density $p(\vec{x})$.

Monte Carlo integration
The integration of a function $f(\vec{x})$ for $\vec{x} \in \Omega \subset \mathbb{R}^D$ can be done thanks to the sampling of a random variable X of probability distribution density $p(\vec{x})$ such that

$$I = \int_{\vec{x}\in\Omega} f(\vec{x})\mathrm{d}^D\vec{x} = \left\langle \frac{f(\vec{x})}{p(\vec{x})}w(\vec{x}) \right\rangle . \tag{20}$$

We can easily implement this method thanks to an algorithm.

Monte Carlo integration algorithm
The algorithm to perform a Monte Carlo integration works as follows.

1. Generate N samples $\{\vec{x_1}, \ldots, \vec{x_N}\}$ of the random variable X of probability distribution density $p(\vec{x})$ (see the section about **sampling**).
2. Compute the integral I as

$$I = \frac{1}{N}\sum_{i=1}^{N} \frac{f(\vec{x_i})}{p(\vec{x_i})}w(\vec{x_i}). \tag{21}$$

Consider a one-dimensional integral on $\Omega = [a, b]$. Then, the simplest choice is to take $p(x)$ uniform on that support, namely $p(x) = 1/(b-a)$. In this case, the data points sampled always lie in the integration domain, and the numerical estimate of I reads

$$I = \frac{b-a}{N}\sum_{i=1}^{N} f(x_i).$$

2. Choice of the probability distribution

We now need to discuss the choice of $p(\vec{x})$ for the Monte Carlo integration. We give several general rules below.

Choice of the probability distribution density for importance sampling
▶ The probability distribution density p for the Monte Carlo integration of a function f in a domain Ω must be a proper probability distribution density with a support which includes the integration domain. In other words $p(\vec{x}) \geq 0$, the support I of $p(\vec{x})$ is such that $\Omega \subset I$ and

$$\int_{\vec{x} \in I} p(\vec{x}) \mathrm{d}^D \vec{x} = 1.$$

▶ The probability distribution density should be taken so that the probability is higher in regions of Ω which contribute the most to the integral of f (regions where f is higher). In other words, regions which contribute the most to the integral are visited more often to increase the accuracy. Said differently, f and p should have similar shapes.

▶ The probability distribution density should be taken so that one never has $f(\vec{x}) \gg p(\vec{x})$. Otherwise the ratio $f(\vec{x})/p(\vec{x})$ becomes very large, leading to a poor accuracy.

Therefore, you need to adapt $p(\vec{x})$ depending on the function $f(\vec{x})$ you want to integrate, but also on the integration domain. This is called **importance sampling**. In particular, the uniform distribution discussed in the example above is not always the distribution which yields the most accurate estimates of integrals.

3. Comparison with other methods

We end this section by comparing the efficiency of the Monte Carlo integration with other deterministic methods to compute integrals (trapezoidal rule, Simpson's rule, etc.).

Numerical accuracy of the Monte Carlo integration
If the integral to compute is

$$I = \int_{\vec{x} \in \Omega} f(\vec{x}) \mathrm{d}^D \vec{x},$$

we approximate it thanks *via*

$$I = \left\langle \frac{f(\vec{x})}{p(\vec{x})} w(\vec{x}) \right\rangle \simeq \frac{1}{N} \sum_{i=1}^{N} \frac{f(\vec{x_i})}{p(\vec{x_i})} w(\vec{x_i}). \tag{22}$$

via the sampling of N points.
According to the Central Limit Theorem, when $N \gg 1$, the numerical error is **of the order** $\sqrt{N}$:

$$\left| I - \frac{1}{N} \sum_{i=1}^{N} \frac{f(\vec{x_i})}{p(\vec{x_i})} w(\vec{x_i}) \right| \sim \frac{1}{\sqrt{N}}. \tag{23}$$

As a conclusion, the **Monte Carlo integration performs badly for one-dimensional integrals**, as compared to the trapezoidal method (error $\sim 1/N^2$) or the Simpson's method (error $\sim 1/N^3$). However, it is important to remember that the error in deterministic methods comes from the step size δ to discretize space. For instance, the error for the trapezoidal method is $\sim \delta^2$. In one dimension, as $\delta \sim 1/N$, we get the above estimate. What about larger dimensions?

In D dimensions, if you keep the number N of points fixed, the step size now behaves as $\delta \sim 1/N^{1/D}$ [$N^{1/D}$ points per dimension such that you have $(N^{1/D})^D = N$ points in total]. This is because you discretize space in N small volumes of size δ^D. As a consequence, the error for the trapezoidal method scales as $\sim 1/N^{2/D}$ in D dimensions and increases with the dimension D ("curse of dimensionality"). On the opposite, the Monte

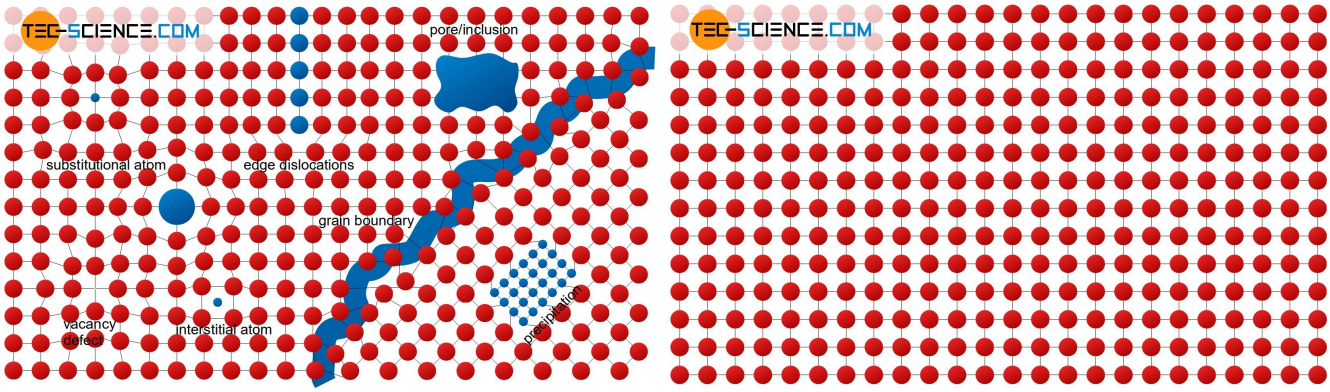


Figure 6: **Illustration of the physical origin of the simulated annealing algorithm.** Crystal with defects (left) and perfect crystal (right). Illustrations from https://www.tec-science.com/material-science/structure-of-metals/crystallographic-defects/ (last checked: 17/09/2024).

Carlo integration has a precision of order $1/\sqrt{N}$ independently of the dimension. Therefore, for $D \geq 4$, the Monte Carlo integration becomes more efficient than the trapezoidal method, at fixed number of samples (and roughly computation time). In other words, the **Monte Carlo integration performs better than any common deterministic method for high-dimensional integrals**. This is typically the case in statistical mechanics.

We finally note that in relatively small dimensions ($2 \leq D \leq 4$), the deterministic methods have a better accuracy than the Monte Carlo integration. However, a **Monte Carlo integration may be more suitable, especially when the integration domain is complicated** (*e.g.*, when integrating on the surface of a sphere or a torus).

We summarize below the conditions to implement a Monte Carlo integration.

> **When to use a Monte Carlo integration?**
> Monte Carlo integration is preferred for **high-dimensional integrals** or for **integrals with complicated domains of integration**.

IV. Optimization

In this section, we discuss a second application of Monte Carlo methods, namely, the **research of the global minimum** of functions. The name of this method is **simulated annealing**. Of course, you can also find the global maximum of functions with the simulated annealing method, you just have to apply the procedure to the opposite of the function you want to maximize.

1. Motivation

Before describing the algorithm, we discuss its origin. Interestingly, as to now, we discussed algorithms made by computer scientists to solve Physics problems. The **simulated annealing** algorithm is instead an algorithm rooted in Physics principles mainly used to solve optimization problems.

Consider a liquid above its melting temperature T_m and cool it. If you cool it fastly, most liquids eventually cristallize but the crystalline structure obtained has a lot of defects, *e.g.*, dislocations, vacancies or interstitials (see Fig. 6 left). Instead, if you cool the liquid very slowly, it cristallizes with almost no defects: the system has found its ground state, the perfect crystal (see Fig. 6 right). Said differently, when you cool the liquid fastly, it converges to a local free energy minimum. Instead, when you cool the liquid slowly, it is able to find its global free energy minimum.

To understand more deeply this phenomenon, consider a simpler problem: one particle in a one-dimensional double-well potential (see Fig. 7). Thermal fluctuations give kinetic energy to the particle which allows it to

explore its potential energy landscape $V(x)$. Said differently, the Boltzmann distribution gives that the probability distribution density of the random variable x (the position of the particle) is $p(x) \propto e^{-V(x)/(k_\mathrm{B}T)}$, with k_B the Boltzmann constant. As a consequence, $k_\mathrm{B}T$ corresponds to the typical excursion of energy such that events remain highly probable.
At high-enough temperature (left panel), the particle can jump from one well to another. In terms of probability distribution, this is because the height of the energy barrier is smaller than $k_\mathrm{B}T$. However, at lower temperature (right panel), *i.e.*, when the height of the energy barrier becomes larger than $k_\mathrm{B}T$, the particle does not have enough kinetic energy (or thermal fluctuations) to jump. It is therefore more likely to remain in the closest minimum, even though it is not the global minimum. As a consequence, if the temperature is decreased fastly, then the system may converge to a sub-optimal situation, corresponding to a local minimum. Instead, if the temperature is decreased slowly, the particle can equilibrate and find the global minimum when its kinetic energy is high enough and remain in it when its kinetic energy is too small to overcome the energy barrier. As a result, it is more likely to find the global minimum.

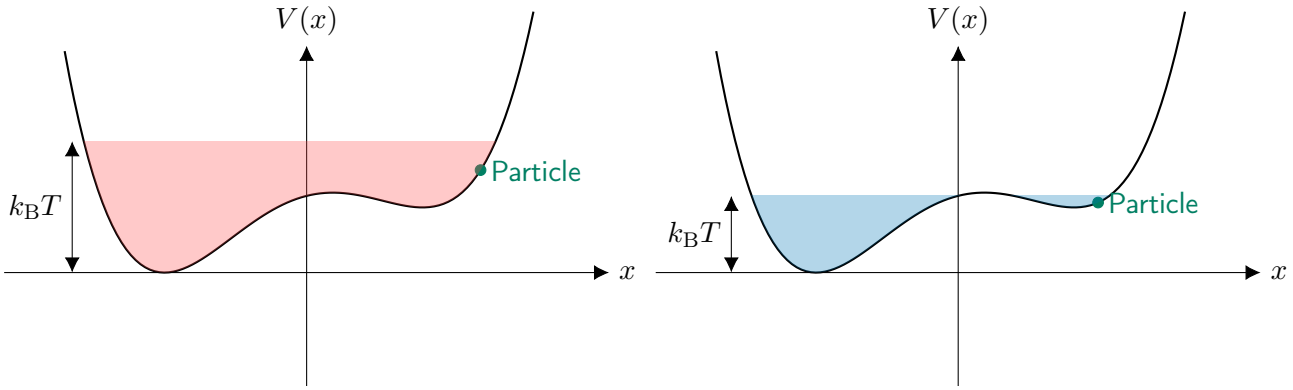


Figure 7: **An intuitive explanation of the behavior of the simulated annealing algorithm.** Particle evolving in a one-dimensional asymmetric double-well potential. The shaded area corresponds to the typical excursion of the particle at temperature T thanks to thermal fluctuations. The high-temperature scenario is represented on the left, the low-temperature scenario on the right.

2. Description of the simulated annealing algorithm

We now describe the simulated annealing algorithm, which combines an "equilibrium-like" dynamics at temperature T with a cooling protocol.

Simulated annealing protocol
To minimize a function $f(X)$ for a given random variable X, we proceed as follows.

1. Simulate the "equilibrium dynamics" of an effective system at "temperature" T of "potential energy" $f(x)$, such that its probability distribution density is $p(x) \propto e^{-f(x)/T}$ (we set $k_\mathrm{B} = 1$). For that, proceed using the Metropolis algorithm (see the section about **sampling**).
2. Slowly decrease the "temperature".
3. Start from a "very high temperature" and end the algorithm at "very low temperature".

3. Practical implementation of the simulated annealing algorithm

We now discuss practical aspects regarding the implementation of the simulated annealing protocol.

Cooling protocol. The choice of a cooling protocol depends on the optimization problem to solve. Simplest cooling protocols imply an arithmetic or a geometric progression of the temperature, but more advanced schemes exist.

Linear cooling for a simulated annealing algorithm
The linear cooling corresponds to an arithmetic progression of the temperature, namely, to a linear decrease in the temperature from T_i to T_f in N steps. Said differently, the temperature is given by

$$T_{n+1} = T_n - \frac{T_\mathrm{i} - T_\mathrm{f}}{N} \Longleftrightarrow T_n = T_\mathrm{i} - \frac{T_\mathrm{i} - T_\mathrm{f}}{N} n \quad \text{(linear cooling)}, \tag{24}$$

at step $n < N$.

Exponential cooling for a simulated annealing algorithm
The exponential cooling corresponds to a geometric progression of the temperature, *i.e.*, to an exponential decay of the temperature from T_i to T_f at a constant relative rate Δ. Said differently, the temperature is given by

$$T_{n+1} = T_n(1 - \Delta) \Longleftrightarrow T_n = T_\mathrm{i}(1 - \Delta)^n \quad \text{(exponential cooling)}, \tag{25}$$

at step n. The number of steps N is set such that $T_N = T_\mathrm{f}$.

Before discussing the different parameters of the simulating annealing technique, we propose a schematic implementation of the exponential cooling to minimize a function $f(x)$.

The simulated annealing algorithm with exponential cooling

$T \leftarrow T_\mathrm{i}$ ▷ *The temperature starts from its initial value.*
while $T > T_\mathrm{f}$ **do**
  $\delta \leftarrow \mathrm{RAND}(-\epsilon, \epsilon)$ ▷ *Draw a random number δ uniformly in the range $I = [-\epsilon, \epsilon[$ with ϵ a parameter which quantifies how far x is randomly kicked.*
  $y \leftarrow x + \delta$ ▷ *Kick x of δ to generate y close to x.*
  if $f(y) < f(x)$ **then** ▷ *$f(y)$ is smaller than $f(x)$ so y is therefore automatically accepted.*
    $x \leftarrow y$
  else
    $r \leftarrow \mathrm{RAND}(0, 1)$ ▷ *Draw a random number r uniformly in the range $I = [0, 1[$.*
    if $\exp\left(-\frac{f(y) - f(x)}{T}\right) > r$ **then** ▷ *Accept y with probability $\exp\left(-\frac{f(y) - f(x)}{T}\right)$.*
      $x \leftarrow y$
  $T \leftarrow T(1 - \Delta)$ ▷ *Cool the system.*

Initial temperature. The choice of the initial temperature is dictated by the fact that initially, the system must be able to overcome all energy barriers. In other words, the initial temperature should ideally be much larger than the difference between the global maximum and the global minimum of the function to minimize. Of course, these maximum and minimum values are usually not known beforehand, so you should take the initial temperature high enough by trying different values.

Final temperature. Similarly, the choice of the final temperature is dictated by the fact that eventually, the system must remain in the global minimum if it finds it. In other words, the final temperature should ideally be much smaller than the energy barriers connected the global minimum to adjacent local minima. Again, these energy barriers are not known beforehand, so you should take the final temperature small enough by trying different values.

Cooling rate. The quality of the optimization is mostly controlled by the value of the cooling rate [which is $(T_\mathrm{i} - T_\mathrm{f})/N$ for the linear cooling, and Δ for the exponential cooling]. The smaller the cooling rate, the better the optimization. In other words, if you take a smaller cooling rate, you will be able to reach smaller and smaller values of the function to optimize.

Efficiency of the simulated annealing protocol
The simulated annealing algorithm is able to find smaller and smaller minima of $f(x)$ as the cooling rate decreases. The optimized value of $f(x)$ found at the end of the simulated annealing protocol slowly converges to the value of the global minimum as the cooling rate goes to 0.

However, **the simulated annealing protocol hardly finds the global minimum itself.** Said differently, the simulated annealing algorithm helps in finding good approximations of the value of the minimum of a function, but not necessarily its actual minimum location.

V. Mathematics of Monte Carlo algorithms

1. The inverse transform sampling

We proove the mathematical result at the origin of the inverse transform sampling. The probability distribution density $q(y)$ of Y is derived as follows. The probability of a given value y of Y is obtained as the sum over all values x of X such that $y = F(x)$, meaning that

$$q(y) = \int_{x\in I} p(x)\delta(y - F(x))\mathrm{d}x,$$

when δ is the Dirac distribution which enforces the constraint $y = F(x)$. We now do the change of variable $y' = F(x)$ in the integral [$\mathrm{d}y' = F'(x)\mathrm{d}x = F'(F^{-1}(y'))\mathrm{d}x$], and we get

$$q(y) = \int_{y'\in F(I)} p(F^{-1}(y'))\delta(y - y')\frac{\mathrm{d}y'}{F'(F^{-1}(y'))}.$$

By definition, $F'(x) = p(x)$ and we eventually obtain that

$$q(y) = \int_{y'\in F(I)} p(F^{-1}(y'))\delta(y - y')\frac{\mathrm{d}y'}{p(F^{-1}(y'))} = \int_{y'\in F(I)} \delta(y - y')\mathrm{d}y' = 1,$$

where we have used the integral property of the δ-distribution. The probability distribution density of Y is constant, hence Y is a uniform random variable. We finally note that because $p(x) \geq 0$, $F(x)$ is an increasing function. Its bounds are $F(a) = 0$ (by definition) and $F(b) = 1$ [because $p(x)$ is normalized]. This proves that Y is a uniform random variable in $[0, 1[$.

2. Markov Chain Monte Carlo

a. Detailed balance

The mathematics behind the Markov Chain Monte Carlo (MCMC) method rely on the concept of **detailed balance** that we will present briefly. The presentation here follows the book from D. Frenkel and B. Smit, *Understanding Molecular Simulation*.

Imagine that you have an algorithm which is able to sample a probability distribution density $p(x)$ of a random variable X. We recall that, by definition, $p(x) \simeq N(x)/N$ for $N \gg 1$, with $N(x)$ the number of times x is visited, and N the total number of steps.
Let's focus on a possible value x of X. If the algorithm is indeed able to sample $p(x)$, it means that the average number of accepted trial moves leaving x should be equal to the average number of accepted trial moves reaching x. Otherwise, the relative number of times $N(x)/N$ that x is visited will change, and the estimated probability $p(x)$ too. The number of accepted trial moves leaving x is

$$\sum_{x'\neq x} p(x)W(x \to x'),$$

where we sum, over all possible values x' the random variable X can take (except x), the probability that X ends up in x' starting from x. The quantity $W(x \to x')$ is the transition probability to go from x to x', or equivalently, the conditional probability that the Markov chain ends in x' knowing that it was at x at the step before. Similarly, the number of accepted trial moves reaching x is

$$\sum_{x'\neq x} p(x')W(x' \to x),$$

where we sum, over all possible values x' the random variable X can take (except x), the probability that X ends up in x starting from x'. As a consequence, the MCMC algorithm samples the probability distribution density $p(x)$ if the equality

$$\sum_{x' \neq x} p(x) W(x \to x') = \sum_{x' \neq x} p(x') W(x' \to x). \tag{26}$$

is verified for all values of x. This condition is called **global balance** and can be satisfied by a proper choice of the transition probabilities $W(x \to x')$.

A particular situation where this condition is satisfied is when we impose that the terms in the two sums equal each other separately. This is a more restrictive condition called **detailed balance**.

> **Detailed balance**
> If a Markov Chain Monte Carlo algorithm verifies the **detailed balance** property, namely,
>
> $$p(x) W(x \to x') = p(x') W(x' \to x) \iff \frac{W(x' \to x)}{W(x \to x')} = \frac{p(x)}{p(x')}, \tag{27}$$
>
> by a proper choice of transition rates $W(x \to x')$, then the algorithm will sample the probability distribution density $p(x)$.

b. Application to the Metropolis algorithm

We now show that the Metropolis algorithm verifies the detailed balance condition. For that, we need to derive the expression of the transition rate $W(x \to x')$. We can decompose it into two factors:

$$W(x \to x') = \alpha(x \to x') \times \mathrm{Acc}(x \to x'), \tag{28}$$

where $\alpha(x \to x')$ stands for the probability to propose a trial move from x to x', while $\mathrm{Acc}(x \to x')$ corresponds to the probability that this trial move is accepted. In the Metropolis algorithm, we do random kicks uniformly distributed around the current position of the Markov chain, and as a consequence $\alpha(x \to x') = \alpha(x' \to x)$ (symmetric). Indeed, $\alpha(x \to x') = \alpha(x' \to x) = 0$ if $|x' - x| > \epsilon$ while $\alpha(x \to x') = \alpha(x' \to x) = 1/(2\epsilon)$ otherwise. As a consequence, we just have to check that

$$\frac{W(x' \to x)}{W(x \to x')} = \frac{\mathrm{Acc}(x' \to x)}{\mathrm{Acc}(x \to x')} = \frac{p(x)}{p(x')}.$$

Consider first the case where $p(x) \geq p(x')$, then $\mathrm{Acc}(x' \to x) = 1$ (the move is automatically accepted), while $\mathrm{Acc}(x \to x') = p(x')/p(x)$, and the above equality is satisfied. Consider then the case where $p(x) < p(x')$, then $\mathrm{Acc}(x' \to x) = p(x)/p(x')$, while $\mathrm{Acc}(x \to x') = 1$, and the above equality is again satisfied.

3. The acceptance-rejection method

The mathematical justification of the acceptance-rejection method can also be done using the concept of detailed balance. For that, we note that we can still define a transition rate $W(x \to x')$ in the form:

$$W(x \to x') = \alpha(x \to x') \times \mathrm{Acc}(x \to x'). \tag{29}$$

Again $\alpha(x \to x') = \alpha(x' \to x)$ since $\alpha(x \to x') = 1/(b-a)$ (uniform probability to pick a value x). As a consequence, we just have to check that

$$\frac{W(x' \to x)}{W(x \to x')} = \frac{\mathrm{Acc}(x' \to x)}{\mathrm{Acc}(x \to x')} = \frac{p(x)}{p(x')}.$$

The acceptance rate $\mathrm{Acc}(x \to x')$ depends on the value of the random variable y. The move $x \to x'$ is accepted if y drawn uniformly in $[0, p_{\max}[$ is smaller than $p(x')$ (it does not depend on the initial state point). The probability that y is indeed smaller than $p(x')$ equals

$$\mathcal{P}(y \leq p(x')) = \int_0^{p(x')} \frac{1}{p_{\max}} \mathrm{d}y' = \frac{p(x')}{p_{\max}},$$

where $1/p_{\max}$ is the probability distribution density of the uniform probability distribution on the support $[0,\, p_{\max}[$. As a conclusion $\mathrm{Acc}(x \to x') = p(x')/p_{\max}$. The same reasoning leads to $\mathrm{Acc}(x' \to x) = p(x)/p_{\max}$, and the detailed balance condition is satisfied.