Notes on Monte Carlo Methods

Enrico M. Malatesta

April 21, 2021

Contents

1 Introduction **2**
1.1 Buffon's needle experiment . . . 2
1.1.1 Analytical solution . . . 3
1.1.2 Monte Carlo code . . . 4

2 Computing volumes **5**
2.1 The basic idea of Monte Carlo method . . . 5
2.1.1 Pseudo-random Numbers . . . 6
2.1.2 Sample size and accuracy . . . 7
2.2 Computing Volumes By Using a Grid . . . 9
2.3 Computing Volumes By Using Monte Carlo Sampling . . . 11
2.4 Counting objects . . . 14
2.5 Multidimensional integration . . . 14
2.5.1 Inverse transform sampling . . . 15
2.5.2 Rejection sampling . . . 17
2.5.3 Importance sampling . . . 18

3 Markov Chains **20**
3.1 Stochastic Processes . . . 20
3.2 Markov Chains . . . 23
3.2.1 Stationary Distributions . . . 25
3.2.2 Averages in ergodic systems . . . 27
3.3 Global Balance and Detailed Balance . . . 30

4 Basic Markov Chain Monte Carlo algorithms **30**
4.1 Metropolis Algorithm . . . 31
4.1.1 Metropolis algorithm in high dimensions . . . 33
4.2 Gibbs Sampling . . . 34

5 The Simulated Annealing Algorithm **35**
5.1 The Boltzmann distribution 35
5.2 Sampling from the Boltzmann distribution 38
5.3 Optimization with the Metropolis algorithm 39
5.4 Simulated Annealing 41

1 Introduction

The family of Monte Carlo methods provide approximate solutions to a variety of mathematical problems by performing statistical sampling experiments on a computer. The modern version of Monte Carlo methods was invented by S. Ulam in the late 1940s, while he was working on nuclear fission at Los Alamos laboratories. Its name was subsequently coined by N. Metropolis and it derives from the fact that those methodologies use extensively random number generators, i.e. numbers obtained through a roulette-like machine of the kind utilized in the gambling Casinos of the Monte Carlo Principate. Despite these numerical techniques have been widely used in physics over the last 70 years, their interest in the context of Bayesian statistics [1] was fully appreciated only in the late eighties [2]. Today Monte Carlo methods are widely used in engineering, business, finance and many other fields.

As we shall see Monte Carlo methods can be helpful in solving two important problems:

- *Sampling*: given a random variable X with probability distribution $p_X(\boldsymbol{x})$, generate samples (or populations) $\{\boldsymbol{x}_i\}_{i=1}^M$ from it (here bold font denotes a m-dimensional vector);
- *Estimation*: Estimate expectations of functions under this probability distribution, i.e.

$$\mathbb{E}[f] \equiv \int d\boldsymbol{x}\, p_X(\boldsymbol{x}) f(\boldsymbol{x}) \tag{1}$$

Those two problems are very common in high-dimensional statistics where we know the form of $p_X(\boldsymbol{x})$ (typically the posterior distribution of the parameters of some model given some observed data), but is difficult to sample from it or to compute expectations in order to be able to make predictions. Exact inference is rarely possible, so one has to resort to some kind of approximations.

1.1 Buffon's needle experiment

The first Monte Carlo simulation was imagined much earlier than Ulam, von Neumann and the computer era, by the French botanist Georges-Louis Leclerc,

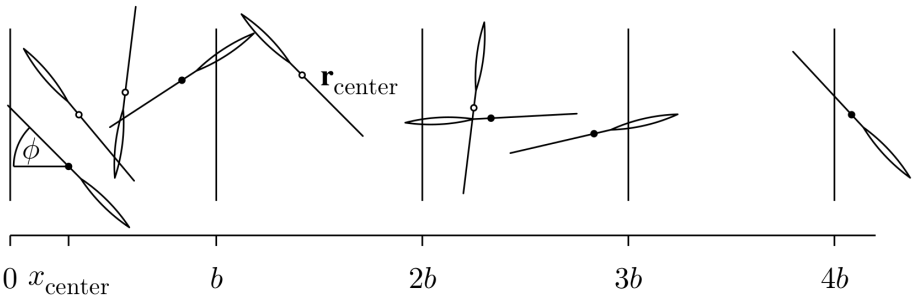


Figure 1: Buffon's parquet with needles. Variables x_{center} and ϕ are also displayed [3].

Comte de Buffon. Imagine we have a (infinite) floor with parallel lines on it, each one distant b from the other and that we drop a needle of length $l \leq b$ on it. With the assumption that the needle is thrown with uniform probability on the floor what is the probability that the needle will hit one of those lines?

1.1.1 Analytical solution

The first thing you have to notice is that, since the length of the needle l is lower or equal than the distance between the lines b, then if we throw a needle it will cross no more than one line. Therefore we can estimate the probability of hitting a line as the mean number of hits $\langle N_{\text{hits}} \rangle$ of an experiment in which the needle is thrown a large number of times on the floor.

In order to write down the mean number of hits, we need to introduce two variables: x_{center}, the x coordinate of the center of the needle and ϕ, that is the angle between the center of the needle and the closest of the lines of the floor, see Fig. 1. The y coordinate of the center of the needle is irrelevant. Since all the lines on the floor are equivalent (there are the symmetries $x_{\text{center}} \to x_{\text{center}} - b$ and $\phi \to -\phi$), and since the needle is thrown with uniform probability on the floor then x_{center} and ϕ can be considered as uniform random variables in

$$0 < x_{\text{center}} < \frac{b}{2} \tag{2a}$$

$$0 < \phi < \frac{\pi}{2} \tag{2b}$$

The tip of the needle has coordinate

$$x_{\text{tip}} = x_{\text{center}} - \frac{l}{2} \cos\phi \,. \tag{3}$$

Therefore if $x_{\text{tip}} < 0$ then the needle hits the line. In order to have $x_{\text{tip}} < 0$ we need that the angle of the needle with respect to the line is such that $\cos\phi > {}^{2x_{\text{center}}}/_{l}$. Now we are ready to write down the number of hits N_{hits} as function

of x_{center} and ϕ. For simplicity we shall rename x_{center} by x from now on. The number of hits is, therefore

$$N_{\text{hits}}(x,\phi) = \begin{cases} 1 & x < \dfrac{l}{2}\cos\phi \\ 0 & \text{otherwise}\,. \end{cases} \tag{4}$$

Introducing the *Heaviside Theta function* $\theta(z)$ whose value is 0 if $z < 0$ and 1 otherwise, we can write N_{hits} in a much compact way as

$$N_{\text{hits}}(x,\phi) = \theta\left(\frac{l}{2}\cos\phi - x\right) \tag{5}$$

Since both x and ϕ have uniform distribution, the mean number of hits per needle is

$$\langle N_{\text{hits}}\rangle = \frac{\int_0^{b/2} dx \int_0^{\pi/2} d\phi\, N_{\text{hits}}(x,\phi)}{\int_0^{b/2} dx \int_0^{\pi/2} d\phi} \tag{6}$$

Computing the integral one gets

$$\langle N_{\text{hits}}\rangle = \frac{2l}{b\pi} \tag{7}$$

For a needle as long as the distance between the lines $l = b$, the probability of crossing is $2/\pi$.

1.1.2 Monte Carlo code

Of course you can verify equation (7) by throwing needles[1] on the floor. This is the method by which Buffon has done it in the 18th century. Now that we have computers, we can get them to do the hard work. Nowadays even the cheapest desktop computer is able throw millions of needles in a few seconds. In listing 1 there is a simple implementation of the Buffon's experiment.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

import numpy as np
from math import pi, cos

def Throw_needle(b):
    x = np.random.uniform(0, b/2.)
    phi = np.random.uniform(0, pi/2)
    return x, phi

def Buffon(l, b, Nsample):
  Nhits = 0
  for i in range(Nsample):
    x, phi = Throw_needle(b)
    x_tip = x - l*cos(phi)/2
```

[1] or even sausages, see for example https://www.youtube.com/watch?v=Q3jVk6k6CGY

```
    if x_tip < 0:
      Nhits += 1

  print(f"Numerical Result: {Nhits/Nsample}")
  print(f"Analytical Result: {2*l/(pi*b)}")


Buffon(1.,1., 1000000)
```

Listing 1: Monte Carlo code to compute the probability of hitting a line on the floor by throwing a needle.

The idea is to simply to extract uniform random variables for the center of the needle x and its angle ϕ and to simply count one if $x_{\text{tip}} < 0$ and 0 otherwise. Finally, note that, if you take the final analytical result for the probability for granted, equation (7) can be used to estimate the value of π.

2 Computing volumes

A nice simple example to understand the power of Monte Carlo (MC) methods is to use random number generators to compute volumes [4]. This may seem like a fancy way of doing a simple task, but we will see that even in trivial cases like this we will achieve interesting results in a very straightforward way. At the end of this section we will extend the ideas behind approximating volumes by Monte Carlo to evaluation of high dimensional integrals.

As a warm-up example we will estimate the value of π.

2.1 The basic idea of Monte Carlo method

Consider the following picture:

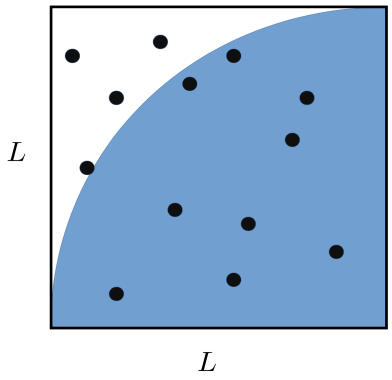


Figure 2

We know that the area of the black-bordered square ("total area") is equal to L^2 and that the "blue area" is equal to $\frac{\pi}{4}L^2$, thus we can obtain the value of

π as

$$\pi = 4 \times \frac{\text{blue area}}{\text{total area}} \tag{8}$$

Now we just need to computes the two areas, or more precisely, their ratio. The ratio can be estimated by generating random points uniformly distributed in the square and counting the ones that fell in the blue are:

$$\pi \approx 4 \times \frac{\text{number of points in blue area}}{\text{total number of points}} \tag{9}$$

The same idea can be applied to every shape that we put inside a box of know dimensions. Take for example the following shape of a lake:

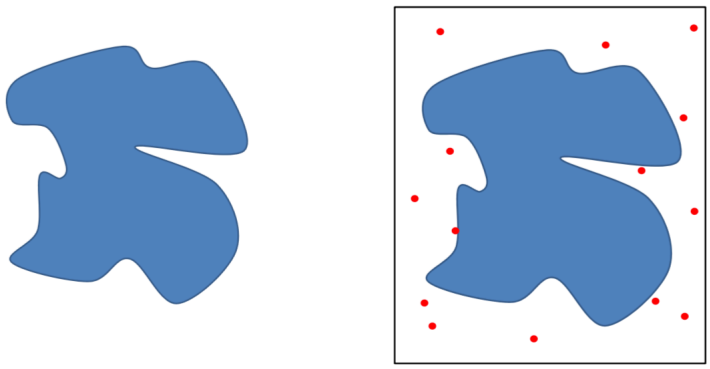


Figure 3

How to estimate the area of this lake? Just put the lake in a framework of known area, then generate random points within it, then count how many of them are in the lake.

From these simple examples it is apparent that the resulting estimate is reliable only when:

- samples are uniformly distributed;
- a sufficiently large number of samples is generated.

Therefore, two main questions arise:

- how to generate uniformly distributed random numbers?
- what is the relationship between the number of samples and the accuracy of the estimate?

2.1.1 Pseudo-random Numbers

Ideally, we would need unlimited sequences of random numbers generated according to a probability distribution with continuous domain. In practice, computers are:

- Discrete → Finite precision
- Finite → Finite domain and limited sequence length
- Deterministic → Sequences can not be truly random

In particular, since computer generated sequences are deterministic, we call the the seemingly random numbers produced by a computer *pseudo-random numbers*. Many software packages include algorithms for generating sequences of pseudo-random numbers. These sequences are almost indistinguishable from sequences of truly random numbers, according to some statistical standards. Generally, an algorithm for pseudo-random number generation can be described as follows: let z_n be the n-th number of the sequence we want to generate, then z_n will be produced according to the (deterministic) rule

$$z_n = f(A, z_{n-1}, ..., z_{n-p}) \tag{10}$$

where f is a function characteristic of the specific generator, A is a set of parameters and $z_{n-1}, ..., z_{n-p}$ are the previous p numbers in the sequence. We can state a number of general properties of pseudo-random number generators:

- they generate a sequence of numbers from a finite set of integers;
- each number is generated as a function of the last p numbers in the sequence;
- the function f has the *topological transitivity property*: every subsequence is produced (early or later) by repeatedly computing f, starting from any subsequence of length p;
- the set of parameters A is chosen in order to maximize the length of the sequence before it starts repeating.

One would want any subsequence of length n to correspond to a random number uniformly and independently distributed in the unit hypercube. Using fast generators, this characterization is less and less satisfied by pseudo-random sequences as n increases. While the error in the estimate obtained with Monte Carlo methods decreases with the number of samples, the error induced by pseudo-random numbers generation does not. This puts a bound to the precision that can be achieved by Monte Carlo methods.

Common pseudo-random number generators used in practice are variation of linear congruential generators. Another class of generators which is very well know for their good sampling properties is that of Mersenne Twister engines.

2.1.2 Sample size and accuracy

In this section we will understand the main advantage of the Monte Carlo method, namely that the error on volumes estimates decreases as $n^{1/2}$, whereas using other (approximated) methods based on the evaluation of n points in m dimensions the error decreases as $n^{1/m}$, therefore much slower when $m > 2$.

We consider two related problems:

- (approximately) computing integrals (areas, volumes,...);
- (approximately) counting.

The former problem arises with continuous functions, the latter one with discrete sets. The Monte Carlo method provides

- a point estimate, i.e. an approximate value for the volume;
- a confidence interval, representing how reliable the estimate is (see also Chap. 2 of [4]).

The number of sample points we use to compute the approximation is called *sample size*. Given a desired accuracy requirement , we would like to know the corresponding minimum sample size. We will see that the probabilistic analysis of the Monte Carlo provides such estimate. Additionally, efficiency-improving techniques have been devised to reduce the sample size and hence the computing time required by the method.

Given a continuous function in an m-dimensional space, we want to (approximately) compute an area or a volume enclosed by the function. Let R denote a region of unknown volume $\lambda(R)$ in the m-dimensional unit hypercube $[0,1]^m$. Now assume that we have two functions:

- the function GeneratePoint(m) generates a point in the m-dimensional unit hypercube. For the time being we don't specify how the functions generates the point and we also allow it to generate a new point based on the previously generated points.
- a boolean function IsInside($\boldsymbol{x}, R$) tests whether any given point $\boldsymbol{x}$ is contained in the region R or not (here the bold font denotes a vector).

Then, we can approximate $\lambda(R)$ using the following MC algorithm:

Algorithm 1 Estimating $\lambda(R)$.

Require: $R \subseteq [0,1]^m, n$
Ensure: $\bar{\lambda}(R) \approx \lambda(R)$
  $S \leftarrow 0$
  for $j \leftarrow 1, ..., n$ **do**
    $\mathbf{x} \leftarrow$ GeneratePoint(m) ▷ m-dimensional vector
    if IsInside($\mathbf{x}, R$) **then**
      $S \leftarrow S + 1$
    end if
  end for
  return S/n

To appreciate the advantage given by the Monte Carlo method, let's now analyze two possible functions GeneratePoint:

1. Each call to GeneratePoint generates a different point on a grid of n points in the unit hypercube

2. The function GeneratePoint generates points drawn from a semi-random sequence, namely a call to GeneratePoint(m) generates a point approximately uniformly distributed in $[0,1]^m$ and independent from all other generated points.

2.2 Computing Volumes By Using a Grid

Let's first think in $m = 2$ dimensions: we set $n = 100$ and divide the unit hypercube into 10×10 smaller identical hypercubes, whose centers form a lattice. We say that the *spacing* of the lattice is $k = 1/10$.

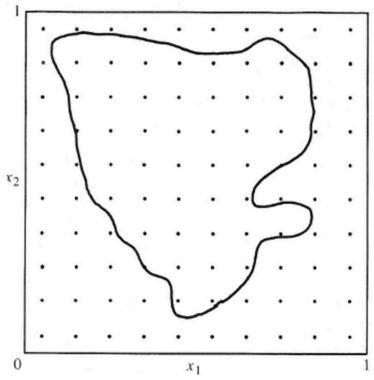


Figure 4

In general, using a spacing k in m dimensions requires $n = 1/k^m$ points in the lattice; therefore the spacing can be written as

$$k = \frac{1}{n^{\frac{1}{m}}} \tag{11}$$

The estimate of the volume is given by (remember that the total volume of the hypercube is 1)

$$\lambda_{\text{grid}} = \frac{\text{n. points inside } R}{n} = k^m \times (\text{n. points inside } R) \tag{12}$$

In the following left picture the estimation of the area is the shaded region. The error of the estimate is the dark area in the picture on the right.

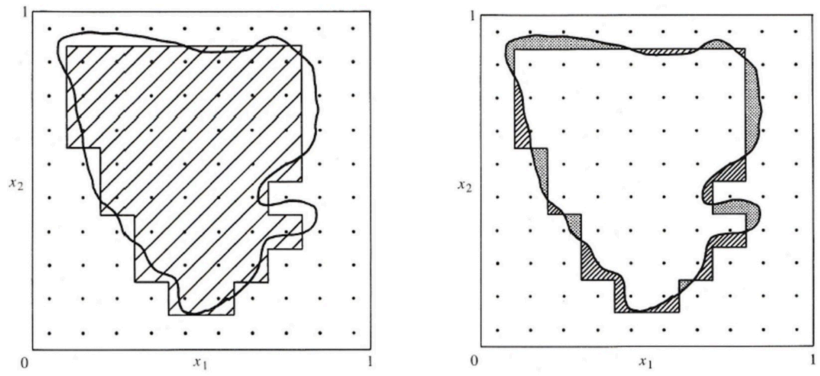


Figure 5

In the worst case, i.e. when errors do not cancel out, the error $|\lambda_{\text{grid}} - \lambda(R)|$ is bounded above by the total dark area (still from the picture on the right):

$$|\lambda_{\text{grid}} - \lambda(R)| \leq \text{dark area}$$

Let $s(R)$ be the length of the frontier of R, that is the set of points that delimit R. Loosely speaking, if R is a volume in m dimensions, it's frontier will typically be a $m-1$ dimensional object. Since here we are in dimension $m = 2$, the frontier is a 1D object, a line. We can use the "hyper-surface" of the frontier (the length of the line) to bound by above the dark area: the dark area cannot be neither longer than $s(R)$, nor larger than k. In the worst case, the error on the estimate is:

$$\text{dark area} \leq s(R) \times k \tag{13}$$

Therefore we obtain a relation for the error on the estimate of the volume:

$$|\lambda_{\text{grid}} - \lambda(R)| \leq s(R) \times k = \frac{s(R)}{n^{\frac{1}{m}}} \tag{14}$$

Now we fix a precision requirement ϵ for the estimate, and look for what values of n the requirement is met. Changing n in this case means refining the grid. We have

$$|\lambda_{\text{grid}} - \lambda(R)| \leq \frac{s(R)}{n^{\frac{1}{m}}} \leq \epsilon \tag{15}$$

The precision requirement is met for those value of n such that

$$n \geq n_{\text{grid}} \equiv \left(\frac{s(R)}{\epsilon}\right)^m$$

Here we see a big problem in this estimator of the volume based on points generated on a grid: the number of points it need for a fixed precision ϵ grows exponentially with the dimension m (provided the surface stays roughly the same). The problem becomes computationally expensive very quickly in high dimension.

2.3 Computing Volumes By Using Monte Carlo Sampling

The aforementioned problem can be overcome using Monte Carlo sampling. That is, instead of using samples from a lattice, we sample n times from a uniform distribution in $[0,1]^m$ (mind that we must sample each one of the m components of the vector $\boldsymbol{x}^j = (x_1^j, ..., x_m^j)$, where $j = 1, ..., n$ is the sample index).
Recall that for Algorithm 1 to work, it is necessary to have an indicator function of R that we called IsInside($\mathbf{x}^j, R$), which checks whether the sampled vector $\mathbf{x}^j$ lies within the region R or not. Let's consider a single random sample $\boldsymbol{x}$. Here for convenience we change the name of this function to $\phi_R(\boldsymbol{x})$:

$$\phi_R(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in R \\ 0 & \text{otherwise} \end{cases} \tag{16}$$

Since $\boldsymbol{x}$ is a uniformly distributed random variable in the box $[0,1]^m$, also $\phi_R(\boldsymbol{x})$ is a random variable. It is easy to check that its expected value is the volume $\lambda(R)$ itself. In fact, we have

$$\mathbb{E}[\phi_R(\boldsymbol{x})] \equiv \int_{[0,1]^m} d\boldsymbol{x}\ \phi_R(\boldsymbol{x}) = \int_R d\boldsymbol{x}\ 1 = \lambda(R). \tag{17}$$

The variance of $\phi_R(\boldsymbol{x})$ is also easy to compute:

$$\begin{aligned} \mathrm{Var}[\phi_R(\boldsymbol{x})] &\equiv \mathbb{E}\left[(\phi_R(\boldsymbol{x}) - \lambda(R))^2\right] \\ &= \mathbb{E}\left[\phi_R^2(\boldsymbol{x})\right] - \lambda^2(R) \\ &= \lambda(R)(1 - \lambda(R)) \end{aligned} \tag{18}$$

Notice that these are the mean and the variance of Bernoulli random variable with parameter $\lambda(R)$.

Now let's go back to our algorithm where we sample n independent points $\boldsymbol{x}^j, j = 1, \ldots, n$. In order to characterize the random output from the algorithm, we have to study the random variable

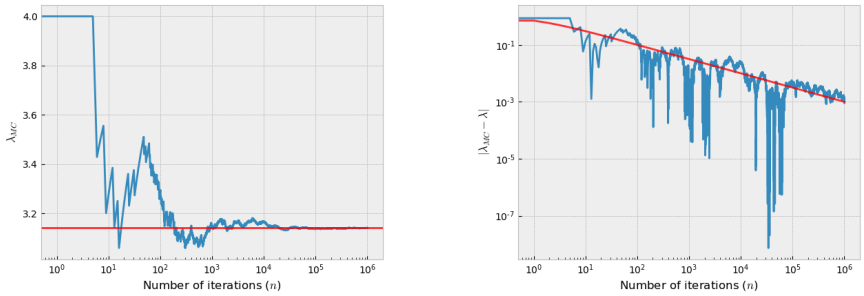
$$S = \sum_{j=1}^{n} \phi_R(\boldsymbol{x}^j)$$

Using the linearity of expectations we have

$$\mathbb{E}[S] = \sum_{j=1}^{n} \mathbb{E}[\phi_R(\boldsymbol{x}^j)] = n\lambda(R).$$

Then, using the fact that the variance of the sum of independent variables is equal to the sum of the variances, we have

$$\mathrm{Var}(S) = n\lambda(R)(1 - \lambda(R))$$



(a) Convergence of λ_{MC} to the expected value $\lambda(R)$ (depicted in red).

(b) The estimation error decreases proportionally to $n^{-1/2}$ (depicted in red).

Figure 6: Behaviour of $\lambda_{\text{MC}}(R)$ (left panel) and of the estimation error $|\lambda_{\text{MC}}(R) - \lambda(R)|$ (right panel, notice the log-log scale) as function of n in the simple case of the computation of π.

Actually, we can go further on and compute the full probability distribution of S, using the fact that S is the sum of independent and identically distributed (i.i.d.) Bernoulli random variables. This leads to the binomial distribution:

$$P(S = s) = \binom{n}{s} \lambda(R)^s (1 - \lambda(R))^{n-s}$$

Since the return value from the Monte Carlo algorithm is $\lambda_{\text{MC}} = S/n$, we obtain the following result:

- λ_{MC} is an *unbiased estimator* of the true volume, meaning that its expected value is the true volume

$$\mathbb{E}\left[\lambda_{\text{MC}}\right] = \frac{\mathbb{E}\left[S\right]}{n} = \lambda(R)$$

- has variance that shrinks with the number of samples n:

$$\text{Var}(\lambda_{\text{MC}}) = \frac{\lambda(R)(1 - \lambda(R))}{n}$$

This two results give us the theoretical guarantee that increasing the sample size λ_{MC} gives us more and more accurate estimates of the true volume value $\lambda(R)$. As an example take into account the simple case of estimating the value of π by throwing points inside a square, as described in subsection 2.1. In Figure 6 you can see the behavior of $\lambda_{\text{MC}}(R)$ and the corresponding estimation error $|\lambda_{\text{MC}}(R) - \lambda(R)|$ as function of the number of iterations n.

Now, in order to obtain an inequality that tells us the minimum n needed to for λ_{MC} to satisfy a certain accuracy requirement, we will use the Chebyshev's inequality, which we state and prove in the following paragraph.

Chebyshev's Inequality Let X be a scalar random variable with distribution function $p_X(x)$, expected value $\mathbb{E}[X] = 0$ and finite variance $\mathrm{Var}(X) = \mathbb{E}[X^2] = \sigma^2$. Then $\forall \epsilon > 0$

$$P\left(|X| \geq \epsilon\right) \leq \frac{\sigma^2}{\epsilon^2} \tag{19}$$

Proof. The proof can be obtained simply using the definition of $P\left(|X| \geq \epsilon\right)$

$$\begin{aligned} P(|X| \geq \epsilon) &= \int_{-\infty}^{-\epsilon} dx\, p_X(x) + \int_{+\epsilon}^{+\infty} dx\, p_X(x) \\ &\leq \int_{-\infty}^{-\epsilon} dx \frac{x^2}{\epsilon^2} p_X(x) + \int_{+\epsilon}^{+\infty} dx \frac{x^2}{\epsilon^2} p_X(x) \\ &\leq \int_{-\infty}^{+\infty} dx \frac{x^2}{\epsilon^2} p_X(x) = \frac{\sigma^2}{\epsilon^2} \end{aligned}$$

□

Now choose the X variable in the Chebyshev's inequality to be $X = \lambda_{MC} - \lambda(R)$. Then from the Chebyshev's inequality we have that:

$$P(|\lambda_{MC} - \lambda(R)| \leq \epsilon) \geq 1 - \frac{1}{\epsilon^2} \frac{\lambda(R)(1 - \lambda(R))}{n} \equiv 1 - \delta$$

where in the last equation we defined the probability confidence level

$$\delta \equiv \frac{1}{\epsilon^2} \frac{\lambda(R)(1 - \lambda(R))}{n}$$

So, to choose a suitable value for n, we have to specify both the accuracy ϵ and the confidence level δ. We find that our sample size n has to satisfy:

$$n \geq \frac{\lambda(R)(1 - \lambda(R))}{\delta \epsilon^2}$$

Since $\lambda(R)(1 - \lambda(R)) \leq \frac{1}{4}$, finally we find that requirement for n given by

$$n \geq n_{\mathrm{MC}} = \frac{1}{4\delta\epsilon^2}$$

Note that this bound does not depend on m. Also recall that the one for the grid algorithm was

$$n_{\mathrm{grid}} = \left(\frac{s(R)}{\epsilon}\right)^m$$

The Monte Carlo sampling algorithm takes polynomial time in m, since all its iterations are polynomial in m and the number n of iterations does not depend on m. On the contrary, the grid algorithm, which samples R with a lattice, takes exponential time. The advantage of the Monte Carlo method is more and more relevant when m is large. A typical occurrence is when m represents the number of variables of a combinatorial problem.

2.4 Counting objects

Another typical application of the Monte Carlo method is for counting objects with specific properties in a (very large) discrete set. Consider the following example.

Consider a vertex of the m-dimensional hypercube $\boldsymbol{x} = (x_1, x_2, ..., x_m)$, where $x_i \in \{0, 1\}\ \forall i = 1, ..., m$. Let's call $\mathcal{X}$ the set of all points $\boldsymbol{x}$. Consider then a set of k inequalities (constraints) of the form

$$a_{j_1} x_1 + a_{j_2} x_2 + ... + a_{j_m} x_m \leq b_j \qquad \forall j = 1, .., k$$

We can be interested in counting how many points in $\mathcal{X}$ satisfy all the inequalities (such points form the feasible region of an *integer programming* problem). Alternatively, we may be interested in counting how many points in $\mathcal{X}$ satisfy at least one of the inequalities (*disjunctive programming*) or a prescribed number of inequalities. If a point satisfies such constraints we say that it has the property P.

The number of points in $\mathcal{X}$ is $|\mathcal{X}| = 2^m$ and an explicit enumeration of all of them is out of question even for relatively small values of m. This is the reason for estimating the number of points with the desired property, instead of explicitly counting them. For this purpose we can resort to the Monte Carlo method. We need two procedures:

- a procedure to generate elements from the ground set $\mathcal{X}$ with uniform probability $1/|\mathcal{X}|$;
- a procedure to test whether an element $x \in \mathcal{X}$ has the property P or not.

Therefore an immediate modification of algorithm 1 can be formulated as follows:

Algorithm 2 Monte Carlo counting

Require: $\mathcal{X}, P, n$
  $S \leftarrow 0$
  for $j \leftarrow 1, ..., n$ **do**
    $\mathbf{x} \leftarrow$ GeneratePoint(X) ▷ uniformly pick from the ground set X
    if HasProperty($\mathbf{x}, P$) **then**
      $S \leftarrow S + 1$
    end if
  end for
  return S/n

2.5 Multidimensional integration

The general problem we wish to address in this section is that of multidimensional integration, i.e. finding the expectation of some function $f(\boldsymbol{x})$ under some

probability distribution $p_X(\boldsymbol{x})$

$$\mathbb{E}[f] \equiv \int d\boldsymbol{x}\, p_X(\boldsymbol{x}) f(\boldsymbol{x})\,. \tag{20}$$

Notice that, in the particular case of f being an indicator function and p_X being a uniform distribution we recover the volume computing method discussed previously. Extending the concepts introduced in the previous subsections, if we are able to sample from the distribution $p_X(\boldsymbol{x})$, we can approximate the expectation with

$$\hat{f} = \frac{1}{n}\sum_{j=1}^{n} f(\mathbf{x}^{(j)}) \tag{21}$$

where $\left\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}\right\}$ are n samples drawn independently from $p_X(\boldsymbol{x})$. As we have seen before, $\hat{f}$ is an unbiased estimator of $\mathbb{E}[f]$, in the sense that $\mathbb{E}[\hat{f}] = \mathbb{E}[f]$ and its variance is

$$\begin{aligned}
\mathrm{Var}(\hat{f}) &= \mathbb{E}\left[(\hat{f} - \mathbb{E}[\hat{f}])^2\right] = \mathbb{E}[\hat{f}^2] - \mathbb{E}[\hat{f}]^2 \\
&= \mathbb{E}\left[\frac{1}{n^2}\sum_{i,j=1}^{n} f(\mathbf{x}^{(i)}) f(\mathbf{x}^{(j)})\right] - \mathbb{E}[f]^2 \\
&= \frac{\mathbb{E}[f^2] - \mathbb{E}[f]^2}{n} = \frac{\mathrm{Var}(f)}{n}
\end{aligned} \tag{22}$$

i.e. is the variance of the function $f(\boldsymbol{x})$ under the probability distribution $p_X(\boldsymbol{x})$, divided by the number of samples n. In going from the second to the third line of the equation above we have used explicitly the independence of the samples $\left\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}\right\}$. It is important to stress that the accuracy of the estimator does not depend on the dimensionality of the integral.

The only problem left in the above discussion is how to sample from a generic distribution $p_X(\boldsymbol{x})$. Of course this is not always possible. In the following subsections we discuss several methods by which one can sample from $p_X(x)$ (both in a direct and indirect way) and, when this is not worth, we will discuss another way to compute the expectation (20). Unfortunately all those methods are basically limited to one-dimensional distributions, but they give some insights of more general strategies. For this reason from now on we will limit the discussion to the $m = 1$ dimensional case, exposing in each case what are the drawbacks in higher dimensions.

2.5.1 Inverse transform sampling

In this subsection we will consider a simple strategy, called *inverse transform method*, to sample from a given distribution $p_X(x)$.

Suppose then that we have a random variable U distributed uniformly in the interval $[0, 1]$ (that we are able to generate easily on a computer). We wish to find an (invertible) transformation f such that $X = g(U)$ is a random variable

distributed as $p_X(x)$. Indicating the cumulative density function of the random variable X as $F_X(x)$, we have

$$F_X(x) = P(X \leq x) = P(g(U) \leq x) = P(U \leq g^{-1}(x)) = F_U(g^{-1}(x)) \tag{23}$$

where with $F_U(\cdot)$ is the cumulative density function of the random variable U, that since is uniform in $[0, 1]$ is simply $F_U(u) = \int_0^u dz = u$. Therefore thanks to equation (23) we finally obtain that the transformation g must be

$$g(x) = F_X^{-1}(x) \tag{24}$$

i.e. the inverse of the cumulative of the random variable X. Therefore, all we need to generate independent random variables X with cumulative density function $F_X(x)$ is to

1. generate a uniform random variable U in $[0, 1]$;
2. find the inverse of the cumulative $F_X(x)$;
3. then $F_X^{-1}(U)$ will be distributed as X

The inverse transforming method can be readily generalized to higher dimension. However the success of the method depend on two factors

1. given $p_X(x)$ it is not always easy to compute the associated cumulative density function $F_X(x)$;
2. even if we are able to compute $F_X(x)$, not necessarily we are able to compute analytically its inverse.

Before turning to other methods, we shall analyze some simple examples.

Exponential Distribution. As an example of the inverse transform method, consider the case of the exponential distribution with parameter λ

$$p_X(x) = \lambda e^{-\lambda x}\,, \qquad x \geq 0$$

The cumulative probability density function is

$$u \equiv F_X(x) = \int_0^x dy\, p_X(y) = 1 - e^{-\lambda x}\,.$$

The inverse of the cumulative is

$$x = F_X^{-1}(u) = -\frac{1}{\lambda}\ln(1-u) = -\frac{1}{\lambda}\ln(u)$$

where in the last step we have used the fact than u is distributed uniformly in $[0, 1]$.

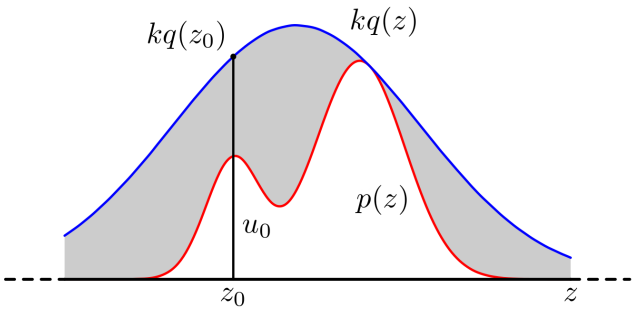


Figure 7: Sketch of $p(x)$ and $kq(x)$ [5]. The shaded area corresponds to the rejection zone.

Box-Muller transformation. It is important to mention, even if we do not prove it, how to generate independent samples from a Gaussian distribution. Suppose U_1 and U_2 are independent uniform random variables in the interval $[0, 1]$. Then let us define the following quantities

$$X_1 = \sqrt{-2 \ln U_1} \cos(2\pi U_2) \tag{25a}$$

$$X_2 = \sqrt{-2 \ln U_1} \sin(2\pi U_2)\,. \tag{25b}$$

It can be proved that X_1 and X_2 are independent Gaussian random variables with zero mean and unit variance. The transformation above is called Box-Muller transformation. To obtain a Gaussian random variable Y with generic mean μ and variance σ^2 from random variable X_1 it suffices to

$$Y = \mu + \sigma X_1\,. \tag{26}$$

2.5.2 Rejection sampling

From now on we suppose that *target density* $p(x)$ is a too complicated function to have any hope of being able to sample from it in a direct way with the inverse transform method. Suppose, instead, that we have a much simpler density $q(z)$ from which we know how to sample. We call $q(z)$ the *proposal density*. We assume also that we know the value of a constant k such that

$$kq(z) \geq p(z)\,, \qquad \forall z \tag{27}$$

A picture of those two functions is given in Figure 7. The rejection sampling consists in extracting two random numbers. The first one, which we call z_0, is generated from the distribution $q(z)$. The second one, u_0, is instead extracted uniformly in $[0, kq(z_0)]$. With this couple of random numbers we generate uniform points in the two-dimensional space under the curve of the function $kq(z)$. Therefore if $u_0 > p(z_0)$ we reject z_0; otherwise it is accepted. The random variable z_0 will be distributed as $p(z)$ since the points (z_0, u_0) that are uniformly distributed in the white area in Fig. 7. In addition, since a z_0 sample

is extracted from the distribution q and accepted with probability $\frac{p(z_0)}{kq(z_0)}$, the probability that a step of the rejection sampling will be accepted is

$$P(\text{acceptance}) = \int dz_0\, q(z_0) \frac{p(z_0)}{kq(z_0)} = \frac{1}{k}$$

that is the ratio of the area of the $p(z)$ curve (which is 1 since is a probability distribution) and the area of $kq(z)$. Therefore k must be chosen as small as possible in order to not have a too small fraction of points that are accepted. For this reason rejection sampling will work best if $q(x)$ is similar to $p(x)$: if not, the constant k can be very large and the rejection rate very high.

Indeed this is what happens in high dimensions. Let's see a very simple example: assume that both the target and the proposal density are m-dimensional Gaussian distributed with zero mean vector and a diagonal covariance matrix with equal entries on the diagonal respectively σ_p^2 and σ_q^2. Let us further assume that $\sigma_q \gtrsim \sigma_p$ so that the two Gaussian distributions are not so dissimilar (if $\sigma_q < \sigma_p$ there exist no k such that $kq(x) \geq p(x)$ in this case). The minimal value of k is easily found by comparing the value of the two distributions in zero, so that we get

$$k = \frac{(2\pi\sigma_q^2)^{m/2}}{(2\pi\sigma_p^2)^{m/2}} = e^{m \ln\left(\frac{\sigma_q}{\sigma_p}\right)}\,.$$

Therefore, even if the σ_q and σ_p are similar, the minimal value of k scales exponentially with the dimension m. For example if $m = 1000$ and ${}^{\sigma_q}/_{\sigma_q} = 1.01$, then $k \simeq 21000$!

2.5.3 Importance sampling

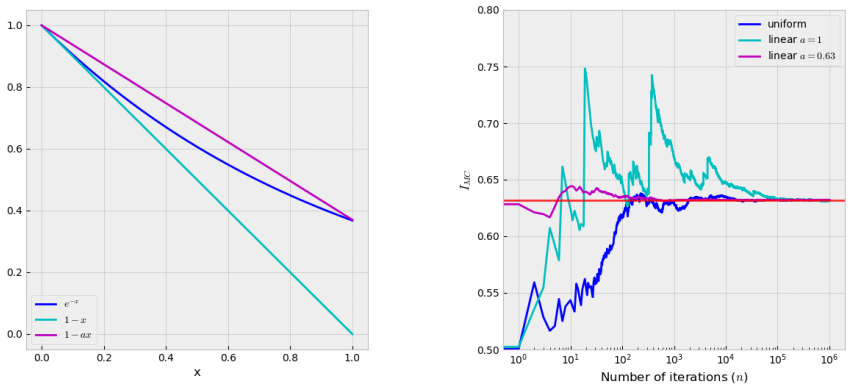
Notice that even if we are able to sample from $p(\boldsymbol{x})$ in a very efficient way, if $p(\boldsymbol{x})$ is small in regions where $f(\boldsymbol{x})$ is large (and vice versa), the integral (20) can be dominated by regions where the integrand $p_X(\boldsymbol{x})f(\boldsymbol{x})$ is small. Importance sampling is simply a technique that tries to remedy this problem by concentrating the sampling where the integral gives a relevant contribution. In order to do that, as in rejection sampling, we will use independent samples $\left\{\mathbf{x}^{(1)}, ..., \mathbf{x}^{(n)}\right\}$ from a proposal density $q(x)$ in this way

$$\mathbb{E}[f] = \int d\boldsymbol{x}\, q(\boldsymbol{x}) \frac{p(\boldsymbol{x})f(\boldsymbol{x})}{q(\boldsymbol{x})} \simeq \frac{1}{n}\sum_{j=1}^{n} f(\mathbf{x}^{(j)})\, \omega(\mathbf{x}^{(j)}) \equiv \hat{f}_\omega\,. \tag{28}$$

In the preceding equation we have defined the *importance weights* $\omega(\boldsymbol{x})$ as

$$\omega(\boldsymbol{x}) = \frac{p(\boldsymbol{x})}{q(\boldsymbol{x})}\,.$$

and they take into account the fact that the samples are extracted from a different distribution (choosing $q(\boldsymbol{x}) = p(\boldsymbol{x})$ we recover the old expression for



(a) (b)

Figure 8: Plot of the integrand and of importance sampling functions used in the main text (left panel) and convergence of the various estimators to the exact result (right panel). In Figure 8b, blue, teal and violet lines are obtained using uniform, linear with $a = 1$ and $a = 0.63$ respectively. The red line is the exact result $1 - 1/e$.

the estimator). As done before you can check that $\hat{f}_\omega$ is an unbiased estimator. Generalizing (22), we have for the variance

$$\mathrm{Var}(\hat{f}_\omega) = \frac{\mathrm{Var}(f\omega)}{n}\,. \tag{29}$$

This equation shows that the variance of the new estimator still goes as $1/n$, but the magnitude of this variance can be reduced greatly with a wise choice of $q(\boldsymbol{x})$. Ideally, we should have $q(\boldsymbol{x}) \simeq p(\boldsymbol{x})f(\boldsymbol{x})$, in which case the variance would vanish altogether. In contrast, if we take $\omega(\boldsymbol{x})$ constant, as in the brute force Monte Carlo sampling, the variance can become very large.

Unfortunately, the simple importance sampling scheme cannot be used to evaluate efficiently multidimensional integrals. The reason is simply that we don't know how to construct a transformation that would enable us to generate points in the domain of integration with a probability density proportional to the complicated probability density $p(\boldsymbol{x})$.

Example Take the simple $m = 1$ dimensional case where $p(x)$ is uniform in $[0, 1]$ and $f(x) = e^{-x}$. The integral to evaluate is

$$I = \int_0^1 dx\, e^{-x}\,. \tag{30}$$

Analitically $I = 1 - 1/e = 0.63212055882\ldots$. The first Monte Carlo scheme we can apply is simply generating uniformly distributed random numbers in $[0, 1]$

to evaluate I. After 10^6 sampling steps we get $I_{\text{MC}} = 0.63204 \pm 0.00018$. Now let's try to find an importance sampling function $q(x)$ that is able to reduce the variance. We can try to use a first order Taylor expansion of the integrand as $e^{-x} \simeq 1 - x$. Normalizing this function in the interval $[0, 1]$ to get a probability distribution we have $q(x) = 2(1 - x)$. Using this function, after 10^6 iterations we get $I_{\text{MC}} = 0.63149 \pm 0.00057$ i.e. a worst outcome with respect to uniform sampling. This is due to the fact that the Taylor expansion approximates well the integrand only near the origin, see Figure 8a. As you can see, the region $x \simeq 1$ even if gives a smaller contribution to the integral with respect to the region $x \simeq 0$, yet it is not exactly zero; our function $2(1 - x)$ weights with infinitesimal probability this region, and the corresponding importance weights are large. As a matter of facts, when a rare event appears, i.e. we sample a value of x close to 1, the importance weight give a huge contribution to the integrand. As you can see in Figure 8b, teal line, this is the meaning of the large spikes that anticipate the convergence to the exact result. How to improve this result?

Consider next, a general linear function $q(x) \propto 1 - ax$, where the parameter a is chosen to give a better representation of e^{-x} over the whole interval of integration. Normalizing we get

$$h(x) = \frac{2(1 - ax)}{2 - a}$$

In the case $a = 1$ we recover the previously analyzed distribution. We can sample from this general linear distribution using the inverse transform method; the cumulative is

$$u \equiv F(x) = \frac{1 - (1 - ax)^2}{a(2 - a)}$$

Inverting the cumulative we get

$$x = \frac{1 - \sqrt{1 - a(2 - a)u}}{a}$$

where u is a uniform random variable in $[0, 1]$. Using $a \simeq 0.63$ (the function is plotted in 8a, violet line), and 10^6 iterations we get $I_{\text{MC}} = 0.63212 \pm 0.00002$ which has a variance one order of magnitude less than uniform sampling. The moral is that one should choose the proposal density $q(x)$ in a careful way. If not, one could get a worst result than uniform sampling.

3 Markov Chains

In this Section we deal with stochastic dynamical systems: variables that evolve in time and whose evolution is not (only) driven by some deterministic law since there is also some randomness at play.

3.1 Stochastic Processes

Consider a possibly infinite sequence of random variables $(X^0, X^1, X^2, ..., X^t, ...)$ indexed by an index $t = 0, 1, \ldots$ that we commonly refer to as *time,* so that we

interpret the sequence as the (stochastic) change of X over time. The random variables take value in some state space $\mathcal{X}$, i.e. $X^t \in \mathcal{X}$. For the time being we don't make any assumptions on $\mathcal{X}$: it can either be some finite set of elements, or an infinite but discrete space as the set of natural number $\mathbb{N}$, or an infinite and continuous space as $\mathbb{R}$, or some d-dimensional space like the Euclidean space $\mathbb{R}^d$ (in the latter case, X^t will be a random vector and we may denote it with $\boldsymbol{X}^t$).

Such sequences are known as stochastic processes, and they are ubiquitous in nature for two main reasons: 1) because laws of physics at the microscopic level (involving elementary particles) are non-deterministic; 2) because at the microscopic level many phenomena involve a large number of actors interacting in such complicated ways that for any practical purpose they can be modeled as stochastic process.

An example of the latter case is given by finance. The price of a final asset, let's say Google's stocks, evolves in time as a result of the sells of Google's products but also as the result of an large number of interaction among traders, rumors, irrational expectations... in such a complex way that no one is able to predict the future evolution of the prices. Looking at the daily or hourly fluctuations of the prices one can't help noticing that, even though the system is purely deterministic it behaves as affected by stochasticity.

Unfortunately in nature, since we cannot go back in time, we often observe a unique realization of the stochastic process, that is just a unique value for X^0, another for X^1, and so on. In other words we observe only a single *trajectory* out of the many possible ones, each of which can be more or less likely. In *computer simulations*, instead, one can sample again and again the same processes, obtaining different trajectories and gaining valuable information on the structure of the process itself.

Stochastic processes are entirely defined by the probability of observing each trajectory. For a given trajectory $(x^0, x^1, x^2, \dots)$ the corresponding probability is written as

$$P(X^0 = x^0, X^1 = x^1, X^2 = x^2, \dots) \equiv P(x^0, x^1, x^2, \dots). \tag{31}$$

When there is no ambiguity, we will typically use the more concise notation on the r.h.s. instead of the more formally correct notation on the l.h.s.. In the case of continuous random variables, the expression above has to be intended as a probability density.

From the probability of a whole trajectory, that is the joint probability over the variables $X^0, X^1, \dots$ one can derive the probabilities over a subset of the variables by marginalization (i.e. by taking sums over all the possible values of those variables). For example, the probability that the process at time t takes some value x^t, i.e. $P(X^t = x^t)$ is given (assuming discrete states) by summing over the all possible values at all times except for t:

$$P(x^t) = \sum_{x^0} \cdots \sum_{x^{t-1}} \sum_{x^{t+1}} \sum_{x^{t+2}} \dots P(x^0, x^1, \dots) \tag{32}$$

$$= \sum_{x^0, \dots, x^{t-1}, x^{t+1} \dots} P(x^0, x^1, \dots) \tag{33}$$

This amounts to sum over all possible trajectories that at time t take values x^t, each trajectory weighted by the corresponding probability.

It is often the case that it is hard to explicitly give the probability distribution $P(x^0, x^1, x^2, \dots)$ for the process, while it may be very easy to define the process by stating how to generate the random variable X^t given the realization of $X^{t-1}, X^{t-2}, \dots$ in the previous steps. This stochastic rule for the evolution implicitly defines a distribution $P(x^0, x^1, x^2, \dots)$, which we may or may not be able to compute.

Let's now give some examples of stochastic process, starting from a trivial one with no time correlation and arriving to a more complicated one with long term memory.

Coin Tosses

In this simple process, we toss a fair coin at each step. The result of the toss is either head or tail, therefore we can define the state space with $\mathcal{X} = \{H, T\}$ (or equivalently $\mathcal{X} = \{0, 1\}$). Therefore, possible trajectories are of the form $(H, H, T, T, H, H, H, H, H, T, H, T, T, \dots)$. It is easy to see that since the coin is fair and there is no history dependence, all trajectories are equiprobable. In fact, if we consider the process up to some time t, corresponding $t + 1$ tosses, the probability of any trajectory $(x^0, \dots, x^t)$ is easy to compute since each toss is independent. Therefore, the joint probability factorizes in the probabilities for the single draws and we have

$$P(x^0, \dots, x^t) = \prod_{t'=0}^{t} P(x^{t'}) = \frac{1}{2^{t+1}} \tag{34}$$

Random Walk (RW)

A random walk in $\mathbb{Z}^d$ is a walk that starts from the origin of the d-dimensional lattice, and takes a step of length one in a random direction, choosing uniformly at random among $2d$ the possible the available directions. This is an example of a possible trajectory for a walk in $\mathbb{Z}^2$: $\boldsymbol{x}^0 = (0, 0)$, $\boldsymbol{x}^1 = (-1, 0)$, $\boldsymbol{x}^2 = (-1, 1)$, $\boldsymbol{x}^3 = (-1, 0)$, $\boldsymbol{x}^4 = (-2, 0), \dots$.

The process can be written as follows

$$\boldsymbol{X}^0 = \boldsymbol{0}, \tag{35}$$

$$\boldsymbol{X}^{t+1} = \boldsymbol{X}^t + \boldsymbol{D}^t \qquad \text{for } t \geq 0, \tag{36}$$

where $\boldsymbol{D}^t$ is a random vector chosen uniformly at time t from the set $\{(1,0,0,\dots),(-1,0,0,\dots),(0,1,0,\dots),(0,-1,0,\dots),\dots\}$ containing $2d$ elements. Notice that for a random walk, the probability of visiting some state at time $t+1$ given the states visited up to time t, depends only on the state at time t and not on the whole past trajectory:

$$P(x^{t+1}|x^t, x^{t-1}, ..., x^0) = P(x^{t+1}|x^t) \tag{37}$$

Self-Avoiding Walk (SAW)

A SAW is a random walk that cannot visit twice the same site. Therefore, at each step he visits uniformly at random one of the neighboring sites he hasn't already visited (they can be less then $2d$). If at a certain step there are no more directions available the process stops. This is an example of a process where the probability of going on a certain states at time $t+1$ depends on the *whole* past trajectory.

3.2 Markov Chains

Generally, stochastic processes can be very complicated and difficult to analyze. Here we consider a restricted class of stochastic processes, named Markov Chain (MC) processes, that are much easier to analyze while still being rich enough to model many real world situations. Markov processes find many applications not only in finance but also in physics, chemistry, biology, and in data science (for example the PageRank algorithm used by Google Search to rank web pages is based on a Markov chain). Also, as we will see in the following paragraphs, one can carefully design and simulate Markov Chains in order to perform hard computational tasks such as sampling from very complicated distributions or solving hard optimization problems.

Markov chains are defined by the property of being *memory-less* stochastic processes, that is stochastic processes in which the probability of the next state given the history of the process depends *only on the current state*. This is called the Markov Property, and is formally written as

$$P(x^{t+1}|x^t, x^{t-1}, ..., x^0) = P(x^{t+1}|x^t)\,. \tag{38}$$

Being $P(x^{t+1}|x^t)$ a conditional probability, we note that it is also normalized, meaning that the probability of transition from a state x^t to any possible state (x^t included) is 1, that is $\sum_{x^{t+1}} P(x^{t+1}|x^t) = 1$.

The coin toss example from last paragraph falls trivially within the Markov Chain (MC) family, since each new toss does not depend at all on the results from the last tosses, not even the last one. Random walks instead are more dignified example of a Markov chain, with dependence of the next state on the current state (and no other past states). This conditional dependence can be easily read in equation (36).

On the other hand, self-avoiding walks are not Markov chains: equation (38) does not hold, since the knowledge of each of the past states $x^0, x^1, \dots$ is relevant to determine which states can be visited at time $t+1$ and which cannot.

Thanks to the Markov property, the probability of a trajectory takes a simple structure. Consider a Markov chain $(X^0, X^1, \dots, X^t)$ running up to some time t. We have

$$P(x^0, x^1, \dots, x^t) = P(x^0) \prod_{t'=0}^{t-1} P(x^{t'+1}|x^{t'}). \tag{39}$$

Therefore, everything is encoded in the initial state distribution $P(x^0)$ and in the *transition probabilities* $P(x^{t'+1}|x^{t'})$. From the sampling point of view, this is very convenient: in order to sample a Markov chain trajectory, one first samples x^0 according to $P(x^0)$, then extracts x^1 according to $P(x^1|x^0)$, x^2 from $P(x^2|x^1)$ and so on.

Moreover, we have the following iterative rule that links the marginal distribution probability $P(x^t)$ at time t – i.e. the probability distribution of the random variable X^t - to the one at time $t+1$:

$$P(x^{t+1}) = \sum_{x^t} P(x^{t+1}|x^t) P(x^t) \qquad \forall x^{t+1} \in \mathcal{X} \tag{40}$$

Those last equations simply say the all the "probability mass" in the state x^{t+1} at time $t+1$ is just given by the mass transitioning to x^{t+1} from any possible state at previous time.

Finite State Spaces

When the state space $\mathcal{X}$ is finite (we call this case *finite* Markov chain) it may be convenient to reframe the concept of Markov processes in the framework of linear algebra. Let M be the size of $\mathcal{X}$, i.e. the total number of states, $M = |\mathcal{X}|$. We can label this states with numbers going from 1 to M. Therefore, for each element $x \in \mathcal{X}$ there is a corresponding label $i \in \{1, \dots, M\}$ and we can essentially use the either i or x to denote the same state.

The transition probabilities at time t, $P(x^{t+1}|x^t)$ can then be encoded into $M \times M$ matrix $P_{ij}^{(t)}$ that we call *transition matrix,* whose elements are given by

$$P_{ij}^{(t)} \equiv P(X^{t+1} = i | X^t = j) \qquad \forall i, j. \tag{41}$$

Since $P^{(t)}$ represents conditional probabilities, it enjoys the property of being column-normalized, in the sense that $\sum_i P_{ij}^{(t)} = 1$ for each column j. This corresponds to saying that $P^{(t)}$ is a stochastic matrix.

On the other hand, the marginal probability distribution $P(x^t)$, can be encoded into a vector of size M, that we call $\boldsymbol{p}^t$ defined as follows by its components p_i^t:

$$p_i^t = P(X^t = i) \qquad \forall i. \tag{42}$$

Up to this point, it seems like we have introduced some redundant definitions without gaining anything. A first understanding of the convenience of this matrix-vector formalism derives from rewriting Eq. (40) in the new formalism:

$$p_i^{t+1} = \sum_{j=1}^{M} P_{ij}^{(t)} p_j^t \qquad \forall i. \tag{43}$$

This is nothing else than a matrix-vector product, that one can conveniently express into the vectorial equation:

$$\boldsymbol{p}^{t+1} = P^{(t)} \boldsymbol{p}^t$$

Using multiple time last equation, we can also write the distribution at time t as function of that at time $t = 0$:

$$\boldsymbol{p}^t = \left(P^{(t-1)} P^{(t-2)} \dots P^{(0)} \right) \boldsymbol{p}^0 \tag{44}$$

were in the parentheses we have a matrix made of the product of t transition matrices.

From now on we will use the matrix-vector notation and the probability distribution notation indistinctly when discussing finite-space Markov chains.

Stationary Markov Chains

If the transition probabilities do not depend on time indexes, the Markov chain is said to be *time-independent* or *time-homogeneous* or *stationary.* The stationary condition can be written as

$$P(X^{t+1} = x' | X^t = x) = P(X^{t'+1} = x' | X^{t'} = x) \quad \forall\, x, x' \in \mathcal{X} \text{ and } \forall t, t'. \tag{45}$$

For finite state spaces, this also means that we can encode the process into a single transition matrix P_{ij}, so that we have

$$\boldsymbol{p}^{t+1} = P\, \boldsymbol{p}^t \tag{46}$$

$$= P^t\, \boldsymbol{p}^0. \tag{47}$$

Notice that P^t in last is the product of t matrices P, while the t in $\boldsymbol{p}^t$ is just a time index. Throughout this notes we will mostly consider stationary Markov processes, the only exception being Simulated Annealing that is non-stationary due to the time dependence of the temperature.

3.2.1 Stationary Distributions

The concept of stationary distribution is central to the study and the applications of Markov chains. Given a stationary Markov chain identified by the transition probabilities $P(x'|x)$, a probability distribution $\pi(x)$ is said to be a *stationary distribution* for the process if it reproduces itself during the temporal

dynamics (40) (or equivalently Eq. (46)); namely, $\pi(x)$ must satisfy the following condition:

$$\pi(x') = \sum_{x \in \mathcal{X}} P(x'|x)\pi(x) \qquad \forall x' \in \mathcal{X},$$

which in vectorial form reads

$$\boldsymbol{\pi} = P\boldsymbol{\pi}.$$

Notice that last equation tells us that if the stationary distribution $\boldsymbol{\pi}$ exists, than it is an eigenvector of the transition matrix P with eigenvalue 1.

At this point, a few questions concerning stationary distributions naturally arise:

1. does any Markov process P admits a stationary distribution?
2. is such distribution unique?
3. starting from an initial condition $P(x^0)$, does $P(x^t)$ converges on $\pi(x)$ at large times?

The answer to all the 3 questions is yes, once we restrict the family of process we consider to *ergodic* processes. A precise definition of ergodicity is technically involved, therefore here we just convey the general intuition; the interested reader can find more details in any specialized stochastic processes textbook.

Essentially, the idea of ergodicity is that the stochastic dynamics is not trapped in some regions of the state space for indefinite time, but is free to explore the whole space as time goes by. Moreover, in an ergodic process there is no periodicity. Formally we state that a finite Markov Chain is *ergodic* if it has the following properties:

- **Irreducibility**: the chain can go from any state j to any state i in a finite number of steps with probability larger than 0. This means that for each pairs of states i and j there exists an integer $k > 0$ such that $(P^k)_{ij} > 0$.
- **Aperiodicity**: there is no state that is visited by the stochastic process with finite probability only periodically, that is at multiples of a fixed period $k > 1$.

If the Markov chain P is ergodic, a theorem due to Perron and Frobenius implies that a stationary distribution exists, and it is unique. Calling $\boldsymbol{\pi}$ such stationary distribution (in vector notation), we also have that, regardless of the initial state distribution $\boldsymbol{p}^0$, the state distribution $\boldsymbol{p}^t$ converges to $\boldsymbol{\pi}$ at large times, that is:

$$\boldsymbol{p}^t = P^t\boldsymbol{p}^0 \xrightarrow[t\to+\infty]{} \boldsymbol{\pi}$$

One can show that convergence is exponentially fast, and the convergence rate is given by the second largest (in modulus) eigenvalues of P (the first eigenvalue being 1).

3.2.2 Averages in ergodic systems

A very important property of ergodic processes is that *time averages* can be replaced by *ensemble averages* over the stationary distribution (actually, this is the very definition of an ergodic process). This means, that for any *observable* $f(x)$ (a function on $\mathcal{X}$) and for any trajectory $(x^t)_{t\geq 0}$ sampled according to an ergodic process with stationary distribution π, we have (almost surely in probability) that

$$\begin{aligned}\bar{f} &\equiv \lim_{T\to\infty} \frac{1}{T}\sum_{t=1}^{T} f(x^t) \\ &= \mathbb{E}_{X\sim\pi}\, f(X) \equiv \sum_{x\in\mathcal{X}} \pi(x) f(x)\end{aligned}$$

Simple 2-state Markov chain. As first example of (stationary) Markov chain let us discuss the simple case with only $M = 2$ states $\mathcal{X} = \{1, 2\}$, and in which the probability to change state is $1 - p$ and to remain in the state is p, see Figure 9.

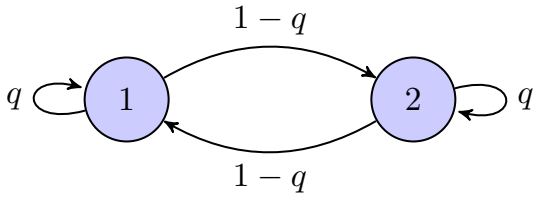


Figure 9: A simple 2-state Markov chain.

This Markov chain is manifestly irriducible (if $q \neq 0, 1$) and aperiodic. Therefore the chain is ergodic and it admits a stationary distribution $\boldsymbol{\pi}$ to which the process $\boldsymbol{p}^t \to \boldsymbol{\pi}$ converges for large times. Let us now compute $\boldsymbol{\pi}$. The transition matrix can be written as

$$P = \begin{pmatrix} q & 1-q \\ 1-q & q \end{pmatrix} \tag{48}$$

Since $p_1^t + p_2^t = 1$ we can describe the Markov evolution only via p_1^t. Applying equation (43) we have

$$\begin{aligned} p_1^t &= q p_1^{t-1} + (1-q)(1-p_1^{t-1}) \\ &= 1 - q + (2q-1)p_1^{t-1}\,. \end{aligned} \tag{49}$$

Iterating the previous equation we obtain

$$p_1^t = \frac{1}{2} + (2q-1)^t \left(p_1^0 - \frac{1}{2}\right)$$

where p_1^0 is the probability to be in the state 1 initially. Since $q < 1$ (if $q = 1$ we will remain in the initial state forever) we have that for $p_1^t \to \frac{1}{2}$ independently of p_1^0 and q. Therefore this Markov chain admits a the stationary distribution which is simply $\boldsymbol{\pi} = (1/2, 1/2)$. The convergence to the stationary distribution is exponentially fast with a rate given by $\tau = -1/\ln|1-2q|$. In fact you can easily check that the eigenvalues of the matrix (48) are $\lambda_1 = 1$ and $\lambda_2 = 1 - 2q$: the first corresponds to the stationary distribution $\boldsymbol{\pi}$ (which is the corresponding eigenvector), whereas the second one controls the convergence rate.

M-cycle Markov Chain. Let us consider now the following Markov chain with M states $\mathcal{X} = \{1, 2, \dots, M\}$, as in Figure 10 in the case $M = 5$.

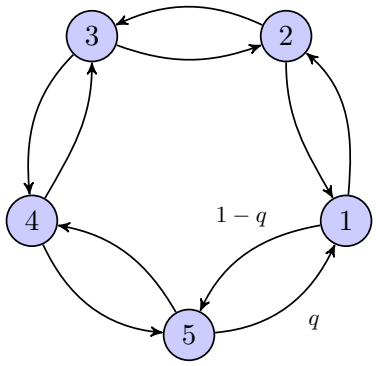


Figure 10: M-cycle with $M = 5$.

The transition probability matrix has components

$$P_{ij} = \begin{cases} q\,, & i = j+1 \\ 1-q\,, & i = j-1 \\ 0\,, & \text{otherwise} \end{cases} \tag{50}$$

and we sub-intend that $P_{M+1,M} = P_{1,M}$ and $P_{0,1} = P_{M,1}$. Equation (43) gives

$$p_j^{t+1} = q p_{j-1}^t + (1-q) p_{j+1}^t\,. \tag{51}$$

It is easy to see that $\pi_j = 1/M$ is a stationary solution to the previous equation. Now we want to see if the state distribution $\boldsymbol{p}^t$ approaches it for large times. To see this let us take for example $M = 4$ and $q = 1/2$, with initial condition $\boldsymbol{p}^0 = (1, 0, 0, 0)$. At time $t = 1$ we will have $\boldsymbol{p}^1 = (0, 1/2, 0, 1/2)$, at time $t = 2$, $\boldsymbol{p}^1 = (1/2, 0, 1/2, 0)$ and so on. Therefore if n is even $\boldsymbol{p}^t$ cannot converge (even if the chain is irreducible) to the invariant distribution, because the chain is periodic. On the other hand, if n is odd, periodicity is lost and the chain is ergodic; therefore $\boldsymbol{p}^t \to \boldsymbol{\pi}$ for large times.

Playing at the casino. Suppose we play at the roulette. The roulette is a disc composed of 37 numbers (or 38 in the american roulette), 18 of which are red, other 18 are black and 1 (2 in the American one) is green. Suppose we choose to bet 1 euro at a time on the color red or black. The probability of winning is clearly $q = {}^{18}/_{37} < {}^{1}/_{2}$. If we win, we gain 1 euro on the initial bet, whereas if we loose we simply loose the initial bet. Of course if we finish money, we sadly leave the casino. On the opposite side, suppose that we will be sufficiently smart to leave the casino if our capital increases above a certain threshold C (or that the casino will force us to stop playing...). Suppose now to have at time $t = 0$ an initial capital $x_0 = j = 500$ euros and we aspire to a capital $C = 2j = 1000$ euros for example. Is our aspiration too risky?

We can model the game by using a Markov chain with $M = C + 1$ states, as represented in Figure 11.

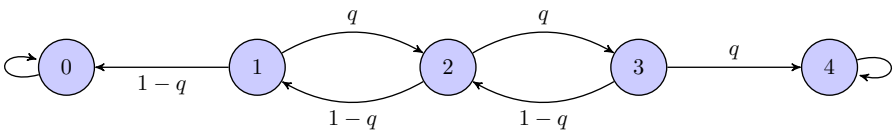


Figure 11: Markov chain for games at the roulette with $M = 5$. If we win the bet our capital raises by 1 euro; if we loose, we loose the 1 euro bet.

We can easily quantify the risk of the game, by computing the probability that we fail (i.e. we loose all our initial money), before reaching the capital C we desire. Denoting by F the "failing event", the probability to fail given an initial amount of money $x_0 = j$ is

$$P(F|x_0 = j) = (1 - q)P(F|x_1 = j - 1) + qP(F|x_1 = j + 1)\,.$$

In fact you can either fail if you win the first bet or if you loose it. However, since the failing event is independent on the time it needs to happen, we have $P(F|x_1 = j) = P(F|x_0 = j) \equiv u_j$. Therefore the previous equation becomes

$$u_j = (1 - q)u_{j-1} + qu_{j+1}\,. \tag{52}$$

This equation must be solved with the boundary conditions $u_0 = 1$ (if we start with no money we already failed) and $u_C = 0$ (if we start with the amount of money we desire we do not play at all). If $q \neq {}^{1}/_{2}$, the solution is

$$u_j = \frac{\left(\frac{1-q}{q}\right)^C - \left(\frac{1-q}{q}\right)^j}{\left(\frac{1-q}{q}\right)^C - 1}\,. \tag{53}$$

As you can see, the probability to fail is almost certain, even if $q \simeq {}^{1}/_{2}$ as in the case of the roulette machine. With 500 euros initially we reach in doubling our capital with probability $1 - u_j \simeq \left(\frac{1-q}{q}\right)^{j-C} \simeq 10^{-12}$. In other words, if you want a better strategy when playing the roulette, it is better to bet all the 500 euros once; in this case the probability to loose is only ${}^{18}/_{37}$.

3.3 Global Balance and Detailed Balance

Consider now a stationary distribution $\boldsymbol{\pi}$ for the Markov chain with transition matrix P, that is a distribution (in vector form, since we consider finite space states now) satisfying the equation

$$\pi_i = \sum_j P_{ij}\pi_j \qquad \forall i.$$

Now we plug in the l.h.s. the identity in the form $1 = \sum_j P_{ji}$ which stems from the fact that P is a stochastic matrix representing a conditional probability: summing over a column of the matrix P_{ji} is equivalent to the probability of going *anywhere* from the state i, which must be 1. If we then subtract from both sides a term $P_{ii}\pi_i$ we arrive to the following equation, known as *global balance* condition:

$$\sum_{j\neq i} P_{ji}\pi_i = \sum_{j\neq i} P_{ij}\pi_j \qquad \forall i. \tag{54}$$

This set of equations (we have one for each i) states than when stationarity is reached, whatever "flows out" of a given state i to all other states is perfectly balanced by what "flows in" from all other states into i. The global balance condition is a necessary and sufficient condition for stationarity.

Now we give a stronger condition called *detailed balance* condition that, being stronger, doesn't apply to any stationary distribution and therefore is only a sufficient condition for stationarity. The *detailed balance* condition reads

$$P_{ji}\pi_i = P_{ij}\pi_j \qquad \forall i, j.$$

This form of balance is "detailed" because the balance of out-flow and in-flow applies to each pair of states i and j. Summing both sides of last equation over j it is easy to see that detailed balance implies global balance, which in turn implies stationarity. Checking if the detailed balance condition holds for a certain distribution $\boldsymbol{\pi}$ and chain P is typically easier than checking the global balance condition, therefore this condition is typically actually used in practical applications, as we will see in the following section.

4 Basic Markov Chain Monte Carlo algorithms

We now turn the problem around: instead of asking ourselves what is the stationary distribution of a Markov chain, we consider a target distribution $\pi(x)$ and our objective is to devise a Markov chain that has $\pi(x)$ as its stationary distribution. In other words, we want to find a Markov chain that allows us to sample from the given distribution $\pi(x)$. For this reason all those Monte Carlo methods that have a proposal density which is constructed using a Markov process are called *Markov Chain Monte Carlo* methods (MCMC). In this section we will see the most popular MCMC methods: the Metropolis method and Gibbs sampling.

4.1 Metropolis Algorithm

Algorithm 3 Metropolis Algorithm

Require: Initial configuration x^0, time t_{max}
Ensure: A configuration x approximately distributed as π

$x \leftarrow x^0$
for $t = 1, ..., t_{max}$ **do**
  $x' \leftarrow$ sample from $g(\cdot|x)$ ▷ *propose a move from x to x'*
  $a \leftarrow \min\left(1, \frac{\pi(x')}{\pi(x)} \frac{g(x|x')}{g(x'|x)}\right)$
  $r \leftarrow$ sample from Uniform$(0, 1)$
  if $r \leq a$ **then** ▷ *accept the move*
    $x \leftarrow x'$
  end if
end for
return x

In order to do devise a Markov chain that at long times samples from $\pi(x)$, we let the detailed balance condition hold:

$$P(x'|x)\pi(x) = P(x|x')\pi(x') \tag{55}$$

$$\Rightarrow \frac{P(x'|x)}{P(x|x')} = \frac{\pi(x')}{\pi(x)} \tag{56}$$

We then split the transition probability $P(x'|x)$ in two parts

$$P(x'|x) = g(x'|x)A(x', x) \tag{57}$$

where:

- $g(x'|x)$ is called the *proposal kernel* or *selection probability*. This is a properly normalized conditional probability distribution, that is

$$\sum_{x'} g(x'|x) = 1\,.$$

- $A(x', x)$ is called the *acceptance ratio*. For each choice of x' and x, this is a number $0 \leq A(x', x) \leq 1$.

It turns out that the decomposition (57) can be interpreted as follows: the probability of transitioning from a certain state x to a different state x' is equal to the probability of sampling x' (the proposal) from the distribution $g(\bullet|x)$, times the probability of accepting the proposal, this given by $A(x', x)$.

Once we have this g and A decomposition, we have to link it to the target distribution π in order to solve our task. The link is done by the detailed balance condition. We keep $g(x'|x)$ generic, since we want the proposal kernel to be

flexible, and obtain the following equation from the detailed balance condition for

$$\frac{g(x'|x)A(x',x)}{g(x|x')A(x,x')} = \frac{\pi(x')}{\pi(x)} \tag{58}$$

$$\Rightarrow \frac{A(x',x)}{A(x,x')} = \frac{g(x|x')\pi(x')}{g(x'|x)\pi(x)} \qquad \forall x \neq x'. \tag{59}$$

There are many different choices for the function $A(x',x)$ that satisfy last equation. The Metropolis choice of $A(x',x)$ reads

$$A(x',x) = \min\left(1, \frac{\pi(x')}{\pi(x)}\frac{g(x|x')}{g(x'|x)}\right) \tag{60}$$

It is easy to see that this rule satisfies the detailed balance condition by simply distinguishing the cases $\pi(x')g(x|x') > \pi(x)g(x'|x)$ and $\pi(x')g(x|x') < \pi(x)g(x'|x)$.

For any target distribution $\pi(x)$ and any proposal kernel $g(x'|x)$, using Eq. (60) and Eq. (57) we can construct a Markov chain that has $\pi(x)$ as its asymptotic distribution. The problem of sampling from π is solved by starting from any state and running the Markov chain for long time. The sampling procedure for this Markov chain is called Metropolis algorithm and presented in Alg. 3.

Consideration 1 Typically g is chosen to be symmetric, that is $g(x'|x) = g(x|x')$, $\forall\ x, x'$. With this choice, the acceptance ratio (60) simplifies to

$$A(x',x) = \min\left(1, \frac{\pi(x')}{\pi(x)}\right)$$

i.e. it depends only on the ratio of the target distribution evaluated in different points.

Consideration 2 From Equation (60), in order to obtain the acceptance ratio we need to compute ratio $\pi(x')/\pi(x)$, which requires the knowledge of π up to an overall normalization factor. This is very important, since in many practical application $\pi(x)$ is in fact known only up to a normalization factor that is hard to compute.

Consideration 3 Typically, proposal kernels $g(x'|x)$ are such that it is computationally cheap to sample from. A very common proposal kernel when dealing with binary variables (e.g. when a configuration x is a string of N bits, $x = (x_1, ..., x_N)$ with $x_n \in \{0, 1\}$) is defined by the following very easy to implement process (which should be used in line 3 in Alg. 3):

- sample a uniformly distributed number $i \in \{1, ..., N\}$
- flip the component i of x, i.e. change x_i from 0 to 1 or viceversa.

Notice that this kernel is symmetric.

Consideration 4 Since subsequent samples are correlated between each other (because the process is a Markov chain), in general MCMC methods, the Markov chain may be run for a long time before samples become uncorrelated samples of $\pi(x)$. This is why we have in Alg. 3 we let the Markov chain evolve for $t_{\max}$ time.

4.1.1 Metropolis algorithm in high dimensions

The choice of the proposal kernel in the Metropolis method is very delicate; here we want to show how different choices of proposal kernels may affect performances of the algorithm. This analysis will be of great help in understanding how to design the proposal kernel in a good way.

Suppose to be, for simplicity, in a continuous m-dimensional space, such as $\mathcal{X} = \mathbb{R}^m$, with m large. In many high-dimensional problem, the proposal kernel is endowed with a length scale ρ. This parameter controls how much the distribution $g(\boldsymbol{x}'|\boldsymbol{x})$ is spread out: if ρ is large we it will much more probable to propose a value of $\boldsymbol{x}' \in \mathbb{R}^m$ which is far from the current state $\boldsymbol{x}$ and vice versa.

Evidently, this parameter must be chosen in such a way to be small with respect to the largest length scale $\sigma_{\max}$ of the probable region of the target distribution $\pi(\boldsymbol{x})$. In fact, if $\rho > \sigma_{\max}$, we can propose frequently new configurations $\boldsymbol{x}'$ for which $\pi(\boldsymbol{x}')$ is low, and therefore by Eq. (60) they will be rejected with high probability. On the other hand if ρ is too small, the proposed move will be always be accepted, but our Markov chain will take a long time to explore all the probable region of the target distribution (i.e. the convergence to $\pi(\boldsymbol{x})$ is slow).

To make a concrete example, consider a target distribution $\pi(\boldsymbol{x})$ which has strong correlations in a given direction, for example a multivariate Gaussian with a large variance $\sigma_{\max}$ in a given direction, see Figure 12, for a picture in $m = 2$ dimensions.

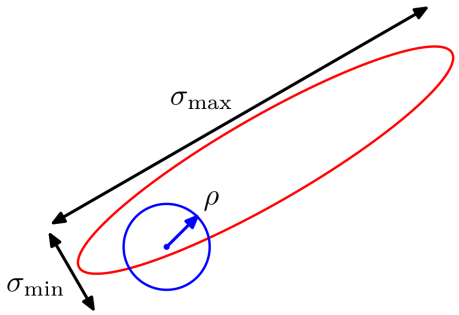


Figure 12

As a proposal kernel let us use a multivariate isotropic (i.e. with the same variance ρ in every direction) Gaussian centered around the current state $\boldsymbol{x}$. As

evident in Fig. 12, if ρ is larger than $\sigma_{\max}$ the points inside the blue circle lie with high probability outside the area enclosed by the red ellipse; therefore we expect low acceptance rates.

If ρ is very small we will explore the area of the red ellipse by a random walk. An estimate of the time needed for a random walk to explore all the target distribution is therefore to obtain independent samples is

$$t_{\max} \simeq \left(\frac{\sigma_{\max}}{\rho}\right)^2 \tag{61}$$

Therefore the scale ρ of the target distribution must be chosen as high as possible to reduce $t_{\max}$. Roughly this suggests ρ must be chosen to be of the order of $\sigma_{\min}$ the smallest standard deviation of the target density, but of course the precise value of ρ depends on the value of the other variances. Note that, differently for what we have seen in the case of rejection and importance sampling, $t_{\max}$ does not depend on the dimensionality m!

4.2 Gibbs Sampling

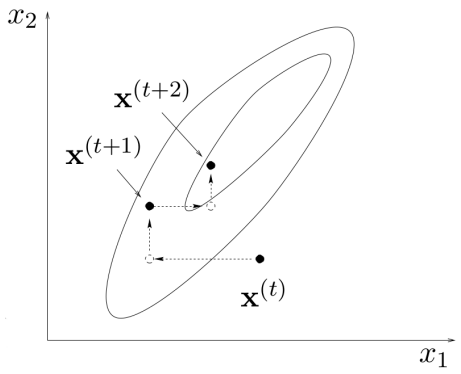


Figure 13: Gibbs Sampling

Gibbs sampling is a Markov chain method that is designed to work in at least two dimensions. As we shall see, it is also a special case of the Metropolis method. We assume that, although sampling from $\pi(\boldsymbol{x})$ is difficult, we know how to sample from one dimensional conditional distributions $\pi(x_i|\{x_k\}_{k\neq i})$, for $i = 1, \ldots, m$. Suppose we start from a state $\boldsymbol{x}^{(t)}$. In Gibbs sampling the first component of $\boldsymbol{x}$ at time $t+1$ will be obtained by sampling from $\pi(x_1|x_2^{(t)}, \ldots, x_m^{(t)})$. The second component will be sampled from $\pi(x_2|x_1^{(t+1)}, x_3^{(t)} \ldots, x_m^{(t)})$ and so on. A single iteration therefore involve sam-

pling from m conditional distributions

$$x_1^{(t+1)} \sim \pi(x_1|x_2^{(t)}, \dots, x_m^{(t)}) \tag{62a}$$
$$x_2^{(t+1)} \sim \pi(x_2|x_1^{(t+1)}, x_3^{(t)} \dots, x_m^{(t)}) \tag{62b}$$
$$\vdots$$
$$x_m^{(t+1)} \sim \pi(x_m|x_1^{(t+1)}, \dots, x_{m-1}^{(t+1)}) \tag{62c}$$

A schematic picture of Gibbs sampling in shown in Fig. 13 in 2 dimensions.

We now see that Gibbs sampling is no more than a particular case of the Metropolis method. Let us consider a Metropolis step, in which we update only the first component of $\boldsymbol{x}$. Let us take, as proposal kernel $g(\boldsymbol{x}'|\boldsymbol{x}) = \pi(x_1'|\{x_k\}_{k\neq 1})$ as in the first step of Gibbs sampling. In this case the term in equation (60) becomes

$$\begin{aligned}\frac{g(\boldsymbol{x}|\boldsymbol{x}')\pi(\boldsymbol{x}')}{g(\boldsymbol{x}'|\boldsymbol{x})\pi(\boldsymbol{x})} &= \frac{\pi(x_1|\{x_k'\}_{k\neq 1})\,\pi(\boldsymbol{x}')}{\pi(x_1'|\{x_k\}_{k\neq 1})\,\pi(\boldsymbol{x})} \\ &= \frac{\pi(x_1|\{x_k'\}_{k\neq 1})\,\pi(x_1'|\{x_k'\}_{k\neq 1})\pi(\{x_k'\}_{k\neq 1})}{\pi(x_1'|\{x_k\}_{k\neq 1})\,\pi(x_1|\{x_k\}_{k\neq 1})\pi(\{x_k\}_{k\neq 1})} \\ &= 1\end{aligned}$$

where we have used $\pi(\boldsymbol{x}) = \pi(x_1|\{x_k\}_{k\neq 1})\pi(\{x_k\}_{k\neq 1})$ in the second line and the fact that $x_k' = x_k$ for every $k \neq 1$ in the last line, since the components different from the first one are not updated. Therefore $A(\boldsymbol{x}', \boldsymbol{x}) = 1$, i.e. the proposed move will be always accepted.

5 The Simulated Annealing Algorithm

In this section we describe Simulated Annealing, a very important and versatile algorithm that allows to perform optimization by means of Markov Chain Monte Carlo (MCMC). To this aim we will use the Metropolis algorithm and see how it can be extended to become a solver for optimization problems.

5.1 The Boltzmann distribution

Consider a discrete probability distribution $\pi(x)$. By the very definition of probability distribution (density), we have the following two properties

- $p(x) \geq 0\ \forall x$. That is, $p(x)$ is non-negative.
- $\sum_x p(x) = 1$.That is, $p(x)$ is normalized (to 1).

Now let's restrict ourselves to the case where $p(x)$ is always strictly larger then zero, $p(x) > 0\ \forall x$ (the general case where $[(x) = 0$ for some x can be easily

accommodated by restricting the configuration space). Then there exists a function $E(x)$ and a constant Z, such that we can express $\pi(x)$ as

$$p(x) = \frac{e^{-E(x)}}{Z}. \tag{63}$$

The normalization factor Z is simply given by $Z = \sum_x e^{-E(x)}$. It is easy to see that this definition satisfies both properties of a well defined probability function.

- $E(x)$ is called *energy function*. It is usually easy to evaluate.
- Z is called *partition function*. It is usually hard to compute since it typically involves the sum over a large number of configurations. Fortunately we don't have to worry about the hardness of computing Z in MCMC algorithms (see Consideration 2 in Section 4.1).

Also, notice that for a given $p(x)$, the corresponding $E(x)$ is not unique, since the partition function can absorb any constant factor in the numerator. In fact, adding to $E(x)$ any constant term C, $E_2(x) = E(x)+C$, we obtain a new energy function that describes the same probability distribution. Therefore the energy function is defined only up to a constant term.

It is useful to consider the converse situation, as people in *statistical physics* often do. The starting point now is the *energy function* $E(x)$ and a parameter $T > 0$ called *temperature*. Also, usually one calls $\beta = \frac{1}{T}$ the *inverse temperature*. Given an energy function and a temperature, one can define the *Boltzmann probability distribution* $\pi(x)$ *as*

$$\pi_T(x) = \frac{e^{-\frac{1}{T}E(x)}}{Z_T}, \tag{64}$$

where as usual the partition function is defined so that it normalizes the distribution to one: $Z_T = \sum_x e^{-\frac{1}{T}E(x)}$.

The temperature T is exactly the same quantity you measure with thermometers, but it may also be considered as a tuning parameter that changes the shape of the Boltzmann distribution, as we will now see. For very large temperature ($T \gg 1$) $\pi_T(x)$ becomes the uniform distribution (Fig. 14), giving to all configurations the same weight:

$$\pi_T(x) \approx \frac{1}{\sum_{x'} 1} \qquad \text{for } T \gg 1 \tag{65}$$

\

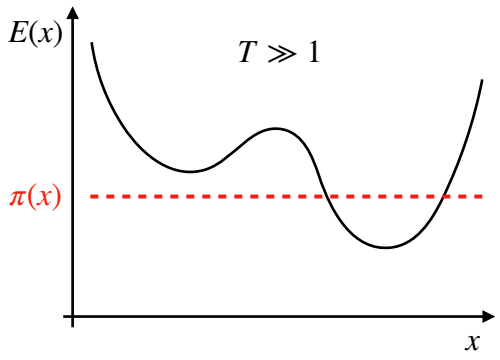


Figure 14: Black line: energy function $E(x)$. Red line: Boltzmann distribution $\pi_T(x)$ in the limit of high temperature T.

At intermediate values of the temperature ($T \approx 1$) the configurations with lower energies are preferred (i.e. the probability density is higher in correspondence of these configurations.) (Fig. 15).

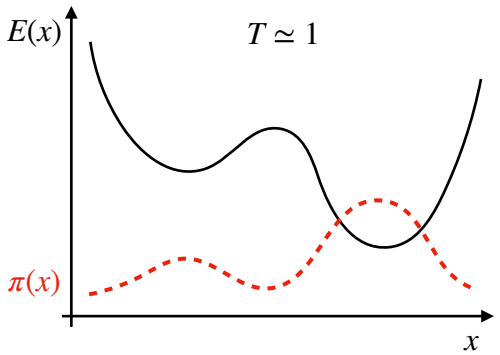


Figure 15: Black line: energy function $E(x)$. Red line: Boltzmann distribution $\pi_T(x)$ for intermediate values of the temperature T.

When the temperature is close to zero ($T \ll 1$) the distribution $\pi_T(x)$ is peaked around the smallest energy configuration (Fig. 16) called *ground state*, namely the *global minimum* of $E(x)$.

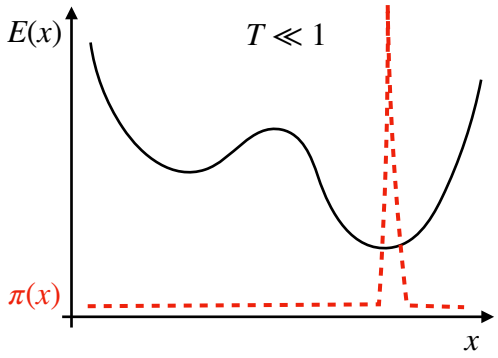


Figure 16: Black line: energy function $E(x)$. Red line: Boltzmann distribution $\pi(x)$ for low values of the temperature T.

Note: For any $T < +\infty$ the configurations with lower energy are favored, but typically there are *many more* high energy configurations than low energy ones, so while sampling from $\pi_T(x)$ you may almost always get high energy configurations: this is called *entropic effect*. Entropy is a measure of how many possible configurations your system may assume, and this is related to disorder. An everyday-life example of this effect is the following: your room may appear always messy, unless you put a lot of effort into tidying it up. This is because there are many ways for it to be messy and just a few ordered configurations. In the Boltzmann distribution the temperature T tunes the balance between energy minimization (low T) and entropy maximization (high T).

5.2 Sampling from the Boltzmann distribution

We can sample from the Boltzmann distribution using Markov Chain Monte Carlo with the Metropolis algorithm

Algorithm 4 Metropolis for Boltzmann sampling

Require: energy function E, temperature T
Require: stopping time t_{max}, initial configuration x^0
Ensure: a configuration sampled from the Boltzmann distribution π_T

1: $x \leftarrow x^0$
2: **for** $t = 1, ..., t_{max}$ **do**
3: $\quad x' \leftarrow$ sample from $g(\bullet \mid x)$ ▷ *propose a move from x to x'*
4: $\quad \Delta E \leftarrow E(x') - E(x)$
5: $\quad r \leftarrow \text{Uniform}(0, 1)$
6: $\quad$ **if** $r \leq \min(1, e^{-\Delta E/T})$ **then** ▷ *accept the move*
7: $\quad\quad x \leftarrow x'$
8: $\quad$ **end if**
9: **end for**
10: **return** x

So here you see that the important quantity is the energy difference ΔE: if $\Delta E < 0$, i.e. the configuration x' has lower energy than the configuration x, you accept the move with probability one (as you see from line 6 of the pseudocode and considering that $e^{-\Delta E/T} < 1$ if $\Delta E < 0$). So you're preferring configurations with lower energy. On the other hand, if $\Delta E > 0$ (i.e. x' has higher energy than x), you can still accept the move, but only with probability $e^{-\Delta E/T}$. The temperature parameter T tunes the probability by which energy increasing proposals are accepted, from always ($T = +\infty$) to never ($T = 0$).

5.3 Optimization with the Metropolis algorithm

For some "objective / cost / loss / energy" function $E(x)$ (the name depends on the scientific field considered) defined on some space $\mathcal{X}$, consider the following optimization problem:

$$x^* = \min_{x \in \mathcal{X}} E(x).$$

We call x^*, the global minimum of $E(x)$, the *ground state* of the the system. According to our previous discussion in Section 5.1, we could solve the problem if we were able to sample from the zero-temperature ($T = 0$) Boltzmann distribution (as it is peaked on the lowest energy configuration). We are therefore tempted to use the following variation of the Metropolis algorithm:

Algorithm 5 Zero-temperature Metropolis algorithm

Require: energy function E
Require: stopping time t_{max}, initial configuration x^0
Ensure: a candidate ground state.

1: $x \leftarrow x^0$
2: **for** $t = 1, ..., t_{max}$ **do**
3: $x' \leftarrow$ sample from the kernel $g(\bullet|x)$ ▷ *propose a move from x to x'*
4: $r \leftarrow \text{Uniform}(0, 1)$
5: $\Delta E \leftarrow E(x') - E(x)$
6: **if** $\Delta E \leq 0$ **then** ▷ *only energy decreasing moves*
7: $x \leftarrow x'$
8: **end if**
9: **end for**

Here you see that the difference w.r.t. to the standard Metropolis algorithm is contained again in line 6 of the pseudocode: you accept the move only if $\Delta E \leq 0$. The $0T$ Metropolis algorithm is a *stochastic greedy heuristic* for solving optimization problems:

stochastic: the proposal x' is stochastic and depends only on the current configuration x (it is a Markov Chain);

greedy: only the moves that decrease (or not-increase) the energy are accepted. Since the moves are typically local (x' close to x) this may not be optimal and it may get stuck in a local minimum;

heuristic: the algorithm is *not* guaranteed to solve the problem, but it may reach a good configuration

For hard combinatorial problem such as the Traveling Salesman Problem, this greedy strategy is very likely to fail, preforming premature optimization and returning some sub-optimal solutions. In the next section we present a much more powerful algorithm to deal with "rough landscape" problems.

5.4 Simulated Annealing

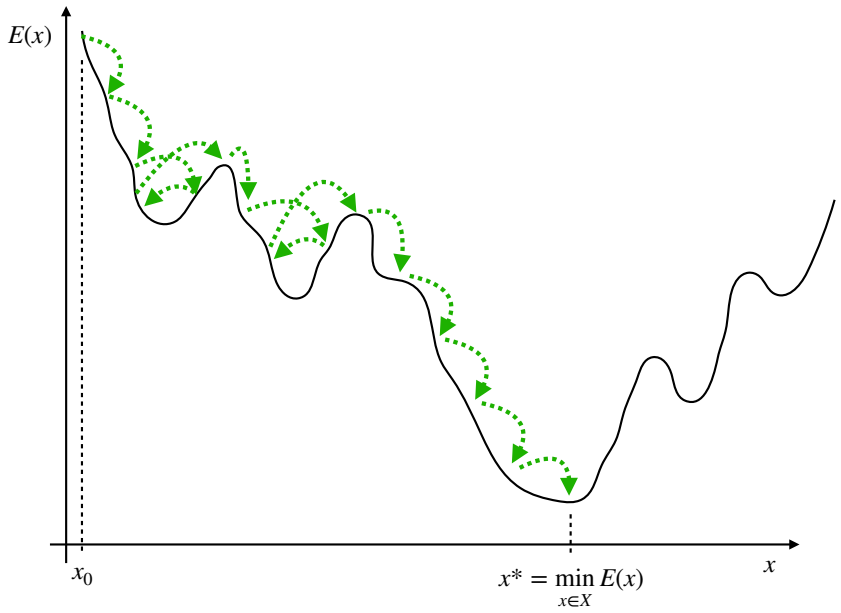


Figure 17: Example of a SA trajectory. A greedy MCMC algorithm would get stuck in the first local minimum valley, while SA is able to overcame the small barriers because it can accept moves that increase the energy while $T > 0$. The algorithm should *ideally* be tuned in such a way that the temperature reaches zero when sampling in proximity of the global minimum.

The idea behind the Simulated Annealing (SA) algorithm is that a MCMC simulation at *high* temperature is able to explore the state space without getting trapped by energy barriers. Since in an optimization problem one wants to ultimately find the ground state (global minimum), it may be convenient to *gradually lower the temperature* to drive the system towards regions of lower and lower energy. The temperature sets the balance between exploration and exploitation: we want to explore a lot at the start of simulation, and we want to exploit the collected information at the end.

We define a decreasing temperature schedule

$$T^0 \geq T^1 \geq T^2 \geq ... \geq T^t \geq ... \geq T^{t_{\max}} = 0$$

and we use the schedule inside our Metropolis algorithm. Ideally, if the schedule is slow enough, at time t the algorithm will be roughly sampling from the Boltzmann distribution with temperature T^t, and as time goes by the sampling will track the changing shape of the distribution.

A possible choice for the temperature schedule is

$$T^{t+1} = T^t - \Delta T$$

for some constant ΔT that becomes a parameter of our SA algorithm. Many other different schedule could be considered. The algorithm is presented as Alg. 6.

Algorithm 6 Simulated Annealing

Require: energy function E, temperature schedule $\{T^t\}_t$
Require: stopping time t_{max}, initial configuration x^0
Ensure: a candidate ground state.

$x \leftarrow x^0$
for $t = 1, ..., t_{max}$ **do**
    $x' \leftarrow$ sample from the kernel $g(\bullet \mid x)$
    $r \leftarrow \text{Uniform}(0, 1)$
    $\Delta E \leftarrow E(x') - E(x)$
    if $r \leq \min(1, e^{-\Delta E / T^t})$ **then** ▷ *accept. prob. depends on t*
        $x \leftarrow x'$
    end if
end for
return x

References

[1] David JC MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003.

[2] Christophe Andrieu. Monte carlo methods for absolute beginners. In *Summer School on Machine Learning*, pages 113–145. Springer, 2003.

[3] Werner Krauth. *Statistical mechanics: algorithms and computations*, volume 13. OUP Oxford, 2006.

[4] George Fishman. *Monte Carlo: concepts, algorithms, and applications*. Springer Science & Business Media, 2013.

[5] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.