

Some Historical Elements

The origins of the Monte Carlo method coincide with the origins of scientific computing and are closely associated with the computational needs of statistical physics. The most cited figure as the father of the Monte Carlo method is Metropolis[1]. Yet, the content of his foundational paper does not focus on the Monte Carlo method itself but on a new sampling technique. It makes sense to attribute the paternity of Monte Carlo to him, even if the method was already in place before his proposal, because this proposal changed everything in terms of the practicality of equilibrium statistical physics calculations, and it was the resulting dynamics that gave the Monte Carlo method all the resonance it subsequently achieved.

During the first twenty years of the method, everything progressed very rapidly, and one could even say that by the end of the 1960s, everything had been said ! Promises about parallelization, infinite dimensions, handling complex physics, and complex data were all formulated[2, 3, 4, 5, 6]. Today, we are revisiting them, almost identically.

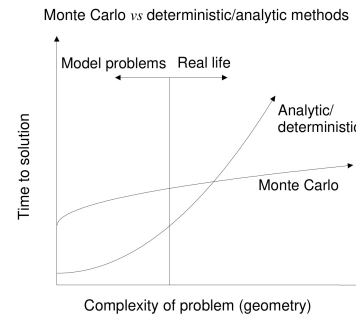


Figure 1: Time to solution using Monte Carlo vs. deterministic/analytic approaches.

FIGURE 1 – First figure [5]

However, one very specific field of application deserves special mention : that of linear kinetic physics. The main reason is that this physics is initially expressed in statistical terms (distributions of mean free paths, probability of collision types, etc.). It is then possible to use the Monte Carlo method without ever writing the slightest additional mathematical development : the code "reproduces" the physics. These are known as *analog algorithms*. Of course, it is also possible to make the effort to formally translate the algorithms in question. This is referred to as an *integral formulation*. This dual perspective on the same numerical physics has greatly contributed to stabilizing practices[3, 7, 8, 9, 10, 11].

In contrast, with the work of Feynman and Kac, followed by the Russian school that developed around Ermakov, an entire expertise emerged concerning the desire to start with physics that initially has nothing statistical and translate it into stochastic processes, then apply the Monte Carlo method in the spirit of linear kinetic physicists. As a work of duality, this research has a scope that far exceeds the Monte Carlo method itself and is found in very diverse application fields, often far removed from physics[12, 13, 14, 15].

A major milestone was the arrival of computer graphics. Today, not a single synthetic image produced by the film industry is created without the Monte Carlo method. However, reaching this stage required a enormous community investment over nearly thirty years. Above all, it was about fulfilling the initial promise of handling complex geometries and infinite scale ratios ! To move from promise to effective practice, many concepts—both geometric and statistical—had to be added, which we now inherit for all engineering, entirely independent of the goal of producing images[16, 17, 18, 19].

More recently, another essential research axis for engineering has been opened simultaneously in several application contexts (statistical physics, finance, energy physics, biology, etc.) : the management of non-linearities. These are mainly of three types : i) when a phenomenon is governed by a non-linear differential or integral equation ;

ii) when multiple phenomena interact in a coupled manner and their coupling is non-linear ; iii) when seeking to understand, even for purely linear phenomenology, how an observable depends on the problem's parameters (sensitivity calculation) and this dependence is non-linear[20, 21, 22, 23].

These last two developments (complexity of geometric data and non-linearity) mean that the method is both highly mature and continually shaken by a renewal of application perspectives, in academic fields as well as in the various industrial contexts where it is implemented.

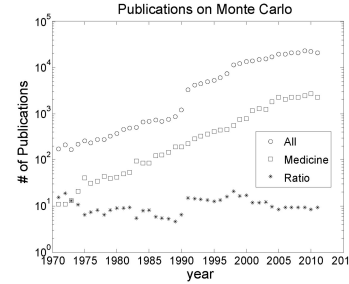


Figure 2: The number of papers published per year garnered from the Web of Knowledge ('All') and MedLine ('Medicine').

FIGURE 2 – Second figure [5]

Evaluate an expected value

$$\mathcal{A} = \mathbb{E}(W) \quad (1)$$

where

$$\mathbb{E}(W) = \int p_W(w) dw \quad (2)$$

W is the *weight*. We use n realizations of the weight $w_1, w_2, \dots, w_n$ to estimate $\mathcal{A}$ as

$$\mathcal{A} \approx m \quad (3)$$

with

$$m = \frac{1}{n} \sum_{i=1}^n w_i \quad (4)$$

m is an *estimate* of $\mathcal{A}$, and we associate an *estimation uncertainty* s with this estimate :

$$s = \frac{1}{\sqrt{n-1}} \sqrt{\left(\frac{1}{n} \sum_{i=1}^n w_i^2 \right) - m^2} \quad (5)$$

Commentaire 1 - The estimator

From a probabilistic perspective, we introduce n random variables $W_1, W_2, \dots, W_n$ (the W are uppercase), which are independent and identically distributed (IID) like the weight W . We then define the *estimator* M as

$$M = \frac{1}{n} \sum_{i=1}^n W_i \quad (1)$$

In equation 3, the estimate m is then seen as a *single realization* of the estimator M . It is easy to prove that $\mathbb{E}(M) = \mathbb{E}(W)$ and therefore that

$$\mathcal{A} = \mathbb{E}(M) \quad (2)$$

Since the expectation of the estimator is exactly equal to the desired quantity $\mathcal{A}$, we say that *the estimator is unbiased*.

A Monte Carlo-type algorithm therefore always has the structure indicated in the pseudo-algorithm 1. The challenge thus reduces to defining the random variable W and constructing an algorithm to sample it.

Algorithm 1 : Common part of all Monte Carlo algorithms

```

1  $sum = 0;$ 
2  $sumOfSquares = 0;$ 
3 pour  $chaque\ i \in \{1 : n\}$  faire
4   | Sample  $w$  from  $W$ ;
5   |  $sum = sum + w$ ;
6   |  $sumOfSquares = sumOfSquares + w^2$ ;
7 fin
8  $m = \frac{sum}{n};$ 
9  $s = \frac{1}{\sqrt{n-1}} \sqrt{\frac{sumOfSquares}{n} - m^2};$ 

```

Similarly, we define the random variable S as follows (as in equation 5 but with uppercase letters) :

$$S = \frac{1}{\sqrt{n-1}} \sqrt{\left(\frac{1}{n} \sum_{i=1}^n W_i^2 \right) - M^2} \quad (3)$$

and we show that

$$\mathbb{E}(S^2) = \sigma^2(M) \quad (4)$$

and thus S^2 is an unbiased estimator of the variance of the estimator. In equation 5, the square of the estimation uncertainty s is therefore seen as a single realization of the estimator of the variance of the estimator. Thus, it is s^2 that is interpreted with the confidence that can be given to an unbiased estimator, and not s (which is only a biased estimator of the standard deviation of the estimator).

Integral Formulation : A Common Language for Physics, Probability, and Algorithms

Our starting point is that when seeking to estimate a quantity $\mathcal{A}$ using the Monte Carlo method, we need a random variable W , defined on a universe $\mathcal{U}_W$ with probability density p_W , such that $\mathcal{A}$ is equal to the expectation of W , i.e.,

$$\mathcal{A} = \int_{\mathcal{U}_W} p_W(w) dw \quad (6)$$

All the subtleties of the method lie in how W is defined and practically sampled using a computer. It is worth noting that W will almost always be defined using intermediate random variables, which are themselves well-known and easy to sample, such as the two random variables X and Y in the following example. There will thus be a sampling algorithm for W that involves sampling X and Y . In a heat transfer context, X might typically be related to a conductive path within a convex solid that can only exchange with the outside through radiation. For example, X could represent the position at which the conductive path reaches the surface of the solid, and Y could represent the radiative path starting from the surface of the solid at X , with the space of radiative paths depending on the endpoint of the conductive path, making Y dependent on X . With such an example, it is immediately clear that the algorithms we will discuss can be highly elaborate, both in terms of phenomenology and in terms of their relationship to data (in this case, primarily geometric data).

It is therefore essential to ensure the rigor of probabilistic choices at every step, even when the work leads us into advanced algorithmic issues, such as accelerating data access and parallelism. Fortunately, this guarantee of rigor is quite simply achieved through the integral formulation.

Let us revisit our example with two intermediate random variables. We typically write

$$\mathcal{A} = \int_{\mathcal{U}_X} p_X(x) dx \int_{\mathcal{U}_Y(x)} p_Y(y; x) dy \tilde{w}(x, y) \quad (7)$$

We thus establish an equality between the studied quantity $\mathcal{A}$ and an integral. Regardless of the origin of this proposal, since X and Y are well-defined, $\mathcal{U}_X$, $\mathcal{U}_Y$, p_X , and p_Y are also well-defined, and we can perform the mathematical verification of the proposed equality. This is then a task of integral formulation that starts from what is known about $\mathcal{A}$ and is entirely independent of the probabilistic meaning assigned to equation 7. This is the first utility of integral formulations : *they allow rigorous justification of probabilization choices*. Even when these choices are derived from analog reasoning, which is highly intuitive (as we described in the introduction regarding physical kinetics), if we make the effort to establish integral formulations, we can always verify a posteriori that the proposed approach is physically rigorous.

The second utility of an integral formulation is that it replaces the algorithm itself : *it reads like an algorithm*. With a bit of practice, one recognizes in equation 7 that $\mathcal{A}$ is the expectation of a random variable W . However, the convention that a random variable is denoted by an uppercase letter is not directly used here for this main random variable : the letter W does not appear. Instead, the letters X and Y appear and do signify two random variables, whose universes $\mathcal{U}_X$ and $\mathcal{U}_Y$ are identified at the integration signs and define W as $W = \tilde{w}(X, Y)$, i.e., as a function of the two random variables X and Y .

The notation $\tilde{w}(X, Y)$ does not appear because the lowercase letters x and y are used in the function $\tilde{w}$. However, these lowercase letters carry meaning (even though they are mute from the perspective of strict mathematical grammar). This meaning comes from the presence of the terms $p_X(x) dx$ and $p_Y(y; x) dy$, which refer to the probability measures of X and Y .

Going a step further, we see that X and Y are correlated (p_Y depends on x) and that the universe of Y depends on X ($\mathcal{U}_Y$ is a function of x). Then, the integral over $\mathcal{U}_Y$ means that the second part of the expression is the expectation of $\tilde{w}(x, Y)$, i.e.,

$$\int_{\mathcal{U}_Y(x)} p_Y(y; x) dy \tilde{w}(x, y) = \mathbb{E}(\tilde{w}(x, Y)) \quad (8)$$

Here, only Y is random because the lowercase letter x is just a variable of integration. But in the complete expression of $\mathcal{A}$, the integration over $\mathcal{U}_X$ signifies a second expectation, and we read

$$\mathcal{A} = \mathbb{E}(\mathbb{E}(\tilde{w}(X, Y))) \quad (9)$$

where X is indeed an uppercase letter and signifies the random variable. Finally, since the expectation of an expectation is an expectation, we have

$$\mathcal{A} = \mathbb{E}(\tilde{w}(X, Y)) \quad (10)$$

and thus

$$\mathcal{A} = \mathbb{E}(W) \quad (11)$$

Therefore, equation 7 directly implies equation 11, which refers to algorithm 1, with

$$W = \tilde{w}(X, Y) \quad (12)$$

meaning that sampling W involves sampling X and Y , then applying the function $\tilde{w}$. We also know that sampling X precedes that of Y because both the universe and the probability density of Y depend on X . Thus, we directly read (after practice) algorithm 2

Importance Sampling and Zero Variance

In the preceding discussion :

- We asserted that $\mathcal{A}$ exists, with a definition derived, for example, from a physical model ;
- We acknowledged that it is also possible to express $\mathcal{A}$ as the expectation of a random variable W ;

Algorithm 2 : Monte Carlo algorithm corresponding to equation 7

```

1  $sum = 0;$ 
2  $sumOfSquares = 0;$ 
3 pour chaque  $i \in \{1 : n\}$  faire
4   Sample  $x$  from  $X$ ;
5   Sample  $y$  from  $Y$  given  $X = x$ ;
6   Compute  $w = \tilde{w}(x, y)$ ;
7    $sum = sum + w$ ;
8    $sumOfSquares = sumOfSquares + w^2$ ;
9 fin
10  $m = \frac{sum}{n}$ ;
11  $s = \frac{1}{\sqrt{n}} \sqrt{\frac{sumOfSquares}{n} - m^2}$ ;
```

— We presented the integral formulation as a way to describe the algorithm and verify that the statistical formulation is rigorously compatible with the initial definition of $\mathcal{A}$.

Let us consider the simplest implementation example : $\mathcal{A}$ is initially defined as the integral of a function f over a domain $\mathcal{D} \subseteq \mathbb{R}$.

$$\mathcal{A} = \int_{\mathcal{D}} f(x) dx \quad (13)$$

We can then choose any random variable X on $\mathcal{U}_X = \mathcal{D}$ with a strictly positive probability density p_X and perform the following transformation :

$$\mathcal{A} = \int_{\mathcal{D}} f(x) dx = \int_{\mathcal{U}_X} p_X(x) dx \frac{f(x)}{p_X(x)} \quad (14)$$

We then define the function $\tilde{w}$ such that

$$\tilde{w}(x) = \frac{f(x)}{p_X(x)} \quad (15)$$

to obtain

$$\mathcal{A} = \int_{\mathcal{U}_X} p_X(x) dx \tilde{w}(x) = \mathbb{E}(\tilde{w}(X)) \quad (16)$$

and by defining the random variable W as $W = \tilde{w}(X)$, we indeed arrive at

$$\mathcal{A} = \mathbb{E}(W) \quad (17)$$

Importance Sampling and Zero Variance

The idea of *importance sampling* stems from the observation that we can influence the choice of X , and thus the choice of the probability density p_X used during sampling, to minimize the variance of W . Indeed, we have seen that the uncertainty in the Monte Carlo calculation is related to the variance of the weights : if all realizations of the Monte Carlo weight led to exactly the same value, the variance of the estimator would be zero ($s = 0$), and the calculation would thus be infinitely precise.

Now, $W = \tilde{w}(X)$. We therefore seek to reduce the variations of the function $\tilde{w}(x) = \frac{f(x)}{p_X(x)}$. To do this, whenever possible, we choose p_X such that it reflects the variations of the function f : if p_X exactly follows f , then $\tilde{w} = \frac{f}{p_X}$ is a constant, and W is a Dirac delta. Regardless of the realization x of X , the corresponding realization $w = \tilde{w}(x)$ of W always takes the same value, and W indeed has zero variance.

We refer to a *zero-variance algorithm* when this ideal limit is achieved. Of course, this is never the case in practical implementation, but in our efforts to optimize convergence, we are always guided by this objective and invest significant effort in the detailed description of these limiting algorithms, even if they are not practical.

Let us therefore take the time to describe a zero-variance algorithm for our example, indexing all corresponding ideal sampling choices with the letter I . For all x , we must have $\frac{f(x)}{p_{X,I}(x)} = K$, where K is a constant, so $p_{X,I}(x) = \frac{1}{K}f(x)$. Additionally, $p_{X,I}$ is a probability density, so it is normalized : $\int_{\mathcal{U}_X} p_{X,I}(x) dx = 1$. This leads to

$$\int_{\mathcal{U}_X} \frac{1}{K} f(x) dx = \frac{1}{K} \int_{\mathcal{U}_X} f(x) dx = \frac{1}{K} \mathcal{A} = 1 \quad (18)$$

hence

$$K = \mathcal{A} \quad (19)$$

$$p_{X,I}(x) = \frac{f(x)}{\mathcal{A}} \quad (20)$$

and

$$W_I = \tilde{w}_I(X) = \frac{f(X)}{p_{X,I}(X)} = \mathcal{A} \quad (21)$$

The ideal sampling probability density is therefore the function f to be integrated, divided by the solution $\mathcal{A}$ of the integral, and the Monte Carlo weight is always strictly equal to the desired solution. The algorithm thus returns the exact solution to the problem from the first realization.

It is clear that this algorithm can only be implemented in practice if the solution is already known. In a typical scientific computing situation, since the solution is unknown, this is not feasible. However, importance sampling can be guided by this limit in the sense that we seek to use an

Commentaire 2 - Does a zero-variance algorithm always exist ?

Regardless of its practicality, can we always refer to a zero-variance algorithm as an ideal for convergence, or are there cases where no zero-variance algorithm can be considered? By setting $p_{X,I}(x) = \frac{f(x)}{\mathcal{A}}$, we assumed that the conditions were met for $p_{X,I}$ to be a valid probability density. Integrability and normalization are ensured, but $f(x)$ is not necessarily positive for all x .

If f is everywhere negative, this is not a problem because we can take $p_{X,I}(x) = -\frac{f(x)}{\mathcal{A}}$ and reach the same conclusions. However, if f changes sign, this trick is insufficient. We must then perform an integral reformulation before considering a zero-variance algorithm. The answer to our question is therefore negative : without changing the form of the integral, it is often impossible to define a zero-variance algorithm.

However, the necessary integral reformulation can be very simple. Suppose, for example, that f is sometimes negative but is bounded below by a known value $f_{\min}$ and that the integration domain $\mathcal{D}$ has a finite measure $\mu_{\mathcal{D}}$. We can then write

$$\mathcal{A} = \int_{\mathcal{D}} f(x) dx = \int_{\mathcal{D}} (f(x) - f_{\min}) dx + \mu_{\mathcal{D}} f_{\min} \quad (18)$$

The problem then reduces to estimating a new integral $\mathcal{A}' = \int_{\mathcal{D}} f'(x) dx$ of the function $f' = f - f_{\min}$, which is now always positive, and we can define a zero-variance algorithm for $\mathcal{A}'$.

Commentaire 3 - Optimizing an algorithm with multiple variables

Let us revisit the example of equation 7 to address the question of importance sampling in the case of multiple sampling, here sampling X and then sampling Y given the rea-

optimized sampling distribution, $p_{X,\text{opt}}$, that is as close as possible to $p_{X,I}$. For example, starting from a non-optimized algorithm, we can write

$$\mathcal{A} = \int_{\mathcal{U}_X} p_X(x) dx \tilde{w}(x) = \int_{\mathcal{U}_X} p_{X,\text{opt}}(x) dx \frac{p_X(x)}{p_{X,\text{opt}}(x)} \tilde{w}(x) = \int_{\mathcal{U}_X} p_{X,\text{opt}}(x) dx \tilde{w}_{\text{opt}}(x) \quad (22)$$

with

$$\tilde{w}_{\text{opt}}(x) = \frac{p_X(x)}{p_{X,\text{opt}}(x)} \tilde{w}(x) \quad (23)$$

Starting from a given Monte Carlo algorithm, the practice is therefore to seek to improve its convergence (reduce the uncertainty of the estimator) by adjusting the sampling distribution and modifying the Monte Carlo weight so that the solution remains exact for an infinite number of realizations (so that the estimator remains unbiased). This modification is nothing other than multiplying the old weight by the ratio of the sampling probability densities.

lization of X :

$$\mathcal{A} = \int_{\mathcal{U}_X} p_X(x) dx \int_{\mathcal{U}_Y(x)} p_Y(y; x) dy \tilde{w}(x, y) \quad (19)$$

The only idea to remember is that importance sampling is applied to each integral as if it were unique, as we described for equation 13. Here, for the random variable X , the function of x that we must try to follow as closely as possible is $f(x) = p_X(x) \int_{\mathcal{U}_Y(x)} p_Y(y; x) dy \tilde{w}(x, y)$. The ideal probability density is therefore

$$p_{X,I}(x) = \frac{f(x)}{\int_{\mathcal{U}_X} f(x) dx} = \frac{1}{\mathcal{A}} p_X(x) \int_{\mathcal{U}_Y(x)} p_Y(y; x) dy \tilde{w}(x, y) \quad (20)$$

To optimize the sampling of x , we must therefore have, for every value of x , a good approximation of the integral of the second variable, i.e., of $\int_{\mathcal{U}_Y(x)} p_Y(y; x) dy \tilde{w}(x, y)$.

Next, to optimize the sampling of y , we return to the single-variable case, but it is important to note that we must be able to approximate

$$p_{Y,I}(y; x) = \frac{p_Y(y; x) \tilde{w}(x, y)}{\int_{\mathcal{U}_Y(x)} p_Y(y; x) dy \tilde{w}(x, y)} \quad (21)$$

for every value of x , which can sometimes be very difficult to achieve.

Integral Reformulation

From simple examples like those in this introduction, it is difficult to convey the essence of the practice in terms of improving convergence. This essence is *integral reformulation*. Importance sampling and zero-variance algorithms come into play only later, after we have established that the chosen integral form is the most suitable for a Monte Carlo calculation.

It is challenging to describe the criteria that justify this judgment ("the chosen formulation is the right one") because they stem from practical experience, which varies across application domains and is heavily reliant on the available physical insights. Here, we provide an example from our practice, first presenting it from a purely mathematical perspective and then briefly explaining how the chosen strategy naturally emerged from the physics of radiation.

We aim to estimate

$$\mathcal{A} = \mathcal{A}_1 - \mathcal{A}_2 \quad (24)$$

with

$$\mathcal{A}_1 = \int_{-a}^0 k f(x_1) e^{+kx_1} (1 - e^{-kb}) dx_1 \quad (25)$$

and

$$\mathcal{A}_2 = \int_0^b k f(x_2) e^{-kx_2} (1 - e^{-ka}) dx_2 \quad (26)$$

Initially, $\mathcal{A}_1$ and $\mathcal{A}_2$ were treated as separate integrals, each estimated independently, and their difference was then considered. However, it quickly became apparent that convergence issues arose when the function f varied slightly around a reference value f_{ref} over the interval $[-a, b]$. Indeed, both integrals take values close to $f_{\text{ref}}(1 - e^{-ka})(1 - e^{-kb})$, and thus each must be estimated very precisely before their difference can be taken; otherwise, only the uncertainties of the two Monte Carlo estimates are observed.

However, we can observe that

$$1 - e^{-kb} = \int_0^b k e^{-kx_2} dx_2 \quad (27)$$

and

$$1 - e^{-ka} = \int_{-a}^0 k e^{+kx_1} dx_1 \quad (28)$$

This suggests expressing both $\mathcal{A}_1$ and $\mathcal{A}_2$ as integrals over the same double domain :

$$\mathcal{A}_1 = \int_{-a}^0 dx_1 \int_0^b dx_2 k e^{+kx_1} k e^{-kx_2} f(x_1) \quad (29)$$

and

$$\mathcal{A}_2 = \int_{-a}^0 dx_1 \int_0^b dx_2 k e^{+kx_1} k e^{-kx_2} f(x_2) \quad (30)$$

We can then express $\mathcal{A}$ as a single integral :

$$\mathcal{A} = \int_{-a}^0 dx_1 \int_0^b dx_2 k e^{+kx_1} k e^{-kx_2} (f(x_1) - f(x_2)) \quad (31)$$

With this new formulation, we can attempt to estimate $\mathcal{A}$ directly using a single Monte Carlo algorithm, typically based on importance sampling that disregards the difference $f(x_1) - f(x_2)$: we seek only to reflect the variations of e^{+kx_1} and e^{-kx_2} as functions of x_1 and x_2 , respectively. This leads to

$$p_{X_1}(x_1) = \frac{e^{+kx_1}}{\int_{-a}^0 e^{+kx_1} dx_1} = \frac{k e^{+kx_1}}{1 - e^{-ka}} \quad (32)$$

and

$$p_{X_2}(x_2) = \frac{e^{-kx_2}}{\int_0^b e^{-kx_2} dx_2} = \frac{k e^{-kx_2}}{1 - e^{-kb}} \quad (33)$$

Substituting these two probability densities, we obtain the following integral formulation, in which the convergence difficulties related to f having only small variations around a reference value f_{ref} have disappeared. Indeed, the algorithm directly samples the difference $f(x_1) - f(x_2)$, and if this difference tends toward zero, the solution tends toward zero but is still well estimated because it is no longer constructed as the difference between two independent estimators.

$$\mathcal{A} = \int_{-a}^0 p_{X_1}(x_1) dx_1 \int_0^b p_{X_2}(x_2) dx_2 \frac{ke^{+kx_1}}{p_{X_1}(x_1)} \frac{ke^{-kx_2}}{p_{X_2}(x_2)} (f(x_1) - f(x_2)) = \int_{-a}^0 p_{X_1}(x_1) dx_1 \int_0^b p_{X_2}(x_2) dx_2 (1 - e^{-ka})(1 - e^{-kb})(f(x_1) - f(x_2)) \quad (34)$$

or

$$\mathcal{A} = \int_{-a}^0 p_{X_1}(x_1) dx_1 \int_0^b p_{X_2}(x_2) dx_2 \tilde{w}(x_1, x_2) \quad (35)$$

with

$$\tilde{w}(x_1, x_2) = (1 - e^{-ka})(1 - e^{-kb})(f(x_1) - f(x_2)) \quad (36)$$

and thus

$$W = (1 - e^{-ka})(1 - e^{-kb})(f(X_1) - f(X_2)) \quad (37)$$

This example is a simplified version of a problem where we seek to evaluate the net power exchanged between two adjacent layers of thicknesses a and b , composed of the same semi-transparent medium with an absorption coefficient k . The function f represents the Planck radiance, which depends on the temperature at each point. $\mathcal{A}_1$ is the power of the radiation emitted by the layer of thickness a and absorbed by the layer of thickness b . Conversely, $\mathcal{A}_2$ is the power of the radiation emitted by the layer of thickness b and absorbed by the layer of thickness a . In the first strategy, these two powers are estimated separately. However, if the two layers are at the same temperature (i.e., $f(x)$ is the same for all x), the second law of thermodynamics tells us that these two powers are rigorously equal, and thus their difference, the net exchanged power $\mathcal{A} = \mathcal{A}_1 - \mathcal{A}_2$, is zero. It is from this physical insight that the reformulation proposed above was designed in practice.

Sensitivity Calculation

Suppose we have a Monte Carlo algorithm that estimates $\mathcal{A}(\alpha)$ as follows (see the section on integral formulation) :

$$\mathcal{A}(\alpha) = \int_{\mathcal{U}_X} p_X(x; \alpha) dx \tilde{w}(x; \alpha) \quad (38)$$

In this algorithm, the Monte Carlo weight w and the probability density p_X of the sampled random variable X both depend on a parameter α . The solution $\mathcal{A}$ therefore also depends on α . We can then aim to estimate the sensitivity of $\mathcal{A}$ to α in a linear sense, i.e., to estimate the derivative $\frac{\partial \mathcal{A}}{\partial \alpha}$.

Observing that the integration domain does not depend on α (otherwise the problem is much more complex), we can write

$$\frac{\partial \mathcal{A}}{\partial \alpha} = \int_{\mathcal{U}_X} dx \left(\frac{\partial p_X}{\partial \alpha} \tilde{w} + p_X \frac{\partial \tilde{w}}{\partial \alpha} \right) \quad (39)$$

In this form, we do not recognize a Monte Carlo algorithm. However, if we decide to use the same sampling distribution as in the initial algorithm, we obtain

$$\frac{\partial \mathcal{A}}{\partial \alpha} = \int_{\mathcal{U}_X} p_X(x; \alpha) dx \left(\frac{1}{p_X} \frac{\partial p_X}{\partial \alpha} \tilde{w} + \frac{\partial \tilde{w}}{\partial \alpha} \right) \quad (40)$$

Comparing equation 39 and equation 40, we see that the algorithms are identical in the sense that they both involve sampling the same random variable X , with the only difference being the Monte Carlo weight :

- For estimating $\mathcal{A}$, we use the weight w ;
- For estimating $\frac{\partial \mathcal{A}}{\partial \alpha}$, we use the weight $w_{sensib} = \frac{1}{p_X} \frac{\partial p_X}{\partial \alpha} \tilde{w} + \frac{\partial \tilde{w}}{\partial \alpha}$.

This property is used to develop algorithms that simultaneously compute the desired quantity $\mathcal{A}$ and its derivatives with respect to all the parameters of the problem. Since the majority of the computation time is often related to the sampling of random variables, and not to the calculation of the weights, these enriched algorithms can have a computational cost almost identical to that of the original algorithm. They provide not only an estimate of the value of $\mathcal{A}$ for a given set of parameters but also information on how $\mathcal{A}$ changes when the parameter values are modified around the chosen set.

Practical Work

Preparation

Log In

- User : etudiant
- Password : "ask for it"

Start the Graphical Server

```
etudiant@machine:~ $ startx
```

Then open a terminal using the combination of the Alt key and the Enter key :

```
[Alt] [Enter]
```

Create a Working Directory with Your Name

```
etudiant@machine:~ $ mkdir your_name
etudiant@machine:~ $ cd your_name
```

Visit the Manual Pages

```
etudiant@machine:~/your_name $ man edix_gestes_de_base
etudiant@machine:~/your_name $ man edix_premiers_pas
etudiant@machine:~/your_name $ man edix_interface_graphique
```

You exit a manual page with the key **q**. The manual page *edix_fenetres* describes the use of the windowing system (navigating among the 9 possible desktops, opening a console, switching from one console to another, etc.). The manual page *edix_gestes_de_base* describes the practical elements you will need for the practical works (shell commands, text editor, visualization tools, etc.).

Practical Work 1

Copy the First Code Example

```
etudiant@machine:~/your_name $ cp -r /home/lafrier/livre_informatique_graphique/Codes/quatre_plus_un_parallelise \
/home/etudiant/your_name/quatre_plus_un_parallelise
etudiant@machine:~/your_name $ cd quatre_plus_un_parallelise
```

You are now in the directory `/home/etudiant/your_name/quatre_plus_un_parallelise`, and you can, for example, use the `ls` command to list the contents of this directory :

```
etudiant@machine:~/your_name/quatre_plus_un_parallelise $ ls
```

Compile the Code and Run It The code we are going to examine proposes estimating the result $\mathcal{A}$ of the sum $4 + 1$ by performing a Bernoulli trial with probability 0.5 (a "heads or tails") and retaining either $4/0.5 = 8$ or $1/0.5 = 2$, then averaging a large number n of such realizations. The average m of a large number of 8s or 2s, with equiprobable outcomes, is indeed close to $\mathcal{A} = 4 + 1 = 5$, so $m \approx \mathcal{A}$ if n is large. The code also provides an estimate s of the statistical uncertainty. This uncertainty is the standard deviation of the estimator, and we know that the estimator is Gaussian. With a probability of 0.9973, we can therefore expect the result of the calculation to be compatible with the exact value 5 within plus or minus three times this uncertainty.

```
etudiant@machine:~/your_name/quatre_plus_un_parallelise $ . ~/lafrier/star-build/local/etc/stardis.profile
etudiant@machine:~/your_name/quatre_plus_un_parallelise $ sh compilation.sh
etudiant@machine:~/your_name/quatre_plus_un_parallelise $ ./a.out
```

The first of these three commands must be re-run each time you open a new shell session (e.g., a new console). It creates the environment variables needed to use the libraries developed by the edstar platform. The second command is the shell script `compilation.sh` that compiles the code (named `quatre_plus_un_parallelise.c`). Without seeking to understand the details of the compilation line, you can display the contents of this script with the `cat` command :

```
etudiant@machine:~/your_name/quatre_plus_un_parallelise $ cat compilation.sh
```

The third command executes the compiled code. You might, for example, get the following output :

```
m=4.995200 +/- s=0.030001
```

The symbol $\pm$ is usual but misleading. As announced above, we can only speak in terms of probabilities. The reasoning is as follows : with a probability of 0.9973, we expect the exact solution to be between $m - 3s$ and $m + 3s$, i.e., between $4.979000 - 3 \times 0.030001$ and $4.979000 + 3 \times 0.030001$.

Open the Code and Read It

```
etudiant@machine:~/your_name/quatre_plus_un_parallelise $ vim quatre_plus_un_parallelise.c
vim:
Move with the arrow keys
Exit with :q!
```

See the manual page *edix_gestes_de_base* for the basic elements of using the text editor *vim*. At this stage, the idea is simply to get an idea of the number of lines required to write such a Monte Carlo code *according to best practices*, i.e., with exception handling and ensuring the independence of random generations when using parallelization (using multiple processors for the same calculation). We will use this first code as a starting point for the following examples. You can already identify some structural elements :

— The declaration of global variables :

```
/* ===== */
/* Specific variables common to all processes      */
/* These variables must not be modified by the     */
/* processes launched in parallel                  */
/* ===== */

double proba = 0.5; // probability of the Bernoulli test
```

— The definition of all quantities that can be passed as input during code execution :

```

/*****
/* Standard input management */
/* Input 1: new_rng_serie (value 0 if not specified) */
/* Input 2: nrealisations (value 10000 if not specified) */
/* Input 3: nthreads (number of logical processors of the machine if */
/* not specified) */
*****/
and all the following tests...

```

— The definition of variables attached to each process (in this example, there are none) :

```

/* ===== */
/* Specific variables for each process */
/* These variables will take different values */
/* for each of the processes launched in parallel */
/* ===== */

```

— The sampling algorithm itself :

```

/* ===== */
/* Weight sampling algorithm */
/* ===== */

if(ssp_rng_canonical(rngs[ithread]) < proba) {
    w = 4./proba;
} else {
    w = 1./(1-proba);
}

/* ===== */
/* End of the weight sampling algorithm */
/* ===== */

```

For all the practical works considered, the rest of the code is generic, and we will not modify it. The parts of the code where we will write to code new Monte Carlo algorithms based on this example are therefore few : two areas for declaring variables (general or process-related), one area for defining inputs, and the sampling algorithm.

Run the Code with Different Parameters If you look at the definition of the inputs, you will see that it is possible to pass three input data. The first is either 0 or 1. If you set it to 0, the same series of random numbers is used each time the code is executed. You therefore get exactly the same result every time. If, on the other hand, you set it to 1, the series of random numbers is different each time the code is executed. For example, if you run the following command three times in a row :

```
etudiant@machine:~/your_name/quatre_plus_un_parallelise $ ./a.out 0
```

you will get the same result three times. If, on the other hand, you run the following command three times in a row :

```
etudiant@machine:~/your_name/quatre_plus_un_parallelise $ ./a.out 1
```

you will get three different simulation results.

The second possible input is the number of realizations. If you run the following two commands successively :

```
etudiant@machine:~/your_name/quatre_plus_un_parallelise $ ./a.out 0 10000
etudiant@machine:~/your_name/quatre_plus_un_parallelise $ ./a.out 0 1000000
```

you will notice that the statistical uncertainty is ten times lower in the second case than in the first. The reason is that we used a hundred times more realizations, and the statistical uncertainty evolves as $\frac{1}{\sqrt{n}}$, where n is the number of realizations. With a hundred times more realizations, we are therefore ten times more precise.

The third input is the number of processes launched in parallel. By default, as many processes as logical processors are used. If you run :

```
etudiant@machine:~/your_name/quatre_plus_un_parallelise $ ./a.out 0 10000 1
```

there will be only one process, and you will therefore not use parallelization.

Open the run.sh Shell Script, Comment on It, and Run It

```
etudiant@machine:~/your_name/quatre_plus_un_parallelise $ vim run.sh
etudiant@machine:~/your_name/quatre_plus_un_parallelise $ sh run.sh
```

This script runs the code five times. You therefore get five different estimates of $\mathcal{A}$, each with the same number of realizations.

Modify the Shell Script to Plot a Histogram of the Estimates of $\mathcal{A}$ Instead of five estimates, you can run a large number of different estimates and plot the statistical distribution of these estimates in the form of a histogram. To do this, you can use the statistical software *R*, which includes visualization solutions, or perform the statistics in a C program and visualize the results with the scientific data visualization software *gnuplot*.

The purpose of such a statistical study is to verify (at least visually) that the distribution of the estimates m (when re-running the Monte Carlo code) is indeed Gaussian and that its standard deviation is indeed the one predicted by s (the estimate of uncertainty). This property justifies the fact that in the following, we will never run multiple Monte Carlo codes with the same parameters : knowing the values of m and s obtained from a single calculation, we already have all the useful information about the distribution of the estimates (the one we would obtain by repeating the simulation). Repeating a Monte Carlo calculation is no more meaningful than conducting several identical surveys in parallel : if you can access multiple samples, you combine them to work on a larger sample, thus accessing more precise statistical conclusions. We never re-run a Monte Carlo calculation to see how the result fluctuates because we know how it will fluctuate. We always work with *a single estimate*. However, when necessary, we do not hesitate to re-run a Monte Carlo calculation with more realizations to access a more precise estimate.

That said, plotting at least once the distribution of the estimates in the form of a histogram, seeing that it is Gaussian, and seeing that s predicts its width, is essential in a training phase for the method. Later in practice, we never do this again, but the image of this Gaussian distribution is at the heart of all reasoning. You can try to plot this histogram your way, or by drawing inspiration from the example provided in the solution.

Solution for Practical Work 1 In the directory

```
~/your_name/quatre_plus_un_parallelise/histogramme
```

you will find three scripts :

- `run.sh`, which is equivalent to the `run.sh` script we just looked at. The difference is that the number of draws is higher, and the calculation results are archived in the directory `/tmp/hist.dat`.
- `hist.R`, which is an R script that plots a histogram from the contents of the file `/tmp/hist.dat`. The figure containing the histogram is `/tmp/hist.png`.
- `hist.sh`, which is a shell script that runs the R script and visualizes the file `/tmp/hist.png`.

You can test these scripts with the following commands :

```
etudiant@machine:~/your_name/quatre_plus_un_parallelise $ cd histogramme
etudiant@machine:~/your_name/quatre_plus_un_parallelise/histogramme $ sh run.sh
etudiant@machine:~/your_name/quatre_plus_un_parallelise/histogramme $ sh hist.sh
```

You will not yet get the histogram you expect because the number of draws is $n = 2$. The possible estimates are therefore $m = 2$ (if both draws each led to the value 2), $m = 8$ (if both draws each led to the value 8), or $m = 5$ (if one draw led to the value 2 and the other to 8, which is achieved with double probability). There are therefore only three bars in the histogram that are populated. However, by modifying the contents of the `run.sh` script, you can increase the value of n (e.g., by replacing `./a.out 1 2` with `./a.out 1 100`) and approach the expected Gaussian. You can then adjust the different parameters and compare the width of the Gaussian to the value of s obtained by running a single calculation.

Practical Work 2

Starting from the previous code, code a new Monte Carlo algorithm that estimates the following integral :

$$\mathcal{A}(p, q) = \int_0^1 dx \int_0^x dy x^p y^q \quad (41)$$

For x and y , use a uniform sampling distribution.

$\mathcal{A}$ is known analytically, and you can therefore verify that your algorithm obtains, within statistical uncertainty,

$$\mathcal{A}(p, q) = \frac{1}{q+1} \frac{1}{p+q+2} \quad (42)$$

Remark In such an example, it is also possible to analytically predict the value of the variance of the estimator. Indeed, we can formally calculate the expectation of the square of the random variable W , and since the expectation of W is known (as it is equal to $\mathcal{A}$), we can calculate the expectation of W^2 minus the square of $\mathcal{A}$ to find

$$\sigma^2(W) = \frac{1}{2q+1} \frac{1}{2p+2q+3} - \frac{1}{(q+1)^2} \frac{1}{(p+q+2)^2} \quad (43)$$

Since $\sigma^2(M) = \frac{1}{n} \sigma^2(W)$, you can thus predict the values of the statistical uncertainty s produced at each execution of the code.

Solution for Practical Work 2 A code corresponding to Practical Work 2 is proposed in the directory `/home/lafrier/livre_informatique_graphique/Codes/somme_xp_yq`

Practical Work 3

Revisit Practical Work 2 with a zero-variance algorithm. It is sufficient to provide the algorithm. There is no need to code it, as this will be done in Practical Work 5.

Practical Work 4

Estimate the following integral :

$$\mathcal{A}(p, q, L, \alpha) = \int_0^1 dx \int_0^x dy x^p y^q \left(1 + \alpha \cos \left(2\pi \frac{y}{L} \right) \right) \quad (44)$$

For x and y , use a uniform sampling distribution. This practical work is very similar to Practical Work 2. It is a step toward the following practical works. For $\alpha = 0$, you should, in particular, retrieve the results of Practical Work 2.

Solution for Practical Work 4 A code corresponding to Practical Work 4 is proposed in the directory `/home/lafrier/livre_informatique_graphique/Codes/somme_xp_yq_co`

Practical Work 5

Revisit Practical Work 4 using the sampling distributions from Practical Work 3. For $\alpha = 0$, you will be exactly in the conditions of Practical Work 3. The algorithm should therefore be zero-variance. For $\alpha \neq 0$, this will no longer be the case. You can attempt to analyze how the estimates s of the statistical uncertainty depend on p , q , and α . When the difficulty of integration is primarily linked to $x^p y^q$, the sampling distributions from Practical Work 3 are well-suited. When, on the other hand, $\cos \left(2\pi \frac{y}{L} \right)$ is the source of the integration difficulty, these sampling distributions may not be suitable, and the statistical uncertainty will be higher. One of the difficulties will be deciding whether you want to reason in terms of absolute uncertainty, i.e., s , or relative uncertainty, i.e., $\frac{s}{m}$.

Practical Work 6

Starting from Practical Work 4, design and code an algorithm that estimates the derivative $\frac{\partial \mathcal{A}}{\partial \alpha}$.

Solution for Practical Work 6 A code corresponding to Practical Work 6 is proposed in the directory `/home/lafrier/livre_informatique_graphique/Codes/sensib_somme_x`

Références

- [1] Nicholas METROPOLIS et al. « Equation of state calculations by fast computing machines ». In : *The journal of chemical physics* 21.6 (1953), p. 1087-1092.
- [2] John Michael HAMMERSLEY, David Christopher HANDSCOMB et Françoise ROSTAND. *Les méthodes de Monte-Carlo*. Dunod, 1967.
- [3] Malvin H KALOS et Paula A WHITLOCK. *Monte carlo methods*. John Wiley & Sons, 2009.
- [4] Michael EVANS et Timothy SWARTZ. *Approximating integrals via Monte Carlo and deterministic methods*. T. 20. OUP Oxford, 2000.
- [5] Alex F BIELAJEW. « History of monte carlo ». In : *Monte Carlo techniques in radiation therapy*. CRC Press, 2021, p. 3-15.
- [6] Jun S LIU et Jun S LIU. *Monte Carlo strategies in scientific computing*. T. 10. Springer, 2001.
- [7] Alireza HAGHIGHAT. *Monte Carlo methods for particle transport*. Crc Press, 2020.
- [8] Oleg N VASSILIEV. *Monte Carlo Methods for Radiation Transport Fundamentals and Advanced Topics*. Springer.
- [9] William L DUNN et J Kenneth SHULTIS. *Exploring monte carlo methods*. Elsevier, 2022.
- [10] Dirk P KROESE, Thomas TAIMRE et Zdravko I BOTEV. *Handbook of monte carlo methods*. John Wiley & Sons, 2013.
- [11] Nick T THOMOPOULOS. *Essentials of Monte Carlo simulation : Statistical methods for building simulation models*. Springer Science & Business Media, 2012.
- [12] Bernard LAPEYRE, Étienne PARDOUX et Rémi SENTIS. *Méthodes de Monte-Carlo pour les équations de transport et de diffusion*. Springer, 1998.
- [13] Ivan T DIMOV. *Monte Carlo methods for applied scientists*. World Scientific, 2008.
- [14] Sergei Mikhailovich ERMAKOV, Vladimir Viktorovich NEKRUTKIN et Aleksandr Stepanovich SIPIN. *Random processes for classical equations of mathematical physics*. T. 34. Springer Science & Business Media, 2013.
- [15] Mervin E MULLER. « Some continuous Monte Carlo methods for the Dirichlet problem ». In : *The Annals of Mathematical Statistics* (1956), p. 569-589.
- [16] Eric VEACH. *Robust Monte Carlo methods for light transport simulation*. Stanford University, 1998.
- [17] Henrik Wann JENSEN et al. « Monte Carlo ray tracing ». In : *ACM SIGGRAPH*. T. 5. 2003, p. 340769537.
- [18] Matt PHARR, Wenzel JAKOB et Greg HUMPHREYS. *Physically based rendering : From theory to implementation*. MIT Press, 2023.
- [19] Wenzel Alban JAKOB. *Light transport on path-space manifolds*. Cornell University, 2013.
- [20] Ivan DIMOV et Todor GUROV. « Monte Carlo algorithm for solving integral equations with polynomial non-linearity. Parallel implementation ». In : *Pliska Studia Mathematica Bulgarica* 13.1 (2000).
- [21] Emmanuel GOBET. *Monte-Carlo methods and stochastic processes : from linear to non-linear*. Chapman et Hall/CRC, 2016.
- [22] Jérémie DAUCHET et al. « Addressing nonlinearities in monte carlo ». In : *Scientific reports* 8.1 (2018), p. 13302.
- [23] Guillaume TERRÉE et al. « Addressing the gas kinetics Boltzmann equation with branching-path statistics ». In : *Physical Review E* 105.2 (2022), p. 025305.

