

## Chapter 1

# Introduction

Monte Carlo methods are a powerful tool in many fields of mathematics, physics and engineering. It is known that the algorithms based on this method give statistical estimates for any linear functional of the solution by performing random sampling of a certain random variable (r.v.) whose mathematical expectation is the desired functional.

In [Binder and Heermann (1998); Bremaud (1999); Berg (2004); Chen and Shao (2000); Dubi (2000); Ermakov and Mikhailov (1982); Evans and Swartz (2000); Fishman (1995); MacKeown (1997); Gilks *et al.* (1995); Gould and Tobochnik (1989); Hellekalek and Larcher (1998); Jacoboni and Lugli (1989); Landau and Binder (2000); Liu (2001); Manly (1997); Manno (1999); Mikhailov and Sabelfeld (1992); Newman and Barkema (1999); Rubinstein (1992); Robert and Casella (2004); Sabelfeld (1991); Shreider (1966); Sloan and Joe (1994); Sobol (1973)] one can find various definitions and different understanding of what Monte Carlo methods are. Nevertheless, in all above mentioned works there is a common understanding that it's a method which uses r.v. or random processes to find an approximate solution to the problem. We use the following definition.

**Definition 1.1.** Monte Carlo methods are methods of approximation of the solution to problems of computational mathematics, by using random processes for each such problem, with the parameters of the process equal to the solution of the problem. The method can guarantee that the error of Monte Carlo approximation is smaller than a given value with a certain probability.

So, Monte Carlo methods always produce an approximation of the solution, but one can control the accuracy of this solution in terms of the probability error. The Las Vegas method is a randomized method which also uses

r.v. or random processes, but it always produces the correct result (not an approximation). A typical example is the well-known Quicksort method [Hoare (1961); Knuth (1997)].

Usually Monte Carlo methods reduce problems to the approximate calculation of mathematical expectations. Let us introduce some notations used in the book: the mathematical expectation of the r.v.  $\xi$  or  $\theta$  is denoted by  $E_\mu(\xi)$ ,  $E_\mu(\theta)$  (sometimes abbreviated to  $E\xi$ ,  $E\theta$ ); the variance by  $D(\xi)$ ,  $D(\theta)$  (or  $D\xi$ ,  $D\theta$ ) and the standard deviation by  $\sigma(\xi)$ ,  $\sigma(\theta)$  (or  $\sigma\xi$ ,  $\sigma\theta$ ). We shall let  $\gamma$  denote the random number, that is a uniformly distributed r.v. in  $[0, 1]$  with  $E(\gamma) = 1/2$  and  $D(\gamma) = 1/12$ . We shall further denote the values of the random point  $\xi$  or  $\theta$  by  $\xi_i$ ,  $\theta_i$  ( $i = 1, 2, \dots, N$ ) respectively. If  $\xi_i$  is a  $d$ -dimensional random point, then usually it is constructed using  $d$  random numbers  $\gamma$ , i.e.,  $\xi_i \equiv (\gamma_i^{(1)}, \dots, \gamma_i^{(d)})$ . The density (frequency) function will be denoted by  $p(x)$ . Let the variable  $J$  be the desired solution of the problem or some desired linear functional of the solution. A r.v.  $\xi$  with mathematical expectation equal to  $J$  must be constructed:  $E\xi = J$ . Using  $N$  independent values (realizations) of  $\xi$ :  $\xi_1, \xi_2, \dots, \xi_N$ , an approximation  $\bar{\xi}_N$  to  $J$ :  $J \approx \frac{1}{N}(\xi_1 + \dots + \xi_N) = \bar{\xi}_N$ , can then be computed. The following definition of the probability error can be given:

**Definition 1.2.** If  $J$  is the exact solution of the problem, then the probability error is the least possible real number  $R_N$ , for which:

$$P = Pr \{ |\bar{\xi}_N - J| \leq R_N \}, \quad (1.1)$$

where  $0 < P < 1$ . If  $P = 1/2$ , then the probability error is called the *probable error*.

The probable error is the value  $r_N$  for which:

$$Pr \{ |\bar{\xi}_N - J| \leq r_N \} = \frac{1}{2} = Pr \{ |\bar{\xi}_N - J| \geq r_N \}.$$

The computational problem in Monte Carlo algorithms becomes one of calculating repeated realizations of the r.v.  $\xi$  and of combining them into an appropriate statistical estimator of the functional  $J(u)$  or solution. One can consider Las Vegas algorithms as a class of Monte Carlo algorithms if one allows  $P = 1$  (in this case  $R_N = 0$ ) in Definition 1.2. For most problems of computational mathematics it is reasonable to accept an error estimate with a probability smaller than 1.

The year 1949 is generally regarded as the official birthday of the Monte Carlo method when the paper of Metropolis and Ulam [Metropolis and Ulam (1949)] was published, although some authors point to earlier dates.

Ermakov [Ermakov (1975)], for example, notes that a solution of a problem by the Monte Carlo method is contained in the Old Testament. In 1777 G. Comte de Buffon posed the following problem: suppose we have a floor made of parallel strips of wood, each the same width, and we drop a needle onto the floor. What is the probability that the needle will lie across a line between two strips [de Buffon (1777)]? The problem in more mathematical terms is: given a needle of length  $l$  dropped on a plane ruled with parallel lines  $t$  units apart, what is the probability  $P$  that the needle will cross a line? (see, [de Buffon (1777); Kalos and Whitlock (1986)]). He found that  $P = 2l/(\pi t)$ . In 1886 Marquis Pierre-Simon de Laplace showed that the number  $\pi$  can be approximated by repeatedly throwing a needle onto a lined sheet of paper and counting the number of intersected lines (see, [Kalos and Whitlock (1986)]). The development and intensive applications of the method is connected with the names of J. von Neumann, E. Fermi, G. Kahn and S. M. Ulam, who worked at Los Alamos (USA) for forty years on the Manhattan project <sup>1</sup>. A legend says that the method was named in honor of Ulam's uncle, who was a gambler, at the suggestion of Metropolis.

The development of modern computers, and particularly parallel computing systems, provided fast and specialized generators of random numbers and gave a new momentum to the development of Monte Carlo algorithms.

There are many algorithms using this essential idea for solving a wide range of problems. The Monte Carlo algorithms are currently widely used for those problems for which the deterministic algorithms hopelessly break down: high-dimensional integration, integral and integro-differential equations of high dimensions, boundary-value problems for differential equations in domains with complicated boundaries, simulation of turbulent flows, studying of chaotic structures, etc..

An important advantage of Monte Carlo algorithms is that they permit the direct determination of an unknown functional of the solution, in a given number of operations equivalent to the number of operations needed to calculate the solution at only one point of the domain [Dimov and Tonev (1993b,a); Dimov *et al.* (1996)]. This is very important for some problems of applied science. Often, one does not need to know the solution on the whole domain in which the problem is defined. Usually, it is only necessary

---

<sup>1</sup>Manhattan Project refers to the effort to develop the first nuclear weapons during World War II by the United States with assistance from the United Kingdom and Canada. At the same time in Russia I. Kurchatov, A. Sakharov and other scientists working on *Soviet atomic bomb project* were developing and using the Monte Carlo method.

to know the value of some functional of the solution. Problems of this kind can be found in many areas of applied sciences. For example, in statistical physics, one is interested in computing linear functionals of the solution of the equations for density distribution functions (such as Boltzmann, Wigner or Schroedinger equation), i.e., probability of finding a particle at a given point in space and at a given time (integral of the solution), mean value of the velocity of the particles (the first integral moment of the velocity) or the energy (the second integral moment of the velocity), and so on.

It is known that Monte Carlo algorithms are efficient when parallel processors or parallel computers are available. To find the most efficient parallelization in order to obtain a high value of the speed-up of the algorithm is an extremely important practical problem in scientific computing [Dimov *et al.* (1996); Dimov and Tonev (1992, 1993a)].

Monte Carlo algorithms have proved to be very efficient in solving multidimensional integrals in composite domains [Sobol (1973); Stewart (1983, 1985); Dimov and Tonev (1993a); Haber (1966, 1967)]. The problem of evaluating integrals of high dimensionality is important since it appears in many applications of control theory, statistical physics and mathematical economics. For instance, one of the numerical approaches for solving stochastic systems in control theory leads to a large number of multidimensional integrals with dimensionality up to  $d = 30$ .

There are two main directions in the development and study of Monte Carlo algorithms. The first is *Monte Carlo simulation*, where algorithms are used for *simulation* of real-life processes and phenomena. In this case, the algorithms just follow the corresponding physical, chemical or biological processes under consideration. In such simulations Monte Carlo is used as a tool for choosing one of many different possible outcomes of a particular process. For example, Monte Carlo simulation is used to study particle transport in some physical systems. Using such a tool one can simulate the probabilities for different interactions between particles, as well as the distance between two particles, the direction of their movement and other physical parameters. Thus, Monte Carlo simulation could be considered as a method for solving *probabilistic* problems using some kind of simulations of *random variables* or *random fields*.

The second direction is *Monte Carlo numerical* algorithms. Monte Carlo numerical algorithms can be used for solving *deterministic* problems by modeling random variables or random fields. The main idea is to construct some *artificial* random process (usually, a Markov process) and to prove that the mathematical expectation of the process is equal to the unknown

solution of the problem or to some functional of the solution. A Markov process is a stochastic process that has the Markov property. Often, the term Markov chain is used to mean a discrete-time Markov process.

**Definition 1.3.** A finite discrete Markov chain  $T_k$  is defined as a finite set of states  $\{\alpha_1, \alpha_2, \dots, \alpha_k\}$ .

At each of the sequence of times  $t = 0, 1, \dots, k, \dots$  the system  $T_k$  is in one of the following states  $\alpha_j$ . The state  $\alpha_j$  determines a set of conditional probabilities  $p_{jl}$ , such that  $p_{jl}$  is the probability that the system will be in the state  $\alpha_l$  at the  $(\tau + 1)^{th}$  time given that it was in state  $\alpha_j$  at time  $\tau$ . Thus,  $p_{jl}$  is the probability of the transition  $\alpha_j \Rightarrow \alpha_l$ . The set of all conditional probabilities  $p_{jl}$  defines a transition probability matrix

$$P = \{p_{jl}\}_{j,l=1}^k,$$

which completely characterizes the given chain  $T_k$ .

**Definition 1.4.** A state is called absorbing if the chain terminates in this state with probability one.

In the general case, iterative Monte Carlo algorithms can be defined as *terminated Markov chains*:

$$T = \alpha_{t_0} \rightarrow \alpha_{t_1} \rightarrow \alpha_{t_2} \rightarrow \dots \rightarrow \alpha_{t_i}, \quad (1.2)$$

where  $\alpha_{t_q}$ , ( $q = 1, \dots, i$ ) is one of the absorbing states. This determines the value of some function  $F(T) = J(u)$ , which depends on the sequence (1.2). The function  $F(T)$  is a random variable. After the value of  $F(T)$  has been calculated, the system is restarted at its initial state  $\alpha_{t_0}$  and the transitions are begun anew. A number of  $N$  independent runs are performed through the Markov chain starting from the state  $s_{t_0}$  to any of the absorbing states. The average

$$\frac{1}{N} \sum_T F(T) \quad (1.3)$$

is taken over all actual sequences of transitions (1.2). The value in (1.3) approximates  $E\{F(T)\}$ , which is the required linear form of the solution.

Usually, there is more than one possible ways to create such an artificial process. After finding such a process one needs to define an algorithm for computing values of the r.v.. The r.v. can be considered as a weight of a random process. Then, the Monte Carlo algorithm for solving the

problem under consideration consists in simulating the Markov process and computing the values of the r.v.. It is clear, that in this case a statistical error appears. The error estimates are important issue in studying Monte Carlo algorithms.

Here the following important problems arise:

- How to define the random process for which the mathematical expectation is equal to the unknown solution?
- How to estimate the statistical error?
- How to choose the random process in order to achieve *high computational efficiency* in solving the problem with a given statistical accuracy (for *a priori* given probable error)?

This book will be primary concerned with *Monte Carlo numerical* algorithms. Moreover, we shall focus on the performance analysis of the algorithms under consideration on different parallel and pipeline (vector) machines. The general approach we take is the following:

- Define the problem under consideration and give the conditions which need to be satisfied to obtain a unique solution.
- Consider a random process and prove that such a process can be used for obtaining the approximate solution of the problem.
- Estimate the probable error of the method.
- Try to find the optimal (in some sense) algorithm, that is to choose the random process for which the statistical error is minimal.
- Choose the parameters of the algorithm (such as the number of the realizations of the random variable, the length (number of states) of the Markov chain, and so on) in order to provide a good balance between the *statistical* and the *systematic* errors.
- Obtain *a priori* estimates for the *speed-up* and the *parallel efficiency* of the algorithm when *parallel or vector machines* are used.

By  $x = (x_{(1)}, x_{(2)}, \dots, x_{(d)})$  we denote a  $d$ -dimensional point (or vector) in the domain  $\Omega$ ,  $\Omega \subset \mathbb{R}^d$ , where  $\mathbb{R}^d$  is the  $d$ -dimensional Euclidean space. The  $d$ -dimensional unite cube is denoted by  $\mathbf{E}^d = [0, 1]^d$ .

By  $f(x)$ ,  $h(x)$ ,  $u(x)$ ,  $g(x)$  we usually denote functions of  $d$  variables belonging to some functional spaces. The inner (scalar) product of functions  $h(x)$  and  $u(x)$  is denoted by  $(h, u) = \int_{\Omega} h(x)u(x)dx$ . If  $\mathbf{X}$  is some Banach space and  $u \in \mathbf{X}$ , then  $u^*$  is the conjugate function belonging to the dual space  $\mathbf{X}^*$ . The space of functions continuous on  $\Omega$  are denoted by  $\mathbf{C}(\Omega)$ .

$\mathbf{C}^{(k)}(\Omega)$  is the space of functions  $u$  for which all  $k$ -th derivatives belong to  $\mathbf{C}(\Omega)$ , i.e.,  $u^{(k)} \in \mathbf{C}(\Omega)$ . As usual  $\|f\|_{\mathbf{L}_q} = (\int_{\Omega} f^q(x)p(x)dx)^{1/q}$  denotes the  $\mathbf{L}_q$ -norm of  $f(x)$ .

**Definition 1.5.**  $\mathbf{H}^\lambda(M, \Omega)$  is the space of functions for which  $|f(x) - f(x')| \leq M|x - x'|^\lambda$ .

We also use the  $\mathbf{W}_q^r$ -semi-norm, which is defined as follows:

**Definition 1.6.** Let  $d, r \geq 1$  be integers. Then  $\mathbf{W}^r(M; \Omega)$  can be defined as a class of functions  $f(x)$ , continuous on  $\Omega$  with partially continuous  $r^{\text{th}}$  derivatives, such that

$$|D^r f(x)| \leq M,$$

where

$$D^r = D_1^{r_1} \dots D_d^{r_d}$$

is the  $r^{\text{th}}$  derivative,  $r = (r_1, r_2, \dots, r_d)$ ,  $|r| = r_1 + r_2 + \dots + r_d$ , and

$$D_i^{r_i} = \frac{\partial^{r_i}}{\partial x_{(i)}^{r_i}}.$$

**Definition 1.7.** The  $\mathbf{W}_q^r$ -semi-norm is defined as:

$$\|f\|_{\mathbf{W}_q^r} = [\int_{\Omega} (D^r f(x))^q p(x) dx]^{1/q}.$$

We use the notation  $L$  for a linear operator. Very often  $L$  is a linear integral operator or a matrix.

**Definition 1.8.**  $Lu(x) = \int_{\Omega} l(x, x')u(x')dx'$  is an integral transformation ( $L$  is an integral operator)

$A \in \mathbb{R}^{n \times n}$  or  $L \in \mathbb{R}^{n \times n}$  are matrices of size  $n \times n$ ;  $a_{ij}$  or  $l_{ij}$  are elements in the  $i^{\text{th}}$  row and  $j^{\text{th}}$  column of the matrix  $A$  or  $L$  and  $Au = f$  is a linear system of equations. The transposed vector is denoted by  $x^T$ .  $L^k$  is the  $k^{\text{th}}$  iterate of the matrix  $L$ .

$(h, u) = \sum_{i=1}^n h_i u_i$  is the inner (scalar) product of the vectors  $h = (h_1, h_2, \dots, h_n)$  and  $u = (u_1, u_2, \dots, u_n)^T$ .

We will be also interested in *computational complexity*.

**Definition 1.9.** Computational complexity is defined by

$$C_N = tN,$$

where  $t$  is the mean time (or number of operations) needed to compute one value of the r.v. and  $N$  is the number of realizations of the r.v..

Suppose there exists different Monte Carlo algorithms to solve a given problem. It is easy to show that the computational effort for the achievement of a preset probable error is proportional to  $t\sigma^2(\theta)$ , where  $t$  is the expectation of the time required to calculate one realization of the random variable  $\theta$ . Indeed, let a Monte Carlo algorithm  $A_1$  deal with the r.v.  $\theta_1$ , such that  $E\theta_1 = J$  (where  $J$  is the exact solution of the problem). Alternatively there is another algorithm  $A_2$  dealing with another r.v.  $\theta_2$ , such that  $E\theta_2 = J$ . Using  $N_1$  realizations of  $\theta_1$  the first algorithm will produce an approximation to the exact solution  $J$  with a probable error  $\varepsilon = c\sigma(\theta_1)N_1^{-1/2}$ . The second algorithm  $A_2$  can produce an approximation to the solution with the same probable error  $\varepsilon$  using another number of realizations, namely  $N_2$ . So that  $\varepsilon = c\sigma(\theta_2)N_2^{-1/2}$ . If the time (or number of operations) is  $t_1$  for  $A_1$  and  $t_2$  for  $A_2$ , then we have

$$C_{N_1}(A_1) = t_1 N_1 = \frac{c^2}{\varepsilon^2} \sigma^2(\theta_1) t_1,$$

and

$$C_{N_2}(A_2) = t_2 N_2 = \frac{c^2}{\varepsilon^2} \sigma^2(\theta_2) t_2.$$

Thus, the product  $t\sigma^2(\theta)$  may be considered as *computational complexity*, whereas  $[t\sigma^2(\theta)]^{-1}$  is a measure for the *computational efficiency*. In these terms the *optimal* Monte Carlo algorithm is the algorithm with the lowest computational complexity (or the algorithm with the highest computational efficiency).

**Definition 1.10.** Consider the set  $\mathcal{A}$ , of algorithms  $A$ :

$$\mathcal{A} = \{A : Pr(R_N \leq \varepsilon) \geq c\}$$

that solve a given problem with a probability error  $R_N$  such that the probability that  $R_N$  is less than *a priori* given constant  $\varepsilon$  is bigger than a constant  $c < 1$ . The algorithms  $A \in \mathcal{A}$  with the smallest computational complexity will be called *optimal*.

We have to be more careful if we have to consider two classes of algorithms instead of just two algorithms. One can state that the algorithms of the first class are better than the algorithms of the second class if:

- one can prove that some algorithms from the first class have a certain rate of convergence and
- there is no algorithm belonging to the second class that can reach such a rate.

The important observation here is that the *lower error bounds* are very important if we want to compare classes of algorithms. We study optimal algorithms in functional spaces in Chapters 2 and 3 of this book.