Advanced Monte Carlo Methods - Computational Challenges

Ivan Dimov

IICT, Bulgarian Academy of Sciences

Sofia, February, 2012

Aims

- To cover important topics from probability theory and statistics. To present advanced Monte Carlo methods for computational problems arising in various areas of Computational Science;

- Examples will be drawn from Integrals, Integral Equations, Linear Algebra and Differential Equations.

Learning Outcomes:

- Become familiar with stochastic methods such as Monte Carlo for solving mathematical problems and be able to apply such methods to model real life problems;

- **Assessable Learning Outcomes:** To be able to efficiently analyze a problem and be able to select and apply an efficient stochastic method for its solution.

Definition:

Monte Carlo methods are methods of approximation the solution by using random processes with parameters of the processes 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.

Lecture Content(1):

- Main Definitions and notation. Error and complexity of algorithms.
- Probability Space, Random Variables. Discrete and Continuous Random Variables.
- Monte Carlo Integration. Error analysis.
- Efficient Monte Carlo Algorithms.
- Super-convergent algorithms. Interpolation Quadratures.
- Quasi-Monte Carlo Algorithms.
- Super-convergent Adaptive Monte Carlo Algorithms. Practical Applications.

Lecture Content(2):

- Solving Linear Equations. Iterative Algorithms. Discrete Markov Chains.
- Matrix Inversion by Monte Carlo.
- Convergence and Mapping.
- Monte Carlo method for Computing Eigenvalues.
- Boundary Value Problems for PDEs.
- Grid Monte Carlo Algorithms.
- Grid-free algorithms. Local Integral Presentation.
- Parallel Implementations. Practical Computations.

Key Texts:

- The course notes: freely distributed.

- I. Dimov, *Monte Carlo Methods for Applied Scientists*, World Scientific, London, 2008, ISBN-10 981-02-2329-3. (http://www.amazon.com/Monte-Carlo-Methods-Applied-Scientists/dp/9810223293).

- M.H. Kalos and P.A. Whitlock, *Monte Carlo Methods*, Wiley-VCH Verlag, Weinheim, 2008, ISBN 978-3-527-40760-6.

- G.R.Grimmett, D.R.Stirzarek, Probability and Random Processes, Second Edition, 1992, Clarendon Press. ISBN: 0-19-853665-8.

- Rubinstein, Simulation and the Monte Carlo Method, John Wiley and Sons, 1984.

Introduction

- 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.

- Usually Monte Carlo methods reduce problems to the approximate calculation of mathematical expectations. Let us introduce some notations used in the course: the mathematical expectation of the r.v. ξ or θ 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 γ 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 ξ or θ by ξ_i, θ_i $(i = 1, 2, \ldots, N)$ respectively. If ξ_i is a d-dimensional random point, then usually it is constructed using d random numbers γ, i.e., $\xi_i \equiv (\gamma_i^{(1)}, \ldots, \gamma_i^{(d)})$. 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. ξ with mathematical expectation equal to J must be constructed: $E\xi = J$. Using N independent values (realizations) of $\xi : \xi_1, \xi_2, \ldots, \xi_N$, an approximation $\overline{\xi}_N$ to $J : J \approx \frac{1}{N}(\xi_1 + \ldots + \xi_N) = \overline{\xi}_N$, can then be computed. The following definition of the probability error can be given:

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

$$P = Pr\left\{|\overline{\xi}_N - J| \leq R_N\right\}, \tag{1}$$

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:

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

The computational problem in Monte Carlo algorithms becomes one of calculating repeated realizations of the r.v. ξ 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 $R_N = 0$). 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 was published, although some authors point to earlier dates. Ermakov, for example, notes that a solution of a problem by the Monte Carlo method is contained in the Old Testament. In 1777 G. Compte 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? 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? He found that $P = 2l/(\pi t)$. In 1886 Marquis Pierre-Simon de Laplace showed that the number π can be

approximated by repeatedly throwing a needle onto a lined sheet of paper and counting the number of intersected lines. 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 [1]. 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

[1] 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, G. Mikhailov and other scientists working on *Soviet atomic bomb project* were developing and using the Monte Carlo method.

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. 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 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.

Monte Carlo algorithms have proved to be very efficient in solving multidimensional integrals in composite domains. 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 are 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: A finite discrete Markov chain T_k is defined as a finite set of states $\{\alpha_1, \alpha_2, \ldots, \alpha_k\}$.

At each of the sequence of times $t = 0, 1, \ldots, k, \ldots$ the system T_k is in one of the following states α_j. The state α_j determines a set of conditional

probabilities p_{jl}, such that p_{jl} is the probability that the system will be in the state α_l at the $(\tau+1)^{th}$ time given that it was in state α_j at time τ. 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: 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*:

$$T = \alpha_{t_0} \to \alpha_{t_1} \to \alpha_{t_2} \to \ldots \to \alpha_{t_i}, \tag{2}$$

where α_{t_q}, $(q = 1, \ldots, i)$ is one of the absorbing states. This determines the value of some function $F(T) = J(u)$, which depends on the sequence (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 α_{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

$$\frac{1}{N}\sum_{T} F(T) \tag{3}$$

is taken over all actual sequences of transitions (2). The value in (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 course 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.

Notation:

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

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 Ω 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: $\mathbf{H}^\lambda(M, \Omega)$ 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: 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 Ω with partially continuous r^{th} derivatives, such that

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

where

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

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

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

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

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

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

Definition: $Lu(x) = \int_\Omega l(x, x')u(x')dx'$ 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^{th} row and j^{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^{th} iterate of the matrix L.

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

We will be also interested in *computational complexity*.

Definition: 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 θ. Indeed, let a Monte Carlo algorithm A_1 deal with the r.v. θ_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. θ_2, such that $E\theta_2 = J$. Using N_1 realizations of θ_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 ε 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: 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 ε is bigger than a constant $c < 1$. The algorithms $A \in \mathcal{A}$ with the smallest computational complexity will be called $optimal$.

Low bounds estimates:

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.

Monte Carlo convergence rate:

The quality of any algorithm that approximate the true value of the solution depends very much of the convergence rate. One needs to estimate how fast the approximate solution converges to the true solution. Let us consider some basic results for the convergence of Monte Carlo methods. Let ξ be a r.v. for which the mathematical expectation of $E\xi = I$ exists. Let us formally define

$$E\xi = \begin{cases} \int_{-\infty}^{\infty} \xi p(\xi) d\xi, \text{ where } \int_{-\infty}^{\infty} p(x)dx = 1, & \text{when } \xi \text{ is a continuous r.v.;} \\ \sum_{\xi} \xi p(\xi), \text{ where } \sum_{x} p(x) = 1, & \text{when } \xi \text{ is a discrete r.v.} \end{cases}$$

By definition $E\xi$ exists if and only if $E|\xi|$ exists. The nonnegative function $p(x)$ (continuous or discrete) is called the probability density function.

To approximate the variable I, a computation of the arithmetic mean must

usually be carried out,

$$\overline{\xi}_N = \frac{1}{N}\sum_{i=1}^{N}\xi_i. \tag{4}$$

For a sequence of uniformly distributed independent random variables, for which the mathematical expectation exists, the theorem of J. Bernoulli (who proved for the first time the Law of Large Numbers Theorem) is valid. This means that the arithmetic mean of these variables converges to the mathematical expectation:

$$\xi_N \xrightarrow{p} I \quad \text{as} \quad N \to \infty$$

(the sequence of the random variables $\eta_1, \eta_2, \ldots, \eta_N, \ldots$ converges to the constant c if for every $h > 0$ it follows that

$$\lim_{N\to\infty} P\{|\eta_N - I| \geq h\} = 0).$$

Thus, when N is sufficiently large

$$\overline{\xi}_N \approx I \tag{5}$$

and (4) may be used to approximate the value of I whenever $E\xi = I$ exists.

Let us consider the problem of estimating the error of the algorithm. Suppose that the random variable ξ has a finite variance

$$D\xi = E(\xi - E\xi)^2 = E\xi^2 - (E\xi)^2.$$

It is well known that the sequences of the uniformly distributed independent random variables with finite variances satisfy the *central limit theorem*. This means that for arbitrary x_1 and x_2

$$\lim_{N\to\infty} P\left\{x_1 < \frac{1}{\sqrt{ND\xi}}\sum_{i=1}^{N}(\xi_i - I) < x_2\right\} = \frac{1}{\sqrt{2\pi}}\int_{x_1}^{x_2} e^{-\frac{t^2}{2}}dt. \qquad (6)$$

Let $x_2 = -x_1 = x$. Then from (6) it follows that

$$\lim_{N\to\infty} P\left\{\left|\frac{1}{N}\sum_{i=1}^{N}(\xi_i - I)\right| < x\sqrt{\frac{D\xi}{N}}\right\} = \Phi(x),$$

where $\Phi(x) = \dfrac{2}{\sqrt{2\pi}}\displaystyle\int_0^x e^{-\frac{t^2}{2}}dt$ is the probability integral.

When N is sufficiently large

$$P\left\{|\overline{\xi}_N - I| < x\sqrt{\frac{D\xi}{N}}\right\} \approx \Phi(x). \tag{7}$$

Formula gives a whole family of estimates, which depend on the parameter x. If a probability β is given, then the root $x = x_\beta$ of the equation

$$\Phi(x) = \beta$$

can be found, e.g., approximately using statistical tables.

Then, from (7), it follows that the probability of the inequality

$$|\overline{\xi}_N - I| \leq x_\beta \sqrt{\frac{D\xi}{N}} \tag{8}$$

is approximately equal to β.

The term on the right-hand side of the inequality (8) is exactly the *probability error* R_N. Indeed, since R_N is the least possible real number, for which $P = Pr\left\{|\overline{\xi}_N - I| \leq R_N\right\}$, $R_N = x_\beta \sqrt{\frac{D\xi}{N}}$. Taking into account that the *probable error* is the value r_N for which: $Pr\left\{|\overline{\xi}_N - I| \leq r_N\right\} = \frac{1}{2} = Pr\left\{|\overline{\xi}_N - I| \geq r_N\right\}$ one can determine r_N:

$$r_N = x_{0.5}\sigma(\xi)N^{-\frac{1}{2}}, \tag{9}$$

where $\sigma(\xi) = (D\xi)^{\frac{1}{2}}$ is the standard deviation and $x_{0.5} \approx 0.6745$.

I. Integral Evaluation:

A Plain (Crude) Monte Carlo Algorithm

Plain (Crude) Monte Carlo is the simplest possible approach for solving multidimensional integrals. This approach simply uses the definition of the mathematical expectation.

Let Ω be an arbitrary domain (bounded or unbounded, connected or unconnected) and let $x \in \Omega \subset \mathbb{R}^d$ be a d-dimensional vector.

Let us consider the problem of the approximate computation of the integral

$$I = \int_\Omega f(x)p(x)dx, \tag{10}$$

where the non-negative function $p(x)$ is called a density function if $\int_\Omega p(x)dx = 1$. Note that every integral $\int_\Omega f(x)dx$, when Ω is a bounded domain, is an integral of the kind (10). In fact, if S_Ω is the area of Ω, then

$p_1(x) \equiv \frac{1}{S_\Omega}$ is the *probability density function* of a random point which is uniformly distributed in Ω. Let us introduce the function $f_1(x) = S_\Omega f(x)$. Then, obviously, $\int_\Omega f(x)dx = \int_\Omega f_1(x)p_1(x)dx$.

Let ξ be a random point with probability density function $p(x)$. Introducing the random variable

$$\theta = f(\xi) \tag{11}$$

with mathematical expectation equal to the value of the integral I, then

$$E\theta = \int_\Omega f(x)p(x)dx.$$

Let the random points $\xi_1, \xi_2, \ldots, \xi_N$ be independent realizations of the random point ξ with probability density function $p(x)$ and $\theta_1 = f(\xi_1), \ldots, \theta_N = f(\xi_N)$.

Then an approximate value of I is

$$\overline{\theta}_N = \frac{1}{N} \sum_{i=1}^{N} \theta_i. \tag{12}$$

If the integral is absolutely convergent, then $\overline{\theta}_N$ would be convergent to I.

A Simple Example

Let us consider a simple example of evaluating multi-dimensional integrals in functional classes which demonstrates the power of the Monte Carlo algorithms. This is a case of practical computations showing high efficiency of the Monte Carlo approach versus the deterministic one. Consider the classical problem of integral evaluation in functional classes with bounded derivatives. Suppose $f(x)$ is a continuous function and let a quadrature formula of Newton or Gauss type be used for calculating the integrals. Consider an example with $d = 30$ (this is a typical number for some applications in control theory and mathematical economics). In order to apply such formulae, we generate a grid in the d-dimensional domain and take the sum (with the respective coefficients according to the chosen formula) of the function values at the grid points. Let a grid be chosen with 10 nodes on the each of the coordinate axes in the d-dimensional cube $\mathbf{E^d} = [\mathbf{0}, \mathbf{1}]^{\mathbf{d}}$. In this case we have to compute about 10^{30} values of the function $f(x)$.

Suppose a time of $10^{-7}s$ is necessary for calculating one value of the function. Therefore, a time of order $10^{23}s$ will be necessary for evaluating the integral

(remember that 1 year $= 31536 \times 10^3 s$, and that there has been less than $9 \times 10^{10} s$ since the birth of Pythagoras). Suppose the calculations have been done for a function $f(x) \in \mathbf{W^{(2)}}(\alpha, [\mathbf{0}, \mathbf{1}]^{\mathbf{d}})$. If the formula of rectangles (or some similar formula) is applied then the error in the approximate integral calculation is

$$\varepsilon \leq cMh^3, \ \ (h = 0.1), \tag{13}$$

where h is the mesh-size and c is a constant independent of h. More precisely, if $f(x) \in C^2$ then the error is $\frac{1}{12} f^{(2)}(c) h^3$, where c is a point inside the domain of integration.

Consider a plain Monte Carlo algorithm for this problem with a probable error ε of the same order. The algorithm itself consists of generating N pseudo random values (points) (PRV) in E^d; in calculating the values of $f(x)$ at these points; and averaging the computed values of the function. For each uniformly distributed random point in $\mathbf{E^d}$ we have to generate 30 random numbers uniformly distributed in $[0, 1]$.

To apply the Monte Carlo method it is sufficient that $f(x)$ is continuous. The

probable error is:

$$\varepsilon \leq 0.6745\sigma(\theta)\frac{1}{\sqrt{N}}, \tag{14}$$

where $\sigma(\theta) = (D\theta)^{1/2}$ is the standard deviation of the r.v. θ for which $E\theta = \int_{E^d} f(x)p(x)dx$ and N is the number of the values of the r.v. (in this case it coincides with the number of random points generated in E^d).

We can estimate the probable error using (14) and the variance properties:

$$\varepsilon \leq 0.6745 \left(\int_{\mathbf{E^d}} f^2(x)p(x)dx - \left(\int_{\mathbf{E^d}} f(x)p(x)dx \right)^2 \right)^{1/2} \frac{1}{\sqrt{N}}$$

$$\leq 0.6745 \left(\int_{\mathbf{E^d}} f^2(x)p(x)dx \right)^{1/2} \frac{1}{\sqrt{N}} = 0.6745 \parallel f \parallel_{\mathbf{L}_2} \frac{1}{\sqrt{N}}.$$

In this case, the estimate simply involves the $\mathbf{L}_2$-norm of the integrand.

The computational complexity of this commonly-used Monte Carlo algorithm

will now be estimated. From (14), we may conclude:

$$N \approx \left(\frac{0.6745 \parallel f \parallel_{\mathbf{L}_2}}{cM} \right)^2 \times h^{-6}.$$

Suppose that the expression in front of h^{-6} is of order 1. (For many problems it is significantly less than 1 as M is often the maximal value of the second derivative; further the Monte Carlo algorithm can be applied even when it is infinity). For our example ($h = 0.1$), we have

$$N \approx 10^6;$$

hence, it will be necessary to generate $30 \times 10^6 = 3 \times 10^7$ PRV. Usually, two operations are sufficient to generate a single PRV. Suppose that the time required to generate one PRV is the same as that for calculating the value of the function at one point in the domain $\mathbf{E}^{\mathrm{d}}$. Therefore, in order to solve the

problem with the same accuracy, a time of

$$3 \times 10^7 \times 2 \times 10^{-7} \approx 6s$$

will be necessary. The advantage of employing Monte Carlo algorithms to solve such problems is obvious. The above result may be improved if additional realistic restrictions are placed upon the integrand $f(x)$.

The plain (or crude) Monte Carlo integration is often considered as the best algorithm for evaluating multidimensional integrals in case of very high dimensions. In the case of low dimensions (less than 3) Gauss product rules seem to be the most efficient choice. Between these two bound cases algorithms that exploit the regularity of the integrand should be used. In fact, we use a variance reduction technique to build Monte Carlo algorithms with an increased rate of convergence. The proposed algorithms are based on piecewise interpolation polynomials and on the use of the control variate method. It allowed us to build two algorithms reaching the optimal rate of convergence for smooth functions.

It is known that for some problems (including one-dimensional problems) Monte Carlo algorithms have better convergence rates than the optimal deterministic

algorithms in the appropriate function spaces. For example, one can show that if $f(x) \in \mathbf{W^1}(\alpha; [\mathbf{0}, \mathbf{1}]^{\mathbf{d}})$, then instead of (14) we have the following estimate of the probability error:

$$R_N = \varepsilon \leq c_1 \alpha \frac{1}{N^{1/2+1/d}}, \tag{15}$$

where c_1 is a constant independent of N and d. For the one dimensional case we have a rate of $O(N^{-\frac{3}{2}})$ instead of $O(N^{-\frac{1}{2}})$, which is a significant improvement. Indeed, one needs just $N = 10^2$ random points (instead of $N = 10^6$) to solve the problem with the same accuracy.

Let us stress on the fact that we compare different approaches assuming that the integrand belongs to a certain functional class. It means that we do not know the particular integrand and our consideration is for the $worst$ function from a given class.

Another approach: adaptive and interpolation type quadrature. To be able to apply adaptive or interpolation quadrature one should know the integrand, so this problem is much easier for consideration.