ARTICLE IN PRESS

Available online at www.sciencedirect.com
ScienceDirect

ELSEVIER

Applied Mathematical Modelling xxx (2007) xxx–xxx

APPLIED MATHEMATICAL MODELLING

www.elsevier.com/locate/apm

What Monte Carlo models can do and cannot do efficiently?

Emanouil Atanassov [a], Ivan T. Dimov [b,*]

[a] *Institute for Parallel Processing, Department of Parallel Algorithms, Bulgarian Academy of Sciences, Acad. G. Bonchev Street, 25 A, 1113 Sofia, Bulgaria*

[b] *ACET, The University of Reading, Whiteknights, P.O. Box 217, Reading RG6 6AH, UK*

Received 1 December 2006; received in revised form 1 April 2007; accepted 20 April 2007

Abstract

The question "*what Monte Carlo models can do and cannot do efficiently*" is discussed for some functional spaces that define the regularity of the *input data*. Data classes important for practical computations are considered: classes of functions with bounded derivatives and Hölder type conditions, as well as Korobov-like spaces.

Theoretical performance analysis of some algorithms with unimprovable rate of convergence is given. Estimates of computational complexity of two classes of algorithms – deterministic and randomized for both problems – numerical multidimensional integration and calculation of linear functionals of the solution of a class of integral equations are presented.

MSC: 45B05; 65C05; 65C30; 65C40; 65Gxx; 65R20; 68Q25; 68W20

Keywords: Monte Carlo algorithms; Deterministic algorithms; Multidimensional integration; Integral equations; Unimprovable rate of convergence

1. Introduction: definitions and basic notations

The Monte Carlo method is 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.

The Monte Carlo method is a method for solving problems using random variables. In [1] one can find the following definition of the Monte Carlo method.

Definition 1.1. The Monte Carlo method consists of solving various problems of computational mathematics by means of the construction of some random process for each such problem, with the parameters of the process equal to the required quantities of the problem.

* Corresponding author.
E-mail addresses: emanouil@parallel.bas.bg (E. Atanassov), I.T.Dimov@reading.ac.uk (I.T. Dimov).
URL: http://www.personal.reading.ac.uk/sis04itd/ (E. Atanassov).

0307-904X/$ - see front matter
doi:10.1016/j.apm.2007.04.010

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

The method can guarantee that the error of Monte Carlo approximation is smaller than a given value with a certain probability. So, Monte Carlo method always produces an approximation of the solution, but one can control the accuracy of this solution in terms of probability error.

Let us introduce some notations used in the paper: 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 $\bar{\xi}_n$ to $J : J \approx \frac{1}{n}(\xi_1 + \cdots + \xi_n) = \bar{\xi}_n$, 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

$$P = \Pr\{|\bar{\xi}_n - J| \leqslant R_n\}, \tag{1}$$

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

The probable error is the value r_n for which

$$\Pr\{|\bar{\xi}_n - J| \leqslant r_n\} = \frac{1}{2} = \Pr\{|\bar{\xi}_n - J| \geqslant r_n\}.$$

So, dealing with randomized algorithms one has to accept that the result of the computation can be close to the real value only with a certain probability. Such a setting of the problem of error estimation may not be acceptable if one needs a guaranteed accuracy or strictly reliable results. But in the most cases it is reasonable to accept an error estimate with a probability smaller than 1. In fact, we shall see that this is a price paid by randomized algorithms to increase their convergence rate. It is important to note here that the value of the probability P $(0 < P < 1)$ in (1) does not reflect on the rate of convergence of the probability error R_n. It reflects only on the constant. That is why the choice of the value of P is not important for the convergence rate (respectively, for the rate of algorithmic complexity). Nevertheless, for practical computations it may be of great importance to have the value of the constant in order to get the number of operations for a given algorithm. Two examples of such estimates are presented in Section 2.4, and one more example is given in Section 3.4.

The year 1949 is generally regarded as the official birthday of the Monte Carlo method when the paper of Metropolis and Ulam [2] was published, although some authors point to earlier dates. Ermakov [3], 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 [4]? 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 [4,5]). 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 (see [5]). The development and intensive applications of the method is connected with the names of John von Neumann, E. Fermi and G. Kahn, who worked at Los Alamos (USA) for 40 years for the Manhattan project.[1] 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.

[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.

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

There are many algorithms using this essential idea for solving a wide range of problems. In our consideration we use both *Monte Carlo* and *randomized* algorithms as equivalent concepts. The randomized 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 the 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 [6–8]. 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, but only a part of the solution or even the solution at a given point. 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 function (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 well known that Monte Carlo algorithms are very efficient when parallel processors or parallel computers are available. Indeed these algorithms are inherently parallel and have minimum dependency. In addition, they are also naturally vectorizable when powerful vector processors are used. Nevertheless, the problem of parallelization of the Monte Carlo algorithms is not a trivial task because different kinds of parallelization can be used. 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 [8,9,7].

Monte Carlo algorithms have proved to be very efficient in solving multidimensional integrals in composite domains [10–12,7,13]. The problem of evaluation 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 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. Usually, there are 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 (usually, a Markov 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.

This paper will be primary concerned with *Monte Carlo numerical* algorithms. We will concentrate on two important problems of numerical analysis: evaluation of integrals and linear functionals of integral equations. Moreover, we shall focus on the performance analysis of the algorithms under consideration. Some important results from this point of view can be found in [14,9,7,15–17]. We should say that the performance analysis of the algorithms is closely connected with the error analysis in functional spaces. Our consideration will allow comparisons between deterministic and randomized algorithms. We are going to discuss the unimprovable limits of the complexity of two big classes of algorithms for computing both integrals and integral equations. Having these unimprovable rates another important question arises: which one of the existing algorithms

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

reaches these unimprovable rates? Another interesting problem will be: if there are no algorithms reaching these unimprovable rates, then develop new algorithms with such optimal behaviours. In this paper we will refer to some results and show some algorithms with unimprovable rates, but we are not going to formulate new algorithms with optimal behaviours.

There are two big classes of Monte Carlo numerical algorithms: *direct* and *iterative*. A typical case of direct Monte Carlo algorithms is the case of Monte Carlo integration. Iterative randomized algorithms are used for calculating linear functionals of the solution of integral equations. Define an iteration of *order* 0.2 mm i as a function of the following form:

$$u^{(k+1)} = F_k(A, b, u^{(k)}, u^{(k-1)}, \ldots, u^{(k-i+1)}),$$

where $u^{(k)}$ is obtained from the kth iteration. It is desired that $u^{(k)} \to u = A^{-1}b$ as $k \to \infty$. The method is called *stationary* if $F_k = F$ for all k, that is, F_k is independent of k. The iterative process is called *linear* if F_k is a linear function of $u^{(k)}, \ldots, u^{(k-i+1)}$. Here we will be interested in *stationary linear iterative Monte Carlo* algorithms.

Consider a general description of the iterative Monte Carlo algorithms. Let $\mathbf{F}$ be a Banach space of real-valued functions. Let $f = f(x) \in \mathbf{F}$ and $u_k = u(x_k) \in \mathbf{F}$ be defined in a close domain $G \subset \mathbb{R}^d$ and $L = L(u)$ be a linear operator defined on $\mathbf{F}$. Consider the sequence $u_1, u_2, \ldots$, defined by the recursion formula

$$u_k = L(u_{k-1}) + f, \quad k = 1, 2, \ldots \tag{2}$$

The formal solution of (2) is the truncated Neumann series

$$u_k = f + L(f) + \cdots + L^{k-1}(f) + L^k(u_0), \quad k > 0, \tag{3}$$

where L^k means the kth iterate of L.

As an example consider the integral iterations. Let $u(x) \in \mathbf{F}$, $x \in G \subset \mathbb{R}^d$ and $l(x, x')$ be a function defined for $x \in G$, $x' \in G$. The integral transformation $Lu(x) = \int_G l(x, x')u(x')\,\mathrm{d}x'$ maps the function $u(x)$ onto the function $Lu(x)$, and is called an *iteration of $u(x)$ by the integral transformation kernel $l(x, x')$*. The second integral iteration of $u(x)$ is denoted by $LLu(x) = L^2u(x)$. Obviously, $L^2u(x) = \int_G \int_G l(x, x')l(x', x'')\mathrm{d}x'\,\mathrm{d}x''$. In this way $L^3u(x), \ldots, L^iu(x), \ldots$ can be defined. When the infinite series converges, the sum is an element u from the space $\mathbf{F}$ which satisfies the equation

$$u = L(u) + f. \tag{4}$$

The truncation error of (3) is $u_k - u = L^k(u_0 - u)$. Let $J(u_k)$ be a functional that has to be calculated. Consider the spaces

$$\mathbf{T}_{i+1} = \underbrace{\mathbb{R}^d \times \mathbb{R}^d \times \ldots \times \mathbb{R}^d}_{i \text{ times}}, \quad i = 1, 2, \ldots, k, \tag{5}$$

where "×" denotes the Cartesian product of spaces.

Random variables $\theta_i,\ i = 0, 1, \ldots, k$ are defined on the respective product spaces $\mathbf{T}_{i+1}$ and have conditional mathematical expectations

$$E\theta_0 = J(u_0), \quad E(\theta_1/\theta_0) = J(u_1), \ldots, E(\theta_k/\theta_0) = J(u_k),$$

where $J(u)$ is a functional of u. The computational problem then becomes one of calculating repeated realizations of θ_k and combining them into an appropriate statistical estimator of $J(u_k)$.

As an approximate value of the linear functional $J(u_k)$ is set up

$$J(u_k) \approx \frac{1}{n}\sum_{s=1}^{n}\{\theta_k\}_s, \tag{6}$$

where $\{\theta_k\}_s$ is the sth realization of the r.v. θ_k.

Note that the nature of the every process realization of θ is a Markov process. We will consider only *discrete Markov processes with a finite set of states*, the so called *finite discrete Markov chains*. Such kind of chains are used to compute linear functionals of the solution of integral equations.

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

Definition 1.3. A finite discrete Markov chain T_i is defined as a finite set of states $\{k_1, k_2, \ldots, k_i\}$.

At each of the sequence of times $t = 0, 1, \ldots, i, \ldots$ the system T_i is in one of the following states k_j. The state k_j determines a set of conditional probabilities p_{jl}, such that p_{jl} is the probability that the system will be in the state k_l at the $(\tau + 1)$th time given that it was in state k_j at time τ. Thus, p_{jl} is the probability of the transition $k_j \Rightarrow k_l$. The set of all conditional probabilities p_{jl} defines a transition probability matrix $P = \{p_{jl}\}_{j,l=1}^{i}$, which completely characterizes the given chain T_i.

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*

$$T = k_{t_0} \to k_{t_1} \to k_{t_2} \to \cdots \to k_{t_i}, \tag{7}$$

where k_{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 (7). The function $F(T)$ is a r.v. After the value of $F(T)$ has been calculated, the system is restarted at its initial state k_{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

$$\frac{1}{n}\sum_{T} F(T) \tag{8}$$

is taken over all actual sequences of transitions (7). The value in (8) approximates $E\{F(T)\}$, which is the required linear form of the solution.

We also will be interested in *performance analysis of algorithms*. The performance analysis deals with *computational cost* of the algorithms.

Definition 1.5. Computational cost of a randomized iterative algorithm A^R is defined by

$$\tau(A^R) = nE(q)t_0,$$

where $E(q)$ is the mathematical expectation of the number of transitions in the sequence (7) and t_0 is the mean time needed for value of one transition.

In case of Monte Carlo integration, which is a direct randomized algorithm we have $E(q) = 1$ and the cost is a product of the number of values of the r.v. and the mean time needed to produce one value of the r.v.

Dealing with randomized algorithms, the general approach one could follow is:

- Define the problem under consideration and give the conditions that need to be satisfied to obtain a unique solution.
- Construct a random process and prove that such a process can be used for obtaining the approximate solution of the problem.
- Estimate the statistical 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 values of the r.v., 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 processors* are used.

In this paper we will not go in details in choosing parameters of the algorithms as it is done in our paper [14]. We also are not going to obtain a priori estimates for the speed-up and the parallel efficiency of the algorithms under consideration. Some results in this direction are given in [8,9,7]. The above mentioned questions are important, but they are considered in other works. Here we shall concentrate on comparing two classes of algorithms – deterministic and randomized from a point of view of their performance for fixed classes of the input data of the problem, i.e., for a priori given smoothness.

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

The question: *what Monte Carlo can do and cannot do efficiently?* frequently arises among people dealing with numerical methods, scientific computing and applications of mathematics in theoretical and applied sciences. Often if there are two algorithms people are interested which one is *better*. What means *better* in the case when one cannot get the exact solution of the problem and one is happy to have an ε-approximation to the true solution? Obviously, one will call *better* the algorithm, which produces the ε-approximation to the solution with a fixed probability *faster*. We assume that an algorithm that uses smaller number of operations is executed faster, and thus we measure the speed of the algorithm with the number of operations required.

Definition 1.6. Consider the set $\mathscr{A}$, of algorithms, A

$$\mathscr{A} = \{A : \Pr(R_n \leqslant \varepsilon) \geqslant c\}$$

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 $A \in \mathscr{A}$ with the smallest computational cost 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, which 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.

By $x = (x_1, \ldots, x_d)$ we denote a point in the domain $G \subset \mathbb{R}^d$, where $\mathbb{R}^d$ is d-dimensional Euclidean space. The d-dimensional unite cube is denoted by $E^d = [0,1]^d$.

By $f(x)$, $h(x)$, $u(x)$, $g(x)$ we denote functions of d variables belonging to some functional spaces. The inner product of functions $h(x)$ and $u(x)$ is denoted by $(h,u) = \int_G h(x)u(x)\,\mathrm{d}x$. $J(u)$ denotes a linear functional of u. 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 G are denoted by $C(G)$. $C^{(k)}(G)$ is the space of functions u for which $u^{(k)} \in C(G)$. As usual $\|f\|_{\mathbf{L}_q} = (\int_G f^q(x)p(x)\,\mathrm{d}x)^{1/q}$ denotes the $\mathbf{L}_q$-norm of $f(x)$.

Definition 1.7. $H_\lambda(\alpha, G)$ is the space of functions for which $|f(x) - f(y)| \leqslant \alpha|x - y|^\lambda$.

Definition 1.8. $\mathbf{W}^r(\alpha; G)$ is a class of functions $f(x)$, continuous on G with partially continuous rth derivatives, such that $|D^r f(x)| \leqslant \alpha$, where $D^r = D_1^{r_1} \ldots D_d^{r_d}$ is the rth derivative, $r = (r_1, r_2, \ldots, r_d)$, $|r| = r_1 + r_2 + \cdots + r_d$, and $D_i = \frac{\partial}{\partial x_{(i)}}$.

Definition 1.9. The $\mathbf{W}_q^r$-norm is defined as

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

Definition 1.10. Define the class $H_\lambda^r(\alpha, G)$, $(0 < \lambda \leqslant 1)$ of functions from C^r, which derivatives of order r satisfy the Hölder condition with a parameter λ

$$H_\lambda^r(\alpha, G) \equiv \left\{ f \in C^r : |D^r f(y_1, \ldots, y_d) - D^r f(z_1, \ldots, z_d)| \leqslant \alpha \sum_{j=1}^{d} |y_j - z_j|^\lambda \right\}.$$

The analysis, studying and finding the number of operations (or the computational cost) we call *performance analysis*. So, the performance analysis deals with the computational cost of algorithms. It should be mentioned here that the performance analysis is connected with the complexity that will be defined latter in Section 2.2. The complexity characterizes the problem for a given class of algorithms and not the algorithms. In Section 2, we will show how the computational cost is connected with the complexity.

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

2. Computational cost and complexity of integration

We will consider a numerical problem as a mapping

$$S : F_0 \to \mathbb{R},$$

where $F_0 \subset F$ and F is some Banach space. We will call S a solution operator following [18]. The elements of F_0 are the data, for which the problem has to be solved; and for $f \in F_0$, $S(f)$ is the exact solution. For a given $f \in F$ we want to compute (or approximate) $S(f)$.

2.1. Problem of integration

Let us consider the following problem of integration:

$$S_I(f) = \int_{E^d} f(x)\mathrm{d}x,$$

where $E^d \equiv [0,1]^d$, $x \equiv (x_1, \ldots, x_d) \in E^d \subset \mathbb{R}^d$ and $f \in C(E^d)$ is an integrable function on E^d.

The computational problem can be considered as a mapping of function $f : \{[0,1]^d \to \mathbb{R}^d\}$ to $\mathbb{R}$ [18]: $S_I(f) : f \to \mathbb{R}$, where $S_I(f) = \int_{E^d} f(x)\mathrm{d}x$ and $f \in F_0 \subset F \equiv C(E^d)$. For a given f we want to compute (or approximate) $S_I(f)$. We will be interested to consider subsets F_0 of $C(E^d)$ and try to study how the smoothness of F_0 can be exploited.

Many problems in science and finance lead to high-dimensional integration and it is not an easy task to select a good (or the best, if possible) algorithm to handle the problem when f belongs to a given functional space.

2.2. Computational model

We will call a *quadrature formula* any expression of the following kind:

$$A_I = \sum_{i=1}^{n} c_i f(x_i),$$

which approximates the value of the integral $S_I(f)$. The real numbers $c_i \in \mathbb{R}$ are called weights and d dimensional points $x_i \in E^d$ are called nodes.

It is clear that for fixed weights c_i and nodes x_i the quadrature formula A_I may be used to define an algorithm. The algorithm A_I belongs to the class of deterministic algorithms $\mathscr{A}$. It is assumed that for each integrand $f \in F_0$ the program can access an oracle $\mathrm{O}(f,t)$ that provides information about f. This information consists of values at f of linear functionals on F. In our particular case, f is a function defined on the domain $E^d \subset \mathbb{R}^d$. At input $t = (t_1, \ldots, t_d)$ the oracle $\mathrm{O}(t,f)$ is supposed to provide the value $f(t)$ [18].

We call a *randomized quadrature formula* any formula of the following kind:

$$A_I^R = \sum_{i=1}^{n} \sigma_i f(\xi_i),$$

where σ_i and ξ_i are random weights and nodes.

To obtain random nodes and weights a pseudo-random generator (PRG) is used (in fact, pseudo-random generators are widely used [19–22]). It is assumed that PRG produces a sequence of independent, uniformly distributed over the domain $[0,1]^d$ random variables (as defined in Section 2.7) $\xi_i(\omega)$, $i = 1, 2, \ldots, \xi_i(\omega) \equiv (\gamma_1, \ldots, \gamma_d)$ on some probability space (Ω, Σ, μ). More precisely, when the algorithm calls PRG-instruction, the number $\xi_1(\omega)$ is produced; the second PRG-call produces $\xi_2(\omega)$, etc.. The dependence of the values produced is given through the dependence on $\omega \in \Omega$ (and the probability μ on (Ω, Σ)).

The computational cost of the algorithm A_I at x will be denoted by $\tau(A_I, f)$. In the case of randomized algorithms the computational cost will be denoted by $\tau(A_I^R, f, \omega)$. The computational cost is the sum of the cost of all instructions carried out, where arithmetic floating point operations, comparisons, PRG-calls are assumed

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

to be of unit cost. As a good approximation we can also use the sum of the cost of all floating point instructions, since they are more easier to be estimated, while the integer arithmetic and logical operations may depend on the particular compiler settings. We refer for more details to [15,23,17]. The computational cost of a deterministic algorithm A will be defined as

$$\tau(A) = \sup_{f \in F_0} \tau(A, f).$$

For a randomized algorithm A^R we will have the expected number of instructions

$$\tau(A^R) = \sup_{f \in F_0} E_\mu \tau(A^R, f, \omega).$$

For a given positive ε the ε-complexity of the integration problems S_I and S_I^R are defined as follows:

$$C_\varepsilon(S_I) = \inf_{A \in \mathcal{A}} \{\tau(A_I) : r(A_I) \leqslant \varepsilon\}$$

and

$$C_\varepsilon(S_I^R) = \inf_{A^R \in \mathcal{A}^R} \{\tau(A_I^R) : r(A_I^R) \leqslant \varepsilon\},$$

where the errors $r(A_I)$ and $r(A_I^R)$ are defined in Section 2.3. As a good measure of the cost can be considered

$$\tau(A_I, f) = kn + c,$$
$$\tau(A_I^R, f, \omega) = k^R n + c^R,$$

where n is the number of nodes and k, k^R are constants depending on the function f, dimensionality d and on the domain of integration (in our case on E^d) and constants c and c^R depend only on d and on the regularity parameter of the problem (in the case of $H_\lambda^p(\alpha, G)$-on $p + \lambda$). These constants describe the so-called preprocessing operations, i.e., operations that are needed to be performed beforehand.

2.3. Error analysis results in functional spaces

Generally, we assume that the problem of integration is not solved exactly, that is $S_I(f)$ differs from $A_I(f)$. We define the error as

$$r(A_I) = \sup_{f \in F_0} |S_I(f) - A_I(f)|$$

in the deterministic case and as

$$r(A_I^R) = \sup_{f \in F_0} E_\mu |S_I(f) - A_I^R(f, \omega)| = \sup_{f \in F_0} \int_{E^d} |S_I(f) - A_I^R(f, \omega)| \mathrm{d}\mu(\omega)$$

(where $A_I(f, \omega)$ is Σ-measurable in ω for each f) in the randomized case.

To get error estimates we will use some results of Bakhvalov and Korobov [24–28].

Let us now define the subset F_0: Let d, p be integers, and $d, p \geqslant 1$. Consider the class C^p of real functions f defined over the unit cube $E^d \equiv [0, 1]^d$, possessing all the partial derivatives

$$D^r f = \frac{\partial^r f(x)}{\partial x_1^{r_1} \ldots \partial x_d^{r_d}}, \quad r_1 + \ldots + r_d = r \leqslant p,$$

which are continuous when $r < p$ and bounded in sub norm when $r = p$. The semi-norm $\|.\|$ on C^p is defined as

$$\|f\| = \sup\{|D^p f|, |r_1, \ldots, r_d| = p, x \equiv (x_1, \ldots, x_d) \in E^d\}.$$

Now define the class $H_\lambda^p(\alpha, E^d)$, $(0 < \lambda \leqslant 1)$ of functions from C^p, which derivatives of order p satisfy Hölder condition with a parameter λ

$$H_\lambda^p(\alpha, E^d) \equiv \left\{ f \in C^p : |D^p f(y_1, \ldots, y_d) - D^p f(z_1, \ldots, z_d)| \leqslant \alpha \sum_{j=1}^{d} |y_j - z_j|^\lambda \right\}.$$

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

So, first we set

$$F_0 \equiv H_\lambda^p(\alpha, E^d).$$

In [24], Bakhvalov proved the following theorem:

Theorem 2.1 (Bakhvalov [24]). *For any deterministic way of evaluation the integral*

$$\sup_{f \in H_\lambda^p(\alpha, E^d)} r(A_I) \geqslant c'(d, p+\lambda)\alpha n^{-\frac{p+\lambda}{d}} \tag{9}$$

and for any randomized way of evaluation the integral

$$\sup_{f \in H_\lambda^p(\alpha, E^d)} r(A^R) \geqslant c''(d, p+\lambda)\alpha n^{-\frac{p+\lambda}{d}-\frac{1}{2}}. \tag{10}$$

The constants $c'(d,p+\lambda)$ and $c'(d,p+\lambda)$ depend only on d and $p+\lambda$.

The proof of the Bakhvalov's theorem is obtained by using a special construction allowing to obtain lower-bound estimates [24–26,29]. This theorem gives the best possible order in the class of deterministic algorithms $\mathscr{A}$, as well as in the class of randomized algorithms $\mathscr{A}^{\mathscr{R}}$.

In our work [14] we construct two algorithms A_1^R and A_2^R of randomized settings $\mathscr{A}^{\mathscr{R}}$: and for both of them the best possible rate (10) is reached. In fact, in [14] we consider the class $W^p(\alpha, E^d)$, but the proposed algorithms allow to extend easily the estimates for the functional class $H_\lambda^p(\alpha, E^d)$, where $0 < \lambda \leqslant 1$. Later we shall show how this could be done. Here we will give an example of two optimal algorithms with unimprovable rate of convergence.

For each integer $q, d, p \geqslant 1$ we define a Monte Carlo integration formula, depending on an integer parameter $m \geqslant 1$ and $\binom{d+p-1}{d}$ points in $\mathbf{E}^d$ in the following way:

The $\binom{d+p-1}{d}$ points $x^{(r)}$ have to fulfill the condition that if for some polynomial $P(x)$ of combined degree less than p

$$P(x^{(r)}) = 0,$$

then $P \equiv 0$. Let $n = q^d$, $q \geqslant 1$. We divide the unit cube $\mathbf{E}^d$ into q^d disjoint cubes $\mathbf{E}^d = \bigcup_{j=1}^n K_j$, where $K_j = \prod_{i=1}^d [a_i^j, b_i^j)$, with $b_i^j - a_i^j = \frac{1}{q}$ for all $i = 1, \ldots, d$. Now in every cube K_j we calculate the coordinates of $\binom{d+p-1}{d}$ points $y^{(r)}$, defined by $y_i^{(r)} = a_i^r + \frac{1}{n} x_i^{(r)}$.

Suppose, we select m random points $\xi(j,s) = (\xi_1(j,s), \ldots, \xi_d(j,s))$ from each cube K_j, such that all $\xi_i(j,s)$ are uniformly distributed and mutually independent, calculate all $f(y^{(r)})$ and $f(\xi(j,s))$ and consider the Lagrange interpolation polynomial of the function f at the point z, which uses the information from the function values at the points $y^{(r)}$. We call it $L_p(f,z)$. For all polynomials P of degree at most $p-1$ we have $L_k(p,z) \equiv z$.

We approximate

$$\int_{K_j} f(x)\mathrm{d}x \approx \frac{1}{mn} \sum_{s=1}^{m} (f(\xi(j,s)) - L_p(f, \xi(j,s))) + \int_{K_j} L_k(f,x)\mathrm{d}x.$$

Then we sum these estimates over all $j = 1, \ldots, n$ to achieve

$$I(f) \approx \frac{1}{mn} \sum_{j=1}^{n} \sum_{s=1}^{m} (f(\xi(j,s)) - L_p(f, \xi(j,s))) + \int_{K_j} L_p(f,x)\mathrm{d}x.$$

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

Now, for the class $H^p_\lambda(\alpha, E^d)$ we can prove the following theorem:

Theorem 2.2. *The cubature formula constructed above satisfies*

$$r_n(f, p+\lambda, d, m) \leqslant c'(d, p+\lambda)\frac{1}{m}\alpha n^{-\frac{1}{2}-\frac{p+\lambda}{d}}$$

and

$$\left(E\left(\int_{\mathbf{E}^d} f(x)\mathrm{d}x - I(f)\right)^2\right)^{1/2} \leqslant c''(d, p+\lambda)\frac{1}{m}\alpha n^{-\frac{1}{2}-\frac{p+\lambda}{d}},$$

where the constants $c'(d,p+\lambda)$ and $c'(d,p+\lambda)$ depend implicitly on the points $x^{(r)}$, but not on n.

Proof. The proof is a modification of the proof given in [14]. Indeed, taking into account that f belongs to the space $H^p_\lambda(\alpha, E^d)$ one can use the following inequality:

$$|f(\xi(s,t) - L_p(f, \xi(j,s)| \leqslant c_{d,p+\lambda}\alpha n^{-p-\lambda}. \tag{11}$$

Using the above inequality and applying the same technique that is used in the proof of Theorem 2.1 from [14] we prove the theorem. □

It is clear that the algorithms A^R_1 and A^R_2 are both unimprovable by rate for all functions from $H^p_\lambda(\alpha, E^d)$. Indeed,

$$r(A^R_{I_1}) \leqslant c''_1(d, p+\lambda)\alpha n^{-\frac{p+\lambda}{d}-\frac{1}{2}}$$

for the algorithm A^R_1 and

$$r(A^R_{I_2}) \leqslant c''_2(d, p+\lambda)\alpha n^{-\frac{p+\lambda}{d}-\frac{1}{2}}$$

for the algorithm A^R_2.

For any particular d and $p+\lambda$ one can chose the algorithm with the smallest $c'(d,p+\lambda)$. Algorithms with the best possible order are called *optimal*. In this sense both algorithms A^R_1 and A^R_2 proposed in [14] are optimal.

2.4. Performance analysis of algorithms with unimprovable convergence rate

In this subsection two algorithms reaching the unimprovable rate of convergence are given. Estimates of the computational cost of both algorithms are presented.

In the fist algorithm the points $x^{(r)}$ are selected so that they fulfill certain conditions that would assure good Lagrange approximation of any function from $H^p_\lambda(\alpha, E^d)$. Let us order all the monomials of d variables and degree less than $p-\mu_1, \ldots, \mu_t$. Note that there are exactly $\binom{d+p-1}{d}$ of them. We use a pseudo-random number generator to obtain many sets of points $x^{(r)}$, then we select the one for which the conditional number of the matrix $A = (a_{ij})$ with $a_{ij} = \mu_i(x^{(j)})$ is the smallest.

Once it is selected for fixed p and d, the same set of points will be used for integrating every function from the space $H^p_\lambda(\alpha, E^d)$. These calculations could be considered as *preprocessing*. The following theorem can be proved:

Theorem 2.3. *The computational cost of the numerical integration of a function from $H^p_\lambda(\alpha, E^d)$ using the above mentioned algorithm A^R_1 can be presented in the following form*:

$$\tau(A^R_1, f, \omega) = k^R n + c^R,$$

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

where

$$k^R \leqslant \left[m + \binom{d+p-1}{d}\right] a_f + m[d(b_r+2)+1] \tag{12}$$

$$+ 2\binom{d+p-1}{d}\left[m+1+d+\binom{d+p-1}{d}\right], \tag{13}$$

where b_r denotes the number of flops used to produce a uniformly distributed random number in [0,1), *a_f stands for the number of flops needed for each calculation of a function value, and $c^R = c^R(d, p+\lambda)$ depends only on d and $p+\lambda$.*

Proof. The proof follows from the proof of Theorem 3.1 given in [14]. □

The second algorithm A_2^R is a modification of the first one. In A_2^R first some p different points $z_i^{(j)} \in (0,1)$ are selected in each dimension, and then the points $x^{(r)}$ have coordinates

$$\{(z_1^{(j_1)}, \ldots, z_d^{(j_d)}) : (j_1 + \cdots + j_d < p+\lambda)\}.$$

In this case the interpolation polynomial is calculated in the form of Newton, namely if $w_r^{(t)} = a_r^j + (b_r^j - a_r^j) z_r^{(t)}$, then

$$L_p(f,\xi) = \sum_{j_1+\ldots+j_d<k} R(j_1, c, j_d, 0, \ldots, 0) \prod_{i=1}^{d} (\xi_i - w_i^{(1)}) \ldots (\xi_i - w_i^{(j_i-1)}),$$

where

$$R(j_1, \ldots, j_d, l_1, \ldots, l_d) = f(w_1^{j_1}, \ldots, w_d^{j_d}) \quad \text{if all } j_i = l_i,$$

and

$$R(j_1, \ldots, j_i, \ldots, j_d, l_1, \ldots, l_i, \ldots, l_d) = \frac{1}{(w_i^{j_i} - w_i^{l_i})} [R(j_1, \ldots, j_i, \ldots, j_d, l_1, \ldots, l_i + 1, \ldots, l_d)$$

$$- R(j_1, \ldots, j_i - 1, \ldots, j_d, l_1, \ldots, l_i, \ldots, l_d)] \quad \text{if } j_i > l_i. \tag{14}$$

This algorithm involves less operations for the same number of points n because the recurrence formula (14) for calculating the divided differences simplifies the calculations exactly $\binom{p+d-1}{d}$ times. In this modification we have the following theorem:

Theorem 2.4. *The computational cost of the numerical integration of a function from $H_\lambda^p(\alpha, E^d)$ using the second algorithm A_2^R can be presented in the form*

$$\tau(A_2^R, f, \omega) = k^R n + c^R,$$

where

$$k^R \leqslant \left[m + \binom{d+p-1}{d}\right] a_f + m[d(b_r+2+k)+1] \tag{15}$$

$$+ 2\binom{d+p-1}{d}(d+1+m), \tag{16}$$

where a_f and b_m are as above and $c^R = c^R(d, p+\lambda)$ depends only on d and $p+\lambda$.

Proof. The proof follows from the proof of Theorem 3.2 given in [14]. □

Remark 2.1. When d and p are sufficiently large the second algorithm A_2^R would be still feasible even if the first one A_1^R is ineffective.

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

Remark 2.2. The performance analysis results of Theorems 2.3 and 2.4 show that the computational cost of both algorithms is linear with the number of nodes n. With such a cost an error of order $n^{-\frac{p+\lambda}{d}-\frac{1}{2}}$ is reached. Such an order is unimprovable in the functional class under consideration.

Optimal algorithms are also proposed in [30–36]. Some of the proposed algorithms use the construction proposed by Dupach in [37]. Various methods for Monte Carlo integration that achieve the order $O(N^{-\frac{1}{2}-\frac{p}{d}})$ are known. While in the case of $p = 1$ and $p = 2$ these methods are fairly simple and are widely used (see, for example, [38,34,36,39]), when $p \geqslant 3$ such methods become much more sophisticated. The first optimal stochastic method was probably proposed by Mrs. Dupach [37] for $k = 1$. This method uses the idea of separation of the domain of integration into uniformly small (according both to the probability and to the sizes) disjoint subdomains and generating one or small number of points in every subdomain. This idea was largely used [34] for creation Monte Carlo methods with high rate of convergence. So, there is at least one algorithm A_I^R for which

$$r(A_I^R) \leqslant c_2''(d, p+\lambda)\alpha n^{-\frac{p+\lambda}{d}-\frac{1}{2}}.$$

Using the same construction as in [14] it is easy to show that for the deterministic case there exists an algorithm for which

$$r(A_I) \leqslant c_A'(d, p+\lambda)\alpha n^{-\frac{p+\lambda}{d}}.$$

As an example of such an algorithm could be considered the algorithm A_1^R proposed in [14] in which the nodes are fixed points.

2.5. Complexity of the integration problem for Hölder spaces

Now we are ready to formulate a theorem given the estimates of the ε-complexity of the problem.

Theorem 2.5. *For $X_0 \equiv H_\lambda^p(\alpha, E^d)$ the ε-complexity of the problem of integration S_I is*

$$C_\varepsilon(S_I) = k(c_A'(d, p+\lambda)\alpha)^{\frac{d}{p+\lambda}}\left(\frac{1}{\varepsilon}\right)^{\frac{d}{p+\lambda}}$$

for the class of deterministic algorithms $\mathscr{A}$, and

$$C_\varepsilon(S_I) = k^R(c_A''(d, p+\lambda)\alpha)^{\frac{d}{p+\lambda+d/2}}\left(\frac{1}{\varepsilon}\right)^{\frac{d}{p+\lambda+d/2}}$$

for the class of randomized algorithms $\mathscr{A}^{\mathscr{R}}$.

Proof. According to the definition of the cost of the algorithm we should take the worst algorithm in sense of $\tau(A_I, f)$ corresponding to $f \in H_\alpha^p(\lambda, E^d)$. According to the Bakhvalov theorem [24] one can write

$$\sup_{x \in H_\alpha^p(\lambda, E^d)} \tau(A_I, f) = k \times n + c = k \times (c_A'(d, p+\lambda)\alpha)^{\frac{d}{p+\lambda}}\left(\frac{1}{r(A_I)}\right)^{\frac{d}{p+\lambda}} + c.$$

Now, for a given $\varepsilon > 0$ we should take

$$\inf\left\{k(c_A'(d, p+\lambda)\alpha)^{\frac{d}{p+\lambda}}\left(\frac{1}{r(A_I)}\right)^{\frac{d}{p+\lambda}} : r(A_I) \leqslant \varepsilon\right\}.$$

Let us note, that this is a non-uniform notion: for each $\varepsilon > 0$ a separate A_I can be designed. However, following the remark that the algorithms A_I are uniform over the set of problems, and the fact that the infimum of the number of preprocessing operations described by c is zero, one can get

$$C_\varepsilon(S_I) = k(c_A'(d, p+\lambda)\alpha)^{\frac{d}{p+\lambda}}\left(\frac{1}{\varepsilon}\right)^{\frac{d}{p+\lambda}},$$

which proves the first part of the theorem concerning the deterministic algorithms. □

The result for the randomized algorithms can be proved by the same way.

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

Corollary 2.1. *The ε-complexity of the problem of integration strongly depends on the dimension of the problem for the class of deterministic algorithms. With the increasing of dimensionality the ε-complexity goes* **exponentially** *to infinity for the class* $F_0 = H_\alpha^p(\lambda, E^d)$.

Corollary 2.2. *In the case of randomized algorithms the ε-complexity of the integration problem for functions from* $F_0 = H_\alpha^p(\lambda, E^d)$ *goes asymptotically to*

$$\left(\frac{1}{\varepsilon}\right)^2.$$

Remark 2.3. The fact that the ε-complexity exponentially depends on d makes the class of deterministic algorithms infeasible for large dimensions. It was demonstrated by the example given in Section 2.7. This conclusion is also confirmed by our practical experience.

Remark 2.4. In the last case the ε-complexity does not increase exponentially with d. This is why for high-dimensional integration Monte Carlo is a right choice. Nevertheless, Bakhvalov's results demonstrated that the smoothness can be exploited to improve the rate of convergence by a factor of $n^{-\frac{p+\lambda}{d}}$ over the rate of standard randomized algorithms $n^{-\frac{1}{2}}$. This fact allows to decrease the ε-complexity from $(1/\varepsilon)^2$ by a factor of

$$\left(\frac{1}{\varepsilon}\right)^{-\frac{4(p+\lambda)}{2(p+\lambda)+d}}.$$

Some authors are interested to relax the assumptions under which the optimal rate of convergence of algorithms are obtained. We shall give an example of how the assumptions can be made weaker without much loss in the rate of convergence. Consider spaces of functions with bounded mixed derivatives corresponding to parameters satisfying the following inequalities:

$$0 \leqslant \alpha_i \leqslant \frac{p}{d},$$

where p is the smoothness parameter and d is the dimensionality. For this class of functions the rate $c(\log n)^\beta n^{p/d}$ can be reached for some parameter $\beta > 0$. Results of this type are obtained and discussed in [40,41].

2.6. Complexity of the integration problem for Korobov-like spaces

Now we shall consider two other functional spaces F_0, in which there are deterministic algorithms with a *good* asymptotic behaviour of their ε-complexity. By *good* we mean linear asymptotic behaviour O(d) or weaker.

Let us introduce some notations following Korobov [27,28]: $d \geqslant 1$, $l > d$, $a_v = a_v(l)$ – integer numbers, $\alpha \geqslant 1$, $c = c(\alpha, d) > 0$ – real numbers, $\{x\}$ – fractional part of x. Define the class $K(\alpha)$ for $\alpha > 0$ by the following condition:

$$\left.\begin{aligned} f(x_1,\ldots,x_d) &= \sum_{m_1,\ldots,m_d=-\infty}^{\infty} c(m_1,\ldots,m_d)\mathrm{e}^{2\pi i(m_1x_1+\cdots+m_dx_d)} \\ |c(m_1,\ldots,m_d)| &\leqslant \frac{c_1}{[(|m_1|+1)\ldots(|m_d|+1)]^\alpha} \end{aligned}\right\},$$

where c_1 is some constant.

In [28], it is shown that for $n = p$ (p is a prime number) there exist a recursive process for finding $a_1, \ldots, a_d$ such that

$$\int_0^1 \ldots \int_0^1 f(x_1,\ldots,x_d)\mathrm{d}x_1 \ldots \mathrm{d}x_d - \frac{1}{n} u m_{k=1}^n f\left(\left|\left\{\frac{ka_1}{p}\right\},\ldots,\left\{\frac{ka_d}{p}\right\}\right|\right) = \mathrm{O}(n^{-\alpha}\log^{\alpha d} n). \tag{17}$$

Such a recursive process is proposed in [28].

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

It is also proved that a set of integer numbers $a_1, \dots a_d$ such that for any function from $K(\alpha)$ one can replace α by α', where $\alpha' > \alpha$ in the above inequality does not exist.

The set of integer numbers $a_1, \dots, a_d$ proposed by Korobov [28] are called *optimal*. It means that the estimation (17) *cannot be improved significantly*.

Let us assume that A_I is the algorithm that uses exactly the numbers $a_1, \dots, a_d$ proposed by Korobov in [28]. Then one can write

$$r(A_I) \leqslant cn^{-\alpha}\log^{\alpha d} n,$$

where c is some constant.

The randomized version of the Korobov's algorithms can be obtained by sampling one uniformly distributed random vector

$$(b_1, \dots, b_d) \in \mathbf{E}^d$$

and adding it to all the points of the sequence (modulo 1) [42] (see, also [43,44]). The integral is thus approximated via

$$\int_0^1 \dots \int_0^1 f(x_1, \dots, x_d)\mathrm{d}x_1 \dots \mathrm{d}x_d \approx \frac{1}{n}\sum_{k=1}^{n} f\left(\left|\left\{\frac{ka_1}{p} + b_1\right\}, \dots, \left\{\frac{ka_d}{p} + b_d\right\}\right|\right). \tag{18}$$

The standard deviation and consequently the probable error of the randomized algorithm can be estimated but we are not dealing with it in this paper.

In [28], Korobov formulated a wider functional class $K'(\alpha)(K(\alpha) \subset K'(\alpha)$ in the following way: Let $\psi(m) \geqslant |m|$ be an even function, for which the inverse series converges, $\psi(0) = 1$; the ratio $\psi(m)/m$ for $m > 0$ is monotonic and for any $\varepsilon > 0$ $\psi(m) = \mathrm{o}(|m|^{1+\varepsilon})$. Now the class $K'(\alpha)$ for $\alpha \geqslant 1$ can be defined by the following conditions:

$$\left.\begin{aligned} f(x_1, \dots, x_d) &= \sum_{m_1,\dots,m_d=-\infty}^{\infty} c(m_1, \dots, m_d)\mathrm{e}^{2\pi i(m_1x_1+\dots+m_dx_d)} \\ |c(m_1, \dots, m_d)| &\leqslant \frac{c_1}{[\psi(m_1)\dots\psi(m_d)]^{\alpha}} \end{aligned}\right\}. \tag{19}$$

In [28], it is shown how to find optimal coefficients $a_1, \dots, a_d$ in order to get the following estimate:

$$\int_0^1 \dots \int_0^1 f(x_1, \dots, x_d)\mathrm{d}x_1 \dots \mathrm{d}x_d - \frac{1}{n}\sum_{k=1}^{n} f\left(\left\{\frac{ka_1}{p}\right\}, \dots, \left\{\frac{ka_d}{p}\right\}\right) = c_1(\alpha, d)n^{-\alpha}, \tag{20}$$

where c_1 is some constant.

It is shown by Korobov [28] that the estimate (20) does not allow significant improvement. Moreover, it is shown [28] that the result (20) *cannot be improved by order*. It means that for the best possible algorithm for solving integrals when $f \in K'(\alpha)$ one can write

$$r(A_I) \leqslant c(\alpha, d)n^{-\alpha}. \tag{21}$$

It is clear that $c(\alpha, d)$ could be different from $c_1(\alpha, d)$ found in the Korobov construction [28]. Next we shall consider only the class $K'(\alpha)$.

Theorem 2.6. *For $F_0 \equiv K'(\alpha)$ the ε-complexity of the problem of integration S_I is*

$$C_\varepsilon(S_I) = k[c(\alpha, d)]^{\frac{1}{\alpha}}\left(\frac{1}{\varepsilon}\right)^{\frac{1}{\alpha}}$$

for the class of deterministic algorithms $\mathscr{A}$.

Proof. Assume that the set of optimal coefficients $a_1, \dots, a_d$ defines the algorithm $A \subset \mathscr{A}$. For the algorithm A we have

$$\sup_{x \in K'(\alpha)} \tau(A_I, f) = k \times n + c = k \times [c(\alpha, d)]^{\frac{1}{\alpha}}\left(\frac{1}{r(A_I)}\right)^{\frac{1}{\alpha}} + c.$$

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

Now we can use the uniformity of the notion

$$C_\varepsilon(S_I) = \inf\{\tau(A_I) : r(A_I) \leqslant \varepsilon\}$$

because, as it was discussed above, the error estimation of the best possible algorithm for solving integrals when $f \in K'(\alpha)$ can be expressed as it is written in (21). Thus,

$$C_\varepsilon(S_I) = k[c(\alpha,d)]^{\frac{1}{2}}\left(\frac{1}{\varepsilon}\right)^{\frac{1}{2}}. \qquad \square$$

Corollary 2.3. *The ε-complexity of the problem of integration for functions from $K'(\alpha)$ has a good asymptotic behaviour with increasing of dimensionality d.*

Remark 2.5. The idea of the Korobov's algorithm is to use the decay of the Fourier coefficients as (19). It is clear that the parameter α is linked to the regularity of the integrand. A practical problem, which appear here is that the constant c_1 grows very quickly with α. This is especially true for high dimensions (see, for instance [45]). To apply the Korobov's algorithm for non-periodic functions one has to use a periodization method (see, for instance [46]), such that the obtained periodical function has to satisfy the same condition of decay as (19) and its integral should be the same as the one of the original function. The growth of the constant c_1 is especially large after applying the periodization method (for more details see [47, p. 226, 48, p. 158]).

In case of non-regular input data (discontinues functions and/or singularities) there are special techniques well developed in Monte Carlo algorithms [14,49,50,19,51,52,10]. These techniques allow the inclusion of the singularity into the density function of choice (see, for instance [30,8,19,20]).

For high-dimensional problems with low regularity Quasi-Monte Carlo sequences [53–56,10,57] are very promising as their speed of convergence is

$$O\left(\frac{(\log n)^d}{n}\right).$$

Although the factor $O(\log n)^d$ seems prohibitively large for reasonable values of n, the method achieves good practical results. These methods are outside the scope of this paper. For more information see [55,53,57]. This class of algorithms is close to algorithms that use good lattice points. They are based on number theoretic constructions. Another interesting approaches of this kind are the quantization [58] and the hyperbolic cross technique developed in [35].

2.7. A simple example

Let us now consider a simple example of evaluating multidimensional integrals which demonstrates the power of the Monte Carlo algorithms. This is a case of practical computations showing high efficiency of the randomized approach versus the deterministic one. Consider the classical problem of integral evaluation. Suppose $f(x)$ is a continuous function and let a quadrature formula of Newton–Cotes 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 in the grid points. Let a grid be chosen with 10 nodes on the each of the coordinate axes in the d-dimensional cube $E^d = [0,1]^d$. 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 (let us remember that 1 year $= 31536.10^3$ s, and that there has been less than 9.10^{10} s since the birth of Pythagoras). Suppose the calculations have been done for a function $f(x) \in W^{(2)}(\alpha,[0,1]^d)$. If the formula of rectangles (or some similar formula) is applied then the error in the approximate integral calculation is

$$\varepsilon \leqslant cMh^3 \quad (h = 0.1), \tag{22}$$

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

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 [47,34].

Consider a Monte Carlo algorithm for this problem with a probable error ε of the same order. We have to generate n random points in E^d and to calculate the values of $f(x)$ at these points. For each uniformly distributed random point in 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

$$\varepsilon \leqslant 0.6745\sigma(\theta)\frac{1}{\sqrt{n}}, \tag{23}$$

where $\sigma(\theta) = (D\theta)^{1/2}$ is the standard deviation of the r.v. θ for which $E\theta = \int_{E^d} f(x)p(x)\mathrm{d}x$ 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 (23) and the variance properties

$$\begin{aligned}\varepsilon &\leqslant 0.6745\left(\int_{\mathbf{E}^d} f^2(x)p(x)\mathrm{d}x - \left(\int_{E^d} f(x)p(x)\mathrm{d}x\right)^2\right)^{1/2}\frac{1}{\sqrt{n}}\\ &\leqslant 0.6745\left(\int_{E^d} f^2(x)p(x)\mathrm{d}x\right)^{1/2}\frac{1}{\sqrt{n}} = 0.6745\|f\|_{L_2}\frac{1}{\sqrt{n}}.\end{aligned}$$

In this case, the estimate simply involves the L_2-norm of the integrand.

The computational complexity of this commonly-used Monte Carlo algorithm will now be estimated. From (22) and (23), we may conclude: $n \approx (\frac{0.6745\|f\|_{L_2}}{cM})^2 \times h^{-6}$.

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 $30 \times 10^6 = 3.10^7$ pseudo random values. 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 E^d. Therefore, in order to solve the problem with the same accuracy, a time of $3.10^7 \times 2 \times 10^{-7} \approx 6$ s will be necessary. The advantage of employing Monte Carlo algorithms to solve such problems is obvious.

The crude Monte Carlo integration is often considered as the best algorithm for evaluating multidimensional integrals in case of very high dimensions (see [45,5,47]). In the case of low dimensions (less than 3) Gauss product rules seems to be the most efficient choice. Between these two bound cases should be used algorithms that exploit the regularity of the integrand. An example of such an approach can be found in our work [14]. In fact, we use a variance reduction technique to build randomized algorithms with an increased rate of convergence. The proposed algorithms are based on piecewise interpolation polynomials and on the use of control variate method. It allowed as for smooth functions in a particular space to build two algorithms reaching the optimal rate of convergence. Randomized algorithms with increased rate of convergence are presented in [59–64].

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 [26,30,65,8,66,6,67,34]. For example, as it is shown in Section 2 if $f(x) \in W^1(\alpha; [0,1]^d)$, then instead of (23) we have the following estimate of the probability error $R_n = \varepsilon \leqslant c_1\alpha\frac{1}{n^{1/2+1/d}}$, 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.

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 is considered in [68], where Monte Carlo and Quasi–Monte Carlo quadratures are compared with adaptive and interpolation deterministic type quadrature. To be able to apply adaptive or interpolation quadrature one should know the integrand, so this problem is much easier for consideration. It is also shown in [68] that some dimensions and test-functions adaptive type

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

quadrature outperform the crude Monte Carlo. Such a result is not surprising since such type of deterministic quadrature should be compared with interpolation and adaptive Monte Carlo quadrature [10,30].

3. Cost and complexity of integral equations

Here we consider the computational cost and complexity of solving integral equations. In Section 3.1, we define what we mean by *solving integral equations*. In fact we consider algorithms for computing on bilinear forms of the Fredholm integral equations of second kind. In Section 3.2, the functional spaces, which characterizes the regularity of the *input data* are defined and error estimates are presented. Some results of ε-complexity of the problem under consideration are presented in Section 3.3. Here we also define the class of *almost optimal randomized algorithms*. Results of the computational cost of a grid-free almost optimal randomized algorithm are given in Section 3.4.

3.1. Formulation of the problem of solving integral equations

Let us consider the following problem: compute the functional

$$(h,u) = \int_G h(x)u(x)\mathrm{d}x, \tag{24}$$

where $h(x)$ is a given function and $u(x)$ is the solution of the Fredholm integral equation of second kind

$$u(x) = \int_G l(x,y)u(y)\mathrm{d}y + f(x) \tag{25}$$

or in an operator form

$$u = Lu + f, \tag{26}$$

where $L : C(G) \to C(G)$ denotes an integral operator. We should note here that such a formulation of the problem is very often used in theoretical and applied sciences. The meaning of the above formulated functional is given in Section 1. It could be the 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) in statistical physics problems, or effect of given pollution levels $u(x)$ (satisfying integral transport equation) on the life matter ($h(x)$ is sensitivity to a given pollutant).

The solution operator for the above formulated problem can be written in the following form:

$$S_{Eq}(l,f) = (h,u) = ((I-L)^{-1}f, h).$$

Sometimes, the adjoint equation

$$v = L^*v + h \tag{27}$$

is used.

In (27) $v, h \in \mathbf{F}^*$, $L^* \in [\mathbf{F}^* \to \mathbf{F}^*]$, $\mathbf{F}^*$ is the dual functional space to $\mathbf{F}$ and L^* is an adjoint operator.

For some important applications $\mathbf{F} = \mathbf{L_1}$ and

$$\|f\|_{\mathbf{L}_1} = \int_G |f(x)|\mathrm{d}x; \quad \|L\|_{\mathbf{L}_1} \leqslant \sup_x \int_G |l(x,x')|\mathrm{d}x'. \tag{28}$$

In this case $h(x) \in \mathbf{L}_\infty$, hence ${\mathbf{L}_1}^* \equiv \mathbf{L}_\infty$ and

$$\|h\|_{\mathbf{L}_\infty} = \sup|h(x)|, \quad x \in G.$$

Obviously, if $u \in \mathbf{L}_1$ and $h \in \mathbf{L}_\infty$ the inner product (24) will be bounded.

For many applications $\mathbf{F} = \mathbf{F}^* = \mathbf{L}_2$. $\mathbf{L_2}$ norms are defined as follows:

$$\|f\|_{\mathbf{L}_2} = \left(\int_G (f(x))^2\mathrm{d}x\right)^{\frac{1}{2}}; \quad \|L\|_{\mathbf{L}_2} \leqslant \sup_x \left(\int_G (l(x,x'))^2\mathrm{d}x'\right)^{\frac{1}{2}}.$$

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

Note also, that if $h(x), u(x) \in \mathbf{L}_2$ then the inner product (24) is finite. One can see, that if $u(x) \in \mathbf{L}_2$ and $l(x,x') \in \mathbf{L}_2(G \times G)$ then $Lu(x) \in \mathbf{L}_2$. It is trivial to show that $L^2u(x), \ldots, L^iu(x), \ldots$ also belong to $\mathbf{L}_2(G)$.

For simplicity and concreteness we assume that $\mathbf{F} = \mathbf{F}^* = \mathbf{L}_2$. If it is also assumed that $\|L^m\| < 1$, where m is any natural number, then the Neumann series $u = \sum_{i=0}^{\infty} L^i f$ converges.

The condition $\|L^m\| < 1$ is not very strong, since, as it was shown by Sabelfeld [33,52], it is possible to construct a Monte Carlo algorithm for which the Neumann series does not converge. Analytically extending the resolvent by a change of the spectral parameter gives a possibility to obtain a convergent algorithm when Neumann series for the original problem does not converge or to accelerate the convergence when it converges slowly. It is easy to show that $J = (h,u) = (f,v)$. This equality means that the solution of the adjoint problem is equivalent to the solution of the original one. The last fact is important for performance analysis studies, because in practice very often the solution of the adjoint problem is easier than the solution of the original one.

Let us consider the Monte Carlo algorithm for evaluating the functional (24). It can be seen that when $l(x,x') \equiv 0$ evaluation of the integrals can pose a problem. Consider a random point $\xi \in G$ with a density $p(x)$ and let there be n values of the random point ξ_i $(i = 1,2,\ldots,n)$. Let a r.v. $\theta(\xi)$ be defined in G, such that

$$E\theta(\xi) = J.$$

Then the computational problem becomes one of calculating repeated values of θ and of combining them into an appropriate statistical estimator of J. The nature of the process for calculating every realization of θ is a Markov process. We will consider *discrete Markov processes with a finite set of states*, the so called *Markov chains* (see Definition 1.3 given in Section 1).

An approximate value of the linear functional J, defined by (24) is

$$J \approx \frac{1}{n}\sum_{s=1}^{n}\{\theta\}_s = \hat{\theta}_n,$$

where $(\theta)_s$ is the sth value of the r.v. θ.

The r.v. whose mathematical expectation is equal to $J(u)$ is given by the following expression:

$$\theta[h] = \frac{h(\xi_0)}{p(\xi_0)}\sum_{j=0}^{\infty} Q_j f(\xi_j),$$

where $Q_0 = 1$; $Q_j = Q_{j-1}\frac{l(\xi_{j-1},\xi_j)}{p(\xi_{j-1},\xi_j)}$, $j = 1,2,\ldots$, and $\xi_0, \xi_1, \ldots$ is a Markov chain in G with initial density function $p(x)$ and transition density function $p(x,y)$.

Iterative randomized algorithms are characterized by two types of errors:

- *systematic* error r_i, $i \geqslant 1$ (obtained from the truncating of the Markov chain) which depends on the number of iterations i of the used iterative process

$$|r_i| \leqslant \frac{\|L\|_{\mathbf{L}_2}^{i+1}\|f\|_{\mathbf{L}_2}}{1 - \|L\|_{\mathbf{L}_2}}$$

  and
- *statistical* error R_n, which depends on the number of samples n of Markov chain

$$R_n = c_\beta \sigma^2(\theta[h])n^{-1/2}, \quad 0 < \beta < 1, \quad \beta \in \mathbb{R}.$$

  The constant c_β (and therefore also the complexity estimates of algorithms) depends on the confidence level β. Probable error r_n is often used, which corresponds to a 1/2 confidence level. Such a level is often acceptable for practical computations.

The problem to achieve a good balance between the systematic and statistical error has a great practical importance. To ensure a statistical error ε, it is necessary to perform i transitions in the Markov process. Assuming $\|f\|_{\mathbf{L}_2} > \varepsilon(1-\alpha)$ one can chose the value of i from the inequality

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

$$i > \log^{-1}\alpha[\log\varepsilon + \log(1-\alpha) - \log\|f\|_{\mathbf{L}_2}] - 1, \tag{29}$$

where $\alpha = \|L\|_2$ and the initial approximation is chosen to be the right-hand side f. To achieve a probable error ε, it is necessary to perform n samples depending on the following inequality:

$$n > \left[c_{0.5}\frac{\sigma(\theta)}{\varepsilon}\right]^2, \quad c_{0.5} \approx 0.6745,$$

where θ is the r.v., whose mathematical expectation coincides with the desired linear functional (24).

3.2. Error analysis results

Theorem 3.1 (Emelyanov and Il'in [69]). *For the problem S_{Eq} of solving d-dimensional Fredholm integral equations of second kind with p smooth data*

$$r(A_{Eq}) \leqslant c' n^{-\frac{p}{2d}} \tag{30}$$

for the deterministic algorithms $\mathscr{A}$ and

$$r(A^R_{Eq}) \leqslant c'' n^{-\frac{p}{2d}-\frac{1}{2}} \tag{31}$$

for the randomized algorithms $\mathscr{A}^{\mathscr{R}}$.

This result can be slightly improved if one assumes that the "input" data (l,f) satisfy a kind of Hölder conditions. Let us define the class $\widehat{H}^p_\lambda(\alpha, E^d)$ for $p, d \in \mathscr{N}$ ($\mathscr{N}$ is the set of natural numbers), $\beta,\gamma > 0$, $0 < \delta < 1$

$$F = W^p(E^{2d}) \times W^p(E^d)$$

$$\widehat{H}^p_\lambda(\alpha, E^d) = \Bigg\{ (l,f) \in F : \|f\|_{W^p(E^d)} \leqslant \gamma, \|l\|_{W^p(E^{2d})} \leqslant \beta, \|l\|_{C^0(E^{2d})} \leqslant \delta, |D^p f(y_1,\ldots,y_d) - D^p f(z_1,\ldots,z_d)| \leqslant \alpha \sum_{j=1}^{d} |y_j - z_j|^\lambda,$$

$$\left| l^{(p)}_{y_1^{r_1}\ldots y_d^{r_d}}(x_1,\ldots,x_d;u_1,\ldots,u_d) - l^{(p)}_{y_1^{r_1}\ldots y_d^{r_d}}(x_1,\ldots,x_d;v_1,\ldots,v_d) \right| \leqslant \alpha(x_1,\ldots,x_d) \sum_{j=1}^{d} |u_j - v_j|^\lambda \Bigg\}.$$

We should comment here that the condition

$$\|l\|_{C^0(E^{2d})} \leqslant \delta$$

ensures the existence and uniqueness of the solution of the integral equation under consideration.

Theorem 3.2. *For the problem S_{Eq} of solving d-dimensional Fredholm integral equations of second kind with p smooth data, which is Hölder with a rate of λ, i.e., $(l,f) \in \widehat{H}^p_\lambda(\alpha, E^d)$ we have*

$$r(A_{Eq}) \leqslant c' n^{-\frac{p+\lambda}{2d}} \tag{32}$$

for the deterministic algorithms $\mathscr{A}$ and

$$r(A^R_{Eq}) \leqslant c'' n^{-\frac{p+\lambda}{2d}-\frac{1}{2}} \tag{33}$$

for the randomized algorithms $\mathscr{A}^{\mathscr{R}}$.

Proof. The proof of the theorem combines the proof of the theorem of Emelyanov and Il'in with the special construction of Bakhvalov used in [24,26], as well as inequalities like (11) used in the proof of Theorem 2.2 for $u(x)$ and $l(x,x')$. □

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

3.3. Complexity of the problem of computing functionals of Fredholm integral equations of second kind

Theorem 3.3. *For $F_0 \equiv \widehat{H}^p_\lambda(\alpha, E^d)$ the ε-complexity of solving d-dimensional Fredholm integral equation of second kind S_{Eq} is*

$$C_\varepsilon(S_{Eq}) = k(c'(d, p+\lambda))^{\frac{2d}{p+\lambda}} \left(\frac{1}{\varepsilon}\right)^{\frac{2d}{p+\lambda}}$$

for the class of deterministic algorithms $\mathscr{A}$, and

$$C_\varepsilon(S_{Eq}) = k^R(c''(d, p+\lambda))^{\frac{2d}{p+\lambda+d}} \left(\frac{1}{\varepsilon}\right)^{\frac{2d}{p+\lambda+d}}$$

for the class of randomized algorithms $\mathscr{A}^{\mathscr{R}}$.

Proof. The proof of this theorem follows the technique used in the proof of Theorem 2.5. □

Corollary 3.1. *If there is not additional regularity, i.e., $p + \lambda = 0$ the deterministic algorithms are not feasible while the randomized algorithms are feasible with a rate of ε-complexity of order*

$$\left(\frac{1}{\varepsilon}\right)^2.$$

According to our best knowledge random algorithms with such a low complexity do not exist at present. Such a complexity corresponds to an iterative method with a number of iterations O(1), i.e. method for which the number of iterations does not depend on ε. Therefore, it is reasonable to consider a class of randomized algorithms with a slightly higher ε-complexity.

Definition 3.1. Randomized algorithms with ε-complexity of order

$$\left(\frac{1}{\varepsilon}\right)^2 \log \varepsilon$$

will be called almost optimal randomized algorithms.

In the next subsection an almost optimal Monte Carlo algorithm with a rate of ε-complexity of order $(\frac{1}{\varepsilon})^2 \log \varepsilon$ for evaluation functionals of solution of Fredholm integral equation of second kind in case $p + \lambda = 0$ will be presented.

3.4. Computational cost of a grid-free randomized algorithm for evaluating functionals of the solution of Fredholm integral equations of second kind

In this subsection we consider a grid-free Monte Carlo algorithm called (in the simplest case) *spherical process* for computing of the bilinear forms of the solution of Fredholm integral equations of second kind. We will denote this algorithm by A^R_{GF}. As a first step of this algorithm a Δ-strip ∂G_Δ of the boundary ∂G is chosen (on the supposition that the solution is known on the boundary) to ensure the convergence of the constructed iterative process. The following number of operations is necessary for one random walk:

- generation of d (this number depends on initial probability π) random numbers to determine the initial point in the Markov chain: $d(k_A + k_L)$ operations (k_A and k_L are the arithmetic and logical operations necessary for the generation of one random number) **or** modeling of an isotropic vector that needs a number of operations of order $R * d(k_A + k_L)$ (the constant R depends on the efficiency of the modeling method and transition probability);
- calculating the coordinates of the initial **or** next point: p_{next} (depends on the modeling method and the dimension d of the domain $B(x)$);

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

- calculating one value of functions: p_f, p_π, p_φ **or** p_k, p_P;
- calculating one sample of the r.v. (it needs less than 3 arithmetic operations);
- calculating the distance from the current point to the boundary ∂G: γ_A arithmetic and γ_L logical operations (depends on the dimension d of the domain G);
- verification if the current point belongs to the chosen Δ-strip ∂G_Δ.

The following logarithmic estimate for the average number $E\{i\}$ of spheres on a single trajectory holds for a wide class of boundaries [19]:

$$E\{i\} \leqslant \text{const}|\log \Delta|, \quad \text{const} > 0, \tag{34}$$

where const depends on the boundary ∂G.

Calculating the linear functional with a preliminary given accuracy ε and attainment of a good balance between the *statistical* and the *systematic* error is an important issue in performance analysis studies. To ensure a statistical error ε, it is necessary to perform i transitions in the Markov process, where i is chosen from the inequality (29). On the other hand, the estimate (34) can be used to chose the value of the parameter Δ of the boundary strip. The value of Δ explicitly depends on the number of transitions i of the Markov process according to (29)

$$\Delta \approx \exp(-i/\text{const}).$$

Therefore, the following estimate holds for the mathematical expectation of the time required to obtain an approximation with accuracy ε using the considered grid-free Monte Carlo algorithm:

$$\begin{aligned}\tau(A^R_{\mathrm{GF}}, f, \omega) \approx \tau n \Bigg[& (\mathrm{d}k_A + p_{\text{next}} + p_f + p_\pi + p_\varphi + \gamma_A + 4) l_A + (\mathrm{d}k_L + \gamma_L + 1) l_L \\ & + ((R\,\mathrm{d}k_A + p_{\text{next}} + p_f + p_k + p_P + 4 + \gamma_A) l_A + (R\,\mathrm{d}k_L + \gamma_L + 1) l_L) \\ & \times \frac{(\log \varepsilon + \log(1-\alpha) - \log^3 F^{(0)}}{\log^3 \alpha} \Bigg] \frac{\{c_\beta \sigma(\Theta[h])\}^2}{\varepsilon^2}.\end{aligned}$$

Remark 3.1. From the above estimate it is easy to see that the computational cost of the grid-free randomized algorithm A^R_{GF} for evaluating bilinear forms of Fredholm integral equations of second kind has a rate of ε-complexity of order $(\frac{1}{\varepsilon})^2 \log \varepsilon$. Since not additional regularity is required this algorithm has almost optimal rate.

Remark 3.2. If some additional regularity is assumed, then it is possible that some deterministic algorithm or grid Monte Carlo algorithm could be applied (see, for instance [70,71,8,7]).

The bounds for the ε-complexity for randomized and deterministic algorithms are given in Theorem 3.3. But it is interesting to compare two randomized algorithms – the above described grid-free Monte Carlo algorithm A^R_{GF} with the grid Monte Carlo algorithm A^R_G. The description of the grid Monte Carlo algorithm is given in [72,70,71]. This algorithm is based on the approximation of the integral equation under consideration by a system of linear algebraic equations. This transformation represents the initial step of the considered class of grid randomized algorithms. The linear system is obtained using some approximate cubature rule (cubature method, Nystrom method [29,73,74]). The next step is to apply the resolvent Monte Carlo algorithm [72,71] for solving linear systems of equations. We should mention here that (according to our knowledge) it is still not proved that the grid Monte Carlo algorithm presented in [70] is almost optimal in sense of reaching the correspondent rate of complexity, as it was shown for the grid-free algorithm A^R_{GF}. Nevertheless, we can mention some conditions under which the grid Monte Carlo algorithm could be competitive with the grid-free Monte Carlo algorithm:

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

Table 1
ε-complexity of problems for the classes of deterministic and randomized algorithms

Problem	F_0	Deterministic	Randomized
Integration	$H^p_\lambda(\alpha, E^d)$	$k(c'_A\alpha)^{\frac{d}{p+\lambda}}(\frac{1}{\varepsilon})^{\frac{d}{p+\lambda}}$	$k^R(c''_A\alpha)^{\frac{d}{p+\lambda+d/2}}(\frac{1}{\varepsilon})^{\frac{d}{p+\lambda+d/2}}$
Integration	$K'(\alpha)$	$k[c(\alpha,d)]^{\frac{1}{\alpha}}(\frac{1}{\varepsilon})^{\frac{1}{\alpha}}$	
Int. eq.	$H^p_\lambda(\alpha, E^{2d})$	$k(c')^{\frac{2d}{p+\lambda}}(\frac{1}{\varepsilon})^{\frac{2d}{p+\lambda}}$	$k^R(c'')^{\frac{2d}{p+\lambda+d}}(\frac{1}{\varepsilon})^{\frac{2d}{p+\lambda+d}}$

- the *input data* functions for the integral equation $l(x,t)$, $f(x)$, $h(x)$ should have comparatively small maximum norm in the corresponding domain and it should be a possibility to calculate their values with a relatively low complexity;
- the initial and transition probability used in the grid-free algorithm are complicated for modeling (acceptance–rejection method);
- the dimension d of the integration domain is large.

It has to be noted that the grid Monte Carlo algorithms are admissible only for integral equations with smooth functions, but some techniques of avoiding singularities of this kind exist (see [29]).

4. Concluding discussion

The results presented here deal with the performance analysis of algorithms for most important data classes: classes of functions with bounded derivatives and Korobov-like spaces. It should be mentioned that in practical computations in physics, bio-informatics and finance the problems usually involve various functions from very different classes. Some problems involve discontinues functions and/or singularities, which makes the problem of choosing the optimal algorithm hard.

Considering the summary of current results in Table 1, one can conclude that as the regularity decreases, the simpler randomized algorithm should be used. Even for small dimensions ($d = 1, 2$) Monte Carlo is a right choice if the functional class has no smoothness. If the computational problem allows probability interpretation, then Monte Carlo will be the best choice. To win in the rate of convergence one has to lose in the reliability (it means that the increased rate of convergence is paid by accepting some uncertainty in the answer).

As a general remark it should be emphasized that for both problems under consideration

- numerical integration and
- evaluation linear functionals of integral equations

the randomized algorithms have better convergence rate for the same regularity of the input data.

But one should be careful because

- The better convergence rate for randomized algorithms is reached with a given probability, so the advantage of Monte Carlo algorithms is a matter of definition of the probability error.
- If the nature of the problem under consideration does not allow the use of the probability error for estimates or the answer should be given with a guaranteed error then the higher convergence order randomized algorithms are not acceptable.
- An important obvious advantage of randomized algorithms is the case of *bad functions*, i.e., functions that do not satisfy some additional conditions of regularity. The main problem with the deterministic algorithms is that normally they need some additional approximation procedure requiring additional regularity. The randomized algorithms do not need such procedures.

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

References

[1] Yu.A. Shreider, The Monte Carlo method, Method of Statistical Testing, Pergamon Press Ltd., Oxford, London, Edinburgh, New York, Paris, Frankfurt, 1966.
[2] N. Metropolis, S. Ulam, The Monte Carlo method, J. Am. Statist. Assoc. 44 (247) (1949) 335–341.
[3] S.M. Ermakov, Monte Carlo Methods and Mixed Problems, Nauka, Moscow, 1985.
[4] G. Comte de Buffon, Essai d'arithmetique morale, Suppl. l'Histoire Naturalle, 4, 1777.
[5] M.H. Kalos, P.A. Whitlock, Monte Carlo Methods, Basics, vol. 1, Wiley, New York, 1986.
[6] I. Dimov, O. Tonev, Random walk on distant mesh points Monte Carlo methods, J. Statist. Phys. 70 (5–6) (1993) 1333–1342.
[7] I. Dimov, O. Tonev, Monte Carlo algorithms: performance analysis for some computer architectures, J. Comput. Appl. Math. 48 (1993) 253–277.
[8] I. Dimov, A. Karaivanova, H. Kuchen, H. Stoltze, Monte Carlo algorithms for elliptic differential equations. data parallel functional approach, J. Parallel Algorithms Appl. 9 (1996) 39–65.
[9] I. Dimov, O. Dimov, Criteria estimators for speed-up and efficiency of some parallel Monte Carlo algorithms for different computer architectures, in: Proc. WP&DP'91 (Sofia, April 16–19), North-Holland Publ. Co., Amsterdam, 1992, pp. 243–262.
[10] I.M. Sobol, Monte Carlo Numerical Methods, Nauka, Moscow, 1973.
[11] L. Stewart, Bayesian analysis using Monte Carlo integration – a powerful methodology for handling some difficult problems, Statistician 32 (1983) 195–200.
[12] L. Stewart, Multiparameter Bayesian inference using Monte Carlo integration – some techniques for bivariate analysis, in: J.M. Bernardo, M.H. de Groot, D.V. Lindley, A.F. Smith (Eds.), Bayesian Statistics, vol. 2, North-Holland, Amsterdam, 1985.
[13] S. Haber, A modified Monte Carlo quadrature, Math. Comput. 20 (95) (1966) 361–368;
S. Haber, A modified Monte Carlo quadrature, Math. Comput. 21 (99) (1967) 388–397.
[14] E. Atanassov, I. Dimov, A new Monte Carlo method for calculating integrals of smooth functions, Monte Carlo Methods Appl. 5 (2) (1999) 149–167.
[15] S. Heinrich, Complexity theory of Monte Carlo algorithms, Lect. Appl. Math. 32 (1996) 405–419.
[16] E. Novak, Deterministic and stochastic error bound in numerical analysis, Lect. Notes Math., vol. 1349, Springer, Berlin, Heidelberg, New York, London, Paris, Tokyo, 1988.
[17] J.F. Traub, G.W. Wasilkowski, H. Woźniakowski, Information-based Complexity, Academic Press, New York, 1988.
[18] Ker-I Ko, Computational Complexity of Real Functions, Birkhauser Boston, Boston, MA, 1991.
[19] S.M. Ermakov, G.A. Mikhailov, Statistical Modeling, Nauka, Moscow, 1982.
[20] S.M. Ermakov, V.V. Nekrutkin, A.S. Sipin, Random Processes for Solving Classical Equations of Mathematical Physics, Nauka, Moscow, 1984.
[21] J.M. Hammersley, D.C. Handscomb, Monte Carlo Methods, John Wiley & Sons, inc., New York, London, Sydney, Methuen, 1964.
[22] Yu.A. Shreider, Method of Statistical Testing. Monte Carlo Method, Elsevier Publishing Co., 335 Jan Van Galenstraat, P.O. Box 211, Amsterdam, Netherlands, 1964.
[23] E. Novak, The real number model in numerical analysis, J. Complex. 11 (1995) 57–73.
[24] N.S. Bakhvalov, On the approximate computation of multiple integrals, Vestnik Moscow State University, Ser. Mat., Mech. 4 (1959) 3–18.
[25] N.S. Bakhvalov, Average estimation of the remainder term of quadrature formulas, USSR Comput. Math. Math. Phys. 1 (1) (1961) 64–77.
[26] N.S. Bakhvalov, On the optimal estimations of convergence of the quadrature processes and integration methods, Numerical Methods for Solving Differential and Integral Equations, Nauka, Moscow, 1964.
[27] N.M. Korobov, Approximate computation of multiple integrals with the aid of methods of the theory of numbers, Dokl. Acad. Nauk. SSSR 115 (6) (1957) 1062–1065 (in Russian).
[28] N.M. Korobov, On the Approximate Computation of Multiple Integrals, vol. 4, Vestnik Moskow State University, 1959 (in Russian).
[29] N.S. Bakhvalov, N.P. Zhidkov, G.M. Kobelkov, Numerical Methods, Nauka, Moscow, 1987.
[30] I. Dimov, Efficient and overconvergent Monte Carlo methods. Parallel algorithms, in: I. Dimov, O. Tonev (Eds.), Advances in Parallel Algorithms, IOS Press, Amsterdam, 1994, pp. 100–111.
[31] J.H. Halton, D.C. Handscome, A method for increasing the efficiency of Monte Carlo integrations, J. Assoc. Comput. Machinery 4 (3) (1957) 329–340.
[32] E. Novak, K. Ritter, Optimal stochastic quadrature formulas for convex functions, BIT 34 (1994) 288–294.
[33] K. Sabelfeld, Algorithms of the Method Monte Carlo for Solving Boundary Value Problems, Nauka, Moscow, 1989.
[34] Bl. Sendov, A. Andreev, N. Kjurkchiev, Numerical solution of polynomial equations, in: P.G. Ciarlet, J.L. Lions (Eds.), Handbook of Numerical Analysis, Proc. WP&DP'91 (Sofia, April 16–19), vol. 3, North-Holland, Amsterdam, NY, 1994.
[35] S.A. Smolyak, Quadrature and interpolation formulas for tensor products of certain classes of functions, Sov. Math. Dokl. 4 (1963) 240–243.
[36] I.M. Sobol, On quadratic formulas for functions of several variables satisfying a general Lipschitz condition, USSR Comput. Math. Math. Phys. 29 (6) (1989) 936–941.
[37] V. Dupach, Stochasticke pocetni metody, Cas. Pro. Pest. Mat. 81 (1) (1956) 55–68.
[38] E. Novak, K. Ritter, High dimensional integration of smooth functions over cubes, Numer. Math. (1996) 1–19.
[39] H. Tanaka, H. Nagata, Quasi-Random number method for the numerical integration, Suppl. Progr. Theoret. Phys. (56) (1974) 121–131.

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010

[40] V.N. Temlyakov, On a way of obtaining lower estimates for the errors of quadrature formulas, Math. Sbornik 181 (1990) 1403–1413 (in Russian); English translation: Math. USSR Sbornik, 71 (1992) 247–257.
[41] V.N. Temlyakov, Approximation of Periodic Functions, Nova Science Publishers, 1993.
[42] R. Cranley, T.N.L. Patterson, Randomization of number theoretic methods for multiple integration, SIAM J. Numer. Anal. 13 (1976) 904–914.
[43] P. L'Ecuyer, C. Lecot, B. Tuffin, Randomized quasi-Monte Carlo simulation of Markov chains with an ordered state space, in: H. Niederreiter, D. Talay (Eds.), Monte Carlo and Quasi-Monte Carlo Methods, 2004, Springer-Verlag, 2006, pp. 331–342.
[44] I.H. Sloan, H. Woniakowski, Tractability of multivariate integration for weighted Korobov classes, J. Complexity 17 (4) (2001) 697–721.
[45] P. Davis, P. Rabinowitz, Methods of numerical integration, second edition, Comp. Sci. Appl. Math., Academic Press, 1984.
[46] M. Beckers, A. Haegmans, Transformations of integrands for lattice rules, in: T.O. Espelid, A. Genz (Eds.), Numerical Integration – Recent Developments, Software and Applications, Kluwer, Dordrecht, the Netherlands, 1992, pp. 329–340.
[47] A.R. Krommer, C.W. Ueberhuber, Computational Integration, SIAM, 1998.
[48] S. Maire, C. De Luigi, Quasi-Monte Carlo quadratures for multivariate smooth functions, Appl. Numer. Math. 56 (2006) 146–162.
[49] J.H. Curtiss, Sampling methods applied to differential and difference equations, in: Proc. Seminar on Sci. Comput. IBM, New York, 1949.
[50] I.T. Dimov, Minimization of the probable error for some Monte Carlo methods, in: Proc. of the Summer School on Mathematical Modelling and Scientific Computations, Bulgaria, Sofia, Publ. House of the Bulg. Acad. Sci., 1991, pp. 159–170.
[51] M.H. Kalos, Importance sampling in Monte Carlo calculations of thick shield penetration, Nucl. Sci. Eng. 2 (1) (1959) 34–35.
[52] K.K. Sabelfeld, I.A. Shalimova, Spherical Means for PDE's, VSP, Utrecht, The Netherlands, 1994.
[53] Y. Levitan, N. Markovich, S. Rozin, I. Sobol, On quasi-random sequences for numerical computations, USSR Comput. Math. Math. Phys. 28 (5) (1988) 755–759.
[54] H. Niederreiter, Quasi-Monte Carlo methods and pseudorandom numbers, Bull. Am. Math. Soc. 84 (1978) 957–1041.
[55] H. Niederreiter, Point sets and sequences with small discrepancy, Monatsh. Math. 104 (1987) 273–337.
[56] H. Niederreiter, Random number generation and Quasi-Monte Carlo methods, Number 63, in: CBMS-NSF Series in Applied Mathematics, SIAM, Philadelphia, 1992.
[57] I.M. Sobol, Quasi-Monte Carlo Methods, in: Bl. Sendov, I. Dimov (Eds.), International Youth Workshop on Monte Carlo Methods and Parallel Algorithms – Primorsko, World Scientific, Singapore, 1990, pp. 75–81.
[58] G. Pages, A space vector quantization for numerical integration, J. Comput. Appl. Math. 89 (1997) 1–38.
[59] S. Maire, Reducing variance using iterated control variates, J. Statist. Comput. Simulat. 73 (1) (2003) 1–29.
[60] S. Maire, An iterative computation of approximatons on Korobov-like spaces, J. Comput. Appl. Math. 157 (2003) 261–281.
[61] K.W. Morton, A generalization of the antithetic variate technique for evaluating integrals, J. Math. Phys. 36 (3) (1957) 289–293.
[62] W.P. Press, G.R. Farrar, Recursive stratified sampling for multidimensional Monte Carlo integration, Comput. Phys. 4 (1990) 190–195.
[63] L. Stewart, Hierarchical Bayesian analysis using Monte Carlo integration computing posterior distributions when there are many possible models, Statistician 36 (1987) 211–219.
[64] L. Stewart, W. Davis, Bayesian posterior distribution over sets of possible models with inferences computed by Monte Carlo integration, Statistician 35 (1986) 175–182.
[65] I. Dimov, T. Gurov, Parallel Monte Carlo algorithms for calculation integrals, in: K. Boyanov (Ed.), Proc. WP&DP 1993, Sofia, pp. 426–434.
[66] I.T. Dimov , O.I. Tonev, Monte Carlo numerical methods with overconvergent probable error, Numerical methods and applications, in: Proc. II Int. Conf. on Numerical Methods and Applications, Sofia, Publ. House of the Bulg. Acad. Sci., Sofia, 1989, pp. 116–120.
[67] S.M. Nikolski, Quadrature Formulas, Nauka, Moscow, 1988.
[68] R. Schurer, A comparison between (Quasi-)Monte Carlo and cubature rule based methods for solving high-dimensional integration problems, Math. Comput. Simulat. 62 (2003) 509–517.
[69] K.V. Emelyanov, A.M. Il'in, On the number of arithmetic operations necessary for the approximate solution of Fredholm integral equations of second kind, J. Vychisl. Math. i Math. Phys. 7 (1967) 905–910 (in Russian).
[70] I. Dimov, R. Georgieva, Complexity of a class of Monte Carlo algorithms for integral equations, in: Proc. of the Int. Conf. on MC&QMC Methods, June 7–10, 2004 Juan-les-Pins, Côte d'Azur, France, 2004.
[71] I.T. Dimov, A.N. Karaivanova, Iterative Monte Carlo algorithms for linear algebra problems, in: L. Vulkov, J. Wasniewski, P. Yalamov (Ed.), Lecture Notes in Computer Science, Springer-Verlag, Berlin, 1196, pp. 150–160.
[72] I.T. Dimov, V.N. Alexandrov, A.N. Karaivanova, Parallel resolvent Monte Carlo algorithms for linear algebra problems, Math. Comput. Simulat. 55 (2001) 25–35.
[73] V.I. Krylov, V.G. Bobkov, P. Monastyrni, Numerical Methods, Nauka, Moscow, 1977.
[74] L.V. Kantorovich, V.I. Krylov, Approximate Methods of Higher Analysis, Physical and Mathematical State Publishing House, Leningrad, 1962.

Please cite this article in press as: E. Atanassov, I.T. Dimov, What Monte Carlo models can do and cannot do efficiently?, Appl. Math. Modell. (2007), doi:10.1016/j.apm.2007.04.010