STOCHASTIC METHODS: MONTE CARLO APPROACH

Ivan Dimov

Contents

Chapter 1

INTRODUCTION

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 the functional of the solution by performing random sampling of a certain random variable whose mathematical expectation is the desired functional.

The Monte Carlo method is a method for solving problems using random variables. In the book [Sh66] edited by Yu.A.Shreider one can find the following definition of the Monte Carlo method.

Definition 1.0.1 *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.*

Usually Monte Carlo methods reduce problems to the approximate calculation of mathematical expectations. Let the variable J be the desired solution of the problem or some desired linear functional of the solution. A random variable 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 to J

$$J \approx \frac{1}{n}(\xi_1 + \xi_2 + \ldots + \xi_n), \tag{1.1}$$

can then be computed.

There are many algorithms using this essential idea for solving a wide range of problems.

The year 1949 is generally regarded as the official birthday of the Monte Carlo method when the paper of Metropolis and Ulam [MU49] was published, although some authors point to earlier dates. Ermakov [Er75], for example, notes that a solution of a problem by the Monte Carlo method is contained in the Old Testament. The development of the method is connected with the names of John von Neumann, E. Fermi and G. Kahn, who worked at Los Alamos (USA) for forty years. 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.

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, at only one point of the domain [So73], [DT93]. This is very important for some problems of applied science. Often, one does not need to know the solution on the whole domain in which the problem is defined. Usually, it is only necessary to know the value of some functional of the solution. Problems of this kind can be found in many areas of the applied sciences. For example, in the statistical physics, one is interested in computing linear functional 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. This is because these algorithms are inherently parallel and have minimum dependency. In addition, they are also naturally vectorizable when powerful vector machines are used. Nevertheless, the problem of parallelization of the Monte Carlo algorithms is not trivial 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.

Monte Carlo algorithms have proved to be very efficient in solving multidimensional integrals in composite domains [So73], [DT93a], [Ha66]. The problem of evaluating integrals of high dimension is important since it appears in many applications of control theory, statistical physics and mathematical economics. For example one of the numerical approaches for solving stochastic systems in control theory leads to a large number of multi-dimensional 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.* Here 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 interactions, 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 usually used for solving *deterministic* problems by modeling random variables or random fields. The main idea here 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 way to create such an artificial process. After finding such a process one needs to define an algorithm for computing realizations of the random variable. Usually, the random variable 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 realizations of the random variables.

Here the following important problems arise:

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

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

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

Let us introduce some notations and definitions used in the course:

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

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 $(h, u) =$

$\int_\Omega h(x)u(x)dx$. $J(u)$ 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 Ω are denoted by $C(\Omega)$. $C^{(k)}(\Omega)$ is the space of functions u for which $u^{(k)} \in C(\Omega)$. As usual $\| f \|_{\mathbf{L}_q} = (\int_\Omega f^q(x)p(x)dx)^{1/q}$ denotes the $\mathbf{L}_q$-norm of $f(x)$.

Definition 1.0.2 *$H^\alpha(M,\Omega)$ is the space of functions for which $|f(x) - f(x')| \leq M|x - x'|^\alpha$.*

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

Definition 1.0.3 *$\mathbf{W}^r(M;\Omega)$ is a a class of functions $f(x)$, continuous on Ω with partially continuous r^{th} derivatives, such that*

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

where

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

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

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

Definition 1.0.4 *The $\mathbf{W}_q^r$-norm is defined as:*
$\| f \|^r_{\mathbf{W}_q} = [\int_\Omega (D^r f(x))^q p(x)dx]^{1/q}$.

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

Definition 1.0.5 *$Lu(x) = \int_\Omega l(x,x')u(x')dx'$ is an integral transformation (L is an integral operator)*

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

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

The probability that $\varepsilon_k = i$ will be denoted by $Pr\{\varepsilon_k = i\}$, while the random variable (r.v.) will be denoted by $\theta(\xi)$.

The mathematical expectation of the r.v. θ will be denoted by $E(\theta)$ (sometimes abbreviated to $E\theta$); the variance by $D(\theta)$ (or $D\theta$) and the standard deviation by $\sigma(\theta)$ (or $\sigma\theta$). 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 realizations (values) of the random points ξ or By $\xi_i (i = 1, 2, \dots, n)$ we denote the realizations of the random point ξ or θ by ξ_i, $\theta (i = 1, 2, \dots, n)$ respectively. The density (frequency) function will be denoted by $p(x)$ and the transition density function by $p(x, y)$. $F(x)$ will denote the distribution function. Finally the mean value of n realizations of the r.v. ξ will be denoted by

$$\bar{\xi}_n = \frac{1}{n} \sum_{i=1}^{n} \xi_i.$$

Definition 1.0.6 *If J is the exact solution of the problem, then the probable error r_n is the value for which:*

$$Pr\left\{|\bar{\xi}_n - J| \leq r_n\right\} = \frac{1}{2} = Pr\left\{|\bar{\xi}_n - J| \geq r_n\right\}.$$

The computational problem in Monte Carlo algorithms becomes one of calculating repeated realizations of the r.v. θ and of combining them into an appropriate statistical estimator of the functional $J(u)$ or solution. Every realization of θ is a Markov process. We shall consider only *discrete Markov processes with a finite set of states*, the so-called *finite discrete Markov chains.*

Definition 1.0.7 *A finite discrete Markov chain T_i is defined as a finite set of states $\{k_1, k_2, ..., k_i\}$.*

At each of the sequence of times $t = 0, 1, \dots, i, \dots$ 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 1.0.8 *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*:

$$T = k_{t_0} \to k_{t_1} \to k_{t_2} \to \dots \to k_{t_i}, \tag{1.2}$$

where k_{t_q}, $(q = 1, \dots, i)$ is one of the absorbing states. This determines the value of some function $F(T) = J(u)$, which depends on the sequence (1.2). The function $F(T)$ is a random variable. After the value of $F(T)$ has been calculated, the system is restarted at its initial state 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

$$\frac{1}{n}\sum_{T} F(T) \tag{1.3}$$

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

We also will be interested in *computational complexity.*

Definition 1.0.9 *Computational complexity is defined by*

$$S_n = nE(q)t_0,$$

where $E(q)$ is the mathematical expectation of the number of transitions in the sequence (1.2) and t_0 is the mean time needed for realization of one transition.

In practice, the definition of computational complexity is only used for obtaining theoretical estimates, because one can only estimate the mathematical expectation of the number of transitions q. Now if for the s^{th} realization of a Markov chain there are q_s transitions then the total number of transitions for n realizations is N, where

$$N = \sum_{s=1}^{n} q_s. \tag{1.4}$$

Suppose there exists different Monte Carlo algorithms to solve a given problem. The computational effort for the achievement of a preset probable error is proportional to $t\sigma^2(\theta)$, where t is the expectation of the time required to calculate one realization of the random variable θ. The product $t\sigma^2(\theta)$ may be considered as *computational complexity*, whereas $[t\sigma^2(\theta)]^{-1}$ is a measure for the *computational efficiency.* In these terms the optimal Monte Carlo algorithm is the algorithm with the lowest computational complexity (or the algorithm with the highest computational efficiency).

Let us now consider a simple example of evaluating multi-dimensional integrals which demonstrates the power of the Monte Carlo algorithms. Consider the classical problem of integral evaluation. Suppose $f(x)$ is a continuous function and let a quadrature formula of Newton or Gauss type be used for calculating the integrals. Consider an example with $d = 30$ (this is a typical number for some applications in control theory, statistical physics 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 $\Omega = C^d = [0,1]^d$. 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 remind 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 $f(x) \in W^{(2)}(M; [0,1]^d)$. If the formula of rectangles (or some similar formula) is applied then the error in the approximate integral calculation is

$$\varepsilon \leq cMh^3, \;\; (h = 0.1), \tag{1.5}$$

where h is the mesh-size and c is a constant independent of h.

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 Ω and to calculate the values of $f(x)$ at these points. For each uniformly distributed random point in Ω we have to generate 30 random numbers uniformly distributed in $[0,1]$.

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

$$\varepsilon \leq 0.6745\sigma(\theta)\frac{1}{\sqrt{n}}, \tag{1.6}$$

where $\sigma(\theta) = (D\theta)^{1/2}$ is the standard deviation of the random variable θ for which:

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

n is the number of the realizations of the random variable (in this case it coincides with the number of random points generated in Ω).

We can estimate the probable error using the variance properties:

$$\begin{aligned}\varepsilon &\leq 0.6745\sigma(\theta)\frac{1}{\sqrt{n}} \\ &\approx 0.6745\left(\int_\Omega f^2(x)p(x)dx - \left(\int_\Omega f(x)p(x)dx\right)^2\right)^{1/2}\frac{1}{\sqrt{n}} \\ &\leq 0.6745\left(\int_\Omega f^2(x)p(x)dx\right)^{1/2}\frac{1}{\sqrt{n}} = 0.6745 \parallel f \parallel_{L_2}\frac{1}{\sqrt{n}}.\end{aligned}$$

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 (1.5) (1.6), we may conclude:

$$n \approx \left(\frac{0.6745 \parallel f \parallel_{L_2}}{cM}\right)^2 \times h^{-6}.$$

Suppose that the expression in front of h^{-6} is of order 1. (For many problems it is significantly less than 1 as M is often the maximal value of the second derivative;

further the Monte Carlo algorithm can be applied even when it is infinity). For our example ($h = 0.1$), we have

$$n \approx 10^6;$$

hence, it will be necessary to generate $30 \times 10^6 = 3.10^7$ pseudo random numbers (PRN). Usually, two operations are sufficient to generate a single PRN. Suppose that the time required to generate one PRN is the same as that for calculating the value of the function at one point in the domain Ω. 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 above result may be improved if additional realistic restrictions are placed upon the integrand $f(x)$.

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 [Ni88], [SAK94], [Ba64], [DT89], [DT93].

For example, as it will be shown in Sections 2.5 and 2.9 if $f(x) \in W^1(M; [0,1]^d)$, then instead of (1.6) we have

$$\varepsilon \leq c_1 M \frac{1}{n^{1/2+1/d}}, \tag{1.7}$$

where c_1 is a constant independent of n and d.

Chapter 2

SOME BASIC FACTS ABOUT MONTE CARLO INTEGRATION

It this chapter some basic facts about Monte Carlo integration are considered. Almost all facts presented here are well known to scientists dealing with Monte Carlo algorithms. Nevertheless, this exposition facilitates the understanding of the algorithms discussed in the course. In this chapter we present results from basic works on the theory of Monte Carlo algorithms [Cu54], [Cu56], [Du56], [EM82], [HH64], [Ka50], [MU49], [Mi87], [Sa89], [So73].

2.1 Convergence and Error Analysis of Monte Carlo Algorithms

Let ξ be a random variable for which a mathematical expectation of $E\xi = I$ exists. Let us formally define

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

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

To approximate the variable I, a computation of the arithmetic mean must usually be carried out,

$$\bar{\xi}_n = \frac{1}{n} \sum_{i=1}^{n} \xi_i.$$

For a sequence of uniformly distributed independent random variables, for which a mathematical expectation exists, the Hinchin theorem (the minimum of large numbers) (Kramer [1948]) is valid. This means that the arithmetic mean of these variables converges to the mathematical expectation when

$$\xi_n \xrightarrow{p} I \quad \text{as} \quad n \to \infty$$

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

$$\lim_{n\to\infty} P\{|\eta_n - c| \geq h\} = 0.)$$

Thus, when n is sufficiently large

$$\xi_n \approx I \tag{2.1}$$

and (1.1) may be used whenever $E\xi = I$ exists.

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

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

It is well known that the sequences of the uniformly distributed independent random variables with finite variances satisfy the *central limit theorem* (see Hammersley and Handscomb [HH64]). This means that for arbitrary x_1 and x_2

$$\lim_{n\to\infty} P\left\{x_1 < \frac{1}{\sqrt{nD\xi}} \sum_{i=1}^{n} (\xi_i - I) < x_2\right\} = \frac{1}{\sqrt{2\pi}} \int_{x_1}^{x_2} e^{-\frac{t^2}{2}} dt. \tag{2.2}$$

Let $x_2 = -x_1 = x$. Then from (2.2) it follows that

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

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

When n is sufficiently large

$$P\left\{|\bar{\xi}_n - I| < x\sqrt{\frac{D\xi}{n}}\right\} \approx \Phi(x). \tag{2.3}$$

Formula (2.3) gives a whole family of estimates, which depend on the parameter x. If a probability β is given then the root $x = x_\beta$ of the equation

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

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

Then from (2.3) it follows that the probability of the inequality

$$|\bar{\xi}_n - I| < x_\beta\sqrt{\frac{D\xi}{n}} \tag{2.4}$$

is approximately equal to β.

The term on the right-hand side of the inequality (2.4) is called the *probability error.*

The symbol r_n is often used to denote the *probable error.* This is the value r_n for which

$$Pr\left\{|\overline{\xi}_n - I| \leq r_n\right\} = \frac{1}{2} = Pr\left\{|\overline{\xi}_n - I| \geq r_n\right\}. \tag{2.5}$$

From (2.5) it follows that

$$r_n = x_{0.5}\sigma(\xi)n^{-\frac{1}{2}}, \tag{2.6}$$

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

2.2 Integral Evaluation

2.2.1 A plain Monte Carlo algorithm

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

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

$$I = \int_\Omega f(x)p(x)dx, \tag{2.7}$$

where the non-negative function $p(x)$ is called the density function if $\int_\Omega p(x)dx = 1$. (Note that every integral $\int_\Omega f(x)dx$, when Ω is a bounded domain, is an integral of the kind (2.7). In fact, if S_Ω is the area of Ω, then $p_1(x) \equiv \frac{1}{S_\Omega}$ is the *probability density function* of a random point which is uniformly distributed in Ω. Let us introduce the function $f_1(x) = S_\Omega f(x)$. Then, obviously, $\int_\Omega f(x)dx = \int_\Omega f_1(x)p_1(x)dx$.)

Let x be a random point with probability density function $p(x)$. Introducing the random variable $\theta = f(x)$ with mathematical expectation equal to the value of the integral I, then

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

Let $x_1, x_2, \ldots, x_n$ be independent realizations of the random point x with probability density function $p(x)$ and $\theta_1 = f(x_1), \ldots, \theta_n = f(x_n)$. Then an approximate value of I is

$$\overline{\theta}_n = \frac{1}{n}\sum_{i=1}^{n}\theta_i. \tag{2.8}$$

According to Section 2.1, if the integral (1.1) were absolutely convergent, then $\overline{\theta}_n$ would be convergent to I.

2.2.2 The geometrical Monte Carlo algorithm

Let the nonnegative function f be bounded, i.e.,

$$0 \leq f(x) \leq c \quad \text{for} \quad x \in \Omega, \tag{2.9}$$

where c is a generic constant.

Consider the cylindrical domain

$$\tilde{\Omega} = \Omega \times [0, c]$$

and the random point $\tilde{x} \equiv (x_{(1)}, x_{(2)}, x_{(3)}) \subset \tilde{\Omega}$ with the following probability density function:

$$\tilde{p}(x) = \frac{1}{c} p(x_{(1)}, x_{(2)}).$$

Let $\tilde{x_1}, \dots, \tilde{x_n}$ be independent realizations of the random point $\tilde{x}$. Introduce the random variable $\tilde{\theta}$, whose dependency on $\tilde{x}$ is clear,

$$\tilde{\theta} = \begin{cases} c, & \text{if} \quad x_{(3)} < f(x_{(1)}, x_{(2)}) \\ 0, & \text{if} \quad x_{(3)} \geq f(x_{(1)}, x_{(2)}). \end{cases}$$

The random variable introduced is a measure of the points below the graph of the function f. Let us calculate $E\tilde{\theta}$:

$$E\tilde{\theta} = cPr\{x_{(3)} < f(x)\} = \int_{\Omega} dx_{(1)} dx_{(2)} \int_0^{f(x_{(1)}, x_{(2)})} \tilde{p}(x_{(1)}, x_{(2)}, x_{(3)}) dx_{(3)} = I.$$

The absolute convergence of the integral follows from (2.9). Therefore,

$$\tilde{\theta}_n = \frac{1}{n} \sum_{i=1}^{n} \tilde{\theta}_i$$

can be used as an approximation to I, since $E\tilde{\theta}_n = I$ and $\tilde{\theta}_n \xrightarrow{p} I$.

2.3 Computational Complexity of Monte Carlo Algorithms

Let us compare the accuracy of the *geometrical* and the *plain* Monte Carlo algorithm.

Let $f \in L_2(\Omega, p)$. This guarantees that the variance

$$D\tilde{\theta} = \int_\Omega f^2(x)p(x)dx - I^2$$

in a plain Monte Carlo algorithm is finite.

For the geometrical Monte Carlo algorithm the following equation holds

$$E(\tilde{\theta}^2) = c^2 P\{x_{(3)} < f(x)\} = cI.$$

Hence the variance is

$$D\tilde{\theta} = cI - I^2,$$

and

$$\int_\Omega f^2(x)p(x)dx \le c\int_\Omega f(x)p(x)dx = cI.$$

Therefore $D\theta \le D\tilde{\theta}$.

The last inequality shows that the plain Monte Carlo algorithm is more accurate than the geometrical one (except for the case when the function f is a constant). Nevertheless, the geometrical algorithm is often preferred from the algorithmic point of view, because its computational complexity is less than the computational complexity of the plain algorithm [So73]. The problem of the computational complexity of different Monte Carlo algorithms is considered in detail in [DT93a].

2.4 Monte Carlo Algorithms with Reduced Error

As has been shown, the probable error in Monte Carlo algorithms when no information about the smoothness of the function is used is

$$r_n = c\sqrt{\frac{D\xi}{n}}.$$

It is important for such computational schemes and random variables that a value of ξ is chosen so that the variance is as small as possible. Monte Carlo algorithms with reduced variance compared to plain Monte Carlo algorithms are usually called *efficient Monte Carlo algorithms.* Let us consider several algorithms of this kind.

2.4.1 Separation of the principal part

Consider again the integral

$$I = \int_\Omega f(x)p(x)dx, \tag{2.10}$$

where $f \in L_2(\Omega, p),\ x \in \Omega \subset I\!R^d$.

Let the function $h \in L_2(\Omega, p)$ be *close* to f with respect to the L_2 norm; i.e. $\|f - h\|_{L_2} \leq \varepsilon$, and the value of the integral

$$\int_\Omega h(x)p(x)dx = I'$$

be known.

The random variable $\theta' = f(x) - h(x) + I'$ generates the following estimate for the integral (2.10)

$$\theta'_n = I' + \frac{1}{n}\sum_{i=1}^{n}[f(x_i) - h(x_i)].$$

Obviously $E\theta'_n = I$. A possible estimate of the variance of θ' is

$$D\theta' = \int_\Omega [f(x) - h(x)]^2 p(x)dx - (I - I')^2 \leq \varepsilon^2.$$

This means that the variance and the probable error can be quite small, if the function h is such that the integral I' can be calculated analytically. The function h is often chosen to be piece-wise linear function.

2.4.2 Integration on subdomain

Let us suppose that it is possible to calculate the integral analytically on $\Omega' \subset \Omega$ and

$$\int_{\Omega'} f(x)p(x)dx = I', \quad \int_{\Omega'} p(x)dx = c,$$

where $0 < c < 1$.

Then the integral (2.10) can be represented as

$$I = \int_{\Omega_1} f(x)p(x)dx + I',$$

where $\Omega_1 = \Omega - \Omega'$.

Let us define in Ω_1 a random point x' with probability density function $p_1(x) = p(x')/(1-c)$ and a random variable

$$\theta' = I' + (1-c)f(x').$$

Obviously $E\theta' = I$. Therefore, the following approximate estimator can be used to compute I

$$\theta'_n = I' + \frac{1}{n}(1+c)\sum_{i=1}^{n} f(x'_i),$$

where x'_i are independent realizations of the d-dimensional random point x'.

The next theorem compares the accuracy of this Monte Carlo algorithm with the plain Monte Carlo.

Theorem 2.4.1 *(Sobol [So73]). If the variance $D\theta$ exists then*

$$D\theta' \leq (1-c)D\theta.$$

P r o o f. Let us calculate the variances of θ and θ'

$$D\theta = \int_{\Omega} f^2 p\, dx - I^2 = \int_{\Omega_1} f^2 p\, dx + \int_{\Omega'} f^2 p\, dx - I^2; \tag{2.11}$$

$$D\theta' = (1-c)^2 \int_{\Omega_1} f^2 p_1\, dx - [(1-c)\int_{\Omega_1} f p_1\, dx]^2$$

$$= (1-c)\int_{\Omega_1} f^2 p\, dx - \left(\int_{\Omega_1} fp\, dx\right)^2. \tag{2.12}$$

Multiplying both sides of (2.11) by $(1-c)$ and subtracting the result from (2.12) yields

$$(1-c)D\theta - D\theta' = (1-c)\int_{\Omega'} f^2 p\, dx - (1-c)I^2 - (I-I')^2.$$

Using the nonnegative value

$$b^2 \equiv \int_{\Omega'} \left(f - \frac{I'}{c}\right)^2 p(x)dx = \int_{\Omega'} f^2 p\, dx - \frac{I'^2}{c},$$

one can obtain the following inequality

$$(1-c)D\theta - D\theta' = (1-c)b^2 + (\sqrt{c}I' - I'/\sqrt{c})^2 \geq 0$$

and the theorem is proved. $\diamondsuit$

2.4.3 Symmetrization of the integrand

For a one-dimensional integral

$$I_0 = \int_a^b f(x)dx$$

on a finite interval $[a,b]$ let us consider the random variables ξ (uniformly distributed in this interval) and $\theta = (b-a)f(\xi)$. Since $E\theta = I_0$, the plain Monte Carlo algorithm leads to the following approximate estimate for I_0:

$$\overline{\theta}_n = \frac{b-a}{n}\sum_{i=1}^{n} f(\xi_i),$$

where ξ_i are independent realizations of ξ.

Consider the symmetric function

$$f_1(x) = \frac{1}{2}[f(x) + f(a+b-x)],$$

the integral of which over $[a,b]$ is equal to I_0. Let also $\theta' = (b-a)f_1(x)$.

Since $E\theta' = I_0$, the following symmetrized approximate estimate of the integral may be employed:

$$\overline{\theta}_n = \frac{b-a}{n}\sum_{i=1}^{n}[f(\xi_i) + f(a+b-\xi_i)].$$

Theorem 2.4.2 *(Sobol [So73]). If the partially continuous function f is monotonic in the interval $a \leq x \leq b$, then*

$$D\theta' \leq \frac{1}{2}D\theta.$$

Proof. The variances of θ and θ' may be expressed as

$$D\theta = (b-a)\int_a^b f^2(x)dx - I_0^2, \tag{2.13}$$

$$2D\theta' = (b-a)\int_a^b f^2 dx + (b-a)\int_a^b f(x)f(a+b-x)dx - I_0^2. \tag{2.14}$$

From (2.13) and (2.14) it follows that the assertion of the theorem is equivalent to establishing the inequality

$$(b-a)\int_a^b f(x)f(a+b-x)dx \leq I_0^2. \tag{2.15}$$

Without loss of generality suppose that f is nondecreasing and $f(b) > f(a)$ and introduce the function

$$v(x) = (b-a)\int_a^x f(a+b-t)dt - (x-a)I_0,$$

which is equal to zero at the points $x = a$ and $x = b$. The derivative of v

$$v'(x) = (b-a)f(a+b-x) - I_0$$

is monotonic and since

$$v'(a) > 0,\ v'(b) < 0, \text{ we see that } v(x) \geq 0$$

for $x \in [a,b]$. Obviously,

$$\int_a^b v(x)f'(x)dx \geq 0. \tag{2.16}$$

Thus integrating (2.16) by parts, one obtains

$$\int_a^b f(x)v'(x)dx \le 0. \tag{2.17}$$

Now (2.15) follows by replacing the expression for $v'(x)$ in (2.17). (The case of a non-increasing function f can be treated analogously.) $\diamondsuit$

2.4.4 A key sampling algorithm

Consider the problem of computing

$$I_0 = \int_\Omega f(x)dx, \quad x \in \Omega \subset I\!R^d.$$

Let Ω_0 be the set of points x for which $f(x) = 0$ and $\Omega_+ = \Omega - \Omega_0$.

Definition 2.4.1 *Define the probability density function $p(x)$ to be tolerant to $f(x)$, if $p(x) > 0$ for $x \in \Omega_+$ and $p(x) \ge 0$ for $x \in \Omega_0$.*

For an arbitrary tolerant probability density function $p(x)$ for $f(x)$ in Ω let

$$\theta_0(x) = \begin{cases} \frac{f(x)}{p(x)}, & x \in \Omega_+, \\ 0, & x \in \Omega_0. \end{cases}$$

The following **problem** arises: *find the tolerant density $p(x)$ which minimises the variance of θ_0?*

Theorem 2.4.3 *(Kahn [Ka50]). The probability density function $\hat{p} = c|f(x)|$ minimises $D\theta_0$ and the value of the minimum variance is*

$$D\hat{\theta}_0 = \left[\int_\Omega |f(x)|dx\right]^2 - I_0^2.$$

P r o o f. Let us note that the constant in the expression for $\hat{p}(x)$ is

$$c = \left[\int_\Omega |f(x)|dx\right]^{-1},$$

because the condition for normalisation of probability density must be satisfied. At the same time

$$D\theta_0 = \int_{G_+} \frac{f^2(x)}{p(x)}dx + I_0^2 = D\hat{\theta}_0.$$

It is necessary only to prove that for other tolerant probability density functions the inequality $p(x)D\theta_0 \ge D\hat{\theta}_0$ holds. Indeed

$$\left[\int_\Omega |f(x)|dx\right]^2 = \left[\int_{\Omega_+} |f|dx\right]^2 = \left[\int_{\Omega_+} |f|p^{-1/2}p^{1/2}dx\right]^2$$

$$\leq \int_{\Omega_+} f^2p^{-1}dx \int_{\Omega_+} p\, dx \leq \int_{\Omega_+} \frac{f^2}{p}dx.$$

$\diamondsuit$

Corollary: *If f does not change sign in Ω, then $D\hat{\theta}_0 = 0$.*

This is clear.

For practical algorithms this assertion allows random variables with a small variances (and consequently small probable errors) to be incurred, using a higher random point probability density in subdomains of Ω, where the integrand has a large absolute value. It is intuitively clear that the use of such an approach should increase the accuracy of the algorithm.

2.5 Superconvergent Monte Carlo Algorithms

As was shown earlier, the probable error usually has the form of (2.6) and the speed of convergence can be increased if an algorithm with a probable error

$$r_n = cn^{-1/2-\varepsilon(d)}$$

can be constructed, where c is a constant, $\varepsilon(d) > 0$ and d is the dimension of the space.

Monte Carlo algorithms with such a probable error are called Monte Carlo algorithms with a *superconvergent probable error.*

Let us consider the problem of computing the integral

$$I = \int_\Omega f(x)p(x)dx,$$

where $\Omega \in I\!R^d$, $f \in L_2(\Omega; p)$ and p is a probability density function, i.e. $p(x) \geq 0$ and $\displaystyle\int_\Omega p(x)dx = 1$.

Let Ω be the unit cube

$$\Omega = C^d = \{0 \leq x_{(i)} < 1; \quad i = 1, 2, \ldots, d\}.$$

Let $p(x) \equiv 1$ and consider the partition of Ω into the subdomains $\Omega_j, j = 1, 2, \ldots, m$, of $n = m^d$ equal cubes with edge $1/m$ (evidently $p_j = 1/n$ and $d_j = \sqrt{d}/m$) so that the following conditions hold:

$$\Omega = \bigcup_{j=1}^{m} \Omega_j, \quad \Omega_i \cap \Omega_j = \emptyset, \quad i \neq j,$$

$$p_j = \int_{\Omega_j} p(x)dx \leq \frac{c_1}{n}, \tag{2.18}$$

and

$$d_j = \sup_{x_1,x_2\in\Omega_j} |x_1 - x_2| \leq \frac{c_2}{n^{1/d}}, \tag{2.19}$$

where c_1 and c_2 are constants.

Then $I = \sum_{j=1}^{m} I_j$, where $I_j = \int_{\Omega_j} f(x)p(x)dx$ and obviously I_j is the mean of the random variable $p_j f(\xi_j)$, where ξ_j is a random point in Ω_j with probability density function $p(x)/p_j$. So it is possible to estimate I_j by the average of n_j observations

$$\overline{\theta}_n = \frac{p_j}{n_j}\sum_{s=1}^{n_j} f(\xi_j), \quad \sum_{j=1}^{m} n_j = n,$$

and I by $\theta_n^* = \sum_{j=1}^{m} \overline{\theta}_{n_j}$.

Theorem 2.5.1 *(Dupac [Du56], Haber [Ha66]).*

Let $n_j = 1$ for $j = 1,\dots,m$ (so that $m = n$). The function f has continuous and bounded derivatives $\left(\left|\frac{\partial f}{\partial x_{(k)}}\right| \leq L \text{ for every } k = 1,2,\dots,d\right)$ and let there exist constants c_1, c_2 such that conditions (2.18) and (2.19) hold.

Then for the variance of θ^* the following relation is fulfilled

$$D\theta_n^* = (dc_1c_2L)^2 n^{-1-2/n}.$$

Using the Chebishev's inequality (see, [So73]) it is possible to obtain

$$r_n = \sqrt{2}dc_1c_2Ln^{-1/2-1/d}.$$

The Monte Carlo algorithm constructed above has a superconvergent probable error, but the conditions of the last theorem are strong. So, the following **problem** arises: *Is it possible to obtain the same result but for functions that are only continuous?*

Let us consider the problem in $I\!R^1$. Let $[a,b]$ be partitioned into n subintervals $[x_{j-1}, x_j]$ and let $d_j = |x_j - x_{j-1}|$. Then if ξ is a random point in $[x_{j-1}, x_j]$ with probability density function $p(x)/p_j$, where $p_j = \int_{x_{j-1}}^{x_j} p(x)dx$, the probable error of the estimator θ_n^* is given by the following:

Theorem 2.5.2 *(Dimov & Tonev [DT89]).*

Let f be continuous in $[a,b]$ and let there exist positive constant c_1, c_2, c_3 satisfying $p_j \leq c_1/n, c_3 \leq d_j \leq c_2/n$ for $j = 1,2,\dots,n$. Then

$$r_n \leq 4\sqrt{2}\frac{c_1 c_2}{c_3}\tau\left(f; \frac{3}{2}d\right)_{L_2} n^{-3/2},$$

where $d = \max_j d_j$ and $\tau(f;\delta)_{L_2}$ is the averaged modulus of smoothness, i.e.

$$\tau(f;\delta)_{L_2} = \| \omega(f, \bullet; \delta) \|_{L_2} = \left(\int_a^b (\omega(f,x;\delta))^q dx\right)^{1/q}, \ 1 \leq q \leq \infty,$$

$\delta \in [0, (b-a)]$ and

$$\omega(f,x;\delta) = sup\{|\Delta_h f(t)| : t, t+h \in [x-\delta/2, x+\delta/2] \cap [a,b]\}.$$

where Δ_h is the restriction operator.

In $I\!R^d$ the following theorem holds:

Theorem 2.5.3 *(Dimov & Tonev [DT89]).*
Let f be continuous in $\Omega \subset I\!R^d$ and let there exist positive constants c_1, c_2, c_3 such that $p_j \leq c_1/n$, $d_j \leq c_2 n^{1/d}$ and $S_j(\bullet, c_3) \subset \Omega_j$, $j = 1, 2, \ldots, n$, where $S_j(\bullet, c_3)$ is a sphere with radius c_3. Then

$$r_n \leq 4\sqrt{2}\frac{c_1 c_2}{c_3}\tau(f; d)_{L_2} n^{-1/2-1/d}.$$

The result [DT89] was improved by Takev [Ta92]. His result is

$$r_n = O(\omega(f, n^{-1})_{L_2})n^{-1/2}.$$

Let us note that the best quadrature formula with fixed nodes in $I\!R^1$ in the sense of Nikolskii [Ni88] for the class of functions $W^{(1)}(l; [a,b])$ is the rectangular rule with equidistant nodes, for which the error is approximately equal to c/n. For the Monte Carlo algorithm given by Theorem 2.5.3 when $n_j = 1$ the rate of convergence is improved by an order of 1/2. This is essentially due to the fact that the randomisation of the quadrature formula (random nodes) increases the rate of convergence by factor of 1/2. At the same time, the estimate given in Theorem 2.5.2 for the rate of convergence attains the lower bound estimate obtained by Bachvalov ([Ba59]), ([Ba61]) (see, also [Ba64], for the error of an arbitrary random quadrature formula for the class of continuous functions in an interval $[a,b]$.

2.6 Random Interpolation Quadrature

If the quadrature formula for computing the integral

$$I = \int_\Omega f(x)p(x)dx, \quad \Omega \subset I\!\!R^d, \quad p(x) \geq 0, \int_\Omega p(x)dx = 1$$

is denoted by the expression

$$I \approx \sum_{j=1}^{n} c_j f(x_j), \tag{2.20}$$

where $x_1, \dots, x_n \in \Omega$ are random nodes and $c_1, \dots, c_n$ are weights, then the random quadrature formula can be written in the following form:

$$I \approx \sum_{j=1}^{n} k_j f(x_j), \tag{2.21}$$

where $x_1, \dots, x_n \in \Omega$ are random nodes and $k_1, \dots, k_n$ are random weights.

The random quadrature formulae (2.21) considered above are a very special case of the (2.20) with fixed (non-random) weights.

All functions considered in this Section are supposed to be partially continuous and belong to the space $L_2(\Omega)$.

Let $\varphi_0, \varphi_1, \dots, \varphi_m$ be a system of orthonormal functions, such that

$$\int_\Omega \varphi_k(x)\varphi_j(x)dx = \delta_{kj}.$$

For $p(x) = \varphi_0(x)$ an approximate solution for the integral

$$I = \int_\Omega f(x)\varphi_0(x)dx \tag{2.22}$$

can be found using a quadrature formula of the type (2.20).

Let us fix arbitrary nodes $x_0, x_1, \dots, x_m \quad (x_i \neq x_j, i \neq j)$ and choose the weights $c_0, c_1, \dots, c_m$ such that (2.20) is exact for the system of orthonormal functions $\varphi_0, \varphi_1, \dots, \varphi_m$. In this case it is convenient to represent the quadrature formula (2.20) as a ratio of two determinants

$$I \approx \frac{W_f(x_0, x_1, \dots, x_m)}{W_{\varphi_0}(x_0, x_1, \dots, x_m)},$$

where

$$W_g(x_0, x_1, \dots, x_m) = \begin{vmatrix} g(x_0) & \varphi_1(x_0) & \dots & \varphi_m(x_0) \\ g(x_1) & \varphi_1(x_1) & \dots & \varphi_m(x_1) \\ \vdots & \vdots & & \vdots \\ g(x_m) & \varphi_1(x_m) & \dots & \varphi_m(x_m) \end{vmatrix}. \tag{2.23}$$

It is easy to check that if $W_{\varphi_0} \neq 0$ then the formula (2.22) is exact for every linear combination of the kind

$$f = a_0\varphi_0 + \dots + a_m\varphi_m.$$

Theorem 2.6.1 *(Sobol [So73]).*
Let $\varphi_0, \varphi_1, \dots, \varphi_m$ be an arbitrary set of orthonormal functions in Ω. Then

$$\int_\Omega \dots \int_\Omega W^2_{\varphi_0} dx_0 \dots dx_m = (m+1)!.$$

Theorem 2.6.2 *(Sobol [So73]).*
Let $\varphi_0, \varphi_1, \dots, \varphi_m, \psi$ be an arbitrary set of orthonormal functions in Ω. Then

$$\int_\Omega \dots \int_\Omega W_{\varphi_0} W_\varphi dx_0 \dots dx_m = \mathbf{0}.$$

For brevity denote by t the $d(m+1)$-dimensional points of the domain

$$B \equiv \underbrace{\Omega \times \Omega \times \dots \times \Omega}_{m+1 \text{ times}}$$

so that $dt = dx_0 ... dx_m$; let B_0 be the set of points t for which $W_{\varphi_0} = 0$ and let $B_+ = B - B_0$. Let us consider the random point $\xi_0, \dots, \xi_m$ in Ω and consider the random variable

$$\hat{\theta}[f] = \begin{cases} \frac{W_f(\xi_0,\xi_1,\dots,\xi_m)}{W_{\varphi_0}(\xi_0,\xi_1,\dots,\xi_m)}, & if\ (\xi_0, \xi_1, \dots, \xi_m) \in B_+, \\ 0, & \text{if } (\xi_0, \xi_1, \dots, \xi_m) \in B_0. \end{cases} \tag{2.24}$$

as an approximate value for (2.22)

Theorem 2.6.3 *(Ermakov and Zolotuchin [EZ60], Ermakov [Er67])*
If the joint probability density function of the random points $\xi_0, \xi_1, \dots, \xi_m$ in B is

$$p(x_0, \dots, x_m) = \frac{1}{(m+1)!} [W_{\varphi_0}(x_0, \dots, x_m)]^2,$$

then, for every function $f \in L_2(D)$, the following is true:

$$E\hat{\theta}[f] = \int_\Omega f(x)\varphi_0(x)dx, \tag{2.25}$$

$$D\hat{\theta}[f] \le \int_\Omega f^2(x)dx - \sum_{j=0}^m \left[\int_\Omega f(x)\varphi_j(x)dx\right]. \tag{2.26}$$

Proof. Let us denote by c_j the Fourier coefficients of the function $f \in L_2(\Omega)$

$$c_j = \int_\Omega f(x)\varphi_j(x)dx,$$

and

$$a^2 = \int_\Omega \left[f(x) - \sum_{j=0}^m c_j\varphi_j(x)\right]^2 dx = \int_\Omega f^2(x)dx - \sum_{j=0}^m c_j.$$

If $a^2 \neq 0$, we can introduce the function

$$f(x) = \sum_{j=0}^{m} c_j \varphi_j(x) + a\psi(x). \qquad (2.27)$$

It is easy to check that $\psi(x)$ is orthogonal to every $\varphi_j(x)$ and

$$\int_\Omega \psi^2(x)dx = 1; \quad \int_\Omega \psi(x)\varphi_j(x)dx = 0, \quad 0 \leq j \leq m.$$

If $a = 0$, the representation (2.27) still holds, because every normed function, which is orthogonal to every φ_j, can be chosen as ψ. Replacing (2.27) in (2.23) one can deduce the following result:

$$W_f = \sum_{j=0}^{m} c_j W_{\varphi_j} + aW_\psi = c_0 W_{\psi_0} + aW_\psi. \qquad (2.28)$$

Using (2.28), it is easy to calculate the mathematical expectation of (2.24)

$$E\hat{\theta}[f] = \int_{B_+} \frac{W_f}{W_{\varphi_0}} p\, dt = \frac{1}{(m+1)!} \int_{B_+} W_f W_{\varphi_0} dt = \frac{1}{(m+1)!} \int_B W_f W_{\varphi_0} dt$$
$$= \frac{c_0}{(m+1)!} \int_B W_{\varphi_0}^2 dt + \frac{a}{(m+1)!} \int_B W_{\varphi_0} W_\psi dt.$$

The first of the last two integrals is equal to $(m+1)!$ by Theorem 2.6.1; the second integral is equal to 0 according to Theorem 2.6.2. Therefore $E\hat{\theta}[f] = c_0$, which is equivalent to (2.25). This proves the statement (2.25) of the theorem.

Let us now compute $E\hat{\theta}^2[f]$,

$$E\hat{\theta}^2[f] = \int_{B_+} \left(\frac{W_f}{W_{\varphi_0}}\right)^2 p\, dt = \frac{1}{(m+1)!} \int_{B_+} W_f^2 dt$$
$$= \frac{1}{(m+1)!} \int_{B_+} [c_0^2 W_f^2 + 2ac_0 W_{\varphi_0} W_\psi + a^2 W_\psi^2] dt.$$

To prove the statement (2.26) we have to show that $D\hat{\theta}[f] \leq a^2$. Using Theorem 2.6.1 we deduce that

$$E\hat{\theta}^2[f] = \frac{c_0^2}{(m+1)!} \int_B W_{\varphi_0}^2 dt + \frac{2ac_0}{(m+1)!} \int_B W_{\varphi_0} W_\psi dt + \frac{a^2}{(m+1)!} \int_{B_+} W_\psi^2 dt.$$

Now

$$\int_{B_+} W_\psi^2 dt \leq \int_B W_\psi^2 dt = (m+1)!,$$

and $E\hat{\theta}^2[f] \leq c_0^2 + a^2$, from whence $D\hat{\theta}[f] = E\hat{\theta}^2[f] - c_0^2 \leq a^2$. $\diamondsuit$

From the proof it is obvious that the inequality (2.26) becomes an equality if and only if $W_{\varphi_0} = 0$ and only for a manifold with dimensions less than $d(m+1)$.

2.7 Some Basic Facts about Quasi-Monte Carlo Algorithms

The key point of Monte Carlo algorithms is the use of a random variable γ, uniformly distributed in the interval $(0,1)$, the so-called ordinary random number. The concept of a *real* random number is a mathematical abstraction. Numbers computed from specified formulae but satisfying an accepted set of tests just as though they were *real* random numbers, are called *pseudo - random*. Numbers which are used instead of random numbers in some Monte Carlo algorithms to achieve convergence are called *quasi-random* (Sobol [So91]). Quasi- random numbers are connected with a certain class of algorithms and their applicability is more restricted than that of pseudo-random numbers (Sobol [So91], Niederreiter [Ni87], [Ni92], see, chapter 4). The reason in favor of quasi-random numbers is the possibility of increasing the rate of convergence: the usual rate of $n^{-1/2}$ can in some cases be replaced by $n^{-1+\varepsilon}$, where $\varepsilon > 0$ is arbitrarily small.

Let $x \equiv (x_{(1)}, \dots, x_{(d)})$ be a point that belongs to the d-dimensional cube $C^d = \{x : 0 \le x_{(i)} \le 1;\ i = 1, 2, ..., d\}$ and $\xi = (\gamma_{(i)}, \dots, \gamma_{(d)})$ be a random point, uniformly distributed in C^d .

An *uniformly distributed sequence* (u.d.s.) of non-random points was introduced by Weyl in 1916 ([We16]).

Denote by $S_n(\Omega)$ the number of points with $1 \le i \le n$ that lie in Ω, where $\Omega \subset C^d$. The sequence $x_1, x_2, \dots$ is called an u.d.s. if, for an arbitrary region Ω,

$$\lim_{n\to\infty} [S_n(\Omega)/n] = V(\Omega),$$

where $V(\Omega)$ is the d-dimensional volume of Ω.

Theorem 2.7.1 *(H. Weyl, see [We16, So91]).*
The relation

$$\lim_{n\to\infty} \frac{1}{n} \sum_{i=1}^{n} f(\xi_j) = \int_{C^d} f(x)dx \tag{2.29}$$

holds for all Riemann integrable functions f if and only if the sequence $x_1, x_2, \dots$ is u.d.s.

Comparing (2.8) with (2.29) one can conclude that if the random points ξ_i , are replaced by the points x_i of u.d.s., then for a wide class of functions f the averages converge. In this case the "i" th trial should be carried out using Cartesian coordinates $(x^{(i)}_{(1)}, ..., x^{(i)}_{(d)})$ of the point x_i , rather than the random numbers $\gamma_1, ..., \gamma_n$ For practical purposes a u.d.s. must be found that satisfied three additional requirements ([So73], [So89]):

(i) the best asymptote as $n \to \infty$;

(ii) well distributed points for small n ;

(iii) a computationally inexpensive algorithm.

All $\prod_\tau$-sequences given in [So89] satisfy the first requirement. The definition of $\prod_\tau$-sequences is as follows: binary estimates are called estimates of the kind $(j-1)2^{-m} \leq x < j2^{-m}$ when $j = 1, 2, ..., 2^m, m = 0, 1, ...$ (for $j = 2^m$ the right-hand end of the estimate is closed). A binary parallelepiped $\prod$ is a multiplication of binary estimates. A net in C^d , consisting of $n = 2^\nu$ points is called a $\prod_\tau$-sequence, if every binary $\prod$ with volume $\frac{2^\tau}{n}$ contains exactly 2^τ points of the net. It is supposed that ν and τ ($\nu > \tau$) are integers.

Subroutines to compute these points can be found in [So79] and Bradley and Fox [1980] ([BF80]). More details are contained in Levitan, Markovitz, Rozin and Sobol [1988] ([LMRS88]).

The problem of determining an error estimate for

$$\frac{1}{n}\sum_{i=1}^{n} f(\xi_j) \approx \int_{C^d} f(x)dx$$

arises.

If one uses the points $x_0, x_1, \dots$ the answer is quite simple. Finite nets $x_0, \dots, x_{n-1}$, $n = 2^m$ (with m a positive integer), have good uniform properties ([So91]).

Non-random estimates of the error

$$\delta_n(f) = \frac{1}{n}\sum_{i=1}^{n-1} f(x_i) - \int_{C^d} f(x)dx \tag{2.30}$$

are known (Sobol [So91], [So89]).

Let us mention two results. First, assume that all partial derivatives of the function $f(x)$, $x \in C^d \subset \mathbb{R}^d$, that include at most one differentiation with respect to each variable,

$$\frac{\partial^s f}{\partial x_{(i_1)} \dots \partial x_{(i_s)}}, \quad 1 \leq i_1 < \dots < i_s \leq d, \quad s = 1, 2, ..., d,$$

are continuous. It follows from [So91] that for $n = 2^m$

$$|\delta_n(f)| \leq A(f) n^{-1} (\log n)^{d-1}. \tag{2.31}$$

If in (2.30) arbitrary values of n are used then in (2.31) $(\log n)^{d-1}$ must be replaced by $(\log n)^d$ ([So91]).

Consider a class of functions $f(x)$, $x \in C^d \subset \mathbb{R}^d$, with continuous first partial derivatives. Denote

$$\sup \left| \frac{\partial f}{\partial x_{(i)}} \right| = L_i, \quad i = 1, ..., d.$$

It follows from the Sobol's result ([So89]) that if $n = 2^m$, then

$$|\delta_n(f)| \leq \max \left(s! \prod_{k=1}^{s} L_{i_k}/n \right)^{\frac{1}{s}},$$

where the maximum is extended over all groups satisfying $1 \leq i_1 < \ldots < i_s \leq d$ and $s = 1, ..., d$.

If, for example, some of the L_i are very small, than the error estimate takes advantage of this fact. The orders of convergence in both cases are optimal for the corresponding classes of functions.

In Doroshkevich and Sobol [DS87] a five-dimensional improper integral is evaluated by using the $\prod_\tau$-approximation and the plain Monte Carlo algorithm.

2.8 Monte Carlo Algorithms for Continual Integrals, Weight Functions and Splitting

2.8.1 Continual integrals

Let μ be a measure that corresponds to the homogeneous process with independent increments $\xi(s),\ 0 \leq s \leq t < \infty$. Let us consider the Monte Carlo algorithm for evaluating the continual integral from the functional F

$$\int F(\xi)\mu(d\xi).$$

First, the random process $\xi(s)$ is approximated by the process

$$\xi^{(m)}(s) = \sum_{k=1}^{m} i^{(k)}(s)[\xi(t_k) - \xi(t_{k-1})],$$

where $0 = t_0 < t_1 < \ldots < t_m = t$ is a splitting of $[0, t]$ into m parts and $i^{(k)}(s)$ is a function that determines the type of the approximation. If, e.g.,

$$i^{(k)}(s) = \frac{s - t_{k-1}}{t_k - t_{k-1}} \mathbf{1}_{[t_{k-1}, t_k)}(s) + \mathbf{1}_{[t_k, t)}(s),$$

where

$$\mathbf{1}_A(\tau) = \begin{cases} 1, & \text{if } \tau \in A \\ 0, & \text{if } \tau \notin A \end{cases},$$

then an approximation of the trajectory of the process is obtained by polygons with nodes at the points $(t_k, \xi(t_k)), k = 0, 1, ..., m$. The continual integral is approximated by

$$\int F(\xi)\mu(d\xi) = EF(x) \approx EF(\xi^{(m)})$$

and $EF(\xi^{(m)})$ can be estimated by the formula

$$EF(\xi^{(m)}) \approx \frac{1}{n}\sum_{k=1}^{n} F(\xi_k^{(m)}), \tag{2.32}$$

where $\xi_k^{(m)}(s)$ are independent realizations of the process $\xi^{(m)}(s)$.

The approximation of $\xi_k^{(m)}(s)$ by $\xi^{(m)}(s)$ is necessitated by the impossibility of obtaining a realization directly from the process $\xi(s)$.

With this mind Lichoded [Li89] showed that

$$\int F(\xi)\mu(d\xi) = \frac{1}{n}\sum_{k=1}^{n} F(\xi^{(m)}) + R^{(m)}(F) + r_n^m(F),$$

where $R^{(m)}(F) = \int F(\xi)\mu(d\xi) - EF(\xi^{(m)})$ is the error arising from the replacement of the random process $\xi(s)$ by the random process $\xi^{(m)}(s)$, and $r_n^m(F) = EF(\xi^{(m)}) - \frac{1}{n}\sum_{k=1}^{n} F(\xi_k^{(m)})$ is the error arising from the computation of $EF(\xi^{(m)})$ using (2.32).

In the above mentioned work the formula (2.32) is made more precise by the approximate determination of the error $R^{(m)}(F)$. Of course, this procedure is meaningful if and only if $R^{(m)}(F) > r_n^m(F)$.

Let us suppose that an exact formula exists for generalized polynomials of degree $2m+1$,

$$\int G(\xi)\mu(d\xi) \approx \sum_{j=1}^{q} A_j \int_0^{t_{(m)}} \dots \int_0^{t} G\left(\sum_{\alpha=1}^{m} x_\alpha^{(j)} \mathbf{1}_{[s_\alpha,t)}(\bullet) + a(\bullet)\right) ds_1 \dots ds_t$$

where q is an integer, $A_j, x_\alpha^{(j)},\ 1 \le j \le q,\ 1 \le \alpha \le m$ are coefficients and $a(s) = \int \xi_s \mu(d\xi)$ is the mean value of the measure μ.

In [Li89] the following theorem is proved.

Theorem 2.8.1 *The approximate equality*

$$R^{(m)}(F) \approx R_1^m(F) \equiv \sum_{j=1}^{q} A_j \int_0^{t_{(m)}} \dots \int_0^{t} F\left(\sum_{\alpha=1}^{l} x_\alpha^{(j)} \mathbf{1}_{[s_\alpha,t)}(.) + a(.)\right) ds_1 \dots ds_t,$$

is exact for generalized polynomials of degree $(2m+1)$.

From Theorem 2.8.1 it follows as a corollary that the approximate formula

$$EF(\xi) \approx EF(\xi^{(m)}) + R_1^{(m)}(F)$$

is exact for $(2m+1)$-degree generalized polynomials.

2.8.2 Weight functions

In the work of Shaw [Sh88] Monte Carlo quadratures with weight functions are considered for the computation of

$$S(g;m) = \int g(\theta)m(\theta)d\theta,$$

where g is some function (possibly vector or matrix valued).

The unnormalized posterior density m is expressed as the product of two functions w and f, where w is called the *weight function* $m(\theta) = w(\theta)f(\theta)$. The weight function w is nonnegative and integrated to one; i.e., $\int w(\theta)d\theta = 1$, and it is chosen to have similar properties to m.

Most numerical integration algorithms then replace the function $m(\theta)$ by a discrete approximation in the form of Shaw [Sh88]:

$$\hat{m}(\theta) = \begin{cases} w_i f(\theta), & \theta = \theta_i, i = 1,2,...,n, \\ 0 & \text{elsewhere,} \end{cases}$$

so that the integrals $S(g;m)$ may by estimated by

$$\hat{S}(g;m) = \sum_{i=1}^{n} w_i f(\theta_i)g(\theta_i). \tag{2.33}$$

Integration algorithms use the weight function w as the kernel of the approximation of the integrand

$$S(g;m) = \int g(\theta)m(\theta)d\theta = \int g(\theta)w(\theta)f(\theta)d\theta = \int g(\theta)f(\theta)dW(\theta) = Ew(g(\theta)f(\theta)).$$

This suggests a Monte Carlo approach to numerical integration (Shaw [Sh88]): generate nodes $\theta_1, \ldots \theta_n$ independently from the distribution w and estimate $S(g;m)$ by $\hat{S}(g;m)$ in (2.33) with $w_i = \frac{1}{n}$. If $g(\theta)f(\theta)$ is a constant then $\hat{S}(g;m)$ will be exact. More generally $\hat{S}(g;m)$ is unbiased and its variance will be small if $w(\theta)$ has a similar shape to $|g(\theta)m(\theta)|$. The above procedure is also known as *importance sampling* (see Section (2.4.3).

In [Sh88] it is noted that the determination of the weight function can be done iteratively using a posterior information.

Monte Carlo quadratures which use posterior distributions are examined in Van Dijk and Kloek [1983, 1985] (see, [VK83, VK85]), Van Dijk, Hop and Louter [1987] (see, [VHL87]), Stewart [1983, 1985, 1987] (see, [St83, St85, St87]), Stewart and Davis [1986] (see, [SD86]), Kloek and Van Dijk [1978] (see, [KV78]).

2.8.3 Splitting

Here we present the approach of Mikhailov, published in [Mi87]. Suppose

$$I = \int_X \int_Y f(x,y)g(x,y)dxdy$$

has to be computed, where $f(x,y)$ is the density of the joint distribution of the random vectors ξ and η .

Following [Mi87], let us introduce the notation:

$$\zeta = g(\xi,\eta), E[\zeta|x] = \int_Y f_2(y|x)d(x,y)dy, \;\; f_1(x) = \int_Y f(x,y)dy,$$

where $f_1(x)$ is the density of the absolute distribution of ξ; $f_2(y|x)$ is the density of the conditional distribution of η when $\xi = x$ and $E[\zeta|x]$ is the conditional mathematical expectation of the random variable ζ when $\xi = x$.

Following this, the problem is reduced to the computation of the integral

$$\int_X f_1(x)E[\zeta|x]dx.$$

Furthermore, the amount of the variance decreases, i.e. $DE[\zeta|\xi] \leq D\xi$, since $D\xi = ED[\zeta|\xi] + DE[\zeta|\xi]$.

It is useful to use several values of η for one value of ξ.

In the work of Mikhailov [Mi87] an approach for optimization of such an algorithm is proposed. Let ξ be distributed with density $f_1(x)$ and let $n(\xi)$ be an integer, which depends on $\xi, n(\xi) \geq 1$. The random variables $\eta_1, \ldots, \eta_{n(\xi)}$ are independent and equally conditionally distributed with density $f_2(y|x)$ under the condition $\xi = x$.

Mikhailov [Mi87] uses for following estimate:

$$\zeta = \frac{1}{n(\xi)} \sum_{k=1}^{n(\xi)} g(\xi,\eta_k),$$

and demonstrates that

$$E\zeta_n = \int_X \int_Y f(x,y)g(x,y)dxdy,$$

$$D\zeta_n = DE[\zeta|\xi] + E\left\{\frac{D[\zeta|\xi]}{n(\xi)}\right\}.$$

When $n(\xi) = \text{const} = n$, the optimal value of n can be found (which minimizes the computational complexity of the algorithm),

$$S_n = t_n D\zeta_n,$$

where $t_n = t_1 + E[n(\xi)t_2(\xi)]$ with t_1 the computational time for the computation of one value of ξ , and $t_2(x)$ the time for the computation of one value of η under the condition $\xi = x$,

$$n = \sqrt{\frac{A_2 t_1}{A_1 t_2}},$$

where A_1 and A_2 are estimated by the result from special a priori calculations.

2.9 Adaptive Monte Carlo Algorithms for Practical Computations

In this section a *superconvergent adaptive algorithm* for practical computations will be presented. The algorithm under consideration combines the idea of separation of the domain into *uniformly small* subdomains with the Kahn approach of *importance sampling*. An estimate of the probable error for functions with bounded derivative will be proved. This estimate improves the existing results. A plain adaptive Monte Carlo algorithm is also considered. It will be shown that for large dimensions d the convergence of the superconvergent adaptive Monte Carlo algorithm goes asymptotically to $O(n^{1/2})$, which corresponds to the convergence of the plain adaptive algorithm.

The both adaptive algorithms - superconvergent and plain - are applied for calculating multidimensional integrals. Numerical tests are performed on the computer CRAY Y-MP C92A. It is shown that for low dimensions (up to $d = 5$) the superconvergent adaptive algorithm gives better results than the plain adaptive algorithm. When the dimension increases the plain adaptive algorithm becomes better. One needs several seconds for evaluating 30-d integrals by plain adaptive algorithm, while the solution of the same integral by Gaussian quadrature would need more than 10^6 billion years if CRAY Y-MP C92A were used.

As it was shown in Section 2.1, the probable error for the plain Monte Carlo algorithm (which does not use any a priori information about the smoothness of $f(x)$) is defined as:

$$r_n = c_{0.5}\sigma(\theta)n^{-1/2},$$

where $c_{0.5} \approx 0.6745$.

In Section 2.5 it was defined that a superconvergent Monte Carlo algorithm is an algorithm for which

$$r_n = cn^{-1/2-\varepsilon(d)},$$

where c is a constant and $\varepsilon(d) > 0$ [So73], [DT89].

The idea of the algorithm consists in separating the domain Ω into subdomains Ω_j that are uniformly small according both to the probability and to the sizes.

Another degree of quality of the Monte Carlo algorithm is the variance of the random variable θ, whose mathematical expectation is equal to I. Let θ be a random variable in the plain Monte Carlo algorithm such that

$$I = E\theta.$$

Let $\hat{\theta}$ be another random variable for which

$$I = E\hat{\theta}$$

and the conditions providing the existence and the finiteness of the dispersion $D\hat{\theta}$ be fulfilled.

As it was shown in 2.4.4 the algorithm for which

$$D\hat{\theta} < D\theta$$

is called *efficient Monte Carlo algorithm* [Ka50], [Mi87], [Di91], [DT93]. An algorithm of this type is proposed by Kahn [Ka50] (for evaluation of integrals) and by Mikhailov and Dimov (for evaluation of integral equations) [Mi87], [Di91].

Here we deal with adaptive Monte Carlo algorithms, which use a priori and a posteriori information obtained during calculations. Both approaches - superconvergent adaptive approach and plain adaptive approach are applied. The works having studied superconvergent Monte Carlo algorithms show that the separation of the domain into *uniformly small* subdomains brings to an increase of the rate of convergence. But this separation does not use any a priori information about parameters of the problem. The *Kahn approach* and the approach in [Mi87], [Di91] use the information about the problem parameters, but do not use the idea of *separation*. In this section a superconvergent algorithm which uses both the idea of *separation* and the idea of *importance sampling* is presented. This algorithm is called *superconvergent adaptive Monte Carlo* (SAMC) algorithm. A *plain adaptive Monte Carlo* (AMC) algorithm will also be presented and studied.

The next subsection 2.9.1 contains theoretical results connected to SAMC algorithms. In fact, the proposed algorithm is an superconvergent Monte Carlo algorithm with the probable error of type $c \times n^{-1/2-\varepsilon(d)}$, but the constant c before $n^{-1/2-\varepsilon(d)}$ is smaller than the constant of the usual superconvergent Monte Carlo algorithm. In 2.9.2 two adaptive Monte Carlo algorithms - plain and superconvergent - will be implemented. This subsection contains numerical results of evaluating multi-dimensional integrals. It will be shown that for low dimensions (up to $d = 5$) the superconvergent adaptive Monte Carlo algorithm gives better results than the plain adaptive algorithm.

2.9.1 Superconvergent adaptive Monte Carlo algorithm and error estimates

The following problem arises:

Problem 1: *Is it possible to combine the idea of separation of the domain into uniformly small subdomains with the Kahn approach of importance sampling?*

Here we consider functions $f(x)$ from the class $W^r(M;\Omega)$. The definition of $W^r(M;\Omega)$ is given in the Introduction (see, Chapter 1).

First, consider the one-dimensional problem of evaluation integrals:

$$I = \int_\Omega f(x)p(x)\,dx, \quad \Omega \equiv [0,1],$$

where the positive function $f(x) \in W^1(L;[0,1])$ and $\int_\Omega p(x)dx = 1$. Consider (as an example of importance separation) the following partitioning of the interval [0,1] into m subintervals ($m \le n$):

$$x_0 = 0; \;\; x_m = 1;$$

$$C_i = 1/2[f(x_{i-1}) + f(1)](1 - x_{i-1}), \quad i = 1, \dots, m-1;$$

$$i = 1, \dots, m-1;$$

$$x_i = x_{i-1} + \frac{C_i}{f(x_{i-1})(n-i+1)}, \quad i = 1, ..., m-1 \tag{2.34}$$

$$\Omega_1 \equiv [x_0, x_1], \ \Omega_i \equiv (x_{i-1}, x_i], \ i = 2, \dots, m.$$

The scheme (2.34) gives us an *importance separation* of the domain $\Omega \equiv [0,1]$. We have:

$$I = \int_0^1 f(x)p(x)\,dx =$$

$$\sum_{i=1}^{m} \int_{x_{i-1}}^{x_i} f(x)p(x)\,dx.$$

Denote by p_i and I_i the following expressions:

$$p_i = \int_{x_{i-1}}^{x_i} p(x)\,dx$$

and

$$I_i = \int_{x_{i-1}}^{x_i} f(x)p(x)\,dx.$$

Obviously

$$\sum_{i=1}^{m} p_i = 1; \ \sum_{i=1}^{m} I_i = I.$$

Consider now a random variable $\xi^{(i)} \in \Omega_i$ with a density function $p(x)/p_i$, where $\Omega_i \equiv (x_{i-1}, x_i]$. In this case

$$I_i = E(p_i f(\xi^{(i)})).$$

Let n_i be a number of random points in $\Omega_i (\sum_{i=1}^m n_i = n)$.

It is easy to show that

$$I_i = E\left[\frac{p_i}{n_i}\sum_{s=1}^{n_i} f(\xi_s^{(i)})\right] = E\theta_{n_i};$$

$$I = E\left[\sum_{i=1}^{m}\frac{p_i}{n_i}\sum_{s=1}^{n_i} f(\xi_s^{(i)})\right] = E\theta_n^*.$$

Let $n_i = 1$ (so, $m = n$). Since $f(x) \in W^1(L; [0,1]$, there exist constants L_i, such that

$$L_i \geq \left|\frac{\partial f}{\partial x}\right| \quad \text{for any} \quad x \in \Omega_i. \tag{2.35}$$

Moreover, for our scheme there exist constants $c_1^{(i)}$ and $c_2^{(i)}$ such that

$$p_i = \int_{\Omega_i} p(x)\,dx \leq c_1^{(i)}/n,\ i = 1,\ldots,n \tag{2.36}$$

and

$$\sup_{x_1^{(i)},x_2^{(i)}\in\Omega_i} |x_1^{(i)} - x_2^{(i)}| \leq c_2^{(i)}/n,\ i = 1,\ldots,n. \tag{2.37}$$

We shall say, that the conditions (2.35)-(2.37) define an *importance separation* of Ω in general. Note that the scheme (2.34) gives us only an example of a possible construction of such importance separation.

Theorem 2.9.1 *Let $f(x) \in W^1(L;[0,1])$ and $m = n$. Then for the importance separation of Ω*

$$r_n \leq \sqrt{2}[1/n\sum_{j=1}^{n}(L_j c_1^{(j)} c_2^{(j)})^2]^{1/2} n^{-3/2}.$$

Proof. Let Ω_j be any subdomain of $[0,1]$. For a fixed point $s^{(j)} \in \Omega_j$ we have:

$$f(\xi^{(j)}) = f(s^{(j)}) + f'(\eta^{(j)})(\xi^{(j)} - s^{(j)}),$$

where $\eta^{(j)} \in \Omega_j$.

Since $f'(\eta^{(j)}) \leq L_j$ we have:

$$Df(\xi^{(j)}) \leq Ef^2(\xi^{(j)}) \leq L_j^2 E(\xi^{(j)} - s^{(j)})^2$$

$$\leq L_j^2 \sup_{x_1^{(i)},x_2^{(i)}\in\Omega_j} |x_1^{(j)} - x_2^{(j)}|^2 \leq L_j^2 (c_2^{(j)})^2/n^2.$$

Now the variance $D\theta_n^*$ can be estimated :

$$D\theta_n^* = \sum_{j=1}^{n} p_j^2 Df(\xi^{(j)}) \leq \sum_{j=1}^{n}((c_1^{(j)})^2 n^{-2} L_j^2 (c_2^{(j)})^2 n^{-2})$$

$$= 1/n \sum_{j=1}^{n}(L_j c_1^{(j)} c_2^{(j)})^2 n^{-3}.$$

To estimate the probable error one can apply the Chebyshev's inequality:

$$\Pr\{|\theta_n^* - E\theta_n^*| < h\} \geq 1 - (D\theta_n^*/h^2),$$

where $h > 0$.

Let

$$h = 1/\varepsilon (D\theta_n^*)^{1/2},$$

where ε is a positive number.

For $\varepsilon = 1/\sqrt{2}$ we have:

$$\Pr\{|\theta_n^* - I| < \sqrt{2}(D\theta_n^*)^{1/2}\} \geq 1/2.$$

The last inequality proves the theorem, since

$$\begin{aligned} r_n &\leq \sqrt{2}(D\theta_n^*)^{1/2} \\ &= \sqrt{2}\left(\frac{1}{n}\sum_{j=1}^{n}(L_j c_1^{(j)} c_2^{(j)})^2\right)^{1/2} n^{-3/2}. \end{aligned} \qquad (2.38)$$

$\diamondsuit$

This result presents a superconvergent adaptive Monte Carlo algorithm. Moreover, the constants $\sqrt{2}\left(\frac{1}{n}\sum_{j=1}^{n}(L_j c_1^{(j)} c_2^{(j)})^2\right)^{1/2}$ in (2.38) is smaller than the constant in the algorithms of the Dupach type [Du56], [DT89] presented in Section 2.5. In fact, $L_i \leq L$ for any $i = 1, \dots, m = n$. Obviously,

$$\frac{1}{n}\sum_{j=1}^{n} L_j c_1^{(j)} c_2^{(j)} \leq L c_1 c_2.$$

Now consider multi-dimensional integrals:

$$I = \int_\Omega f(x)p(x)\,dx\,,\; x \in \Omega \subset I\!R^d,$$

where the positive function $f(x)$ belongs to $W^1(L;\Omega)$.

Let

$$f(x) \in W^1(L_i;\Omega_i), \text{ for any } x \in \Omega_i. \qquad (2.39)$$

Let there exist vectors

$$L_i = (L_{i_1}, \dots, L_{i_d}), \quad i = 1, \dots, n,$$

such that

$$L_{i_l} \geq \left|\frac{\partial f}{\partial x_{(l)}}\right|, \quad l = 1, \dots, d; \quad x = (x_1, \dots, x_d) \in \Omega_i. \qquad (2.40)$$

Let there are positive constants $c_1^{(i)}$ and a vector

$$c_2^{(i)} = (c_{2(1)}^{(i)}, c_{2(2)}^{(i)}, \dots, c_{2(d)}^{(i)}), \qquad (2.41)$$

such that,

$$p_i = \int_{\Omega_i} p(x)\,dx \le c_1^{(i)}/n,\ i = 1,\dots,n \tag{2.42}$$

and

$$d_{il} = \sup_{x_{1(l)}^{(i)},x_{2(l)}^{(i)} \in \Omega_i} |x_{1(l)}^{(i)} - x_{2(l)}^{(i)}| \le c_{2(l)}^{(i)}/n^{1/d};\quad i = 1,\dots,n,\quad l = 1,\dots,d. \tag{2.43}$$

The conditions (2.40 – 2.43) define an importance separation of the domain Ω in the d-dimensional case.

Consider, as an example, the following importance separation of the domain Ω: on each dimension we use the scheme (2.34).

The following statement is fulfilled:

Theorem 2.9.2 *Let there exist an importance separation of Ω such that (2.39) is satisfied and $m = n$. Then*

$$r_n \le \sqrt{2}d \left[\frac{1}{n}\sum_{i=1}^{n}\sum_{l=1}^{d}(L_{il}c_1^{(i)}c_{2(l)}^{(i)})^2\right]^{1/2} n^{-1/2-1/d}.$$

The proof of this theorem follows the same techniques as the proof of Theorem 2.9.1.

Let us formulate one important corollary from this theorem. Denote by

$$L_j = max_l L_{jl},\quad j = 1,\dots,n \tag{2.44}$$

and let

$$c_2^{(j)} = max_l c_{2(l)}^{(j)}.$$

Corollary: *If L_j is defined by (2.44), then for the probable error of the importance separation algorithm the following estimate holds:*

$$r_n \le \sqrt{2}d \left\{\frac{1}{n}\sum_{j=1}^{n}(c_1^{(j)}c_2^{(j)}L_j)^2\right\}^{1/2} n^{-1/2-1/d},$$

where $L_j = max_l L_{jl}$.

The proof is obvious since,

$$\sum_{l=1}^{d} L_{jl} \le d \cdot max_l L_{jl} = dL_j.$$

2.9.2 Implementation of adaptive Monte Carlo algorithms. Numerical tests

The algorithms are realized on CRAY Y-MP C92A machine with two vector processors. Two Monte Carlo algorithms are considered.

1. *Plain adaptive Monte Carlo algorithm*

This approach does not use any a priori information about the smoothness of the integrand. It deals with n uniformly distributed random points $x_i \in [0,1]^d,\ i = 1,\dots,n$ into d- dimensional cube $[0,1]^d$. For generating of any point, d uniformly distributed random numbers into interval $[0,1]$ are produced. The algorithm is adaptive: it starts with a relatively small number n, which is given as an input data. During the calculations the variance on each dimension coordinate is estimated. The above mentioned information is used for increasing the density of the new generated points. This approach leads to the following estimate

$$\varepsilon \leq c\frac{1}{n^{1/2}}$$

instead of standard estimate for the probable error

$$\varepsilon \leq 0.6745\sigma(\theta)\frac{1}{n^{1/2}},$$

where

$$c \leq 0.6745\sigma(\theta).$$

2. *Superconvergent adaptive Monte Carlo algorithm.*

As a first step of this algorithm the domain of integration is separated into subdomains with identical volume. For every subdomain the integral I_j is evaluated and a posteriori information for the variance is also received. After that an approximation for the integral $I = \sum_j I_j$ and the total variance is obtained. The total variance is compared with local a posteriori estimates for the variance in every subdomain. The obtained information is used for the next refinement of the domain and for increasing the density of the random points.

The probable error for this approach has the following form

$$\varepsilon \leq cn^{-\frac{1}{2}-\frac{1}{d}}. \tag{2.45}$$

Obviously, for a large dimension d, the convergence goes asymptotically to $O(n^{-1/2})$.

Since, the SAMC algorithm is more time consuming, for large dimensions d it is better to use the AMC algorithm. This was observed during the performed calculations. But for relatively small d ($d = 1,\dots,5$) and for "bad" (not very smooth) functions (say, functions with bounded first derivative) the SAMC algorithm successfully competes with the standard Gaussian quadratures.

Both algorithms have been used for evaluating integrals of different dimensions and integrands with large and small variance, as well as high and low smoothness.

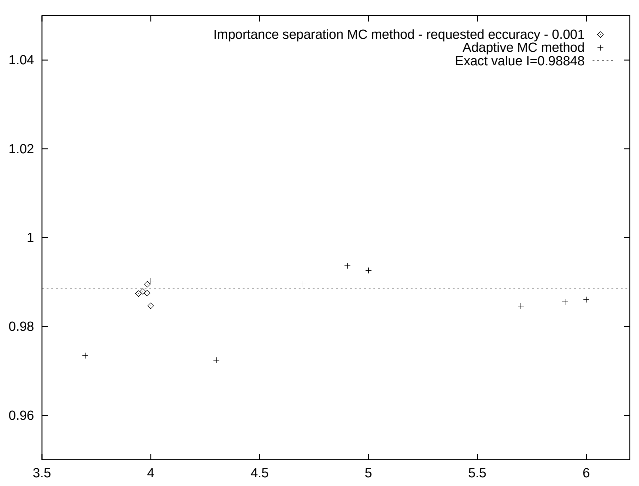

Figure 2.1
Superconvergent AMC integration
$(d = 5)$
(comparison with the AMC)

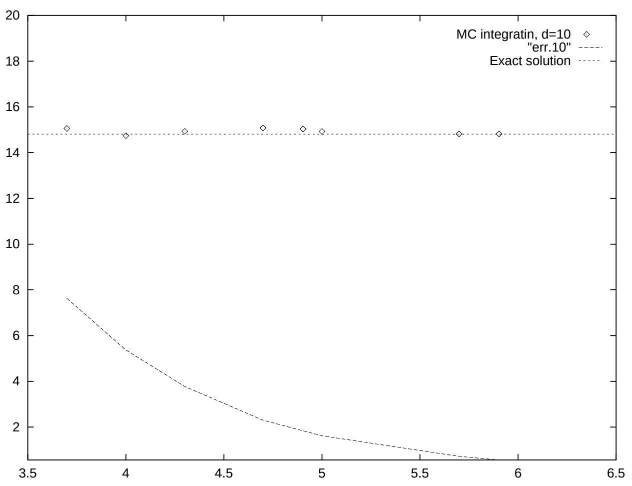

Figure 2.2
AMC integration, $d = 10$

Here some results of numerical experiments are given. We will present four examples (the exact values of integrals are presented to be able to compare the real error with the a posteriori estimated error).

Example 1.
$n = 4$

$$I_1 = \int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{4x_1 x_3^2 \exp\{2x_1 x_3\}}{(1 + x_2 + x_4)^2} dx_1 dx_2 dx_3 dx_4 = 0.5753.$$

Example 2.
$n = 5$

$$I_2 = \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{4x_1 x_3^2 \exp\{2x_1 x_3\} \exp\{x_5\}}{(1 + x_2 + x_4)^2} dx_1 dx_2 dx_3 dx_4 = 0.98848.$$

Example 3.
$n = 25$

$$I_3 = \int_0^1 \dots \int_0^1 \frac{4x_1 x_3^2 \exp\{2x_1 x_3\}}{(1 + x_2 + x_4)^2} \exp\{x_5 + \dots + x_{20}\} \times$$
$$x_{21} x_{22} \dots x_{25} \; dx_1 \dots dx_{25} = 103.8$$

Example 4.
$n = 30$

$$I_4 = \int_0^1 \dots \int_0^1 \frac{4x_1 x_3^2 \exp\{2x_1 x_3\}}{(1 + x_2 + x_4)^2} \exp\{x_5 + \dots + x_{20}\} \times$$
$$x_{21} x_{22} \dots x_{30} \; dx_1 \dots dx_{30} = 3.244$$

Some results are presented in Table 2.1.

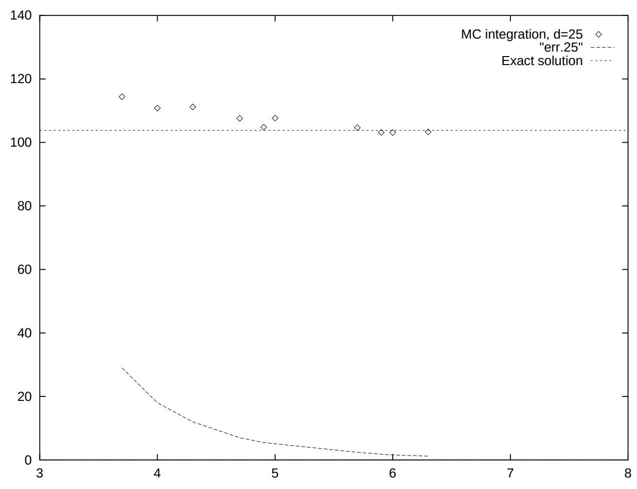

Figure 2.3
AMC integration ($d = 25$)
(comparison with the AMC)

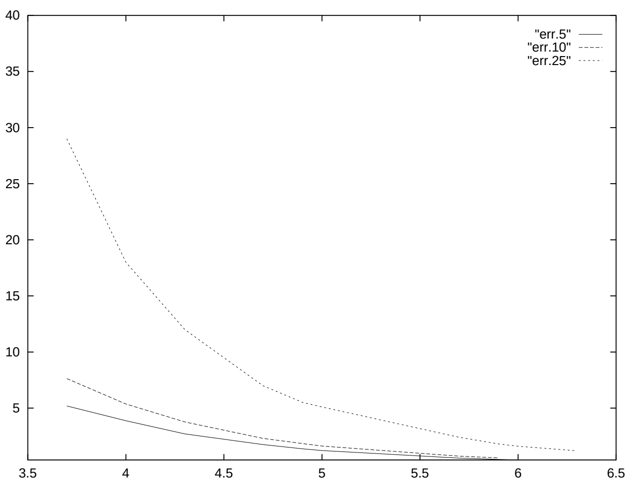

Figure 2.4
Errors for AMC integration (different dimensions)

Table 2.1 contains information about the dimension of the integral, the exact solution, the applied Monte Carlo approach, calculated solution, CP-time, estimated error, relative error really obtained and number of random points.

Some of numerical results are presented on Fig. 2.1 - 2.4. Figure 2.1 shows the results of implementation of SAMC algorithm for solving 5-d integral. The dependence of the calculated values and the error from the logarithm of the number of random points (log n) is presented. In this case about 9000 random points are sufficient for obtaining a relatively good accuracy (the requested accuracy in this case is 0.001). Figure 2.2 presents results of calculated values and the error for 10-d integral for different numbers of random points (here again on the x-coordinate the values of log n are presented). Figure 2.3 expresses the results for the solution as well as for the estimated error obtained for 25-d integral. Some numerical results for the implementation of AMC algorithm for 30-d integral are also presented. It is clear that in this case the estimated error can not be very small. Nevertheless, such accuracy is sufficient for applications which are considered in control theory.

2.9.3 Conclusion

1. For low dimensions SAMC algorithm gives better results than AMC algorithm. For example, for $d = 5$, SAMC algorithm needs 9000 realizations for reaching an error of 0.03%, while AMC algorithm needs 20000 realizations for reaching an error of 0.17%. The CP-time of the SAMC algorithm is about 7 times less than the corresponding CP-time of the AMC algorithm.

2. When the dimension d is high AMC gives better results than SAMC algorithm. It can be explain with the fact that for large dimensions d the error of the SAMC algorithm asymptotically goes to $O(n^{-1/2})$ which corresponds to the error of AMC algorithm.

Table 2.1: **Results of numerical experiments performed on CRAY Y-MP C92A**

Dim. d	Exact sol.	Method	Calcul. solution	$\mathrm{CP}-\mathrm{time}, s$	Estim. error	Rel. error	Num.of points
4	0.5753	AMC	0.5763	0.184	0.025	0.0017	20000
4	0.5753	$SAMC$	0.5755	0.025	$0.82.10^{-2}$	0.0003	1728
25	103.8	AMC	107.66	3.338	0.05	0.036	10^5
25	103.8	AMC	104.6	17.2	0.024	0.0077	5.10^5
25	103.8	AMC	103.1	54.9	0.017	0.0069	10^6
30	3.244	AMC	3.365	4.07	0.099	0.037	10^5
30	3.244	AMC	3.551	0.879	0.23	0.095	2.10^4

3. It is very important that Monte Carlo algorithm permits to integrate numerically high-dimensional integrals with relatively good accuracy. For example, the value of 25-d integral can be computed for 54.9 s only with an error less than 1%. The same integral can be evaluated for 3.34 s with an error of 3.6%. For evaluation of 30-d integral with an error of 3.7% CP-time of 4.07 s is needed. For reaching the same accuracy with quadrature formula of Gaussian type one needs at least 10 nodes on each direction, which means that 10^{30} values of the integrand have to be calculated. It means that 10^{23} s or more than 10^6 billion years are needed if supercomputer CRAY Y-MP C92A is used.

Chapter 3

SOLVING LINEAR EQUATIONS

In general, Monte Carlo numerical algorithms may be divided into two classes – *direct* algorithms and *iterative* algorithms. The direct algorithms provide an estimate of the solution of the equation in a finite number of steps, and contain only a stochastic error. For example, direct Monte Carlo algorithms are the algorithms for evaluating integrals [HH64], [So73]. Iterative Monte Carlo algorithms deal with an approximate solution obtaining an improved solution with each step of the algorithm. In principle, they require an infinite number of steps to obtain the exact solution, but usually one is happy with an approximation to say k significant figures. In this latter case there are two errors - *stochastic* and *systematic*. The systematic error depends both on the number of iterations performed and the characteristic values of the iteration operator, while the stochastic errors depend on the probabilistic nature of the algorithm.

Iterative algorithms are preferred for solving integral equations and large sparse systems of algebraic equations (such as those arising from approximations of partial differential equations). Such algorithms are good for diagonally dominant systems in which convergence is rapid; they are not so useful for problems involving dense matrices, for example.

Define an iteration of degree j as

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

where $u^{(k)}$ is obtained from the kth iteration. It is desired that

$$u^{(k)} \to u = A^{-1}b \quad \text{as} \quad k \to \infty.$$

Usually the degree of j is kept small because of storage requirements.

The iteration is called *stationary* if $F_k = F$ for all k, that is, F_k is independent of k.

The iterative Monte Carlo process is said to be *linear* if F_k is a linear function of $u^k, \ldots, u^{(k-j+1)}$.

We shall consider *iterative stationary linear Monte Carlo algorithms* and will analyze both systematic and stochastic errors.

3.1 Iterative Monte Carlo Algorithms

Consider a general description of the iterative Monte Carlo algorithms. Let $\mathbf{X}$ be a Banach space of real-valued functions. Let $f = f(x) \in \mathbf{X}$ and $u_k = u(x_k) \in \mathbf{X}$ be defined in $I\!R^d$ and $L = L(u)$ be a linear operator defined on $\mathbf{X}$.

Consider the sequence $u_1, u_2, ...$, defined by the recursion formula

$$u_k = L(u_{k-1}) + f, \quad k = 1, 2, \ldots \tag{3.1}$$

The formal solution of (3.1) is the truncated Neumann series

$$u_k = f + L(f) + \ldots + L^{k-1}(f) + L^k(u_0), \qquad k > 0, \tag{3.2}$$

where L^k means the k-th iterate of L.

As an example consider the integral iterations.

Let $u(x) \in \mathbf{X}$, $x \in \Omega \subset I\!R^d$ and $l(x, x')$ be a function defined for $x \in \Omega, x' \in \Omega$. The integral transformation

$$Lu(x) = \int_\Omega l(x, x')u(x')dx'$$

maps the function $u(x)$ into 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,

$$L^2u(x) = \int_\Omega \int_\Omega l(x, x')l(x', x'')dx'dx''.$$

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 $\mathbf{X}$ which satisfies the equation

$$u = L(u) + f. \tag{3.3}$$

The truncation error of (3.2) is

$$u_k - u = L^k(u_0 - u).$$

Let $J(u_k)$ be a linear functional that is to be calculated. Consider the spaces

$$\mathbf{T}_{i+1} = \underbrace{I\!R^d \times I\!R^d \times \ldots \times I\!R^d}_{i \text{ times}}, \qquad i = 1, 2, \ldots, k, \tag{3.4}$$

where "$\times$" 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:

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

where $J(u)$ is a linear 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

$$J(u_k) \approx \frac{1}{n} \sum_{s=1}^{n} \{\theta_k\}_s, \tag{3.5}$$

where $\{\theta_k\}_s$ is the s-th realization of the random variable θ_k.

The probable error r_n of (3.5) [EM82] is then

$$r_n = c\ \sigma(\theta_k) n^{-\frac{1}{2}},$$

where $c \approx 0.6745$ and $\sigma(\theta_k)$ is the standard deviation of the random variable θ_k.

There are two approaches which correspond to two special cases of the operator L :

- (**i**) L is a matrix and u and f are vectors;
- (**ii**) L is an ordinary integral transform

$$L(u) = \int_\Omega l(x,y)u(y)dy$$

  and $u(x)$ and $f(x)$ are functions.

First consider the second case. Equation (3.3) becomes

$$u(x) = \int_\Omega l(x,y)u(y)dy + f(x) \ \text{ or } \ u = Lu + f. \tag{3.6}$$

Monte Carlo algorithms frequently involve the evaluation of linear functionals of the solution of the following type

$$J(u) = \int\limits_\Omega h(x)u(x)dx = (u,h). \tag{3.7}$$

In fact, the equation (3.7) defines an inner product of a given function $h(x) \in \mathbf{X}$ with the solution of the integral equation (3.3).

Sometimes, the adjoint equation

$$v = L^* v + h \tag{3.8}$$

will be used.

In (3.8) $v, h \in \mathbf{X}^*$, $L^* \in [\mathbf{X}^* \to \mathbf{X}^*]$, $\mathbf{X}^*$ is the dual functional space to $\mathbf{X}$ and L^* is an adjoint operator.

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

$$\| f \|_{\mathbf{L_1}} = \int\limits_\Omega | f(x) | dx;$$

$$\| L \|_{\mathbf{L}_1} \leq \sup_x \int_\Omega | l(x,x') | \, dx'. \tag{3.9}$$

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

$$\| h \|_{\mathbf{L}_\infty} = \sup |h(x)| \;\; x \in \Omega.$$

For many applications $\mathbf{X} = \mathbf{X}^* = \mathbf{L}_2$. Note also, that if $h(x), u(x) \in \mathbf{L}_2$ then the inner product (3.7) is finite. In fact,

$$\left| \int_\Omega h(x)u(x)dx \right| \leq \int_\Omega |h(x)u(x)|dx \leq \left\{ \int_\Omega h^2 dx \int_\Omega u^2 dx \right\}^{1/2} < \infty.$$

One can see, that if $u(x) \in \mathbf{L}_2$ and $l(x,x') \in \mathbf{L}_2(\Omega \times \Omega)$ then $Lu(x) \in \mathbf{L}_2$:

$$|Lu(x)|^2 \leq \left\{ \int_\Omega |lu| dx' \right\}^2 \leq \int_\Omega l^2(x,x')dx' \int_\Omega u^2(x')dx'.$$

Let us integrate the last inequality with respect to x:

$$\int_\Omega |Lu|^2 dx \leq \int_\Omega \int_\Omega l^2(x,x')dx'dx \int_\Omega u^2(x')dx' < \infty.$$

From the last inequality it follows that $L^2u(x), \ldots, L^iu(x), \ldots$ belong also to $\mathbf{L}_2(\Omega)$.

Obviously, if $u \in \mathbf{L}_1$ and $h \in \mathbf{L}_\infty$ the inner product (3.7) will be bounded.

If it is 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 K. Sabelfeld [Sa89], 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

$$J = (h,u) = (f,v).$$

In fact, let us multiply (3.6) by v and (3.8) by u and integrate. We obtain

$$(v,u) = (v,Lu) + (v,f) \;\text{ and }\; (v,u) = (L^*v,u) + (h,u).$$

Since

$$(L^*v,u) = \int_\Omega L^*v(x)u(x)dx = \int_\Omega \int_\Omega l^*(x,x')v(x')u(x)dxdx'$$

$$= \int_\Omega \int_\Omega l(x', x) u(x) v(x') dx dx' = \int_\Omega Lu(x') v(x') dx' = (v, Lu),$$

we have

$$(L^* v, u) = (v, Lu).$$

Thus, $(h, u) = (f, v)$. (Usually, the kernel $l(x', x)$ is called *transposed kernel.*)

Consider the Monte Carlo algorithm for evaluating the functional (3.7). It can be seen that when $l(x, x') \equiv 0$ evaluation of the integrals can pose a problem. Consider a random point $\xi \in \Omega$ with a density $p(x)$ and let there be n realizations of the random point $\xi_i (i = 1, 2, ..., n)$. Let a random variable $\theta(\xi)$ be defined in Ω, such that

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

Then the computational problem becomes one of calculating repeated realizations of θ and of combining them into an appropriate statistical estimator of J. 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 *Markov chains* (see, the definition given in the introduction 1).

An approximate value of the linear functional J, defined by (3.7) is

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

where $(\theta)_s$ is the s-th realization of the random variable θ.

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

For the first case, when the linear operator L is a matrix, the equation (3.2) can be written in the following form :

$$u_k = L^k u_0 + L^{k-1} f + \ldots + Lf + f = (I - L^k)(I - L)^{-1} f + L^k u_0, \qquad (3.10)$$

where I is the unit (identity) matrix;

$L = (l_{ij})_{i,j=1}^m$; $u_0 = (u_1^0, \ldots, u_m^0)^T$ and matrix $I - L$ is supposed to be non-singular.

It is well known that if all eigenvalues of the matrix L lie within the unit circle of the complex plane there exists a vector u such that

$$u = \lim_{k \to \infty} u_k \ ,$$

which satisfies the equation

$$u = Lu + f \tag{3.11}$$

(see, for example, [GV83]).

Now consider the problem of evaluating the inner product

$$J(u) = (h, u) = \sum_{i=1}^{m} h_i u_i \,, \tag{3.12}$$

where $h \in I\!R^{m\times 1}$ is a given vector-column.

To construct a random variable whose mathematical expectation coincides with the functional (3.12) for the system (3.14 first consider the integral equation (3.6) for which $\Omega = [0, m)$ is an one-dimensional interval divided into equal subintervals $\Omega_i = [i-1, i)$, $i = 1, 2, \ldots m$ such that

$$\begin{cases} l(x,y) = l_{ij} & , \ x \in \Omega_i, \ y \in \Omega_j \\ f(x) = f_i & , \ x \in \Omega_i \end{cases}$$

Then the integral equation (3.6) becomes

$$u_i = \sum_j \int_{\Omega_j} l_{ij} u(y) dy + f_i$$

for $u_i \in \Omega_i$. Denote

$$u_j = \int_{\Omega_j} u(y) dy \tag{3.13}$$

so that one obtains, for $u(x) \in \Omega_i$,

$$u(x) = \sum_{j=1}^{m} l_{ij} u_j + f_i.$$

From the last equation it follows that $u(x) = u_i$ and so,

$$u_i = \sum_{j=1}^{m} l_{ij} u_j + f_i \ ,$$

or in a matrix form

$$u = Lu + f \ , \tag{3.14}$$

where $L = \{l_{ij}\}_{i,j=1}^{m}$.

The above permits the construction of the following random variable

$$\theta[h] = \frac{h_{k_0}}{p_0} \sum_{\nu=0}^{\infty} Q_\nu f_{k_\nu} \ , \tag{3.15}$$

where

$$Q_0 = 1; \qquad Q_\nu = Q_{\nu-1}\frac{l_{k_{\nu-1},k_\nu}}{p_{k_{\nu-1},k_\nu}}, \qquad \nu = 1, 2, \dots \tag{3.16}$$

and $k_0, k_1, \dots$ is a Markov chain on elements of the matrix L constructed by using an initial probability p_0 and a transition probability $p_{k_{\nu-1},k_\nu}$ for choosing the element $l_{k_{\nu-1},k_\nu}$ of the matrix L.

3.2 Solving Linear Systems and Matrix Inversion

Consider a matrix L:

$$L = \{l_{ij}\}_{i,j=1}^m, \quad L \in I\!\!R^{m\times m}$$

and a vector

$$f = (f_1, \dots, f_m)^T \in I\!\!R^{m\times 1}$$

The matrix L can be considered as a linear operator $L[I\!\!R^m \to I\!\!R^m]$, so that the linear transformation

$$Lf \in I\!\!R^{m\times 1} \tag{3.17}$$

defines a new vector in $I\!\!R^{m\times 1}$.

Since iterative Monte Carlo algorithms using the transformation (3.17) will be considered, the linear transformation (3.17) will be called *iteration*. The algebraic transformation (3.17) plays a fundamental role in iterative Monte Carlo algorithms.

Now consider the following two problems **Pi (i=1,2)** for the matrix L:

Problem P1. Evaluating the inner product

$$J(u) = (h, u) = \sum_{i=1}^m h_i u_i$$

of the solution $u \in I\!\!R^{m\times 1}$ of the linear algebraic system

$$Au = b,$$

where $A = \{a_{ij}\}_{i,j=1}^m \in I\!\!R^{m\times m}$ is a given matrix; $b = (b_1, \dots, b_m)^T \in I\!\!R^{m\times 1}$ and $h = (h_1, \dots, h_m)^T \in I\!\!R^{m\times 1}$ are given vectors.

It is possible to choose a non-singular matrix $M \in I\!\!R^{m\times m}$ such that $MA = I - L$, where $I \in I\!\!R^{m\times m}$ is the identity matrix and $Mb = f$, $f \in I\!\!R^{m\times 1}$.

Then

$$u = Lu + f.$$

It will be assumed that

$$(i) \qquad \begin{cases} 1. & \text{The matrices } M \text{ and } L \text{ are both non-singular;} \\ 2. & |\lambda(L)| < 1 \ \text{ for all eigenvalues } \ \lambda(L) \ \text{ of } \ L, \end{cases}$$

that is, all values $\lambda(L)$ for which

$$Lu = \lambda(L)u$$

is satisfied. If the conditions (i) are fulfilled, then (3.15), (3.16) become a *stationary linear iterative Monte Carlo algorithm.*

As a result the convergence of the Monte Carlo algorithm depends on truncation error of (3.10).

Problem P2. Inverting of matrices, i.e. evaluating of matrix

$$C = A^{-1},$$

where $A \in I\!\!R^{m\times m}$ is a given real matrix.

Assumed that the following conditions are fulfilled:

$$(ii) \qquad \begin{cases} 1. & \text{The matrix } A \text{ is non-singular;} \\ 2. & ||\lambda(A)| - 1| < 1 \text{ for all eigenvalues } \lambda(A) \text{ of } A. \end{cases}$$

Obviously, if the condition (i) is fulfilled, the solution of **the problem P1** can be obtained using the iterations (3.10).

For **problem P2** the following *iterative* matrix:

$$L = I - A$$

can be constructed.

Since it is assumed that the conditions (ii) are fulfilled, the inverse matrix $C = A^{-1}$ can be presented as

$$C = \sum_{i=0}^{\infty} L^i.$$

For the problems $\mathbf{P_i}(\mathbf{i} = \mathbf{1}, \mathbf{2})$ one can create a stochastic process using the matrix L and vectors f and h.

Consider an initial density vector $p = \{p_i\}_{i=1}^m \in I\!\!R^m$, such that $p_i \geq 0, i = 1, \dots, m$ and $\sum_{i=1}^m p_i = 1$.

Consider also a transition density matrix $P = \{p_{ij}\}_{i,j=1}^m \in I\!\!R^{m\times m}$, such that $p_{ij} \geq 0,\ i, j = 1, \dots, m$ and $\sum_{j=1}^m p_{ij} = 1$, for any $i = 1, \dots, m$.

Define sets of *permissible* densities $\mathcal{P}_h$ and $\mathcal{P}_L$.

Definition 3.2.1 *The initial density vector $p = \{p_i\}_{i=1}^m$ is called permissible to the vector $h = \{h_i\}_{i=1}^m \in I\!\!R^m$, i.e. $p \in \mathcal{P}_h$, if*

$$p_i > 0, \ \textit{when } h_i \neq 0 \ \textit{ and } p_i = 0, \ \textit{ when } \ h_i = 0 \ \textit{ for } \ i = 1, \dots, n.$$

The transition density matrix $P = \{p_{ij}\}_{i,j=1}^m$ is called permissible to the matrix $L = \{l_{ij}\}_{i,j=1}^m$, i.e. $P \in \mathcal{P}_L$, if

$$p_{ij} > 0, \textit{when } \ l_{ij} \neq 0 \ \textit{ and } \ p_{ij} = 0, \ \textit{ when } \ l_{ij} = 0 \ \textit{ for } i, j = 1, \dots, m.$$

Note that the set of *permissible* densities is a subset of *tolerant* densities, defined in Section 2.4.4.

Consider the following Markov chain:

$$T_i = k_0 \rightarrow k_1 \rightarrow \ldots \rightarrow k_i, \tag{3.18}$$

where $k_j = 1, 2, \ldots, i$ for $j = 1, \ldots, i$ are natural random numbers.

The rules for constructing the chain (3.18) are:

$$Pr(k_0 = \alpha) = p_\alpha, \quad Pr(k_j = \beta | k_{j-1} = \alpha) = p_{\alpha\beta}. \tag{3.19}$$

Assume that

$$p = \{p_\alpha\}_{\alpha=1}^m \in \mathcal{P}_h, \quad P = \{p_{\alpha\beta}\}_{\alpha,\beta=1}^m \in \mathcal{P}_L.$$

Now define the random variables Q_ν using the formula (3.16). One can see, that the random variables $Q_\nu, \nu = 1, \ldots, i$ can also be considered as weights on the Markov chain (3.19).

From all possible permissible densities we choose the following

$$p = \{p_\alpha\}_{\alpha=1}^m \in \mathcal{P}_h, \quad p_\alpha = \frac{|h_\alpha|}{\sum_{\alpha=1}^m |h_\alpha|};$$

$$P = \{p_{\alpha\beta}\}_{\alpha,\beta=1}^m \in \mathcal{P}_L, \quad p_{\alpha\beta} = \frac{|l_{\alpha\beta}|}{\sum_{\beta=1}^m |l_{\alpha\beta}|}, \alpha = 1, \ldots, m. \tag{3.20}$$

Such a choice of the initial density vector and the transition density matrix leads to an *Almost Optimal Monte Carlo* (MAO) algorithm. The initial density vector $p = \{p_\alpha\}_{\alpha=1}^m$ is called *almost optimal initial density vector* and the transition density matrix $P = \{p_{\alpha\beta}\}_{\alpha,\beta=1}^m$ is called *almost optimal density matrix* [Di91].

Let us consider Monte Carlo algorithms *with absorbing states*: instead of the finite random trajectory T_i in our algorithms we consider an infinite trajectory with a state coordinate $\delta_q (q = 1, 2, \ldots)$. Assume $\delta_q = 0$ if the trajectory is broken (absorbed) and $\delta_q = 1$ in other cases. Let

$$\Delta_q = \delta_0 \times \delta_1 \times \ldots \times \delta_q.$$

So, $\Delta_q = 1$ up to the first break of the trajectory and $\Delta_q = 0$ after that.

It is easy to show, that under the conditions (i) and (ii), the following equalities are fulfilled:

$$E\{Q_i f_{k_i}\} = (h, L^i f), \quad i = 1, 2, \ldots;$$

$$E\{\sum_{i=0}^n Q_i f_{k_i}\} = (h, u), \quad (P1),$$

$$E\{\sum_{i|k_i=r'} Q_i\} = c_{rr'}, \quad (P2),$$

where $(i|k_i = r')$ means a summation only for weights Q_i for which $k_i = r'$ and $C = \{c_{rr'}\}_{r,r'=1}^n$.

3.3 Convergence and Mapping

In this section we consider Monte Carlo algorithms for solving linear systems of equations and matrix inversion in the case when the corresponding Neumann series does not converge, or converge slowly.

To analyze the convergence of Monte Carlo algorithms consider the following functional equation

$$u - \lambda L u = f, \tag{3.21}$$

where λ is some parameter. Note that the matrices can be considered as linear operators. Define resolvent operator (matrix) R_λ by the equation

$$I + \lambda R_\lambda = (I - \lambda L)^{-1},$$

where I is the *identity operator.*

Let $\lambda_1, \lambda_2, \ldots$ be the eigenvalues of the equation (3.21), where it is supposed that

$$|\lambda_1| \geq |\lambda_2| \geq \ldots$$

Monte Carlo algorithms are based on the representation

$$u = (I - \lambda L)^{-1} f = f + \lambda R_\lambda f,$$

where

$$R_\lambda = L + \lambda L^2 + \ldots\,, \tag{3.22}$$

The systematic error of (3.22) when μ terms are used is

$$r_s = O[(|\lambda|/|\lambda_1|)^{\mu+1} \mu^{\rho-1}], \tag{3.23}$$

where ρ is the multiplicity of the roots of λ_1.

From (3.23) is follows that when λ is approximately equal to λ_1 the sequence (3.22) and the corresponding Monte Carlo algorithm converges slowly. When $\lambda \geq \lambda_1$ the algorithm does not converge.

Obviously, the representation (3.22) can be used for $\lambda : |\lambda| < |\lambda_1|$ to achieve convergence.

Thus, there are two problems to consider.

Problem 1. How can the convergence of the Monte Carlo algorithm be accelerated when the corresponding Neumann series converges slowly,

and

Problem 2. How can a Monte Carlo algorithm be constructed when the sequence (3.22)

$$R_\lambda = L + \lambda L^2 + \ldots\,,$$

does not converge.

To answer these questions we apply a *mapping* of the spectral parameter λ in (3.21).

The algorithm under consideration follows an approach which is similar to the algorithm used by L. Kantorovich & G. Akilov [KA77] and K. Sabelfeld [Sa89] for integral equations. In [KA77] the mapping approach is used for solving some problems of numerical analysis.

To extend these results it is necessary to show that the mapping approach can be extended for any linear operators (including matrices).

Consider the problem of constructing the solution of (3.21) for $\lambda \in \Omega$ and $\lambda \neq \lambda_k, k = 1, 2, \ldots$, where the domain Ω is a domain lying inside the definition domain of the $R_\lambda f$, such that all eigenvalues are outside of the domain Ω. In the neighborhood of the point $\lambda = 0$ ($\lambda = 0 \in \Omega$) the resolvent can be expressed by the series

$$R_\lambda f = \sum_{k=0}^{\infty} c_k \lambda^k,$$

where

$$c_k = L^{k+1} f.$$

Consider the variable α in the unit circle on the complex plane $\Delta(|\alpha| < 1)$.

The function

$$\lambda = \psi(\alpha) = a_1\alpha + a_2\alpha^2 + \ldots,$$

maps the domain Δ into Ω. Now it is possible to use the following resolvent

$$R_{\psi(\alpha)} f = \sum_{j=0}^{\infty} b_j \alpha^j \ , \tag{3.24}$$

where

$$b_j = \sum_{k=1}^{j} d_k^{(j)} c_k$$

and

$$d_k^{(j)} = \frac{1}{j!} \left[\frac{\partial^j}{\partial \alpha^j} [\psi(\alpha)]^k \right]_{\alpha=0} .$$

It is clear, that the domain Ω can be chosen so that it will be possible to map the value $\lambda = \lambda_*$ into point $\alpha = \alpha_* = \psi^{-1}(\lambda_*)$ for which the sequence (3.24) converges; hence the solution of the functional equation (3.21) can be presented in the following form:

$$u = f + \lambda_* R_{\psi(\alpha_*)} f,$$

where the corresponding sequence for $R_{\psi(\alpha)} f$ converges absolutely and uniformly in the domain Δ.

This approach is also helpful when the sequence (3.22) converges slowly.

To apply this approach one needs some information about the spectrum of the linear operator (respectively, the matrix). Let us assume, for example, that all eigenvalues λ_k are real and $\lambda_k \in (-\infty, -a]$, where $a > 0$. Consider a mapping for the case of interest ($\lambda = \lambda_* = 1$):

$$\lambda = \psi(\alpha) = \frac{4a\alpha}{(1-\alpha)^2}. \tag{3.25}$$

The sequence $R_{\psi(\alpha)}f$ for the mapping (3.25) converges absolutely and uniformly [KA77].

In Monte Carlo calculations we cut the sequence in (3.24) after μ terms

$$R_{\lambda_*}f \approx \sum_{k=1}^{\mu} b_k \alpha_k^k = \sum_{k=1}^{\mu} \alpha_*^k \sum_{i=1}^{k} d_i^{(k)} c_i = \sum_{k=1}^{\mu} g_k^{(\mu)} c_k, \tag{3.26}$$

where

$$g_k^{(\mu)} = \sum_{j=k}^{\mu} d_k^{(j)} \alpha_*^j. \tag{3.27}$$

The coefficients

$$d_k^{(j)} = (4a)^k q_{k,j}$$

and $g_k^{(\mu)}$ can be calculated in advance.

The coefficients $d_k^{(j)}$ for the mapping (3.26) are calculated and presented in Table 3.1 (for $k, j \leq 9$) .

k/j	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2		1	4	10	20	35	42	84	120
3			1	6	21	56	126	252	462
4				1	8	36	120	330	792
5					1	10	55	220	715
6						1	12	78	364
7							1	14	105
8								1	16
9									1

Table 3.1: **Table of the coefficients** $q_{k,j} = (4a)^{-k} d_k^{(j)}$ **for** $k, j \leq 9$

It is easy to see that the coefficients $q_{k,j}$ are the following binomial coefficients

$$q_{k,j} = C_{k+j-1}^{2k-1}.$$

In order to calculate the iterations $c_k = L^{k+1}f$ a Monte Carlo algorithm has to be used.

The mapping (3.25) creates the following Monte Carlo iteration process

$$\begin{aligned} u_0 &= f \\ u_1 &= 4aLu_0 \\ u_2 &= 4aLu_1 + 2u_1 \\ u_3 &= 4aLu_2 + 2u_2 - u_1 \\ u_j &= 4aLu_{j-1} + 2u_{j-1} - u_{j-2}, \;\; j > 2. \end{aligned} \tag{3.28}$$

and from (3.28) we have

$$u^{(k)} = 4a\alpha Lu^{(k-1)} + 2\alpha u^{(k-1)} - \alpha^2 u^{(k-2)} + f(1-\alpha^2), \;\; k > 2.$$

3.4 A Highly Convergent Algorithm for Systems of Linear Algebraic Equations

Suppose we have a Markov chain with i states. The random trajectory (chain) T_i of length i starting in the state k_0 was defined in Section 3.2 as follows

$$T_i = k_0 \to k_1 \to \cdots \to k_j \to \cdots \to k_i,$$

where k_j means the number of the state chosen, for $j = 1, 2, \cdots, i$.

Assume that

$$P(k_0 = \alpha) = p_\alpha, \quad P(k_j = \beta | k_{j-1} = \alpha) = p_{\alpha\beta},$$

where p_α is the probability that the chain starts in state α and $p_{\alpha\beta}$ is the transition probability to state β after being in state α. Probabilities $p_{\alpha\beta}$ define a transition matrix $P \in I\!R^{m\times m}$. We require that

$$\sum_{\alpha=1}^{m} p_\alpha = 1, \quad \sum_{\beta=1}^{m} p_{\alpha\beta} = 1, \quad \text{for any} \quad \alpha = 1, 2, ..., m.$$

Suppose the distributions created from the density probabilities p_α and $p_{\alpha\beta}$ are *permissible*, i.e. $p \in \mathcal{P}_h$ and $P \in \mathcal{P}_L$.

Now consider the problem of evaluating the inner product (3.12) $J(u) = (h, u) = \sum_{\alpha=1}^{m} h_\alpha u_\alpha$ of a given vector h with the vector solution of the system (3.14).

Define the random variable $\theta^*_\mu[h]$

$$\theta^*_\mu[h] = \frac{h_{k_0}}{p_0} \sum_{\nu=0}^{\mu} g_\nu^{(\mu)} Q_\nu f_{k_\nu}, \tag{3.29}$$

where $Q_0 = 1$, $g_0^{(\mu)} = 1$ and

$$Q_\nu = Q_{\nu-1}\frac{l_{k_{\nu-1},k_\nu}}{p_{k_{\nu-1},k_\nu}}, \qquad \nu = 1, 2, \ldots,$$

$(k_0, k_1, k_2, \ldots$ is a Markov chain with initial density function p_{k_0} and transition density function $p_{k_{\nu-1},k_\nu})$ and coefficients $g_j^{(\mu)}$ are defined by (3.27) for $j \geq 1$.

The following theorem is proved:

Theorem 3.4.1 *Consider matrix L, whose Neumann series (3.22) does not converge. Let (3.25) be the required mapping, so that the presentation (3.26) exists. Then*

$$E\left\{\lim_{\mu\to\infty}\frac{h_{k_0}}{p_0}\sum_{\nu=0}^{\mu} g_\nu^{(\mu)} Q_\nu f_{k_\nu}\right\} = (h, u).$$

Sketch of proof:

First consider the density of the Markov chain $k_0 \to k_1 \to \ldots \to k_i$ as a point in $m(i+1)$-dimensional Eucledian space $\mathbf{T_{i+1}} = \underbrace{I\!R^m \times \ldots \times I\!R^m}_{\mathbf{i+1}}$:

$$P\{k_0 = t_0, k_1 = t_1, \ldots, k_i = t_i\} = p_0 p_{t_0 t_1} p_{t_1 t_2} \ldots p_{t_{i-1} t_i}.$$

Now calculate the mathematical expectation of the random variable

$$\frac{h_{k_0}}{p_0} g_\nu^{(\mu)} Q_\nu f_{k_\nu}.$$

From the definition of the mathematical expectation it follows that:

$$E\left\{\frac{h_{k_0}}{p_0} g_\nu^{(\mu)} Q_\nu f_{k\nu}\right\} = \sum_{t_0,\ldots,t_\nu=1}^{\mu} \frac{h_{t_0}}{p_0} g_\nu^{(\mu)} Q_\nu f_{t_\nu} p_0 p_{t_0 t_1} \ldots p_{t_{\nu-1} t_\nu} =$$

$$\sum_{t_0,\ldots,t_\nu=1}^{\mu} h_{t_0} l_{t_0 t_1} l_{t_1 t_2} \ldots l_{t_{\nu-1} t_\nu} f_{t\nu} = (h, L^\nu f).$$

The existence and convergence of the sequence (3.27) ensures the following representations:

$$\sum_{\nu=0}^{\mu} E\left|\frac{h_{k_0}}{p_0} g_\nu^{(\mu)} Q_\nu f_{k\nu}\right| = \sum_{\nu=0}^{\mu} (|h|, |L|^\nu |f|) = \left(|h|, \sum_{\nu=0}^{\mu} |L|^\nu |f|\right),$$

$$E\left\{\lim_{\mu\to\infty}\frac{h_{k_0}}{p_0}\sum_{\nu=0}^{\mu} g_\nu^{(\mu)} Q_\nu f_{k_\nu}\right\} =$$

$$\sum_{\nu=0}^{\infty} E\left\{\frac{h_{k_0}}{p_0} g_\nu^{(\mu)} Q_\nu f_{k_\nu}\right\} = \sum_{\nu=0}^{\infty} (h, L^\nu f) = (h, u). \quad \diamondsuit$$

This theorem permits the use of the random variable $\theta_\mu^*[h]$ for calculating the inner product (3.12).

For calculating one component of the solution , for example the "r"th component of u, we must choose

$$h = e(r) = (0, ..., 0, 1, 0, ..., 0)^T,$$

where the one is in the "r"th position. It follows that

$$(h, u) = \sum_{\alpha}^{m} e_\alpha(r) u_\alpha = u_r$$

and the corresponding Monte Carlo algorithm is given by

$$u_r \approx \frac{1}{n} \sum_{s=1}^{n} \theta_\mu^*[e(r)]_s,$$

where n is the number of chains and

$$\theta_\mu^*[e(r)]_s = \sum_{\nu=0}^{\mu} g_\nu^{(\mu)} Q_\nu f_{k_\nu};$$

$$Q_\nu = \frac{l_{rk_1} l_{k_1k_2} \dots l_{k_{\nu-1}k_\nu}}{p_{rk_1} p_{k_1k_2} \dots p_{k_{\nu-1}p_\nu}}.$$

To find the inverse $C = \{c_{rr'}\}_{r,r'=1}^{m}$ of some matrix A we must first compute the elements of the matrix

$$L = I - A, \tag{3.30}$$

where I is the identity matrix. Clearly the inverse matrix is given by $C = \sum_{i=0}^{\infty} L^i$, which converges if $\|L\| < 1$. If the last condition is not fulfilled or if the corresponding Neumann series converges slowly we can use the same technique for accelerating the convergence of the algorithm.

Estimate the element $c_{rr'}$ of the inverse matrix C

Let the vector f given by (3.21) be the following unit vector

$$f_{r'} = e(r').$$

Theorem 3.4.2 *Consider matrix L, whose Neumann series (3.22) does not converge. Let (3.25) be the required mapping, so that representation (3.26) exists. Then*

$$E\left\{\lim_{\mu\to\infty} \sum_{\nu=0}^{\mu} g_\nu^{(\mu)} \frac{l_{rk_1} l_{k_1k_2} \dots l_{k_{\nu-1}k_\nu}}{p_{rk_1} p_{k_1k_2} \dots p_{k_{\nu-1}p_\nu}} f_{r'}\right\} = c_{rr'}.$$

Sketch of proof: The proof is similar to the proof of Theorem 3.4.1, but in this case we need to consider an unit vector $e(r)$ instead of vector h and vector $e(r')$ instead of $f_{k\nu}$:

$$E\left\{\frac{e(r)}{1} g_\nu^{(\mu)} Q_\nu f_{k_\nu}\right\} = (e(r), L^\nu f) = (L^\nu f)_r.$$

So, in this case the "r"-th component of the solution is estimated:

$$u_r = \sum_{i=1}^{m} c_{ri} f_i$$

When $f_{r'} = e(r')$, one can get:

$$u_r = c_{rr'},$$

that is:

$$E\left\{\lim_{\mu\to\infty} \sum_{\nu=0}^{\mu} g_\nu^{(\mu)} \frac{l_{rk_1} l_{k_1k_2} \dots l_{k_{\nu-1}k_\nu}}{p_{rk_1} p_{k_1k_2} \dots p_{k_{\nu-1}k_\nu}} e(r')\right\}$$

$$= \lim_{\mu\to\infty} \sum_{\nu=0}^{\infty} E\left\{g_\nu^{(\mu)} \frac{l_{rk_1} l_{k_1k_2} \dots l_{k_{\nu-1}k_\nu}}{p_{rk_1} p_{k_1k_2} \dots p_{k_{\nu-1}k_\nu}} e(r')\right\} = \sum_{\nu=0}^{\infty} (e(r), L^\nu e(r'))$$

$$= \left(e(r), \sum_{\nu=0}^{\infty} L^\nu e(r')\right) = \sum_{i=1}^{m} c_{ri} e(r') = c_{rr'}. \quad \diamondsuit$$

Theorem 3.4.2 permits the use of the following Monte Carlo algorithm for calculating elements of the inverse matrix C:

$$c_{rr'} \approx \frac{1}{n} \sum_{s=1}^{n} \left[\sum_{(\nu|k_\nu = r')}^{\mu} g_\nu^{(\mu)} \frac{l_{rk_1} l_{k_1k_2} \dots l_{k_{\nu-1}k_\nu}}{p_{rk_1} p_{k_1k_2} \dots p_{k_{\nu-1}p_\nu}} \right]_s,$$

where $(\nu|k_\nu = r')$ means that only the variables

$$Q_\nu^{(\mu)} = g_\nu^{(\mu)} \frac{l_{rk_1} l_{k_1k_2} \dots l_{k_{\nu-1}k_\nu}}{p_{rk_1} p_{k_1k_2} \dots p_{k_{\nu-1}p_\nu}}$$

for which $k_\nu = r'$ are included in the sum (3.30).

Observe that since $Q_\nu^{(\mu)}$ is only contained in the corresponding sum for $r' = 1, 2, \dots, m$ then the same set of n chains can be used to compute a single row of the inverse matrix, an important saving in computation which we exploit later.

3.4.1 Balancing of the errors

As it was mentioned in the introduction of Chapter 3 there are two errors in Monte Carlo algorithms: systematic and stochastic. It is clear that in order to obtain good results the stochastic error r_n (the probable error) must be approximately equal to the systematic one r_s, that is

$$r_n = O(r_s).$$

The problem of balancing the error is closely connected with the problem of obtaining an optimal ratio between the number of realizations n of the random variable and the mean value T of the number of steps in each random trajectory μ, i.e., $T = E(\mu)$.

Let us consider the case when the algorithm is applied to **Problem 1**. Using the mapping procedure and a random variable, defined by (3.29) we accelerate the convergence of the algorithm proposed by Curtiss [Cu54],[Cu56]. This means that for a fixed number of steps μ

$$r_s(\mu) < r_s^{(C)}(\mu), \tag{3.31}$$

where $r_s^{(C)}(\mu)$ is the systematic error of the Curtiss algorithm and $r_s(\mu)$ is the systematic error of the algorithm under consideration. A similar inequality holds for the probable errors. Since $g_k^{(\mu)}$ it follows that

$$\sigma(\theta^*) < \sigma(\theta) \tag{3.32}$$

and thus

$$r_n(\sigma(\theta^*)) < r_n^{(C)}(\sigma(\theta)), \tag{3.33}$$

where $r_n^{(C)}$ is the probable error for the Curtiss algorithm.

Next consider the general error

$$R = r_n(\sigma) + r_s(\mu)$$

for matrix inversion by our Monte Carlo approach. Let R be fixed. Obviously from (3.31) and (3.32) it follows that there exist constants $c_s > 1$ and $c_n > 1$, such that

$$r_s^{(C)}(\mu) = c_s r_s,$$

$$r_n^{(C)}(\sigma) = c_n r_n.$$

Since we are considering the problem of matrix inversion for a fixed general error R, we have

$$R = R^{(C)} = r_n^{(C)}(\sigma) + r_s^{(C)}(\mu) = c_n r_n(\sigma) + c_s r_s(\mu).$$

This expression shows that both parameters n and $T = E(\mu)$ or one of them, say n, can be reduced. In fact

$$\begin{aligned} c\sigma(\theta)/n_c^{1/2} + r_s^{(C)}(\mu) &= cc_n\sigma(\theta^*)/n_c^{1/2} + c_s r_s(\mu) \\ &= c\sigma(\theta^*)/n^{1/2} + r_s(\mu), \end{aligned}$$

or

$$c\sigma(\theta^*)/n^{1/2} = cc_n\sigma(\theta^*)/n_c^{1/2} + (c_s - 1)r_s(\mu)$$

and

$$\frac{1}{n^{1/2}} = \frac{c_n}{n_c^{1/2}} + \frac{(c_s - 1)r_s(\mu)}{c\sigma(\theta^*)}, \tag{3.34}$$

where n_c is the number of realizations of the random variable for Curtiss' algorithm.

Denote by b the following strictly positive variable

$$b = \frac{(c_s - 1)r_s(\mu)}{c\sigma(\theta^*)} > 0. \tag{3.35}$$

From (3.34) and (3.35) we obtain:

$$n = \frac{n_c}{\left(c_n + b n_c^{1/2}\right)^2}. \tag{3.36}$$

The result (3.36) is an exact result, but from practical point of view it may be difficult to estimate $r_s(m)$ exactly. However, it is possible using (3.35) to obtain the following estimate for n

$$n < \frac{n_c}{c_n^2}.$$

This last result shows that for the algorithm under consideration the number of realizations of the Markov chain n can be at least c_n^2 times less than the number of realizations n_c of the existing algorithm. Thus it is seen that there are a number of ways in which we can control the parameters of the fixed size array in [MAD94] In section 3.4.2 we explore some of these possibilities and develop some comparisons.

3.4.2 Estimators

Some estimates of n and the mathematical expectation for the length of the Markov chains T for Monte Carlo matrix inversion will now be outlined.

Using an almost optimal frequency function and according to the principle of collinearity of norms [Di91] $p_{\alpha\beta}$ is chosen proportional to the $|l_{\alpha\beta}|$ (see, (3.20)). Depending on estimates of the convergence of the Neumann series one of the following stopping rules can be selected to terminate Markov chains:

- **(i)** when $|Q_\nu^{(\mu)}| < \delta$;
- **(ii)** when a chain enters an absorbing state (see, Definition 1.0.8 in the Introduction).

In the case of a Monte Carlo algorithm without any absorbing states ($f_{k_j} = \delta_{k_j\beta}$ if $\alpha\beta$-th entry of inverse matrix can be computed) the bounds of T and $D\theta^*$ are

$$T \leq \frac{|\log \delta|}{|\log \|L\||}$$

and

$$D\theta^* \leq \frac{\|f\|^2}{(1 - \|L\|)^2} \leq \frac{1}{(1 - \|L\|)^2}.$$

Consider the Monte Carlo algorithms with absorbing states (see, Section 3.2) where $\hat{\theta}[h]$ denotes a random variable $\hat{\theta}_T[h]$ (T is the length of the chain when absorption takes place) taken over an infinitely long Markov chain.

The bounds on T and $D\hat{\theta}[h]$ [Cu54, Cu56] if the chain starts in state $r = \alpha$ and $p_{\alpha\beta} = |l_{\alpha\beta}|$, for $\alpha, \beta = 1, 2, ..., m$ are

$$E(T|r = \alpha) \leq \frac{1}{(1 - \|L\|)},$$

and

$$D\hat{\theta}[h] \leq \frac{1}{(1-\|L\|)^2}.$$

According to the error estimation (see, Section 2.1, formula (2.6))

$$n \geq \frac{0.6745^2}{\varepsilon^2} D\hat{\theta}[h]$$

for a given error ε. Thus

$$n \geq \frac{0.6745^2}{\varepsilon^2} \frac{1}{(1-\|L\|)^2}$$

is a lower bound on n.

If low precision solutions (e.g. $10^{-2} < \varepsilon < 1$) are accepted it is clear that $n >> m$ as $n \mapsto \infty$. Consider n and T as functions of

$$\frac{1}{(1-\|L\|)}.$$

Thus, in both algorithms T is bounded by $O(\sqrt{n})$, since in the Monte Carlo algorithm without any absorbing states

$$T < \sqrt{n} \frac{\varepsilon |\log \delta|}{0.6745}$$

and in the Monte Carlo algorithm with absorbing states

$$T \leq \sqrt{n} \frac{\varepsilon}{0.6745}.$$

Results in [DT93] show that $T \approx \sqrt{n}$, for sufficiently large n.

3.5 A New Iterative Monte Carlo Approach for Linear Systems and Matrix Inversion Problem

In this Section a new approach of the iterative Monte Carlo algorithms for the well known matrix inversion problem will be presented. The algorithms are based on special techniques of iteration parameter choice (refined stop-criteria), which permits to control the convergence of the algorithm for any row (column) of the matrix using a fine iterative parameter. The choice of this parameter is controlled by a posteriori criteria for every Monte Carlo iteration. The algorithms under consideration are also well parallelized.

3.5.1 Formulation of the problem

Here we deal again with Monte Carlo algorithms for calculating the inverse matrix A^{-1} of a square matrix A, i.e.

$$AA^{-1} = A^{-1}A = I,$$

where I is the identity matrix.

Consider the following system of linear equations:

$$Au = b, \tag{3.37}$$

where

$$A \in I\!R^{m\times m}; \quad b, u \in I\!R^{m\times 1}.$$

The inverse matrix problem is equivalent to solving m-times the problem (3.37), i.e.

$$Ac_j = b_j, \quad j = 1, \ldots, m \tag{3.38}$$

where

$$b_j \equiv e_j \equiv (0, \ldots, 0, 1, 0, \ldots, 0)$$

and

$$c_j \equiv (c_{j1}, c_{j2}, \ldots, c_{jm})^T$$

is the j-th column of the inverse matrix $C = A^{-1}$.

Here we deal with the matrix $L = \{l_{ij}\}_{ij=1}^m$, such that

$$L = I - DA, \tag{3.39}$$

where D is a diagonal matrix $D = diag(d_1, \ldots, d_m)$ and

$$d_i = \frac{\gamma}{a_{ii}}, \quad \gamma \in (0, 1] \;\; i = 1, \ldots, m.$$

The system (3.37) can be presented in the following form:

$$u = Lu + f, \tag{3.40}$$

where

$$f = Db.$$

Let us suppose that the matrix A has diagonally dominant property. In fact, this condition is too strong and the presented algorithms work for more general matrices, as it will be shown in Section 3.5.2. Obviously, if A is a diagonally dominant matrix, then the elements of the matrix L must satisfy the following condition:

$$\sum_{j=1}^{m} |l_{ij}| \leq 1 \qquad i = 1, \ldots, m. \tag{3.41}$$

3.5.2 New iterative Monte Carlo algorithms

Here new iterative Monte Carlo algorithms are considered. The first algorithm evaluates every component of the solution u of the following linear algebraic system (3.37).

Algorithm 3.5.1 *:*

```
1. Input initial data: the matrix $A$, the vector $\mathbf{b}$, the constants $\varepsilon, \gamma$ and $n$.
2. Preliminary calculations (preprocessing):
    2.1. Compute the matrix $L$ using the parameter $\gamma \in (0,1]$:
         $$\{l_{ij}\}_{i,j=1}^m = \begin{cases} 1-\gamma \text{ when } & i=j \\ -\gamma\frac{a_{ij}}{a_{ii}} \text{ when } & i \neq j\,. \end{cases}$$
    2.2. Compute the vector lsum:
         $$lsum(i) = \sum_{j=1}^m |l_{ij}| \quad for \;\; i = 1,2,\dots,m.$$
    2.3. Compute the transition probability matrix $P = \{p_{ij}\}_{i,j=1}^m$, where
         $$p_{ij} = \frac{|l_{ij}|}{lsum(i)}, \;\; i = 1,2,\dots,m \;\; j = 1,2,\dots,m\,.$$
3. For $i_0 := 1$ to $m$ do step 4 and step 5.
4. While $(W < \varepsilon)$ do the trajectory
    4.1. Set initial values $X := 0$ , $W := 1$ ;
    4.2. Calculate $X := X + Wf_{i_0}$ ;
    4.3. Generate an uniformly distributed random number $r \in (0,1)$;
    4.4. Set $j := 1$;
    4.5. If $(r < \sum_{k=1}^j p_{i_0 k})$ then
        4.5.1. Calculate $W := W sign(l_{i_0 j}) \times lsum(i_0)$ ;
        4.5.2. Calculate $X := X + Wf_j$ ( one move in trajectory );
        4.5.3. Update the index $i_0 := j$ and go to step 4.3.
                          else
        4.5.4. Update $j := j + 1$ and go to step 4.5.
5. Calculate the mean value based on $n$ independent trajectories:
    5.1. Do "$n$"-times step 4;
```

```
        5.2. Calculate $\overline{X}_n$ and $u_{i_0} := \overline{X}_n$.
    6. End of the Algorithm 3.5.1.
```

The Algorithm 3.5.1 describes the evaluation of every component of the solution of the problem (3.37), which is, in fact, linear algebraic system. Algorithm 3.5.1 is considered separately, since it (or some of its steps) will be used in next algorithms. For finding the corresponding "i"th component of the solution the following functional is used

$$V(u) = (v, u),$$

where $v = e_i = (0, 0, \ldots, \underbrace{1}_{i}, 0, \ldots, 0)$.

We consider the general description of the algorithm - the iteration parameter γ is inside the interval $(0, 1]$

The second algorithm computes the approximation $\hat{C}$ to the inverse matrix $C = A^{-1}$. The algorithm is based on special techniques of iteration parameter choice. The choice of the iteration parameter γ can be controlled by a posteriori criteria for every column of the approximate inverse matrix $\hat{C}$. The every column of this matrix is computed independently using Algorithm 3.5.1.

Algorithm 3.5.2 :

```
    1. Input initial data: the matrix $A$, the constant $\varepsilon$, $n$ and the vector $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_l) \in (0, 1]^l$.
    2. For $j_0 := 1$ to $m$ do
    Calculate the elements of $j_0$-th column of the approximate matrix $\hat{C}$:
        2.1. While $(k \leq l)$ do
        2.2. Apply the Algorithm 3.5.1 for $\gamma = \gamma_k$, $n$ and the right-hand side vector $b_{j_0} = (0, \ldots, 0, \underbrace{1}_{j_0}, 0, \ldots, 0)$ to obtain the column - vector $\hat{c}^k_{j_0} = (c^k_{1j_0}, \ldots, c^k_{mj_0})$.
        2.3. Compute the $l_2$-norm of the column - vector $\hat{c}^k_{j_0}$:
                 $$r^k_{j_0} = \sum_{j=1}^{m} \{ \sum_{i=1}^{m} a_{ji} c^k_{ij_0} - \delta_{jj_0} \}^2 .$$
        2.4. If $(r^k_{j_0} < r)$ then
                 $\hat{c}_{j_0} := c^k_{j_0}$;
                 $r := r^k_{j_0}$.
```

```
3. End of the Algorithm 3.5.2.
```

Algorithm 3.5.2 is based on Algorithm 3.5.1 finding different columns of the matrix $\hat{C}$ by using corresponding values of the iteration parameter $\gamma = \gamma_i, i = 1, 2, \ldots, l$. The values of γ_i are chosen such that to minimize the l_2-norm of the following vectors:

$$E_j = A\hat{C}_j - I_j^T, \quad j = 1, 2, \ldots, m, \tag{3.42}$$

where $I_j = (0, \ldots, 0, \underbrace{1}_{j}, 0, \ldots, 0)$.

The use of the criteria of minimization of the l_2-norm of the vector E_j permits to find better approximation of the error matrix

$$E = A\hat{C} - I.$$

This procedure allows to minimize the norm (for example, the Frobenius norm) of E. In practice, the parameter γ runs a finite numbers of values in the interval $(0, 1]$.

The evaluation of different columns can be realized in parallel and independently.

The algorithm presented above uses a deterministic approach, which is independent of the statistical nature of the algorithm.

Algorithm 3.5.3 :

```
1. Input initial data: the matrix $A$, the constant $n$ and the vectors $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_l) \in (0,1]^l$, $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_m)$ and $r = (r_1, r_2, \ldots, r_m)$.
2. While $(k \leq l)$ do
    2.1. Step 2 of Algorithm 3.5.1 for $\gamma := \gamma_k$.
    tem2.2. While $(i_0 \leq m)$ and $(j_0 \leq m)$ do step 4 and step 5 of the Algorithm 3.5.1 to compute the elements $\hat{c}^k_{i_0 j_0}$ for $\gamma = \gamma_k$ , $\varepsilon := \varepsilon_{i_0}$ and the right-hand side vector $b_{j_0} := (0, \ldots, 0, \underbrace{1}_{j_0}, 0, \ldots, 0)$ .
    2.3. For $i_0 := 1$ to $m$ do
        2.3.1. Calculate
            $r^k_{i_0} = \max_{i \in \{1,2,\ldots,m\}} |\sum_{j=1}^{m} c_{i_0 j} a_{ji} - \delta_{i_0 i}|.$
        2.3.2. If $(r^k_{i_0} < r_{i_0})$ then
            $\hat{c}_{i_0} := c^k_{i_0}$;
            $r_{i_0} := r^k_{i_0}$.
3. End of the Algorithm 3.5.3.
```

The difference between the last two algorithms is that the Algorithm 3.5.3 can not be applied in traditional (non-stochastic) iterative algorithms. The traditional algorithms allow to evaluate the columns of the inverse matrix in parallel, but they do not allow to obtain their elements independently from each other. The advantage of the Monte Carlo algorithms consists in possibilities to evaluate every element of the inverse matrix in an independent way. This property allows to apply different iteration approaches for finding the matrix $\hat{C}$ using a priori information for the rows of the given matrix A (for example, the ratio of the sum of the modulus of the non-diagonal entrances to the value of the diagonal element).

One has to mention that the computational complexity of Algorithm 3.5.3 also depends on "how ill-conditioned" is the given row of the matrix A. The given row A_i of the matrix A_i is "ill-conditioned", when the condition

$$|a_{ii}| < \sum_{j=1}^{i-1} |a_{ij}| + \sum_{j=i+1}^{m} |a_{ij}|$$

of *diagonally dominating* is not fulfilled (but all the eigenvalues lay inside of the unit circle).

The **Algorithm 3.5.3** presented above is very convenient for such matrices since it chooses the value of the iterative parameter γ for every row of the matrix A. As a measure of the ill-conditioning of a given row we use the following parameter:

$$b_i = \sum_{j=1}^{i-1} |a_{ij}| + \sum_{j=i+1}^{m} |a_{ij}| - |a_{ii}|.$$

The possibility to treat non-diagonally dominated matrices increases the set of the problems treated using Monte Carlo algorithms. For finding different rows of the approximation of the inverse matrix $\hat{C}$ different number of moves (iterations) can be used. The number of moves are controlled by some parameter ε. For an a posteriori criteria we use the minimization of the C-norm of the following row-vector

$$E_i = \hat{C}_i A - I_i, \quad i = 1, \dots, m,$$

where $\hat{C}_i = (0, \dots, 0, \underbrace{1}_{i}, 0, \dots, 0)$.

The use of the criteria of minimization of the C-norm of the vector E_i permits to find better approximation of the error matrix

$$E = \hat{C}A - I. \tag{3.43}$$

The above mentioned procedure allows to minimize the norm (for example, the Frobenius norm) of the matrix E.

One can also control the number of moves in the Markov chain (that is the number of iterations) such that to have a good balance between the stochastic and systematic error (i.e., the truncation error). The problem of balancing of both - systematic and stochastic error is very important when Monte Carlo algorithms are used. It is clear that in order to obtain good result s the stochastic error (the probable error) r_n must be approximately equal to the systematic one r_s, that is

$$r_n = O(r_s).$$

The problem of balancing the errors is closely connected with the problem of obtaining an optimal ratio between the number of realizations n of the random variable and the mean value T of the number of steps in each random trajectory. The balancing allows to increase the accuracy of the algorithm for a fixed computational complexity, because in this case one can control the parameter $E(Y)$ by choosing different lengths of the realizations of the Markov chain. In practice, we choose the absorbing state of the random trajectory using the well known criteria

$$|W| < \varepsilon.$$

Such a criteria is widely used in iterative algorithms, but obviously it is not the best way to define the absorbing states of the random trajectory. It is so, because for different rows of the matrix A the convergence of the corresponding iterative process (and, thus, the truncation error) may be different. If the rate of convergence is higher it is possible to use a higher value for the parameter ε and to cut the random trajectory earlier then in the case of lower convergence. Our approach permits to use different stop-criteria for different random trajectories, which allows to optimize the algorithm is the sense of balancing of errors.

3.5.3 Discussion of the numerical results

As an example we consider matrices arising after applying the mixed finite element algorithm for the following boundary value problem

$$\left| \begin{array}{l} -\mathrm{div}(a(x)\underline{\nabla} p) = f(x), \text{ in } \Omega \\ p = 0, \text{ on } \partial\Omega, \end{array} \right. \tag{3.44}$$

where $\underline{\nabla} w$ denotes the gradient of a scalar function w, $div\underline{v}$ denotes the divergence of the vector function $\underline{v}$ and $a(x)$ is a diagonal matrix whose elements satisfy the requirements $a_i(x) \geq a_0 > 0, \ i = 1, 2$.

We set

$$\underline{u} \equiv (u_1, u_2) = a(x)\underline{\nabla} p, \alpha_i(x) = a_i(x)^{-1}, i = 1, 2.$$

Let us consider the spaces $\underline{V}$ and W defined by

$$\begin{array}{rcl} \underline{V} & = & \underline{H}(div;\Omega) = \{\underline{v} \in L^2(\Omega)^2 \ : \ div\underline{v} \in L^2(\Omega)\}, \\ W & = & L^2(\Omega) \end{array}$$

provided with the norms

$$\begin{array}{rcccl} \|\underline{v}\|_{\underline{V}} & \equiv & \|\underline{v}\|_{\underline{H}(div;\Omega)} & = & (\|\underline{v}\|^2_{0,\Omega} + \|div\underline{v}\|^2_{0,\Omega})^{1/2} \text{ and} \\ \|w\|_W & = & \|w\|_{L^2(\Omega)} & = & \|w\|_{0,\Omega} \end{array}$$

respectively.

Then the mixed variational formulation of the problem (3.44) is given by characterizing the pair $(\underline{u}, p)$, as the solution of

$$\left| \begin{array}{ll} a(\underline{u}, \underline{v}) + b(\underline{v}, p) = 0, & \forall \underline{v} \in \underline{V}; \\ b(\underline{u}, w) = -(f, w), & \forall w \in W, \end{array} \right. \tag{3.45}$$

where

$$a(\underline{u}, \underline{v}) = (\alpha u_1, v_1) + (\alpha u_2, v_2), \quad b(\underline{u}, w) = (div\underline{u}, w)$$

and $(\cdot, \cdot)$ indicated the inner product in $L^2(\Omega)$.

The mixed finite element approximation of the problem (3.45) in the Raviart-Thomas spaces leads to the following linear algebraic system:

$$Ku = \begin{pmatrix} A_1 & 0 & B_1 \\ 0 & A_2 & B_2 \\ B_1^T & B_2^T & 0 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ p \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -f \end{pmatrix}, \tag{3.46}$$

where A_i are $m \times m$ matrices, B_i are $m \times m_1$ matrices $(m_1 < m)$, $u_i \in I\!R^m$ and $p, f \in I\!R^{m_1}$, $i = 1, 2$.

If $A_i^{-1}\,(i = 1, 2)$ is obtained then the system (3.46) becomes

$$Bp = f,$$

where

$$B = B_1^T A_1^{-1} B_1 + B_2^T A_2^{-1} B_2$$

Thus we reduce the $2m + m_1$-dimensional linear algebraic system to the m_1-dimensional system.

For matrices $A = A_i, i = 1, 2$ the Algorithm 3.5.2 is applied. Numerical examples for a matrix $A \in I\!R^{16\times16}$, for different values of the parameter γ are presented.

The values of the parameter γ for different columns of the matrix $\hat{C}$ are shown in Table 3.2.

As a basic test example for applying Algorithm 3.5.3 a matrix of of size 7 is used. The size of the matrix is relatively small, because our aim was only to demonstrate how the Algorithm 3.5.3 works. Here we also have to mention that the computational complexity of the algorithm practically does not depend of the size of the matrix. Using the technique [DK96] it is possible to show that the computational complexity of our algorithms depends linearly of the mean value of the number of non-zero entrances per row. This is very important, because it means that very large sparse matrices could be treated efficiently using the algorithms under consideration.

During the numerical tests we control the Frobenius norm of the matrices, defined by

$$\| A \|_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{m} a_{ij}^2.$$

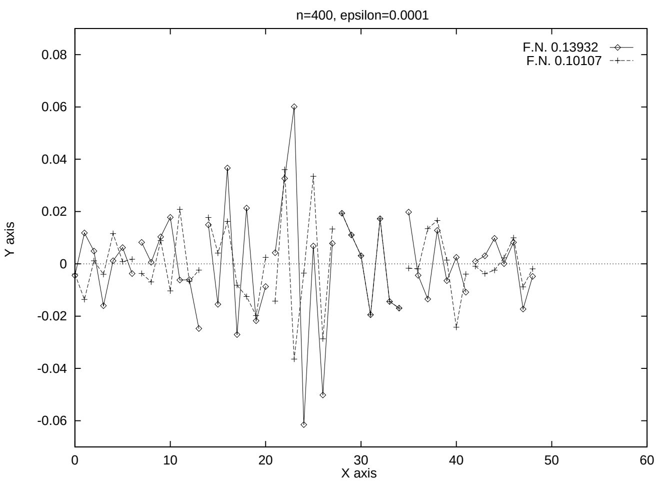

Figure 3.1
Frobenius norm– non-balanced case

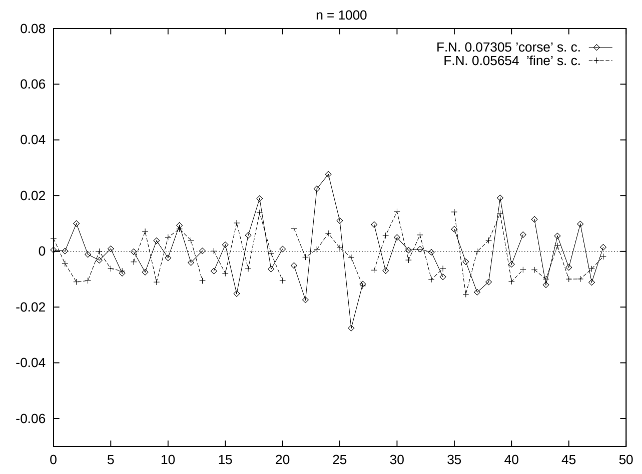

Figure 3.2
Frobenius norm– balanced case

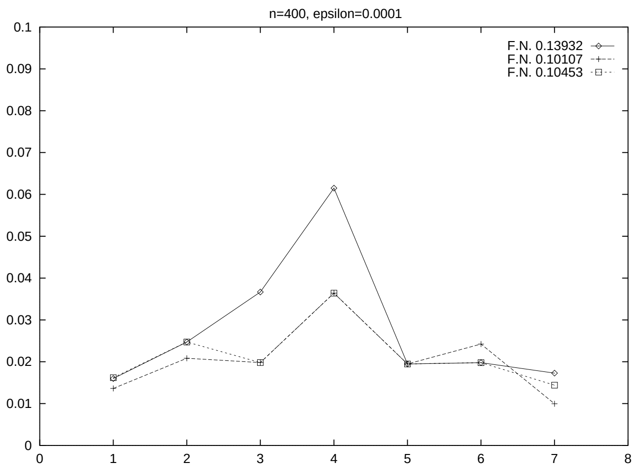

Figure 3.3
Non-controlled balance

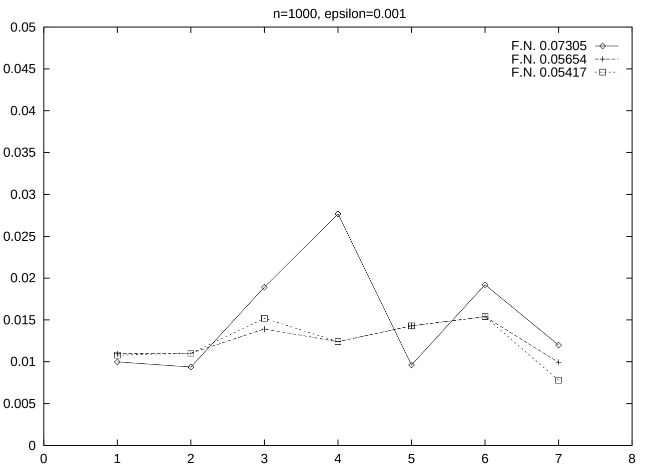

Figure 3.4
Controlled balance

Some of the numerical results performed are shown on Figure 3.1 – 3.8, provided by Table 3.3. On all Figures the value of the Frobenius norm is denoted by F.N., the number of realizations of the random variable (i.e., the number of random trajectories) is denoted by n and the value of the stop-criteria is denoted by ε. In **Algorithm 3.5.3** we use m values of ε (in our case $m = 7$).

Figure 3.1 presents the values of the error matrix (3.43) in the both cases under consideration – coarse stop criteria (' $\diamondsuit$ ') and fine stop criteria ('+'). The first set of connected points corresponds to values of the first row of the error matrix, the second set – to the second row of the same matrix, etc. When the coarse stop criteria is used $\varepsilon = 0.0001$. When the fine stop criteria is used different values of ϵ are applied such that the computational complexity is smaller (in comparison with the case if the coarse stop criteria) (see, also Table 3.3). The values of the Frobenius norm for both cases when the number of realizations n is equal to 400 are also given. For such

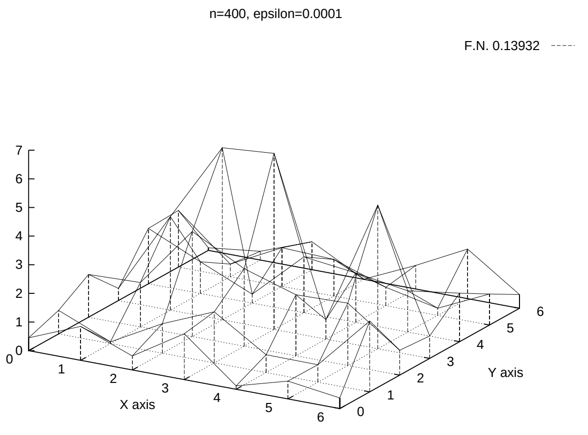

Figure 3.5
Coarse stop-criteria

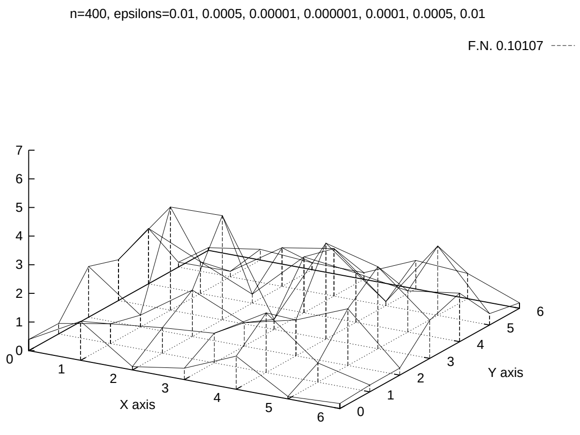

Figure 3.6
Use of different fine stop-criteria

number of realizations the stochastic error is relatively large in comparison with the systematic one. So, the results on Figure 3.1 correspond to the non-balanced case.

The similar results, but for the case of $n = 1000$ and $\varepsilon = 0.001$ (for the coarse stop criteria) are presented on Figure 3.2. One can see, that

- ε is 10 times large then in the previous case, but the Frobenius norm is about two times smaller, because the number of realizations is larger.

The results presented on Figure 3.1 and Figure 3.2 show the statistical convergence of the algorithm, i.e. the error decreases when n increases (even in the case when the parameter ε increases).

These results show how important is to have a good balancing between the stochastic and systematic error. The computational effort for the cases presented on Figure 3.1 and Figure 3.2 is approximately equal, but the results in the case of Figure 3.2, when we have a good balancing are almost 2 times better.

Let us discuss the result presented on Figures 3.3 and 3.4. Here instead of elements of the error matrix the maximum of the modulo element for every row are shown. If the computational complexity for a constant ε is denoted by R_c and and the computational complexity when different values of $\varepsilon = \varepsilon_i, i = 1, \ldots, m$ is denoted by R_f we consider the case when

$$R_c \geq R_f = 1.$$

The results presented on Figures 3.3 and 3.4 show that apart from the less computational complexity R_f of the fine stop criteria algorithm it gives better results than the coarse stop criteria algorithm with complexity R_c. This fact is observed in both cases - balanced (Figure 3.3) and non-balanced (Figure 3.4).

- the variations of the estimations are smaller when the balancing is better;
- the Frobenius norm is smaller, when the control "row per row" is realized.

column number	l_1 norm				C norm			
	$\varepsilon =$ 0.05	0.01	0.005	0.001	0.05	0.01	0.005	0.001
1	1	0.9	0.6	0.2	0.5	0.1	1	1
2	0.4	0.8	0.9	0.8	0.5	0.9	0.6	1
3	1	0.8	0.8	0.9	0.5	1	0.3	0.1
4	0.5	1	0.7	0.7	0.3	0.3	1	0.9
5	0.9	0.5	0.9	0.9	1	1	0.9	0.8
6	0.8	0.1	0.6	0.6	1	0.8	0.9	0.8
7	0.5	0.1	0.9	0.9	0.8	0.4	0.9	1
8	0.5	0.1	0.6	0.9	0.8	0.8	0.3	1
9	0.5	0.1	0.6	0.6	0.6	1	1	0.2
10	0.5	0.1	0.6	0.3	0.6	1	0.4	0.5
11	0.5	0.1	0.6	0.3	0.5	0.1	1	0.5
12	0.5	0.1	0.6	0.3	0.7	0.8	1	0.8
13	0.5	0.1	0.6	0.3	1	1	0.4	0.9
14	0.5	1	0.6	0.3	0.9	0.9	0.4	1
15	1	0.1	0.8	0.9	1	1	1	0.4
16	0.9	0.3	0.6	0.1	1	1	0.9	0.6

Table 3.2: Connection between ε and the parameter γ. Here $m = 16$, $n = 24$.

	not balanced case		balanced case	
γ	'coarse' s. c.	'fine' s. c.	'coarse' s. c.	'fine' s. c.
0.2	1.0806	1	1.0368	1
0.4	1.0903	1	1.0351	1
0.6	1.0832	1	1.0348	1
0.8	1.0910	1	1.0360	1
1	1.0862	1	1.0342	1
generally	1.0848	1	1.0358	1

Table 3.3: Computational complexity

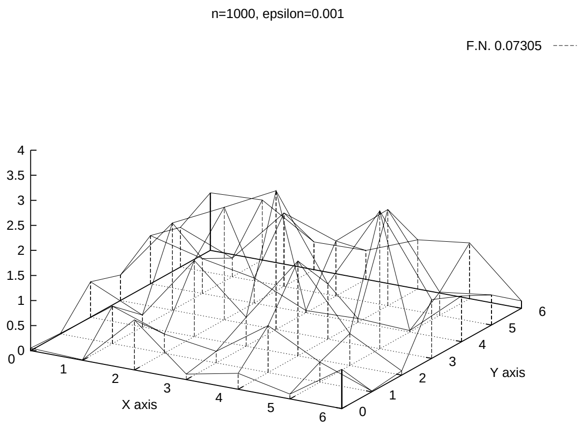

Figure 3.7
Controlled balancing – coarse stop criteria

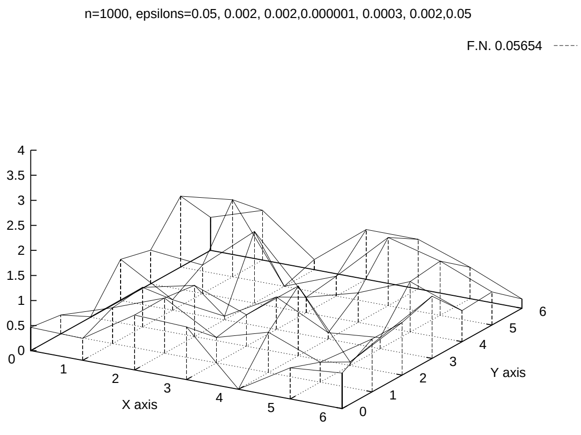

Figure 3.8
Controlled balancing – fine stop criteria

Figures 3.5 and 3.6 present test results for the modulo of every element of the error matrix (3.43) when the coarse stop criteria and fine stop criteria respectively are used in the non-balanced case.

One can see, that

- the Frobenius norm of the estimate in the case of fine stop criteria is about 1.4 times smaller than the corresponding value for the coarse stop criteria, and
- the variances of the estimate of the case of fine stop criteria are smaller.

Figures 3.7 and 3.8 show the corresponding results as on Figures 3.5 and 3.6 in the balanced case. One can make the same conclusion as in the non balanced case, but here

- the Frobenius norm is almost 2 times smaller.

3.5.4 Conclusion

An iterative Monte Carlo algorithm is presented and studied. This algorithm can be applied for solving of inverse matrix problems.

The following conclusion can be done:

- Every element of the inverse matrix A^{-1} can be evaluated independently from the other elements (this illustrates the inherent parallelism of the algorithms under consideration);
- Parallel computations of every column of the inverse matrix A^{-1} with different iterative procedures can be realized;

- It is possible to optimize the algorithm using error estimate criterion ”column by column”, as well as ”row by row”;
- The balancing of errors (both, systematic and stochastic) allows to increase the accuracy of the solution if the computational effort is fixed or to reduce the computational complexity if the error is fixed.

The studied algorithm is easily programmable and parallelizable and can be efficiency implemented on MIMD-machines.

3.6 Monte Carlo Algorithms for Computing Eigenvalues

In this section a new Monte Carlo approach for evaluating the eigenvalues of real symmetric matrices shall be proposed. Algorithms for both - largest and smallest eigenvalue will be considered. It is known that the problem of calculating the smallest eigenvalue of a matrix A is more difficult from numerical point of view than the problem of evaluating the largest eigenvalue. Nevertheless, for many important applications in physics and engineering it is necessary to estimate the value of the smallest eigenvalue, because it usually defines the most stable state of the system which is described by the considered matrix. Therefore we shall consider algorithms for the smallest eigenvalue.

There are, also, many problems in which it is important to have an efficient algorithm which is parallel and/or vectorizable. And for matrices with a large size which often appear in practice it is not easy to find efficient algorithms for evaluating the smallest eigenvalue when modern high–speed vector or parallel computers are used. Let us consider as an example, the problem of plotting the spectral portraits of matrices which is one of the important problems where highly efficient vector and parallel algorithms are needed [1].

[1]In fact, the pseudo-spectrum or ϵ-spectrum [Go91], [Tr91] of a matrix B is defined by

$$\sigma_\epsilon = \left\{ z \in \mathbf{C} : \|(zI - B)^{-1}\|_2 \geq \frac{1}{\epsilon} \right\}.$$

The main problem related to computing the spectral portrait, consists in the evaluation of

$$z \to \|(zI - B)^{-1}\|_2 \quad \text{for} \quad z \in \mathbf{C}.$$

It is clear that (3.4) can be represented as

$$z \to \frac{1}{\sigma_{\min}(zI - B)},$$

where $\sigma_{\min}(G)$ is the smallest singular value of G. The evaluation of the smallest singular value of a matrix can be performed in different ways. We use the following representation for evaluating $\sigma_{\min}(zI - B)$:

$$\sigma_{\min}(zI - B) = \sqrt{\lambda_{\min}((zI - B)^*(zI - B))},$$

The spectral portraits are used in stability analysis. The above mentioned problem leads to a large number of subproblems of evaluating the smallest eigenvalue of symmetric matrices.

Two Monte Carlo Almost Optimal (MAO) algorithms will be presented. The first one is called *Resolvent Monte Carlo algorithm* (RMC) and uses Monte Carlo iterations by the resolvent matrix. The second one is called *Inverse Monte Carlo Iterative algorithm* (IMCI) and uses the representation of the smallest eigenvalue by inverse Monte Carlo iterations.

Estimators for speedup as well as for parallel efficiency are considered for different typical models of computer architectures.

Results of numerical tests for a number of matrices - general symmetric dense matrices, sparse symmetric matrices (including band sparse symmetric matrices) with different behaviors will be discussed.

3.6.1 Formulation of the problem

Consider the following problem of evaluating eigenvalues:

$$Au = \lambda(A)u. \tag{3.47}$$

It is assumed that

$$(iii) \qquad \begin{cases} 1. \;\; A \text{ is a symmetric matrix, i.e. } a_{ij} = a_{ji} \text{ for all } i,j = 1,\dots,m; \\ 2. \;\; \lambda_{min} = \lambda_m < \lambda_{m-1} \le \lambda_{m-2} \le \dots \le \lambda_2 < \lambda_1 = \lambda_{max}. \end{cases}$$

under the conditions (iii) the following equality is fulfilled:

$$\frac{E\{W_i f_{k_i}\}}{E\{W_{i-1} f_{k_{i-1}}\}} \approx \lambda_1(A), \quad \text{for sufficiently large ''i''}.$$

3.6.2 The Resolvent Monte Carlo algorithm (RMC)

Now consider an algorithm based on Monte Carlo iterations by the matrix A resolvent operator $[I - qA]^{-1}$.

It is known [KA77] that

$$[I - qA]^{-r} = \sum_{i=0}^{\infty} q^i C^i_{r+i-1} A^i, \;\; |q|\lambda < 1;$$

the eigenvalues of the operators $[I - qA]^{-1}$ and A are connected with the equality $\mu = \frac{1}{(1-q\lambda)}$, and the eigenfunctions coincide. Accordingly eigenvalues of the matrix A,

where $(zI - B)^*$ denotes the conjugate transpose of $(zI - B)$.

So, the problem consists in evaluating the smallest eigenvalue of a matrix A. Here we consider the case, when $A = (zI^* - B)(zI - B)$ is a symmetric matrix.

assuring the convergence of the Monte Carlo iterations, yield the following expression

$$\mu = \frac{([I-qA]^{-r}f,h)}{([I-qA]^{-(r-1)}f,h)} \to_{r\to\infty} \mu = \frac{1}{1-q\lambda}, \quad f \in I\!R^{m\times 1}, h \in I\!R^{m\times 1}.$$

For negative values of q, the largest μ, μ_1, corresponds to the smallest eigenvalue λ_{min} of the matrix A.

Now, for constructing the algorithm it is sufficient to note that

$$([I-qA]^{-r}f,h) = \sum_{i=0}^{\infty} q^i C^i_{r+i-1}(A^i f,h)$$

$$\approx E\sum_{i=0}^{n} q^i C^i_{r+i-1} W_i h(x_i).$$

After some calculations we obtain

$$\lambda \approx \frac{1}{q}\left(1-\frac{1}{\mu^{(r)}}\right) = \frac{(A[I-qA]^{-r}f,h)}{([I-qA]^{-r}f,h)}$$

$$= \frac{E\sum_{i=1}^{\infty} q^{i-1}C^{i-1}_{i+r-2}W_i h(x_i)}{E\sum_{i=0}^{\infty} q^i C^i_{i+r-1}W_i h(x_i)}.$$

The coefficients C^n_{n+m} are calculated using the representation

$$C^i_{i+r} = C^i_{i+r-1} + C^{i-1}_{i+r-1}.$$

Let $A = \{a_{\alpha\beta}\}^m_{\alpha,\beta=1}$ be $m \times m$ symmetric matrix. Consider a *permissible* to the vector $h = \{h_\alpha\}^m_{\alpha=1} \in I\!R^{m\times 1}$ transition density vector $p = (p_1, p_2, \dots, p_m) = \{p_\alpha\}^m_{\alpha=1} \in \mathcal{P}_h$ and a *permissible* to the matrix $A = \{a_{\alpha\beta}\}^m_{\alpha,\beta=1} \in I\!R^{m\times m}$ transition density matrix $P = \{p_{\alpha\beta}\}^m_{\alpha,\beta=1} \in \mathcal{P}_A$, defined in Section 3.2.

Obviously, p_α and $p_{\alpha\beta}$ are to be nonnegative and satisfy the conditions

$$\sum_{\alpha=1}^{m} p_\alpha = 1, \quad \sum_{\beta=1}^{m} p_{\alpha\beta} = 1, \quad \text{for any} \quad \alpha = 1,2,...,m. \tag{3.48}$$

The value p_α might be interpreted as the probability that the initial point of the trajectory will fall on h_α, and $p_{\alpha\beta}$ as the probability that the random process will pass on the element of β-th column of the α-th row. Under this interpretation it is sufficient to consider the following Markov chain:

$$T_i = k_0 \to k_1 \to \dots \to k_i. \tag{3.49}$$

Construct the chain of random variables (3.49), whose values can be ranged from 1 to n. The rules for constructing the chain (3.49) are:

$$P(k_0 = \alpha) = p_\alpha, \quad P(k_j = \beta | k_{j-1} = \alpha) = p_{\alpha\beta}, \tag{3.50}$$

where the initial probabilities p_α and the transition probabilities $p_{\alpha\beta}$ satisfy the conditions (3.48). The chains (3.50) are Markov chains with finite number of states.

Now consider the random variable W_j introduced above (W_j can also be accepted as weights on the Markov chain states). Let

$$W_j = \frac{a_{k_0k_1}a_{k_1k_2}\dots a_{k_{j-1}k_j}}{p_{k_0k_1}p_{k_1k_2}\dots p_{k_{j-1}k_j}}, \quad W_0 = \frac{h_{k_0}}{p_{k_0}}, \tag{3.51}$$

or, using a recursion formula,

$$W_j = W_{j-1}(a_{k_{j-1}k_j}/p_{k_{j-1}k_j}), \quad j = 1,\dots,n, \quad W_0 = \frac{h_{k_0}}{p_{k_0}}.$$

From all possible permissible initial density vectors $p = \{p_\alpha\}_{\alpha=1}^m$ and from all possible permissible transition density matrices $P = \{p_{\alpha\beta}\}_{\alpha,\beta=1}^m$ we choose the following

$$p = \{p_\alpha\}_{\alpha=1}^m, \quad p_\alpha = |h_\alpha| / \left(\sum_{\alpha=1}^m |h_\alpha|\right),$$

and

$$P = \{p_{\alpha\beta}\}_{\alpha,\beta=1}^m, \quad p_{\alpha\beta} = |a_{\alpha\beta}| / \left(\sum_{\beta=1}^m |a_{\alpha\beta}|\right), \quad \text{for} \quad \alpha = 1,2,\dots,m,$$

so that we have a MAO algorithm, defined in Section 3.2.

From the representation

$$\mu^{(r)} = \frac{1}{1-|q|\lambda^{(r)}} \approx \frac{(h,[I-qA]^{-r}f)}{(h,[I-qA]^{-(r-1)}f)},$$

we obtain the following Resolvent Monte Carlo (RMC) algorithm for evaluating the smallest eigenvalue:

$$\lambda \approx \frac{1}{q}\left(1-\frac{1}{\mu^{(r)}}\right) \approx \frac{E\sum_{i=0}^{l} q^i C_{i+r-1}^i W_{i+1} h(x_i)}{E\sum_{n=0}^{l} q^i C_{i+r-1}^i W_i h(x_i)},$$

where $W_0 = \frac{h_{k_0}}{p_{k_0}}$ and W_i are defined by (3.51).

The parameter $q < 0$ has to be chosen so as to minimize the following expression

$$J(q,A) = \frac{1+|q|\lambda_1}{1+|q|\lambda_2},$$

or, if $\lambda_1 = \alpha\lambda_2, \quad (0 < \alpha < 1)$,

$$J(q,A) = 1 - \frac{|q|\lambda_2(1-\alpha)}{1+|q|\lambda_2}.$$

We choose

$$q = -\frac{1}{2\|A\|},$$

but sometimes slightly different value of q may give better results when a number of realizations of the algorithm is considered.

Since the initial vector f can be any vector $f \in I\!R^{m\times 1}$ the following formula for calculating λ_{min} is used

$$\lambda \approx \frac{E\{W_1 + qC_r^1 W_2 + q^2 C_{r+1}^2 W_3 + \ldots + q^l C_{l+r-1}^l W_{l+1}\}}{E\{1 + qC_r^1 W_1 + q^2 C_{r+1}^2 W_2 + \ldots + q^l C_{l+r-1}^l W_l\}},$$

that is

$$\lambda \approx \frac{\frac{1}{N}\sum_{s=1}^n \{\sum_{i=0}^l q^i C_{i+r-1}^i W_{i+1}\}_s}{\frac{1}{n}\sum_{s=1}^n \{\sum_{i=0}^l q^i C_{i+r-1}^i W_i\}_s}$$

Now we can formulate a **Resolvent Monte Carlo algorithm (RMC) for evaluating the smallest eigenvalue of symmetric matrices**

Algorithm 3.6.1

1. Choose *an initial row of the matrix* $A = \{a_{ij}\}_{i,j=1}^m$ *as a realization of the density* p_0 *permissible to the vector* h*: divide the interval* $[0,1]$ *in subintervals with lengths proportional of the vector coordinates* $h_1, h_2, \ldots, h_m$*:*

$$\triangle p_i = c|h_i|, \qquad i = 1, \ldots, n, \tag{3.52}$$

where c is a constant. Then generate a random number in $[0,1]$*.*

Suppose the element h_{k_0} *has been chosen.*

2. *Consider the "*k_0*"th row of the matrix A.* **Choose** *an element of "*k_0*"th row as a realization of the transition density function* $p_{k_0k_1}$ *in the following way: divide the interval* $[0,1]$ *into subintervals* $\triangle p_i$*,* $\quad i = 1, \ldots, m,$ *proportional to the row elements*

$$\triangle p_i = c_{k_0}|a_{k_0 i}|, \qquad i = 1, 2, \ldots, m, \tag{3.53}$$

where c_{k_0} *is a constant, and generate a random number in* $[0,1]$*. Let the chosen element be* $a_{k_0k_1}$*.*

3. *Consider the "*k_1*"th row of the matrix A and repeat the procedure* **2** *using density functions* $p_{k_0k_1}$*.* **Continue** *this procedure until determining the element* $a_{k_ik_{i+1}}$ *according to the density* $p_{k_ik_{i+1}}$ *.*

4.Compute *the random variables (for a fixed integer value of* l *and* m*)*

$$\theta[u]_\nu^{(1)}[h, A] = \sum_{i=0}^{\nu} q^i C_{i+r-1}^i W_{i+1}, \quad \nu = l-1, l \tag{3.54}$$

$$\theta[d]_\nu^{(1)}[h, A] = \sum_{i=0}^{\nu} q^i C_{i+r-1}^i W_i, \quad \nu = l-1, l \tag{3.55}$$

where

$$W_i = \frac{h_{k_0}}{p_{k_0}} \frac{a_{k_0k_1} a_{k_1k_2} \ldots a_{k_{i-1}k_i}}{p_{k_0k_1} p_{k_1k_2} \ldots p_{k_{i-1}k_i}}.$$

5. Go to 1 *and repeat steps* **1, 2, 3** *and* **4**. **Do** *$n-1$ new realizations of the random variables $\theta[u]_\nu^{(s)}[h,A]$: $\theta[u]_\nu^{(2)}$, $\theta[u]_\nu^{(3)}$, ..., $\theta[u]_\nu^{(n)}$ and $\theta[d]_\nu^{(s)}[h,A]$: $\theta[d]_\nu^{(2)}$, $\theta[d]_\nu^{(3)}$, ..., $\theta[d]_\nu^{(n)}$.*

6. Calculate

$$\overline{\theta}[u]_\nu[h,A] = \frac{1}{n}\sum_{s=1}^{n}\theta[u]_\nu^{(s)}[h,A], \text{ for } \nu = l-1, l. \tag{3.56}$$

$$\overline{\theta}[d]_\nu[h,A] = \frac{1}{n}\sum_{s=1}^{n}\theta[d]_\nu^{(s)}[h,A], \text{ for } \nu = l-1, l. \tag{3.57}$$

7. Compute *the current approximation of the eigenvalue:*

$$\lambda^{(l-1)} = \frac{\overline{\theta}[u]_{l-1}[h,A]}{\overline{\theta}[d]_{l-1}[h,A]}$$

and

$$\lambda^{(l)} = \frac{\overline{\theta}[u]_{l}[h,A]}{\overline{\theta}[d]_{l}[h,A]}$$

8. If

$$|\lambda^{(l)} - \lambda^{(l-1)}| < \varepsilon \tag{3.58}$$

then stop *iterations* . **Else** *continue iterations using steps* **4, 5, 6** *untill reaching new values of the parameters l and r and check the inequality (3.58).*

The process **continue until** *the inequality (3.58) is fulfilled or* **until** *a very large number of iterations l or very high power r of the resolvent matrix is reached.*

3.6.3 The Inverse Monte Carlo Iterative algorithm (IMCI)

Here an **Inverse Monte Carlo Iterative algorithm (IMCI)** is considered.

This algorithm can be applied when A is a non-singular matrix. The algorithm has a high efficiency when the smallest by modulus eigenvalue of A is much smaller then other eigenvalues. This algorithm can be realized as follows:

1. Calculate the inverse of the matrix A.

2. Starting from the initial vector $f_0 \in R^{m\times 1}$ calculate the sequence of Monte Carlo iterations:

$$f_1 = A^{-1}f_0,\; f_2 = A^{-1}f_1\; \ldots,\; f_i = A^{-1}f_{i-1}, \ldots$$

The vectors $f_i \in R^{m\times 1}$ converge to the eigenvector corresponding to the smallest by modulus eigenvalue of A. In fact, we calculate the functionals

$$\frac{(Af_i,h_i)}{(f_i,h_i)} = \frac{(f_{i-1},h_i)}{(f_i,h_i)}.$$

It is not necessary to calculate A^{-1} because the vectors f_k can be evaluated by solving the following system of equations:

$$Af_1 = f_0$$

$$Af_2 = f_1$$

$$\dots$$

$$Af_i = f_{i-1}.$$

When using Monte Carlo algorithms it is more efficient first to evaluate the inverse matrix using the algorithm proposed in [MAD94] and after that to apply the Monte Carlo iterations.

3.6.4 Numerical tests

In this subsection numerical results obtained by means of both RMC and IMCI algorithms will be presented. The code is written in FORTRAN 77 and C^{++} and is performed on supercomputers Intel-PARAGON and CRAY Y–MP C92A.

Numerical tests are performed for a number of test matrices – general symmetric sparse matrices and band sparse symmetric matrices. The test matrices are produced using a specially created generator of symmetric matrices called *MATGEN*. This generator allows to generate matrices with a given size, given sparsity and fixed largest and smallest eigenvalue (fixed condition number). All other eigenvalues are chosen to be randomly distributed. Using MATGEN-program it is also possible to put a gap of a given size into the spectrum of the matrix. For producing such matrices in MATGEN Jacobi rotations are used such that the angle of rotation and the place of appearing the non-zero entrances are randomly chosen. The test matrices used in our numerical experiments are of size 128, 512, 1000, 1024, and 2000 and have different number of non-zero elements. Some of the most important parameters of the matrices are shown in Tables 4.3. The name of matrices contains the size of the matrix and also a parameter which indicates the sparsity. So, we a able to control the parallel behaviors of the algorithm for different levels of sparsity and to study the dependence between the computational time and the size of the matrices.

Implementation on Intel-PARAGON

The numerical tests are performed on an Intel PARAGON machine. The Intel PARAGON is a particular form of a parallel machine which consists of a set of independent processors, each with its own memory, capable of operating on its own data. Each processor has its own program to execute and processors are linked by communication channels. The PARAGON machine on which our experiments are performed consists of a mesh-grid of $16 \times 4 = 64$ nodes. 8 processors are devoted

Table 3.4: **Test matrices**

Name	Size	Non − zero el. per row	λ_{min}	λ_{max}
tr128.min	128	52	1.0000	64.0000
tr512.min	512	178	1.0000	64.0000
tr1000.2min	1000	39	−1.9000	1.0000
tr1024.2min	1024	56	1.0000	64.0000
tr1024.min	1024	322	1.0000	64.0000
tr2000.2min	2000	56	1.0000	64.0000

to the system management (service and I/O processors) while the 56 remain for applications purposes. An amount of 16 MB of RAM is also available on each node but approximately 8 to 9 are consumed by OSF so that 7 MB are really available for the users. Data needed for processing by PARAGON must be shared between processors by *message passing*. There are a number of libraries available for message passing on the PARAGON system. Here we use the proprietary NX message passing library provided by Intel. It offers the best message passing performance for this system. The Task-to-Task communication latency is about $50\mu s$ while the bandwidth is approximately 92 MB/s under OSF.

In this section numerical tests obtained using RMC algorithm are presented. The code is written in FORTRAN 77 and is performed on Intel PARAGON using p processors. Each processor executes the same program for n/p number of trajectories, i.e. - n/p independent realizations of the random variable. At the end the host processor collects the results of all realizations and computes the average value.

Numerical tests are performed for a large number of test matrices - general symmetric sparse matrices and band sparse symmetric matrices produced using *MATGEN*.

Some information about the computations complexity, speed-up and parallel efficiency of the algorithm is presented in Tables 3.5 – 3.7 as well as on on Figures 3.9 and 3.10.

Tables 3.5 and 3.6 present results for matrices of different size and sparsity when a **small number** of Monte Carlo iterations is needed to receive a good accuracy. Subtables "a" show the dependence between calculated values and the number of random trajectories. Subtables "b" contain an information about the dependence of the computational complexity, speed-up and parallel efficiency on the number of processors p.

Table 3.7 presents results for a matrix of size 1024 with a given sparsity (322 non-zero elements per row) when a **large** number of iterations are needed ($r = 129$ for Table 3.7). Here an information about the computational error, time, speed-up and parallel efficiency is given. The results for the parallel efficiency are not very good,

Table 3.5: **Resolvent Monte Carlo Method (RMC) for m512.52** ($\lambda_{min} = 1.00000$)**.**

a) The solution when the number of trajectories increases.

Number of trajectories	10^3	10^4	10^5	10^6
Calculated λ_{min}	1.0278	0.9958	0.999984	1.000010

b) Implementation on PARAGON (Num. of tr. $n = 10^6$). Calculated $\lambda_{min} = 1.000010$.

Number of nodes	1	2	3	4	5	6	7	8	9	10
Time (s)	70.75	35.97	24.9	18.3	14.78	12.96	11.49	9.63	8.67	7.68
Speedup S	1	1.97	2.84	3.86	4.78	5.46	6.16	7.34	8.16	9.21
Efficiency E	1	0.98	0.95	0.97	0.96	0.91	0.88	0.92	0.91	0.92

Table 3.6: **Resolvent Monte Carlo Method (RMC) for m1000.39** ($\lambda_{min} = -1.9000$)**.**

a) The solution when the number of trajectories increases.

Number of trajectories	10^3	10^4	10^5	10^6
Calculated λ_{min}	-1.9068	-1.9011	-1.90021	-1.900057

b) Implementation on PARAGON (Num. of tr. $n = 10^6$). Calculated $\lambda_{min} = -1.900057$.

Number of nodes	1	2	3	4	5	6	7	8	9	10
Time (s)	26.72	14.05	10.05	7.97	6.75	5.02	4.51	3.75	3.36	3.30
Speedup S	1	1.90	2.66	3.35	3.96	5.32	5.921	7.13	7.95	8.11
Efficiency E	1	0.95	0.89	0.84	0.79	0.89	0.84	0.89	0.88	0.81

Table 3.7: **Resolvent Monte Carlo Method (RMC) for m1024.322 ($\lambda_{min} = 1.0$). Parameters of the problem: r=129, l=5, f - unit vector.**

a) The solution when the number of trajectories increases.

Number of trajectories	10^3	10^4	10^5	10^6
Calculated λ_{min}	0.9268	0.9865	0.9935	0.9959

b) Implementation on PARAGON (Number of trajectories $n = 10^5$).

Number of nodes	λ_{min}	Time (s)	Speed $-$ up	Efficiency
1	0.9935	12.896	1	1
2	0.9935	6.896	1.870	0.935
3	0.9954	4.736	2.722	0.907
4	0.9923	3.648	2.722	0.680
5	0.9946	3.616	3.535	0.707
6	0.9966	3.648	3.566	0.707
7	0.9931	3.552	3.535	0.594
8	0.9935	3.104	3.630	0.505
9	0.9964	3.008	4.154	0.453
10	0.9945	2.880	4.287	0.461

Table 3.8: **Computing time and matrix size**

a) Rezults for matrix 128×128. Number of nonzero elements = 6714.
Exact $\lambda_{max} = 64.0$. Parameters: r=47, l=7.

Number of trajectories	Calculated λ_{max}	Time s
10^3	64.1350	0.256
10^4	63.3300	2.112
10^5	63.1843	21.600
10^6	63.189	208.256

b) Rezults for matrix 1000×1000. Number of nonzero elements = 38748.
Exact $\lambda_{max} = 1.0$. Parameters: r=47, l=7.

Number of trajectories	Calculated λ_{max}	Time s
10^3	0.9981	0.128
10^4	0.9997	1.344
10^5	1.000051	13.184
10^6	1.000033	132.288

c) Rezults for matrix 1024×1024. Number of nonzeroelements = 57538.
Exact $\lambda_{max} = 64.0$. Parameters: r=17, l=5.

Number of trajectories	Calculated λ_{max}	Time s
10^3	64.3462	0.256
10^4	64.1883	1.856
10^5	64.1724	18.176
10^6	64.1699	181.504

d) Rezults for matrix 2000×2000. Number of nonzero elements = 112594.
Exact $\lambda_{max} = 64.0$. Parameters: r=17, l=5.

Number of trajectories	Calculated λ_{max}	Time s
10^3	63.9943	0.192
10^4	64.0050	1.408
10^5	64.0204	13.312
10^6	64.0265	133.248

Table 3.9: **Resolvent Monte Carlo Method (RMC) for m1000.39**
Exact $\lambda_{max} = 1.0$.

l	r	*Calculated* λ_{max}	*Time* s
2	2	1.0921	0.064
2	5	1.0833	0.064
2	10	1.0773	0.064
2	20	1.0729	0.064

l	r	*Calculated* λ_{max}	*Time* s
5	2	1.0658	0.128
5	5	1.0338	0.128
5	10	1.0191	0.128
5	20	1.0133	0.128

l	r	*Calculated* λ_{max}	*Time* s
10	2	1.0599	0.192
10	5	1.0195	0.192
10	10	1.0040	0.192
10	20	0.9983	0.192
10	30	0.9956	0.192

Remarks: 1. The number of trajectories n is 1000.
2. The time is measured during the program implementation on Intel PARAGON (one processor).

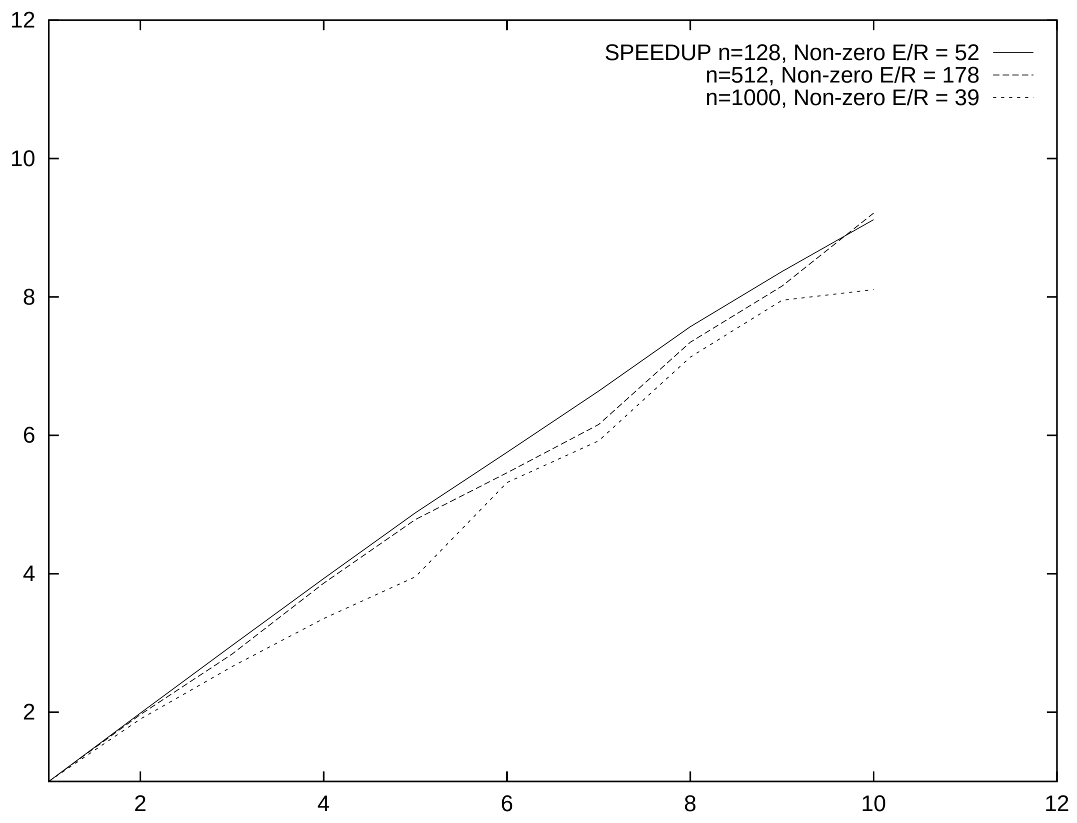

Figure 3.9
Dependence of the Speedup on the number of processors for different matrices (the computational time is relatively large in comparison with communicational time for all matrices; the number of realizations n for all matrices is relatively large).

because the computational time is very small when a large number of processors is used. In fact, the computational time is comparable with the time needed for communications. It means that even for matrices of large size (up to 2000×2000) it is not necessary to use a lot of processors. Our results show that the parallel efficiency increases when the number of random trajectories increases.

Our numerical tests performed for a large number of matrices confirm the independence of the algorithmic complexity from the size of the matrix. The results presented on Table 3.8 show that:

- the computational time is almost independent of the size of the matrix;
- a linear dependence between the computational time and the number of the random trajectories is observed;
- a linear dependence between the computational time and the mean value of the number of non-zero entries of each row of the matrix is realized.

Some numerical results showing the dependence of the accuracy from the parameters r and l are presented on Table 3.9.

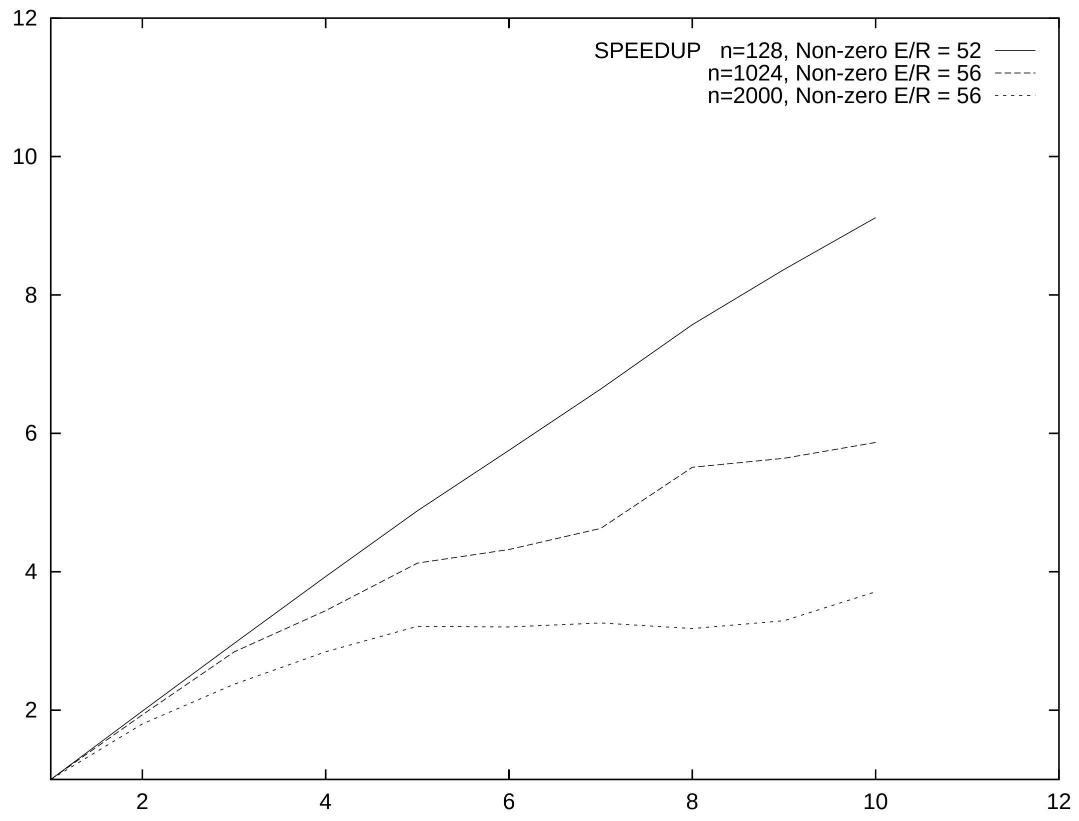

Figure 3.10
Dependence of the Speedup from the number of processors for different matrices (the computational time is relatively small in comparison with communicational time for matrices of size 1024 and 2000 since the number of realizations n is 10 times smaller than in the case of matrix m128.52).

Figures 3.9, 3.10 present some results for speed-up when different numbers of processors of Intel PARAGON machine are used. One can see, that when the number of the random trajectories is large the speed-up is almost linear and it is close to the best value of the speed-up, i.e. to the value of p (Figure 3.9). When the number of processors is small (with respect to the computational complexity) the speed-up is linear. If the number of processors p is large and the computational time is small the speed-up is not so good (see, Figure 3.10). It may happen for tasks with a very small computational complexity that such a large number of processors is not needed. The case of small computational complexity may happen not only because of small number of non-zero elements per row, but also because of small number of realizations n needed for realizing the algorithm. The last case is shown on Figure 3.10.

Implementation on CRAY Y–MP C92A

The results for a matrix of size $m = 512$ are shown in Tables 3.10 - 3.11.

The experimental results show that both IMCI and RMC algorithms give good

Table 3.10: **Inverse Monte Carlo Iterative algorithm (IMCM) for MS512.2** ($\lambda_{min} = 0.2736$)**. (A general symmetric matrix of size 512.)**

a) The number of Markov chains is fixed $n = 80$.

l	Calculated λ_{min}	Error,
2	0.2736	0.0000
3	0.2733	0.0011
4	0.2739	0.0011
5	0.2740	0.0015
10	0.2732	0.0015
50	0.2738	0.0007
100	0.2757	0.0076

b) The number of iterations (number of moves in every Markov chain) l is fixed - $l = 50$.

n	Calculated λ_{min}	Error,	CP − time, s	HWM− memory
20	0.2729	0.0026	5.356	1137378
40	0.2742	0.0022	5.396	1137378
60	0.2748	0.0044	5.468	1137378
80	0.2739	0.0011	5.524	1137378
100	0.2736	0.0000	5.573	1137378
500	0.2737	0.0004	6.666	1137378
1000	0.2739	0.0011	8.032	1137378

Remark: The values for CP-time and HWM-memory are for CRAY Y-MP C92A.

Table 3.11: **The number of iterations (number of moves in every Markov chain) for MS512.2 ($\lambda_{min} = 0.2736$) l is small and fixed - $l = 4$.**

n	Calculated λ_{min}	Error,	CP − time, s	HWM− memory
20	0.2737	0.0004	5.296	1137378
40	0.2749	0.0058	$\star$	1137378
60	0.2754	0.0066	$\star$	1137378
80	0.2739	0.0011	$\star$	1137378
100	0.2736	0.0000	$\star$	1137378
500	0.2737	0.0004	$\star$	1137378
1000	0.2738	0.0007	5.514	1137378

Remarks:
1. The values of CP-time and HWM-memory are for CRAY Y–MP C92A.
2. "$\star$" - no estimated CP-time; the values of CP-time are between 5.296 s and 5.514 s.
3. In comparison with case b), CP-time decreases very slowly for more then 10-times decreasing of the number of moves l.
4. The corresponding NAG-routine for solving the same problem needs CP-time = 5.452 s and HWM-mem = 1 220 676.

results even in the case of small values of the parameters r and n.

An information about the vectorizing of the vectorization of the algorithms is received. This information shows that the studied algorithms are well-vectorized. For matrices A with *well distributed* eigenvalues, the RMC algorithm works well, when the parameter q is *not too large.* When the largest eigenvalue of the resolvent matrix $(I - qA)^{-1}$ is *well isolated* a small number for the power r is needed ($r = 2, \dots, 5$). In this case the number of moves in every Markov chain can also be small ($l = 5, \dots, 10$) and the accuracy of calculations is high. In this case the results are not very sensitive to the value of the parameter q, which controls the speed of convergence of the Monte Carlo iterative procedure. When $r > 35$ the results are sensitive to q. This case occurs place when $\lambda_1((I - qA)^{-1}/\lambda_2((I - qA)^{-1}$ is approximately equal to 1. In such a case it is better to apply the IMCI algorithm.

IMCI algorithm is very efficient when the smallest eigenvalue is *well isolated.* For example the matrix

$$C = \begin{pmatrix} -2.0 & 0.20 & 0.30 & 0.40 & 0.50 & 0.60 & 0.70 \\ 0.20 & -2.0 & 0.25 & 0.35 & 0.40 & 0.45 & 0.50 \\ 0.30 & 0.25 & -2.0 & 0.15 & 0.20 & 0.25 & 0.40 \\ 0.40 & 0.35 & 0.15 & -2.0 & 0.10 & 0.20 & 0.30 \\ 0.50 & 0.40 & 0.20 & 0.10 & -2.0 & 0.08 & 0.15 \\ 0.60 & 0.45 & 0.25 & 0.20 & 0.08 & -2.0 & 0.10 \\ 0.70 & 0.50 & 0.40 & 0.30 & 0.15 & 0.10 & -2.0 \end{pmatrix}$$

the results for the smallest eigenvalue obtained after two iterations by matrix C^{-1} are between $-4.349 * 10^{-2}$ and $-4.385 * 10^{-2}$ (the "exact" value is $-4.3649 * 10^{-2}$).

In this case the accuracy is high for arbitrary by small power i and number of moves l in every Markov chain . A good accuracy can by reached for a number of random trajectories $n \in [20, 80]$. In some cases (for example, for the general symmetric matrix MS512.2 of size 512) the CP-time is less than the corresponding CP-time for the NAG-routine when CRAY Y–MP C92A machine is used.

The two-steps power Monte Carlo algorithm has also been applied. It has a good efficiency when the ratio between the largest eigenvalue and the next eigenvalue is not "too close" to 1 for both matrices A and B.

3.6.5 Concluding remarks

Two different parallel and vectorizable Monte Carlo algorithms - RMC and IMCI for calculating eigenvalues of symmetric matrices have been studied.

The algorithms under consideration are *almost optimal* from statistical point of view, i.e the variance of the randon variable, which is equal to the eigenvalue is *almost optimal.*

The convergence of RMC algorithm depends on the spectrum of the matrix. The systematic error is

$$O\left[\left(\frac{2\lambda_1 + \lambda_m}{2\lambda_1 + \lambda_{m-1}}\right)^r\right], \tag{3.59}$$

where r is the power (or the number of iterations).

When $\lambda_m \approx -2\lambda_1$ the convergence is very good. It is clear from (3.59) that for a positive or negative defined matrices the convergence decreases, so that the best convergence which can be reached is $O[(2/3)^r]$. The studied algorithm has strong requirements about matrices for which it can be applied: the error from the power algorithm used on the resolvent matrix determines the value of the of the parameter r; the error comming from the representation of the resolvent matrix as a series determines the parameter l. The values of r and l are not independent, since they determine the binomial coefficients C^l_{r+l-1} which grow exponentially with l.

The results obtained by RMC for sparce matrices show that:

- the computational time is almost independent from the size of the matrix;
- there is a linear dependence between the computational time and the number of the random trajectories;
- there is a linear dependence between the computational time and the mean value of the number of the non-zero entrances of each row of the matrix;
- the speedup is almost linear when the computational time for every processor is not too small.

For matrices A with "well distributed" eigenvalues, when the parameter q is "not too large" the RMC algorithm works well. When the largest eigenvalue of the resolvent matrix $(I - qA)^{-1}$ is "well isolated" a small number of the power "r" is needed $(r = 2, \dots, 5)$. In this case the number of moves in every Markov chain can be also small $(l = 5, \dots, 10)$ and the accuracy of calculations is high. In this case the results are not very sensitive to the value of the parameter q, which controls the speed of convergence of the Monte Carlo iterative procedure. When $r > 35$ the results are sensitive to q. This case takes place when $\lambda_1((I - qA)^{-1}/\lambda_2((I - qA)^{-1}$ is approximately equal to 1. In this case it is better to apply the Inverse Monte Carlo iterative algorithm.

IMCI algorithm is efficient when the smallest eigenvalue is "well isolated". In this case the accuracy is high for arbitrary small power i and number of moves in every Markov chain l.

Chapter 4

MONTE CARLO ALGORITHMS FOR BOUNDARY-VALUE PROBLEMS (BVP)

4.1 BVP for Elliptic Equations

There are essentially two approaches to numerically solving elliptic equations. The first one is the so-called *grid approach*, while the second one might be called the *grid-free approach*. In this section we consider both approaches.

Let $\Omega \subset I\!\!R^d$ be a bounded domain with a boundary $\partial\Omega$.

The following notations are used:

$x = (x_{(1)}, x_{(2)}, \ldots, x_{(d)})$ is a point in $I\!\!R^d$;

$D^\alpha = D_1^{\alpha_1} D_2^{\alpha_2} \ldots D_d^{\alpha_d}$ is an $|\alpha| = \alpha_1 + \alpha_2 + \ldots + \alpha_d$ derivative, where $D_i = \partial/\partial x_{(i)},\ i = 1, \ldots, d$ and $C^k(\bar{\Omega})$ is a space of functions $u(x)$ continuous on $\bar{\Omega}$ such that $D^\alpha u$ exists in Ω and admits a continuous extension on $\bar{\Omega}$ for every $\alpha : |\alpha| \leq k$.

We consider the linear boundary value problem

$$Lu \equiv \sum_{|\alpha| \leq 2m} a_\alpha(x) D^\alpha u(x) = -f(x), \qquad x \in \Omega \tag{4.1}$$

$$u(x) = \varphi(x), \qquad x \in \partial\Omega\,, \tag{4.2}$$

where L is an arbitrary linear elliptic operator in $I\!\!R^d$ of order $2m$, $a_\alpha(x) \in C^\infty(I\!\!R^d)$ and the function $f(x)$ belongs to the Banach space $\mathbf{X}(\Omega)$.

We use the following definition of *ellipticity*:

Definition 4.1.1 *The equation*

$$\sum_{|\alpha| \leq 2m} a_\alpha(x) D^\alpha u(x) = -f(x)$$

is called elliptic in a domain Ω *if*

$$\sum_{|\alpha| \leq 2m} a_\alpha(x) \xi_{\alpha_1} \xi_{\alpha_2} \ldots \xi_{\alpha_d} \neq 0 \text{ when } |\xi| \neq 0$$

holds for every point $x \in \Omega$. The corresponding operator $\sum_{|\alpha|\leq 2m} a_\alpha(x)D^\alpha$ is called elliptic in Ω.

Assume that $f(x)$, $\varphi(x)$, and the boundary $\partial\Omega$ satisfy conditions ensuring that the solution of the problem (4.1), (4.2) exists and is unique [Mi83], [Mi55].

We shall study Monte Carlo algorithms for calculating linear functionals of the solution of the problem (4.1), (4.2)

$$J(u) = (h, u) = \int_\Omega u(x)h(x)dx, \tag{4.3}$$

where $h \in \mathbf{X}^*(\Omega)$ ($\mathbf{X}^*(\Omega)$ is the dual functional space to $\mathbf{X}(\Omega)$).

For many applications $\mathbf{X} = L_1$ and thus $\mathbf{X}^* = L^\infty$, or $\mathbf{X} = L_2$, $\mathbf{X}^* = L_2$.

There are two approaches for calculating (4.3). The first approach uses a discretization of the problem (4.1, 4.2) on a mesh and solves the resulting linear algebraic system, which approximates the original problem (4.1, 4.2). This is the so-called *grid Monte Carlo algorithm*, or *grid walk algorithm.* The second approach (grid-free approach) uses an integral representation for the problem (4.1, 4.2).

4.2 Grid Monte Carlo Algorithm

Consider a regular mesh (lattice) with step-size h in $I\!R^d$. Let Ω_h be the set of all inner mesh points ($\gamma \in \Omega_h$ if and only if $\gamma \in \Omega$); $\partial\Omega_h$ be the set of all *boundary* mesh points ($\gamma \in \partial\Omega_h$ if there exists a neighboring mesh point γ^* which does not belong to $I\!R^d \setminus \bar{\Omega}$) and u_h be a function defined on a set of mesh points (a mesh function).

The differential operator L at the mesh point $x_i \in \Omega_h$ is approximated by a difference operator L_h as follows:

$$(L_h u_h)_i = \sum_{x_j \in P_k(x_i)} a_h(x_i, x_j)u_h(x_j) \ , \tag{4.4}$$

where $a_h(x_i, x_j)$ are coefficients; and $P_k(x_i)$ is a set of mesh points with center in $x_i \in \Omega_h$ called *scheme.*

Since L is a linear differential operator, after the discretisation of (4.4), the following system of linear equation arises:

$$Au = b \ ,$$

where $b = (b_1, \dots, b_m)^T \in I\!R^{m\times 1}$ is an m-dimensional vector and $A \in I\!R^{m\times m}$ is an $m \times m$-dimensional matrix.

4.3 Grid-Free Monte Carlo Algorithms

Consider two approaches for constructing grid-free Monte Carlo algorithms. The first one consists in obtaining a *global integral representation* both on the boundary and on the domain.

Let us consider the following linear elliptic boundary value problem :

$$\Delta u(x) - c^2 u(x) = -\varphi(x),\ x \in \Omega \tag{4.5}$$

$$u(x) = \psi(x),\ x \in \partial\Omega, \tag{4.6}$$

where Δ is the Laplacian and the functions $\varphi(x)$, $\psi(x)$ and the boundary satisfy all conditions, which provide the existence of a unique solution of the problem (4.5), (4.6).

From the theory of fundamental solutions it follows that the solution of the problem (4.5), (4.6) can be represented as the integral equation (3.6) (see, Section 3.1) [Bi82], [EM82], where

$$k(x,y) = \begin{cases} \dfrac{cd(x)}{\sinh[cd(x)]}\delta(y-x) & \text{, when } x \in \Omega \setminus \partial\Omega \\ 0 & \text{, when } x \in \partial\Omega \end{cases}$$

$$f(x) = \begin{cases} \dfrac{1}{4\pi}\displaystyle\int \dfrac{\sinh((d(x)-|y-x|)c}{|y-x|\sinh[cd(x)]}\varphi(y)dy & \text{, when } x \in \Omega \setminus \partial\Omega \\ \psi(x) & \text{, when } x \in \partial\Omega \end{cases}$$

where $d = d(x)$ is the distance from x to the boundary $\partial\Omega$.

It will it be necessary to calculate the functional (4.3), where u is the solution of the problem (4.5), (4.6) and h is a given function.

This representation permits the use of a random variable for calculating the functional (4.3). Unfortunately, this approach is not successful when one deals with more complicated operators for which it is impossible to find an integral representation.

The second grid-free Monte Carlo approach is based on use of local integral representation of the solution. In this case the Green's function for standard domains, lying inside the domain Ω (for example - ball, sphere, ellipsoid) is used.

Consider the elliptic boundary value problem:

$$Mu = -\phi(x),\ x \in \Omega,\ \Omega \subset I\!R^3 \tag{4.7}$$

$$u = \psi(x),\ x \in \partial\Omega, \tag{4.8}$$

where

$$M = \sum_{i=1}^{3}\left(\frac{\partial^2}{\partial x_{(i)}^2} + b_i(x)\frac{\partial}{\partial x_{(i)}}\right) + c(x).$$

Define the class of domains $\mathbf{A}^{(k,\lambda)}$.

Definition 4.3.1 *The domain Ω belongs to the class $\mathbf{A}^{(k,\lambda)}$ if for any point $x \in \partial\Omega$ (from the boundary $\partial\Omega$) the boundary $\partial\Omega$ can be presented as a function $z_3 = \sigma(z_1, z_2)$ in the neighborhood of x for which $\sigma^{(k)}(z_1, z_2) \in \mathbf{C}^{(0,\lambda)}$, i.e.*

$$|\sigma^{(k)}(y) - \sigma^{(k)}(y')| \le N|y - y'|^{\lambda}$$

where the vectors $y \equiv (z_1, z_2)$ *and* $y' \equiv (z'_1, z'_2)$ *are* 2*-dimensional vectors,* N *is constant and* $\lambda \in (0,1]$.

If in the closed domain $\bar{\Omega} \in \mathbf{A}^{(1,\lambda)}$ the coefficients of the operator M satisfy the conditions $b_j,\ c(x) \in \mathbf{C}^{(0,\lambda)}(\bar{\Omega}),\ c(x) \leq 0$ and $\phi \in \mathbf{C}^{(0,\lambda)}(\Omega) \cap \mathbf{C}(\bar{\Omega}),\ \psi \in \mathbf{C}(\partial\Omega)$, the problem (4.7), (4.8) has an unique solution $u(x)$ in $\mathbf{C}^2(\Omega) \cap \mathbf{C}(\bar{\Omega})$.

The conditions for uniqueness of a solution can be found in ([Mi83], p. 179, [Bi82], p. 79).

We obtain an integral representation of the solution $u(x)$. This representation allows for the use of the random variable for calculating the functional (4.3).

4.3.1 Local integral representation

We have to estimate the functional (4.3) by means of grid-free Monte Carlo approach. This approach is based on the use of a local integral representation of the solution $u(x)$ in the problem (4.7), (4.8). The representation uses the Green's function approach for standard domains, lying inside the domain Ω.

The initial step in studying the grid-free Monte Carlo approach is to obtain an integral representation of the solution in the form:

$$u(x) = \int_{B(x)} k(x,y)u(y)dy + f(x) \tag{4.9}$$

assuming that a representation exists.

The iterative Monte Carlo process converges when the condition

$$\| K(u) \|_{L_1} = \max_{x\in\Omega} \int_{\Omega} | k(x,y) | \, dy \leq q < 1 \tag{4.10}$$

holds.

For the existence of the integral representation, (4.9) might be obtained using the result of C. Miranda [Mi55] taking into consideration that the domain $B(x)$ belongs to the space $\mathbf{A}^{(1,\lambda)}$ and that the operator M is of elliptic type.

We seek a representation of the integral kernel $k(x,y)$ using Levy's function and the adjoint operator M^* for the initial differential operator M. The following Lemma holds:

Lemma 4.3.1 *Let the components of the vector-function* $\mathbf{b}(x)$ *satisfy the conditions* $b_j(x) \in C^{(1)}(\Omega),\ (j = 1,2,3)$ *and* $div\, \mathbf{b}(x) = 0$.

Then the adjoint operator M^* *applied on functions* $v(x)$, *where* $v \in C^2(\Omega)$ *and*

$$\frac{\partial v(x)}{\partial x_{(i)}} = v(x) = 0 \ \textit{for any} \ \ x \in \partial\Omega,\ i = 1,2,3$$

has the following form:

$$M^* = \sum_{i=1}^{3} \left(\frac{\partial^2}{\partial x_{(i)}^2} - b_i(x) \frac{\partial}{\partial x_{(i)}} \right) + c(x).$$

Proof. Let us show that M^* is an adjoint operator to M, i.e. we have to prove that

$$\int_\Omega v(x) M u(x) dx = \int_\Omega u(x) M^* v(x) dx. \tag{4.11}$$

To prove (4.11) we use the Green's formulas:

$$\int_\Omega u(x) \sum_{i=1}^{3} \frac{\partial v(x)}{\partial x_{(i)}} dx = - \int_\Omega v(x) \sum_{i=1}^{3} \frac{\partial u(x)}{\partial x_{(i)}} dx + \int_{\partial\Omega} \sum_{i=1}^{3} u(x) v(x) n_i d_x S \tag{4.12}$$

and

$$\int_\Omega u(x) \Delta v(x) dx = - \int_\Omega \operatorname{grad} u(x) \operatorname{grad} v(x) dx + \int_{\partial\Omega} u(x) \sum_{i=1}^{3} n_i \frac{\partial v(x)}{\partial x_{(i)}} d_x S,$$

where

$$\Delta = \sum_{i=1}^{3} \frac{\partial^2}{\partial x_{(i)}^2}, \;\; \operatorname{div} \mathbf{b}(x) = \sum_{i=1}^{3} \frac{\partial b_i(x)}{\partial x_{(i)}}, \;\; \operatorname{grad} u(x) \equiv \left(\frac{\partial u(x)}{\partial x_{(1)}}, \frac{\partial u(x)}{\partial x_{(2)}}, \frac{\partial u(x)}{\partial x_{(3)}} \right)$$

and $\mathbf{n} \equiv (n_1, n_2, n_3)$ is the exterior normal for the boundary $\partial\Omega$.
Taking into consideration that

$$\operatorname{div} \mathbf{b}(x) = 0 \;\text{ and }\; \frac{\partial v(x)}{\partial x_{(i)}} = v(x) = 0 \;\; \textit{for any} \;\; x \in \partial\Omega, \; i = 1, 2, 3,$$

we have

$$\begin{aligned}
\int_\Omega v(x) M u(x) dx &= \int_\Omega v(x) \left(\Delta u(x) + \mathbf{b}(x) \operatorname{grad} u(x) + c(x) u(x) \right) dx \\
&= - \int_\Omega \operatorname{grad} v(x) \operatorname{grad} u(x) dx + \int_{\partial\Omega} v(x) \sum_{i=1}^{3} n_i \frac{\partial u(x)}{\partial x_{(i)}} d_x S \\
&\quad + \int_\Omega v(x) \mathbf{b}(x) \operatorname{grad} u(x) dx + \int_\Omega v(x) c(x) u(x) dx \\
&= - \int_\Omega \operatorname{grad} v(x) \operatorname{grad} u(x) dx + \int_\Omega v(x) \sum_{i=1}^{3} b_i(x) \frac{\partial u(x)}{\partial x_{(i)}} dx + \int_\Omega v(x) c(x) u(x) dx.
\end{aligned}$$

On the other hand

$$\begin{aligned}
\int_\Omega u(x)M^*v(x)dx &= \int_\Omega u(x)\left[\Delta v(x) - \mathbf{b}(x)\mathrm{grad}\, v(x) + c(x)v(x)\right]dx \\
&= -\int_\Omega \mathrm{grad}\, u(x)\mathrm{grad}\, v(x)dx + \int_{\partial\Omega} u(x)\sum_{i=1}^{3} n_i \frac{\partial v(x)}{\partial x_{(i)}} d_x S \\
&\quad - \int_\Omega u(x)\mathbf{b}(x)\mathrm{grad}\, v(x)dx + \int_\Omega u(x)c(x)v(x)dx \\
&= -\int_\Omega \mathrm{grad}\, u(x)\mathrm{grad}\, v(x)dx - \int_\Omega u(x)\sum_{i=1}^{3} b_i(x)\frac{\partial v(x)}{\partial x_{(i)}}dx + \int_\Omega u(x)c(x)v(x)dx \\
&= -\int_\Omega \mathrm{grad}\, u(x)\mathrm{grad}\, v(x)dx + \int_\Omega v(x)\sum_{i=1}^{3}\frac{\partial(u(x)b_i(x))}{\partial x_{(i)}}dx \\
&\quad - \int_{\partial\Omega}\sum_{i=1}^{3} n_i b_i(x)u(x)v(x)dx + \int_\Omega v(x)c(x)u(x)dx \\
&= -\int_\Omega \mathrm{grad}\, u(x)\mathrm{grad}\, v(x)dx + \int_\Omega v(x)\sum_{i=1}^{3} b_i(x)\frac{\partial u(x)}{\partial x_{(i)}}dx + \int_\Omega v(x)c(x)u(x)dx.
\end{aligned}$$

From the last result there follows the proof of the lemma. ♢

The Levy's function for the problem (4.7), (4.8) is

$$L_p(y,x) = \mu_p(R)\int_r^R (1/r - 1/\rho)p(\rho)d\rho, \quad r \le R, \tag{4.13}$$

where the following notations are used:

$p(\rho)$ is a density function;

$$r = \mid x - y \mid = \left(\sum_{i=1}^{3}(x_{(i)} - y_{(i)})^2\right)^{1/2};$$

$$\mu_p(R) = [4\pi q_p(R)]^{-1};$$

$$q_p(R) = \int_0^R p(\rho)d\rho.$$

It is clear that Levy's function $L_p(y,x)$, and the parameters $q_p(R)$ and $\mu_p(R)$ depend on the choice of the density function $p(\rho)$. In fact, the equality (4.13) defines a family of functions.

We seek a choice of $p(\rho)$ which leads to a representation of type (4.9). Moreover, the kernel of the integral transform should be a transition density function, i.e. $k(x,y) \ge 0$.

From an algorithmic point of view the domain $B(x)$ must be chosen in such a way that the coordinates of the boundary points $y \in \partial B(x)$ could be easily calculated.

Denote by $B(x)$ the ball:

$$B(x) = B_R(x) = \{y : r = \mid y - x \mid \le R(x)\} \tag{4.14}$$

where $R(x)$ is the radius of the ball.

For Levy's function $L_p(y,x)$ the following representation holds (see, [Mi55]):

$$u(x) = \int_{B(x)} \Big(u(y) M_y^* L_p(y,x) + L_p(y,x)\phi(y) \Big)\, dy +$$

$$+ \int_{\partial B(x)} \sum_{i=1}^{3} n_i \left[\left(\frac{L_p(y,x)\partial u(y)}{\partial y_{(i)}} - \frac{u(y)\partial L_p(y,x)}{\partial y_{(i)}} \right) - b_i(y)u(y)L_p(y,x) \right] d_y S \,, \quad (4.15)$$

where $\mathbf{n} \equiv (n_1, n_2, n_3)$ is the exterior normal to the boundary $\partial T(x)$.

Formula (4.15) holds for any domain $T(x) \in \mathbf{A}^{(1,\lambda)}$ contained in Ω.

Obviously, $B(x) \in \mathbf{A}^{(1,\lambda)}$ and therefore for every ball lying inside the domain Ω the representation (4.15) holds.

Now we express the solution $u(x)$ by Green's function $G(x,y)$. It is known, that Green's function is a solution of the problem:

$$M_y^* G(x,y) = -\delta(x-y), \;\; y \in \Omega \setminus \partial\Omega \setminus \{x\},$$

$$G(x,y) = 0, \quad y \in \partial\Omega, \;\; x \in \Omega \setminus \partial\Omega$$

Green's function is Levy's function, $L_p(y,x)$, for which (4.7), (4.8) hold.

Under the condition $L_p(y,x) = G(x,y)$ from (4.15) it is possible to get the integral representation:

$$u(x) = \int_{B(x)} G(x,y) f(y) dy - \int_{\partial B(x)} \sum_{i=1}^{3} n_i \frac{\partial G(x,y)}{\partial y_{(i)}} u(y) d_y S. \quad (4.16)$$

The representation (4.16) is the basis for the Monte Carlo algorithm.

For achieving this aim it is necessary to have a non-negative integral kernel. Next we show that it is possible to construct the Levy's function choosing the density $p(\rho)$ such that $M_y^* L_p(y,x)$ is non-negative in $B(x)$ and such that $L_p(y,x)$ and its derivatives vanish on $\partial B(x)$, i.e.

$$L_p(y,x) = \partial L_p(y,x)/\partial y_i = 0 \;\; \text{for} \;\; y \in \partial B(x), \;\; i = 1,2,3.$$

Lemma 4.3.2 *The conditions*

$$M_y^* L_p(y,x) \geq 0 \;\; \textit{for any} \;\; y \in B(x)$$

and

$$L_p(y,x) = \partial L_p(y,x)/\partial y_{(i)} = 0, \;\; \textit{for any} \;\; y \in \partial B(x), \;\; i = 1,2,3$$

are satisfied for

$$p(r) = e^{-kr},$$

where

$$k \geq \max_{x\in\Omega} \mid \mathbf{b}(x) \mid + R \max_{x\in\Omega} \mid c(x) \mid \tag{4.17}$$

and R is the radius of the maximal ball $B(x) \subset \bar{\Omega}$.

Proof. The condition

$$L_p(y,x) = 0, \quad \text{for any} \quad y \in \partial B(x)$$

obviously holds. It follows from (4.13), (4.14), since in case $y \in \partial B(x)$, then $r = R$ and $L_p(y,x) = 0$.

The condition

$$\partial L_p(y,x)/\partial y_{(i)} = 0, \quad \text{for any} \quad y \in \partial B(x), \quad i = 1,2,3$$

can be checked immediately. Indeed,

$$\frac{\partial L_p(y,x)}{\partial y_{(i)}} = \frac{\partial L_p}{\partial r}\frac{\partial r}{\partial y_{(i)}} = \mu_p(R)\frac{\partial}{\partial r}\left(\int_r^R (1/r - 1/\rho)p(\rho)d\rho\right)\frac{\partial r}{\partial y_{(i)}}$$

$$= \mu_p(R)\frac{\partial}{\partial r}\left(\frac{1}{r}\int_r^R p(\rho)d\rho - \int_r^R \frac{1}{\rho}p(\rho)d\rho\right)\frac{\partial r}{\partial y_{(i)}}$$

$$= \mu_p(R)\left[-\frac{1}{r^2}\int_r^R p(\rho)d\rho + \frac{1}{r}\left(-p(r)\right) - \left(-\frac{1}{r}p(r)\right)\right]\frac{\partial r}{\partial y_{(i)}}$$

$$= \mu_p(R)\left(-\frac{1}{r^2}\int_r^R p(\rho)d\rho\right)\frac{\partial r}{\partial y_{(i)}}.$$

Taking into consideration that

$$\frac{\partial r}{\partial y_{(i)}} = \frac{-(x_{(i)} - y_{(i)})}{r}$$

one can get:

$$\frac{\partial L_p(y,x)}{\partial y_{(i)}} = \mu_p(R)\frac{(x_{(i)} - y_{(i)})}{r^3}\int_r^R p(\rho)d\rho. \tag{4.18}$$

The last expression vanishes when $r = R$, i.e. for every boundary point $y \in \partial B(x)$. Thus we obtain

$$\partial L_p(y,x)/\partial y_{(i)} = 0, \quad \text{for any} \quad y \in \partial B(x), \quad i = 1,2,3.$$

Now calculate $M_y^* L_p(y,x)$. The operator M_y^* has the following form:

$$M_y^* = \sum_{i=1}^{3}\left(\frac{\partial^2}{\partial y_{(i)}^2}\right) - \sum_{i=1}^{3}\left(b_i(y)\frac{\partial}{\partial y_{(i)}}\right) + c(y)$$

and $M_y^* L_p(y,x)$ has the form:

$$M_y^* L_p(y,x) = \sum_{i=1}^{3} \left(\frac{\partial^2 L_p(y,x)}{\partial y_{(i)}^2} \right) - \sum_{i=1}^{3} \left(b_i(y) \frac{\partial L_p(y,x)}{\partial y_{(i)}} \right) + c(y) L_p(y,x). \qquad (4.19)$$

The second term is calculated using (4.18), i.e.

$$\sum_{i=1}^{3} b_i(y) \frac{\partial L_p(y,x)}{\partial y_{(i)}} = \mu_p(R) \sum_{i=1}^{3} b_i(y) \frac{(x_{(i)} - y_{(i)})}{r^3} \int_r^R p(\rho) d\rho. \qquad (4.20)$$

Calculate the first term in (4.19). That can be done easily when we use spherical coordinates:

$$y_{(1)} - x_{(1)} = r \sin\theta \cos\varphi, \quad y_{(2)} - x_{(2)} = r \sin\theta \sin\varphi, \quad y_{(3)} - x_{(3)} = r \cos\theta,$$

where $0 < r < R(x)$, $\theta \in [0, \pi)$ and $\varphi \in [0, 2\pi)$.
Thus the Laplacian

$$\Delta_y = \sum_{i=1}^{3} \left(\frac{\partial^2}{\partial y_{(i)}^2} \right)$$

written in spherical coordinates has the following form ([TS77], p. 282):

$$\Delta_{r,\theta,\varphi} = \frac{1}{r^2} \frac{\partial}{\partial r} \left(r^2 \frac{\partial}{\partial r} \right) + \frac{1}{r^2 \sin\theta} \frac{\partial}{\partial \theta} \left(\sin\theta \frac{\partial}{\partial \theta} \right) + \frac{1}{r \sin^2\theta} \frac{\partial^2}{\partial \varphi^2}.$$

The Levy's function in spherical coordinates depends on the radius r, (see, (4.13)). Thus,

$$\Delta_y L_p(y,x) = \Delta_{r,\theta,\varphi} L_p(r) = \frac{1}{r^2} \frac{\partial}{\partial r} \left(r^2 \frac{\partial L_p(r)}{\partial r} \right)$$

$$= \mu_p(R) \frac{1}{r^2} \frac{\partial}{\partial r} r^2 \frac{\partial}{\partial r} \left(\int_r^R (1/r - 1/\rho) p(\rho) d\rho \right) = \mu_p(R) \frac{1}{r^2} \frac{\partial}{\partial r} \left(r^2 \left(-\frac{1}{r^2} \right) \int_r^R p(\rho) d\rho \right)$$

$$= \mu_p(R) \left(-\frac{1}{r^2} \right) \frac{\partial}{\partial r} \int_r^R p(\rho) d\rho = \mu_p(R) \frac{p(r)}{r^2}.$$

Taking into consideration (4.19), (4.20) we obtain:

$$M_y^* L_p(y,x) = \mu_p(R) \frac{p(r)}{r^2} - \mu_p(R) c(y) \int_r^R \frac{p(\rho)}{\rho} d\rho$$

$$+ \frac{\mu_p(R)}{r^2} \left[c(y) r + \sum_{i=1}^{3} b_i(y) \frac{y_{(i)} - x_{(i)}}{r} \right] \int_r^R p(\rho) d\rho.$$

Next we prove that $M_y^* L_p(y,x)$ is non-negative for every point of the ball $B(x)$. Write $M_y^* L_p(y,x)$ in the following form:

$$M_y^* L_p(y,x) = \frac{\mu_p(R)}{r^2}\Gamma_p(y,x),$$

where

$$\Gamma_p(y,x) = p(r) + c(y)r\left(\int_r^R p(\rho)d\rho - \int_r^R \frac{p(\rho)r}{\rho}d\rho\right)$$
$$+\sum_{i=1}^{3} b_i(y)\frac{y_{(i)} - x_{(i)}}{r}\int_r^R p(\rho)d\rho.$$

It is necessary to show that for all $y \in B(x)$ the function $\Gamma_p(y,x)$ is non-negative. From the condition $c(y) \leq 0$ it follows that

$$\Gamma_p(y,x) = p(r) - \left|c(y)r\left(\int_r^R p(\rho)d\rho - \int_r^R \frac{p(\rho)r}{\rho}d\rho\right)\right|$$
$$+\sum_{i=1}^{3} b_i(y)\frac{y_{(i)} - x_{(i)}}{r}\int_r^R p(\rho)d\rho \geq 0. \tag{4.21}$$

So, it is necessary to prove (4.21). For

$$p(r) = e^{-kr}$$

we have

$$p(r) \geq e^{-kr} - e^{-kR} = k\int_r^R p(\rho)d\rho.$$

Choosing

$$k \geq \max_{x\in\Omega} \mid \mathbf{b}(x) \mid + R\max_{x\in\Omega} \mid c(x) \mid$$

one can obtain

$$p(r) \geq \left(\max_{x\in\Omega} \mid \mathbf{b}(x) \mid + R\max_{x\in\Omega} \mid c(x) \mid\right)\int_r^R p(\rho)d\rho$$
$$\geq \mid c(y) \mid r\int_r^R p(\rho)d\rho + \mid \mathbf{b}(y) \mid \int_r^R p(\rho)d\rho$$
$$\geq \mid c(y) \mid r\left(\int_r^R p(\rho)d\rho - \int_r^R \frac{p(\rho)r}{\rho}d\rho\right) + \left|\sum_{i=1}^{3} b_i(y)\frac{y_{(i)} - x_{(i)}}{r}\right|\int_r^R p(\rho)d\rho. \tag{4.22}$$

Now (4.21) follows from (4.22). $\diamondsuit$

It follows that the representation (4.9) can be written in the form:

$$u(x) = \int_{B(x)} M_y^* L_p(y,x)u(y)dy + \int_{B(x)} L_p(y,x)\phi(y)dy. \tag{4.23}$$

The last representation enables the construction of an unbiased estimate for the solution of our problem.

4.3.2 Monte Carlo algorithms

Simple numerical examples for performing grid and grid-free Monte Carlo algorithms will now be considered.

Let the operator L in the equation (4.1) be the Laplacian :

$$L = \Delta.$$

Using a regular discretisation with a step–size h equation (4.1) is approximated by the following difference equation

$$\Delta_h^{(d)} u = -f_h \tag{4.24}$$

or solved for the i-th point $i = (i_1, \dots, i_d)$

$$u_i = L_h u + \frac{h^2}{2d} f_i,$$

where $\Delta_h^{(d)}$ is the Laplace difference operator, and L_h is an averaging operator. For example, the operator L_h in $I\!R^2$ is

$$L_h u = \frac{1}{4}[u_{i-1,j} + u_{i+1,j} + u_{i,j-1} + u_{i,j+1}] = \frac{1}{4}\Lambda_1(i,j)$$

and then (4.24) becomes

$$u_{ij} = \frac{1}{4}\Lambda_1(i,j) + \frac{h^2}{4} f_{i,j}. \tag{4.25}$$

The matrix form of equations (4.25) has only $2d$ non-zero elements in every row and they all are equal to $\frac{1}{2d}$.

The grid Monte Carlo algorithm for solving (4.25) consists in simulating a Markov chain with initial density p_0 which is *tolerant* to the vector h (see, Definition 2.4.1 in Section 2.4.4) from the integral (4.3), and the probability $p_{\alpha\beta}$ for the transition from the point α to the next point β is equal to $\frac{1}{2d}$ if the point is inside of the domain, that is $((x_{(1)})_\alpha, (x_{(2)})_\beta) \in \Omega_h$, and $p_{\alpha\beta} = 0$ for boundary points (the boundary is an absorbing barrier for the process). Then the random variable whose mathematical expectation coincides with the solution of the problem is:

$$\theta = \frac{h^2}{2d} \sum_{i=1}^{i^*-1} f_i + \varphi_{i^*},$$

where f_i are values of the function f in the points of the Markov chain, and i^* is the point where Markov chain reaches the boundary $\partial\Omega_h$

The well known grid algorithm (see, for example, [Sh64] can be described in pseudo-code notation:

```
**Start** at the grid point $(x_{10}, x_{20})$, i.e. $(x_{(1)}, x_{(2)}) := (x_{10}, x_{20})$
**While** $(x_{(1)}, x_{(2)})$ is not at the boundary
```

Move to a neighboring point $(x'_{(1)}, x'_{(2)}) \in \{(x_{(1)} - h, x_{(2)}),$
$(x_{(1)} + h, x_{(2)}), (x_{(1)}, x_{(2)} - h), (x_{(1)}, x_{(2)} + h)\}$ (i.e.$(x_{(1)}, x_{(2)}) := (x'_{(1)}, x'_{(2)})$)
such that each neighboring is selected with
the same probability $p = 1/4$

Let $(x^*_{(1)}, x^*_{(2)})$ be the final point at the boundary. Then, the searched random variable is:

$$\theta := u(x^*_{(1)}, x^*_{(2)}).$$

In order to compute $E\theta$, we start n Markov processes of the above kind, delivering n realizations $\theta_1, \ldots, \theta_n$ of the random variable θ and approximate the solution by their mean as described above. The behavior of the algorithm is depicted in Figure 4.1.

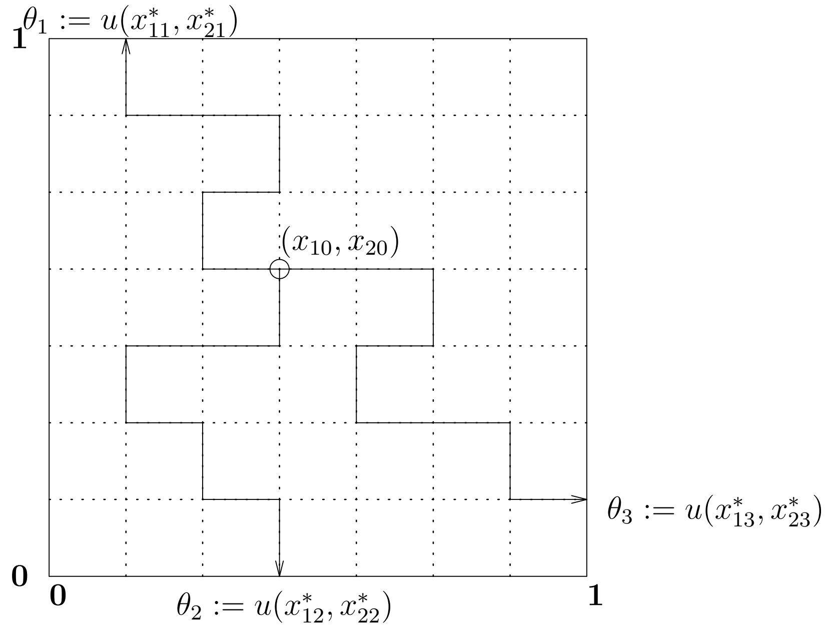
$\theta_1 := u(x^*_{11}, x^*_{21})$
1
(x_{10}, x_{20})
$\theta_3 := u(x^*_{13}, x^*_{23})$
0
0
$\theta_2 := u(x^*_{12}, x^*_{22})$
1

Figure 4.1: **Grid Monte Carlo algorithm (n trajectories from the initial point (x_{10}, x_{20}) to the boundary are constructed and the mean value of the encountered boundary values is computed)**

Now consider the *grid-free Monte Carlo algorithm* based on the local integral representation of the problem.

First, let us describe the selection algorithm in general case. Suppose that $v_1(x)$ and $v_2(x)$ are given functions, $0 \le v_1(x) \le v_2(x)$ and

$$\int_\Omega v_1(x)dx = V_1 < \infty, \quad \int_\Omega v_2(x)dx = V_2 < \infty,$$

where $\Omega \subset I\!R^3$.

Consider an algorithm for simulation of the random variable with density function $v_2(x)/V_2$ and simulate other random variable with the density function $v_1(x)/V_1$. It is necessary to give a realization ξ of the random variable with density $v_2(x)/V_2$ and an independent realization γ of the random variable uniformly distributed in $(0,1)$, as well as to check the inequality $\gamma v_2(x) \leq v_1(x)$. If the last inequality holds, ξ is the needed realization. Otherwise, the process have to be repeated. The efficiency of the selection algorithm is measured by $E = V_1/V_2$.

A local integral representation (4.23) for the boundary value problem (4.7), (4.8) is obtained. Comparing (4.9) with (4.23) one can get

$$k(x,y) = \begin{cases} M_y^* L_p(y,x), & \text{when } x \in \Omega \setminus \partial\Omega, \\ 0, & \text{when } x \in \partial\Omega, \end{cases}$$

and

$$f(x) = \begin{cases} \int_{B(x)} L_p(y,x)\phi(y)dy & \text{when } x \in \Omega \setminus \partial\Omega, \\ \psi(x), & \text{when } x \in \partial\Omega. \end{cases}$$

The Monte Carlo procedure for solving this problem can be defined as a *ball process.* To ensure the convergence of the process, we introduce the ε-strip of the boundary, i.e.

$$\partial\Omega_\varepsilon = \{x \in \Omega : B(x) = B_\varepsilon(x)\}, \text{ where } B_\varepsilon(x) = \{y : r = \mid y - x \mid \leq \varepsilon\}.$$

Consider a transition density function

$$p(x,y) = k(x,y) = M_y^* L_p(y,x) \geq 0. \tag{4.26}$$

This transition density function defines a Markov chain $\xi_1, \xi_2, \ldots, \xi_i$ such that every point $\xi_j,\ j = 1, \ldots, i-1$ is chosen on the maximal ball $B(x_{j-1})$, lying in Ω in accordance with the density (4.26). The Markov chain stops when it reaches $\partial\Omega_\varepsilon$. So, $\xi_i \in \partial\Omega_\varepsilon$.

Let us consider the random variable

$$\theta[\xi_0] = \sum_{j=0}^{i} Q_j \int_{B(\xi_j)} L_p(y,\xi_j) f(y) dy + \varphi(\xi_i),$$

where

$$Q_0 = 1 \tag{4.27}$$

$$Q_j = Q_{j-1} M_y^* L_p(\xi_j, \xi_{j-1})/p(\xi_{j-1}, \xi_j), j = 1, 2, \ldots, i, \tag{4.28}$$

$\varphi(\xi_i)$ is the value of the boundary function at the last point of the Markov chain ξ_i.

It is easy to see that the solution of the problem at the point ξ_0 can be presented as

$$u(\xi_0) = E\theta[\xi_0]. \tag{4.29}$$

To ensure the convergence of the process we consider the next estimate:

$$\int \int k(x,y)k(y,z)dydz < \int \delta(y-z) \int \delta(z-y)dzdy$$

$$= \int \delta(y-x)dy < 1 - \frac{\varepsilon^2}{4R_m^2},$$

where $k(x,y) \geq 0$ is defined by (4.26) and R_m is the supremum of all radii of the spheres lying in Ω.

The above estimate ensures the convergence of the Neumann series and, therefore of the process (4.28) as well.

Obviously, all non-zero values of Q_j are equal to 1 and the problem consists in simulating a Markov chain with a transition density function $p(x,y)$ in the form (4.26). Thus, the problem of calculating $u(\xi_0)$ is reduced to estimating the expectation (4.29). The computational problem then becomes one of calculating repeated realizations of $\theta[\xi_0]$ and combining them into an appropriate statistical estimator of $u(\xi_0)$.

As approximate value of $u(\xi_0)$ is set up

$$\theta_n = 1/n \sum_{s=1}^{n} \{\theta[\xi_0]\}_s,$$

where $\{\theta[\xi_0]\}_s$ is the s-th realization of the random variable $\theta[\xi_0]$ on a Markov chain with initial point ξ_0 and transition density function (4.26).

As it was shown in section 3.1, the probable error for this type of random processes is

$$r_n = c\sigma(\theta[\xi_0])n^{-1/2},$$

where $c \approx 0.6745$ and $\sigma(\theta[\xi_0])$ is the standard deviation.

The direct simulation of a random variable with the stationary density function $p(x,y)$ is unsuitable since the complexity of the expression for $M_y^* L(y,x)$ would sharply increase the algorithm's computational complexity. In this case it is advisable to use the *selection algorithm.*

Denote by $p_0(x,y)$ the transition density function of the Markov chain $M_y^* L_p$ with $c(x) \equiv 0$.

It is easy to see, that

$$p(x,y) \leq p_0(x,y).$$

The function $p_0(x,y)$ satisfies the condition for a transition density function of the Markov chain.

$$\int_{B(x)} p_0(x,y)dy = 1. \tag{4.30}$$

Indeed,

$$\int_{B(x)} p_0(x,y)dy = \int_{B(x)} M_y^* L_p(y,x)\bigg|_{c(y)\equiv 0} dy =$$

$$= \int_{B(x)} \sum_{i=1}^{3} \frac{\partial^2 L_p(y,x)}{\partial y_{(i)}^2} dy - \int_{B(x)} \sum_{i=1}^{3} \left(b_i(y) \frac{\partial L_p(y,x)}{\partial y_{(i)}} \right) dy.$$

Apply the Green's formula (4.12) for the second integral:

$$\int_{B(x)} \sum_{i=1}^{3} \left(b_i(y) \frac{\partial L_p(y,x)}{\partial y_{(i)}} \right) dy$$

$$= \int_{\partial B(x)} \sum_{i=1}^{3} n_i b_i(y) L_p(y,x) d_y S - \int_{B(x)} L_p(y,x) \text{ div } \mathbf{b}(y) dy = 0,$$

because $\text{div } \mathbf{b}(y) = 0$ and $L_p(y,x)|_{y \in \partial B(x)} = 0$, where $\mathbf{n} \equiv (n_1, n_2, n_3)$ is the exterior normal to the boundary $\partial B(x)$.
Calculate the first integral using spherical coordinates:

$$\int_{B(x)} \sum_{i=1}^{3} \frac{\partial^2 L_p(y,x)}{\partial y_{(i)}^2} dy$$

$$= \int_0^R \int_0^\pi \int_0^{2\pi} \frac{r^2 \sin\theta}{4\pi q_p(R)} \frac{1}{r^2} \frac{\partial}{\partial r} r^2 \frac{\partial}{\partial r} \left(\int_r^R (1/r - 1/\rho) p(\rho) d\rho \right) dr d\theta d\varphi$$

$$= \frac{1}{q_p(R)} \int_0^R \frac{\partial}{\partial r} r^2 \frac{\partial}{\partial r} \left(\int_r^R (1/r - 1/\rho) p(\rho) d\rho \right) dr$$

$$= \frac{1}{q_p(R)} \left(r^2 \frac{\partial}{\partial r} \int_r^R (1/r - 1/\rho) p(\rho) d\rho \right) \Bigg|_{r=0}^{r=R}$$

$$= \frac{1}{q_p(R)} (-1) \int_r^R p(\rho) d\rho \Bigg|_{r=0}^{r=R} = \frac{q_p(R)}{q_p(R)} = 1.$$

Thus, we proved (4.30).

The function $p_0(x,y)$ can be expressed in Cartesian coordinates as

$$p_0(x,y) = \frac{\mu_p(R)}{r^2} \left[p(r) + \sum_{i=1}^{3} b_i(y) \frac{y_{(i)} - x_{(i)}}{r} \right] \int_r^R p(\rho) d\rho.$$

Taking into consideration that

$$dy_1 dy_2 dy_3 = r^2 \sin\theta dr d\theta d\varphi \text{ and } y_{(i)} - x_{(i)} = r w_i, \;\; i = 1,2,3$$

one can write:

$$p_0(r, \mathbf{w}) = \mu_p(R) \sin\theta \left[p(r) + \sum_{i=1}^{3} b_i(x + r\mathbf{w}) w_i \right] \int_r^R p(\rho) d\rho,$$

or

$$p_0(r, \mathbf{w}) = \frac{\sin\theta}{4\pi q_p(R)} \left[p(r) + \sum_{i=1}^{3} b_i(x + r\mathbf{w}) w_i \right] \int_r^R p(\rho) d\rho.$$

Here $\mathbf{w}\equiv (w_1, w_2, w_3)$ is an unique isotropic vector in $I\!R^3$, where $w_1 = \sin\theta\cos\varphi$, $w_2 = \sin\theta\sin\varphi$ and $w_3 = \cos\theta$.

Now write $p_0(r, \mathbf{w})$ in the following form:

$$p_0(r, \mathbf{w}) = p_0(r)p_0(\mathbf{w}/r),$$

where

$$p_0(r) = \frac{p(r)}{q_p(R)} = \frac{ke^{-kr}}{1 - e^{-kR}}$$

is a density function and

$$p_0(\mathbf{w}/r) = \frac{\sin\theta}{4\pi}\left[1 + \frac{\mid \mathbf{b}(x + r\mathbf{w}) \mid \cos(\mathbf{b}, \mathbf{w})}{p(r)} \int_r^R p(\rho)d\rho\right]$$

is a conditional density function.

In [ENS84] it is proved that $E \geq \frac{1}{2}$ for the same density function and for the boundary value problem in $I\!R^d (d \geq 2)$.

In [Di89] a majorant function $h_r(\mathbf{w})$ for $p_0(\mathbf{w}/r)$ was found and the following theoretical result for the algorithm efficiency of the selection grid-free Monte Carlo algorithm was proved:

$$E \geq \frac{1 + \alpha}{2 + \alpha}, \tag{4.31}$$

where

$$\alpha = \frac{\max_{x\in\Omega} \mid c(x) \mid R}{\max_{x\in\Omega} \mid \mathbf{b}(x) \mid},$$

and R is the radius of the maximal sphere lying inside Ω.

The following result holds

Theorem 4.3.1 *For the algorithm efficiency of the selection grid-free Monte Carlo algorithm the inequality:*

$$E \geq \frac{1 + \alpha}{2 + \alpha - \varepsilon_R}, \quad 0 < \varepsilon_R = \frac{1}{e^{kR}} < 1,$$

holds, when the majorant function

$$h_r(\mathbf{w}) = \frac{\sin\theta}{4\pi}\left[1 + \frac{\max_{x\in\Omega} \mid \mathbf{b}(x) \mid}{p(r)} \int_r^R p(\rho)d\rho\right]$$

is used.

Proof. Estimate the conditional density function $p_0(\mathbf{w}/r)$:

$$p_0(\mathbf{w}/r) = \frac{\sin\theta}{4\pi}\left[1 + \frac{\mid \mathbf{b}(x + r\mathbf{w}) \mid \cos(\mathbf{b}, \mathbf{w})}{p(r)} \int_r^R p(\rho)d\rho\right]$$

$$\leq \frac{\sin\theta}{4\pi}\left[1+\frac{B}{p(r)}\int_r^R p(\rho)d\rho\right] = h_r(\mathbf{w}), \quad B = \max_{x\in\Omega} \mid \mathbf{b}(x) \mid .$$

On the other hand

$$h_r(\mathbf{w}) = \frac{\sin\theta}{4\pi}\left[1+\frac{B}{p(r)}\int_r^R p(\rho)d\rho\right] = \frac{\sin\theta}{4\pi}\left[1+\frac{B}{k}(1-e^{-k(R-r)})\right]$$

$$= \frac{\sin\theta}{4\pi}\left[1+\frac{B}{k}\left(1-\frac{e^{kr}}{e^{kR}}\right)\right] \leq \frac{\sin\theta}{4\pi}\left[1+\frac{B}{k}\left(1-\frac{1}{e^{kR}}\right)\right] = H(\mathbf{w}). \tag{4.32}$$

The functions $h_r(\mathbf{w})$ and $H(\mathbf{w})$ are majorants for the $p_0(\mathbf{w}/r)$. For the efficiency of the selection Monte Carlo algorithm in the case when $c(y) \equiv 0$ one can obtain:

$$E = \frac{\int_0^{2\pi}\int_0^{\pi} p_0(\mathbf{w}/r)d\theta d\varphi}{\int_0^{2\pi}\int_0^{\pi} H(\mathbf{w})d\theta d\varphi} = \frac{1}{1+\frac{B}{k}\left(1-\frac{1}{e^{kR}}\right)}$$

$$= \frac{k}{k+B(1-\frac{1}{e^{kR}})} = \frac{B+R\max_{x\in\Omega}\mid c(x)\mid}{2B+R\max_{x\in\Omega}\mid c(x)\mid - \frac{B}{e^{kR}}} = \frac{1+\alpha}{2+\alpha-\varepsilon_R},$$

where

$$k = B + R\max_{x\in\Omega} \mid c(x) \mid, \quad \text{(see (4.17))},$$

$$\alpha = \frac{\max_{x\in\Omega}\mid c(x)\mid R}{B} \quad \text{and } \varepsilon_R = \frac{1}{e^{kR}}.$$

Taking into consideration (4.32) , one can get

$$E \geq \frac{1+\alpha}{2+\alpha-\varepsilon_R}, \tag{4.33}$$

when the majorant function $h_r(\mathbf{w})$ is used. This completes the proof.
It is clear that if $\varepsilon_R \to 0$ then the result (4.31) follows. ♢

Denote by $\bar{p}(x,y)$ the following function:

$$\bar{p}(x,y) = \frac{p(x,y)}{V}, \text{ where } \int_{B(x)} p(x,y)dy = V < 1,$$

This is a density function in the case when $c(y) \neq 0$.
The function $p(x,y)$ can be expressed in spherical coordinates as:

$$p(r,\mathbf{w}) = \frac{\sin\theta}{4\pi q_p(R)}\times$$

$$\times\left[p(r) + \left(\sum_{i=1}^{3} b_i(x+r\mathbf{w})w_i + c(x+r\mathbf{w})r\right)\int_r^R p(\rho)d\rho - c(x+r\mathbf{w})r^2\int_r^R \frac{p(\rho)}{\rho}d\rho\right].$$

The following inequalities hold:

$$p(r,\mathbf{w}) \le p_0(r,\mathbf{w}) \le \frac{p(r)}{q_p(R)} h_r(\mathbf{w}) \tag{4.34}$$
$$= \frac{\sin\theta\, p(r)}{4\pi q_p(R)} \left[1 + \frac{\max_{x\in\Omega} \mid \mathbf{b}(x) \mid}{p(r)} \int_r^R p(\rho)d\rho\right] \equiv h(r,\mathbf{w}).$$

It is easy to prove that in the case when $h(r,\mathbf{w})$ is a majorant of the function $p(r,\mathbf{w})$ the efficiency of the selection algorithm is

$$E \ge V \frac{1+\alpha}{2+\alpha-\varepsilon_R}.$$

This estimation follows from Theorem 4.3.1 and (4.34). Clearly, the efficiency E depends on the norm of the kernel $k(x,y)$, because $p(x,y)=k(x,y)$.

In the selection algorithm it is necessary to simulate a random variable η with a density

$$\bar{p}_r(\mathbf{w}) = 1 + \left[\frac{\mid \mathbf{b}(x+r\mathbf{w}) \mid \cos(\mathbf{b},\mathbf{w}) + c(x+r\mathbf{w})r}{p(r)}\right] \times$$
$$\times \int_r^R p(\rho)d\rho - \frac{c(x+r\mathbf{w})r^2}{p(r)} \int_r^R \frac{p(\rho)}{\rho} d\rho.$$

Since

$$\bar{p}_r(\mathbf{w}) \le 1 + \frac{B}{p(r)} \int_r^R p(\rho)d\rho = h(r)$$

the function $h(r)$ is a majorant for the selection algorithm.

Here a Monte Carlo algorithm for the selection algorithm is described:

Consider a point $x \in \Omega$ with the initial density $p(x)$. Suppose that $p(x)$ is tolerant to $g(x)$.

Algorithm 4.3.1 *:*

Grid-free Monte Carlo Algorithm

1. **Calculate** *the radius $R(x)$ of the maximal sphere lying inside Ω and having center x.*

2. **Calculate** *a realization r of the random variable τ with the density*

$$\frac{p(r)}{q_p(R)} = \frac{ke^{-kr}}{1-e^{-kR}}. \tag{4.35}$$

3. **Calculate** *the function*

$$h(r) = 1 + \frac{B}{p(r)} \int_r^R p(\rho)d\rho = 1 + \frac{B}{k}(1-e^{-k(R-r)}).$$

4. **Construct** *independent realizations $\mathbf{w}_j$ of an unique isotropic vector in $I\!R^3$.*

5. **Construct** *independent realizations γ_j of an uniformly distributed random variable in the interval $[0,1]$.*

6. **Calculate** *the parameter j_0, given by*

$$j_0 = \min\{j : h(r)\gamma_j \leq \bar{p}_r(\mathbf{w}_j)\},$$

and **stop** *the execution of the steps 4 and 5. The random vector $\mathbf{w}_{j_0}$ has the density $\bar{p}_r(\mathbf{w})$.*

7. **Calculate** *the random point y, with a density $\bar{p}_r(\mathbf{w})$, using the following formula:*

$$y = x + r\mathbf{w}_{j_0}.$$

The value $r = \mid y - x \mid$ is the radius of the sphere lying inside Ω and having center at x.

8. **Stop** *the algorithm when the random process reaches the ε - strip $\partial\Omega_\varepsilon$, i.e. $y \in \partial\Omega_\varepsilon$. The random variable is calculated.* **If** *$y \bar{\in} \partial\Omega_\varepsilon$* **then** *the algorithm has to be repeated for $x = y$.*

An illustration of the considered *grid-free* algorithm is given on Figure 4.2.

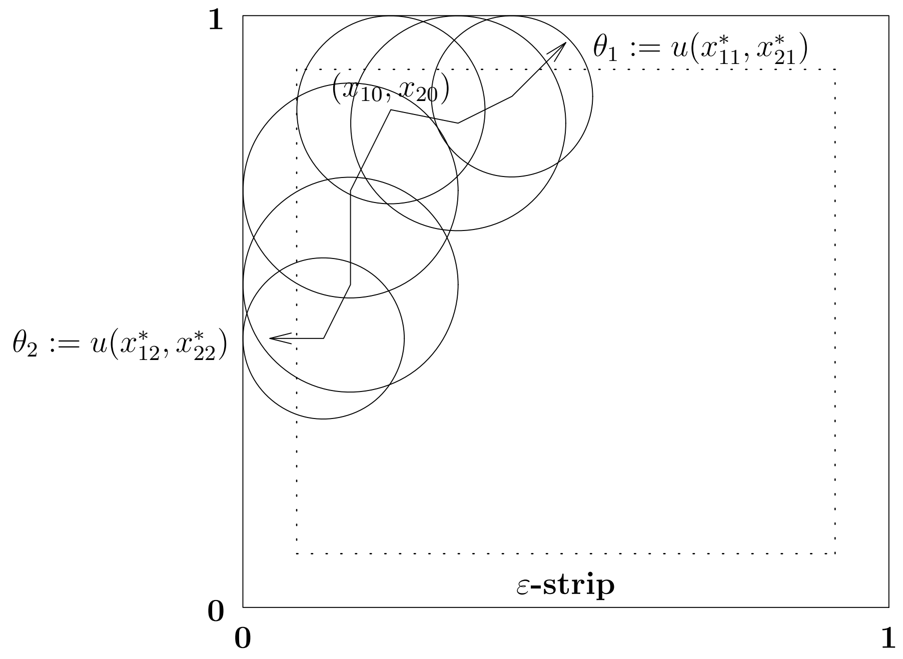
1
$\theta_1 := u(x^*_{11}, x^*_{21})$
(x_{10}, x_{20})
$\theta_2 := u(x^*_{12}, x^*_{22})$
ε-strip
0
0
1

Figure 4.2: **Grid-free Monte Carlo algorithm.**

It is clear that the algorithmic efficiency depends on the expectation of the position of the point y. The location of y depends of the random variable τ with a density (4.35). When the random point is near to the boundary of the ball, the process goes to the boundary of the domain $\partial\Omega_\varepsilon$ quickly. So, it will be important to have an estimate of the mathematical expectation of τ.

One can calculate that

$$E\tau = \int_0^R r\frac{p(r)}{q_p(R)}dr = \int_0^R \frac{rke^{-kr}}{1-e^{-kR}}dr = \frac{1}{k} + \frac{R}{1-e^{kR}}.$$

Obviously, the sequentional algorithmic efficiency depends of the product kR (where R is the radius of the maximal ball, lying inside of the domain Ω for the starting point of the random process). Therefore, the computational results given in the next section are performed for different values of the product kR.

4.3.3 Parallel implementation of the grid-free algorithm and numerical results

It is well known that Monte Carlo algorithms are well suited for parallel architectures. In fact, if we consider the calculation of a trajectory as a single computational process, it is straightforward to regard the Monte Carlo algorithm as a collection of asynchronous processes evolving in parallel. Clearly, MIMD (multiple instruction, multiple data) - machines are the "natural" hardware platform for implementing such algorithms; it seems to be interesting to investigate the feasibility of a parallel implementation on such type of machines. There are two main reasons:

- since Monte Carlo algorithms are many times used from, within or in conjunction with more complex and large existing codes (usually written in FORTRAN, C), the easiness in programming makes the use of these machines very attractive;
- the peak performance of every processor of these machines is usually not very good, but when a large number of processors is efficiently used a high general computational performance can be realized.

The MIMD computer used for our tests is a IBM SP1 with 32 processors. The environment for parallel programming is ATHAPASCAN which is developed by the research group on Parallel algorithms in LMC/IMAG, Grenoble. ATHAPASCAN environment is developed using C-language and a special library for message passing which is similar to well - known MPI-Message Passing Interface and PVM-Parallel Virtual Machine. ATHAPASCAN allows to distribute the computational problem on different type of processors or/and computers. This environment provides use of dynamic distribution of common resources and has a high level of parallel efficiency if the numerical algorithm is well parallelized. (For more information, see, ([Pl94]).

In the previous section a general description of the Monte Carlo algorithm for the selection algorithm has been provided. Note that, in the case of an implementation on a sequential computer, all the steps of the algorithm and all the trajectories are executed iteratively, whereas on a parallel computer the trajectories can be carried concurrently.

Example. A numerical examples are considered. The example deals with the following problem

$$\sum_{i=1}^{3}\left(\frac{\partial^2 u}{\partial x_{(i)}^2}+b_i(x)\frac{\partial u}{\partial x_{(i)}}\right)+c(x)u=0, \text{ in } \Omega=[0,1]^3.$$

Note that the cube $\Omega = [0,1]^3$ does not belong to the $\mathbf{A}^{(1,\lambda)}$, but this restriction is not important for our algorithm since an ε-strip of the domain Ω is considered. In fact now we consider another domain Ω_ε which belongs to the class $\mathbf{A}^{(1,\lambda)}$.

The boundary conditions for the example are:

$$u(x_{(1)},x_{(2)},x_{(3)})=e^{a_1x_{(1)}+a_2x_{(2)}+a_3x_{(3)}}\,,\ (x_{(1)},x_{(2)},x_{(3)})\in\partial\Omega.$$

In our tests

$$b_1(x)=a_2a_3(x_{(2)}-x_{(3)}),\ b_2(x)=a_3a_1(x_{(3)}-x_{(1)}),\ b_3(x)=a_{(1)}a_2(x_{(1)}-x_{(2)})$$

(thus, the condition *div* $\mathbf{b}(x)=0$ is valid) and

$$c(x)=-(a_1^2+a_2^2+a_3^2),$$

where a_1 , a_2 , a_3 are parameters.

The problem is solved using selection grid-free Monte Carlo algorithm.
We consider three cases for the coefficients:

- the first case when

$$a_1=0.25\,,\ a_2=0.25\,,\ a_3=0.25 \text{ and } k*R=0.101;$$

- the second case when

$$a_1=0.5\,,\ a_2=0.5\,,\ a_3=0.5 \text{ and } k*R=0.40401;$$

- the third case

$$a_1=-1\,,\ a_2=-1\,,\ a_3=-1 \text{ and } k*R=1.61603.$$

Four different ε-strip are used:

$$\varepsilon=0.01,\ 0.05,\ 0.1,\ 0.3.$$

The results of evaluating the linear functional (3.7) for the above mentioned parameters and functions are presented in Figures 4.3 - 4.5, the case when

$$h(x)=\delta[(x_{(1)}-1/2),(x_{(2)}-1/2),(x_{(3)}-1/2)].$$

The efficiency of the selection grid-free Monte Carlo does not depend on the number of trajectories (see, Table 4.1). The result of selection efficiency confirms our corresponding theoretical result.

Table 4.1: **Selection efficiency and number of the steps to the boundary domain.**

Epsilon strip	$k * R$	No of steps	Selection efficiency
0.01	0.101	$36 - 37$	0.99
0.01	0.40401	$35 - 36$	0.97123
0.01	1.61603	$43 - 44$	0.91071
0.05	0.101	$17 - 18$	0.99
0.05	0.40401	$17 - 18$	0.9596
0.05	1.61603	$20 - 21$	0.85829
0.10	0.101	$8 - 9$	0.9887
0.10	0.40401	$9 - 10$	0.95371
0.10	1.61603	$12 - 13$	0.83596
0.30	0.101	$1 - 2$	0.97
0.30	0.40401	$2 - 3$	0.92583
0.30	1.61603	$2 - 3$	0.75561

Remarks:

1. The number, which starts generator is 653;
2. The number of realizations of the random ball process is 600.

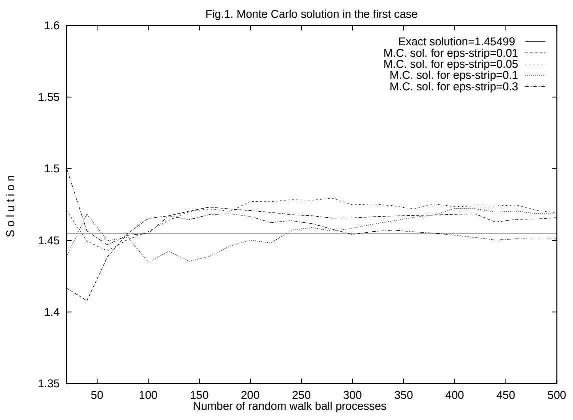

Figure 4.3
Monte Carlo solution in the first case.

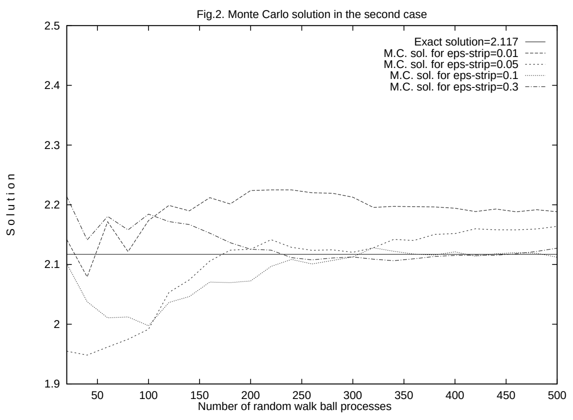

Figure 4.4
Monte Carlo solution in the second case.

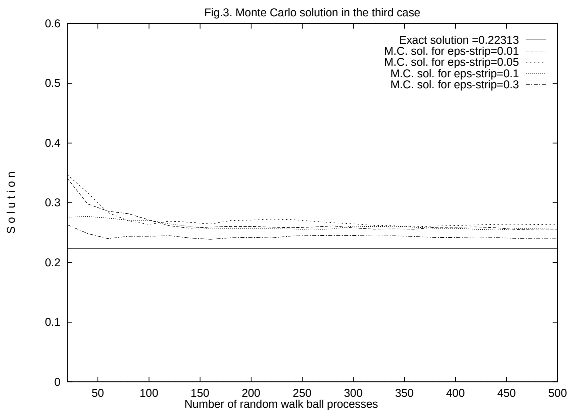

Figure 4.5
Monte Carlo solution in the third case.

The efficiency of the presented grid-free algorithm is studied in the case when

$$h(x) = \delta[(x_{(1)} - 1/4), (x_{(2)} - 1/4), (x_{(3)} - 1/4)],$$

$$h(x) = \delta[(x_{(1)} - 1/2), (x_{(2)} - 1/2), (x_{(3)} - 1/2)],$$

respectively.
We investigate the parallel efficiency for the following values of the coefficients:

$$a_1 = 1\,,\ a_2 = -0.5\,,\ a_3 = -0.5 \ \text{ and } \ \varepsilon - \text{strip} = 0.01\,,\ 0.05\,,\ 0.15.$$

4.3.4 Concluding remarks

- An iterative Monte Carlo algorithm using Green's function is presented and studied. It is proved that the integral transformation kernel in local integral presentation can be used as transition density function in the Markov chain. An algorithm called *ball process* is presented. This algorithm is a grid-free Monte Carlo algorithm and uses the so-called selection.

- One of the advantages of the grid-free Monte Carlo algorithm is that it has the rate of convergence $(|log\ r_n|/r_n^2)$ (where r_n is the statistical error) which is better than the rate r_n^{-3} of the grid algorithm. This means that the same error can be reached with a smaller number of trajectories.
- It is preferable to use the selection algorithm when it is difficult to calculate the realizations of the random variable directly.
- The studied algorithm has high parallel efficiency. It is easily programmable and parallelizable.
- The tests performed show that Monte Carlo algorithms can be efficiently implemented on MIMD-machines.

4.4 Adjoint Formulation for Convection-diffusion Problem

The problem of calculating the functional (4.3), where u is a solution of (4.1), (4.2) includes many boundary value problems as well.

Consider the problem of convection-diffusion particle's transport [ZCM91]:

$$\frac{\partial u}{\partial t} = -\sum_{i=1}^{3} v_i(x,t)\frac{\partial u}{\partial x_{(i)}} + \sum_{i=1}^{3} \mu_i \frac{\partial^2 u}{\partial {x_{(i)}}^2} + E(x,t) - \sigma u,$$

where $x \equiv (x_{(1)}, x_{(2)}, x_{(3)}) \in \Omega$ and $t \in [0,T]$ with the boundary conditions

$$u(x,0) = 0; -\frac{\partial u}{\partial x_{(3)}} + \alpha u = 0, x \in \partial\Omega.$$

Following the approach of Sabelfeld [Sa89] we use the substitution

$$u(x,t) = e^{\alpha x_{(3)}} u(\lambda_1 x_{(1)}, \lambda_2 x_{(2)}, \lambda_3 x_{(3)}, t);$$

$$\lambda_i = \sqrt{\mu/\mu_i}$$

(μ is a given constant)

One can obtain

$$\frac{\partial u}{\partial t} = -\sum_{i=1}^{3} a_i(x,t)\frac{\partial u}{\partial x_{(i)}} + \mu\Delta u - b(x,t)u + g(x,t) \tag{4.36}$$

$$u(x,0) = 0,\ \frac{\partial u}{\partial x_{(3)}} = 0, \tag{4.37}$$

where

$$a_i(x,t) = \lambda_i v_i(x_{(1)}/\lambda_1, x_{(2)}/\lambda_2, x_{(3)}/\lambda_3, t), i = 1, 2;$$

$$a_3(x,t) = \lambda_3 v_3(x_1/\lambda_1, x_{(1)}/\lambda_2, x_{(3)}/\lambda_3, t) - 2\mu_3\alpha\lambda_3;$$

$$b(x,t) = \sigma(x_{(1)}/\lambda_1, x_{(2)}/\lambda_2, x_{(3)}/\lambda_3, t) - \mu_3\alpha^2 + \alpha v_3(x_{(1)}/\lambda_1, x_{(2)}/\lambda_2, x_{(3)}/\lambda_3, t)$$

In order to solve this problem by a Monte Carlo algorithm the first step involves obtaining an integral representation of the solution.

The solution of the problem (4.36), (4.37) can be represented as *volume potential* with a kernel $u_0(x,t,x',t')$:

$$u(x,t) = \int_0^t dt' \int_\Omega u_0(x,t,x^{'},t^{'})q(x^{'},t^{'})dx^{'}. \tag{4.38}$$

Suppose $u_0(x,t,x^{'},t^{'})$ is a given function from the space L_∞ and $q(x,t)$ belongs to the space L_1.

The density q(x,t) satisfies the following equation [Sa89]:

$$q(x,t) + \int_0^t dt^{'} \int_\Omega k(x^{'},t^{'},x,t)q(x^{'},t^{'})dr^{'} = g(x,t), \tag{4.39}$$

where

$$\begin{gathered}
k(x',t',x,t) = \\
-\sum_{i=1}^{3} \frac{(a_i - a_i{}')[x_{(i)} - x_{(i)}{}' - a_i'(t-t')]}{2\mu(t-t')}, Z_0 \\
-\sum_{i=1}^{3} \frac{(a_i - a_i')[x_{(i)} - \hat{x}_{(i)}' - \hat{a}_i'(t-t')]}{2\mu(t-t')} \hat{Z}_0 \\
-\frac{a_3{}'[x_{(3)} + x_{(3)}{}' + a_3{}'(t-t')]}{\mu(t-t')} \hat{Z}_0 + (b-b')(Z_0 + \hat{Z}_0) - \\
\rho Z_0 \left[\sum_{i=1}^{3} (a_i - a_i{}')\alpha_i / 2\mu(t-t') \right] - \hat{\rho}\hat{Z}_0 [\sum_{i=1}^{3} (a_i - a_i{}')\beta_i / 2\mu(t-t') \\
-2a_3{}'\beta_3/(2\mu(t-t'))] + (b-b')(Z_0 + \hat{Z}_0),
\end{gathered} \tag{4.40}$$

where

$$Z^0(x,t,x',t') = \frac{e^{-\frac{\sum_{i=1}^{3}\left[x_{(i)} - x'_{(i)} - \alpha_i(t-t')\right]^2}{4\mu(t-t')} - \beta(t-t')}}{[4\pi\mu(t-t')]^{3/2}},$$

$$\hat{Z}^0(x,t,x',t') = e^{-\frac{\sum_{i=1}^{2}\left[x_{(i)}-x'_{(i)}-a_i(x',t)(t-t')\right]^2}{4\mu(t-t')}} \times \frac{e^{[x_{(3)}+x'_{(3)}+a_3(x',t')(t-t')]^2 4\mu(t-t')-b(t-t')}}{[4\pi\mu(t-t')]^{3/2}}$$

with

$$\rho = |x - x' - a'(t-t')|, \hat{\rho} = |x - \hat{x}' - \hat{a}'(t-t')|,$$

$$\alpha_i = [x - {x'}_i - {a'}_i(t-t')]/\rho, \beta_i = [x - \hat{x}'_i - \hat{a}'_i(t-t')]/\rho.$$

The last representation allows us to use Monte Carlo algorithm for solving the problem since for every fixed point $y = (x,t)$ (4.38) is a finite linear functional of the type (4.3).

A similar approach is valid when the initial condition $u(r,0)$ is given as two-dimensional mesh-function. We deal with noisy data $z^0_{i,j}$, defined on the mesh $\Delta = \Delta_{x_{(1)}} \times \Delta_{x_{(2)}}$, where

$$\Delta_{x_{(1)}} : a = x_{1,0} < x_{1,1} < \ldots < x_{1,N} = b;$$

$$\Delta_{r_2} : c = x_{2,0} < x_{2,1} < \ldots < x_{2,M} = d; \tag{4.41}$$

Suppose

$$u(x_{(1)}, x_{(2)}, x_{(3)}, 0) = S(x_{(1)}, x_{(2)}), \tag{4.42}$$

where $S(x_{(1)}, x_{(2)})$ is a cubic spline, which satisfies the conditions:

$$\begin{aligned} D^{2,0}S(x_{1,i}; x_{2,j}) &= 0,\ i = 0, N; j = 0, 1, ..., M; \\ D^{0,2}S(x_{1,i}; x_{2,j}) &= 0,\ i = 0, 1, ..., N; j = 0, 1, ..., M; \\ D^{2,2}S(x_{1,i}; x_{2,j}) &= 0,\ i = 0, 1, ..., N; j = 0, 1, ..., M; \end{aligned} \tag{4.43}$$

$$(D^{p,q}f(x_{1,i}; x_{2,j}) = \frac{\partial^{p+q}}{\partial x_1^p \partial x_2^q} f(x_{(1)}, x_{(2)})\Big|_{x_{(1)}=x_{1,i}, x_2=x_{2,j}}) \tag{4.44}$$

We use this approach since among the functions $f(x_{(1)}, x_{(2)})$ from the functional space $W_2^{2,2}(\Omega)$, the cubic spline $S(x_{(1)}, x_{(2)})$, which satisfies the conditions (4.43) minimizes the functional [ZKM80]:

$$J(f) = \int_a^b \int_c^d \left[D^{2,2}f(x_{(1)}, x_{(2)})\right]^2 dx_{(1)}dx_{(2)} + \sum_{i=0}^{N} \rho_i^{-1} \int_c^d \left[D^{0,2}f(x_{1,i}, x_{(2)})\right]^2 dx_2 +$$

$$\sum_{j=0}^{M} \sigma_j^{-1} \int_a^b \left[D^{2,0}f(x_{(1)}, x_{2,j})\right]^2 dx_{(1)} + \sum_{i=0}^{N}\sum_{j=0}^{M} \rho_i^{-1}\sigma_j^{-1}(f_{ij} - z^0_{ij})^2 \tag{4.45}$$

The construction of the spline $S(x_{(1)}, x_{(2)})$ is now reduced to the solution of one-dimensional smoothing problems. Consider a space of two-dimensional splines $S(\Delta)$ in the domain Ω with a mesh Δ as a tensor product of two spaces

$$S(\Delta) = S(\Delta_{x_{(1)}}) \otimes S(\Delta_{x_{(2)}}). \tag{4.46}$$

The basis of the space $S(\Delta)$ is

$$\Phi_p(x_{(1)})\Psi_q(x_{(2)}), \quad p = 0, \ldots, N; q = 0, \ldots, M. \tag{4.47}$$

Any function from $S(\Delta)$ can be represented in the form

$$S(x_{(1)}, x_{(2)}) = \sum_{p=0}^{N}\sum_{q=0}^{M} z^0_{pq}\Phi_p(x_{(1)})\Psi_q(x_{(2)}) = \sum_{q=0}^{N} S_q(x_{(1)})\Psi_q(x_{(2)}),$$

where

$$S_q(x_{(1)}) = \sum_{p=0}^{N} z^0_{pq}\Phi_p(x_{(1)}).$$

The functions $S_q(x_{(1)})$ are smoothing splines for smoothing initial data z^0_{pq} on the lines $x_{(2)} = x_{2,q}, q = 0, \ldots, M$. The function $S(x_{(1)}, x_{(2)})$ is a spline for smoothing values of $S_q(x_{(1)})$, which depends on $x_{(1)}$ as a parameter in the nodes of the mesh $\Delta_{x_{(2)}}$. Construct the track of this spline $\tilde{S}_p(x_{(2)}) = S(x_{1,p}; x_{(2)}), p = 0, \ldots, N$ giving as initial data $S_q(x_{1,p}), q = 0, \ldots, M$. It is possible to obtain new values: $z^0_{pq} = \tilde{S}_p(x_{2,q}) = S(x_{1,p}; x_{2,q}), p = 0, \ldots, N$. For these values we construct the interpolation spline, which is a smoothing spline $S(x_{(1)}, x_{(2)})$ for the initial data $z^0_{pq}, p = 0, \ldots, N; q = 0, \ldots, M$. Now we can obtain an integral representation in the form (4.39) for the equation (4.36) using the condition (4.42) instead of condition (4.37).

4.5 Stationary Problem of Particle's Transport

Now we shall show that the grid Monte Carlo algorithm can be considered for solving the stationary problem of particle transport.

First, consider the stationary problem of particle transport [Ma82]:

$$\mu\nabla^2 u(x_{(1)}, x_{(2)}) - b(x_{(1)}, x_{(2)})u = e(x_{(1)}, x_{(2)}) \text{ or}$$

$$\nabla^2 u(x_{(1)}, x_{(2)}) - c(x_{(1)}, x_{(2)})u(x_{(1)}, x_{(2)}) = -f(x_{(1)}, x_{(2)}), \tag{4.48}$$

$$(x_{(1)}, x_{(2)}) \in \Omega,$$

where $c(x_{(1)}, x_{(2)}) = b(x_{(1)}, x_{(2)})/\mu$; $f(x_{(1)}, x_{(2)}) = e(x_{(1)}, x_{(2)})/\mu$, with the boundary conditions:

$$u(x_{(1)}, x_{(2)})|_{\partial\Omega} = g(x_{(1)}, x_{(2)}). \tag{4.49}$$

Consider a regular mesh Ω_h with a step h, defined on the domain Ω. Suppose $u \in C^2(\Omega)$. Since the domain Ω is bounded, the condition $u \in L_1(\Omega)$ is fulfilled. Obviously, Ω_h contains a finite number of nodes. Let the number of nodes be m (if j is a node number, $j = 1, 2, \ldots, m$). Denote by u_j the value of the function $u(x_{(1)}, x_{(2)})$

when r_1 and $x_{(2)}$ coincide with coordinates of the j-th node. This can always be done since u is continuous (moreover, $u \in C^2(\Omega)$).

Discretisation of the problem (4.48) and (4.49) leads to the following system of linear algebraic equations:

$$u_j = \sum_{l=1}^{m} l_{jl} u_l + f_j, \qquad j = 1, \ldots, m,$$

or

$$u = Lu + f. \tag{4.50}$$

In the last equation:

$$u = \begin{bmatrix} u_1 \\ u_2 \\ \vdots \\ u_m \end{bmatrix}; f = \begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ f_m \end{bmatrix}$$

and

$$L = \begin{bmatrix} l_{11} & l_{12} & \ldots & l_{1m} \\ l_{21} & l_{22} & \ldots & l_{2m} \\ & & \vdots & \\ l_{m1} & l_{m2} & \ldots & l_{mm} \end{bmatrix}.$$

It is easy to prove that the system (4.51) is equivalent to the following integral equation:

$$u(x) = \int_0^m l(x, x')u(x')dx\prime + f(x), \tag{4.51}$$

where $u(x)$ and $f(x)$ are one - dimensional step - functions and $l(x, x')$ is a two - dimensional step - function.

In actual fact , if $u(x) = u_j$ for $x \in \Omega_j \equiv [j-1, j]$, $f(x) = f_j$ for $x \in \Omega_j$ and $l(x, x') = l_{jl}$ for $x \in \Omega_j$ and $x' \in \Omega_l$ then the corresponding integral equation can be written in the form (4.50).

In this case, the grid Monte Carlo algorithm consists in moving from one coefficient to another of the matrix (4.51) with a transition density function $p(x_{j-1}, x_{(j)}) = p_{j-1,j}$ which is chosen to be proportional to $|l_{j-1,j}|$:

$$p_{j-1,j} = c|l_{j-1,j}| \tag{4.52}$$

The process terminates when boundary node b is chosen. The matrix L contains approximately $\sqrt{m}$ boundary nodes.

Obviously, a random process can not visit coefficients which are equal to zero. This fact increases the efficiency of the algorithm.

Appendix A

Consider the boundary-value problem

$$\Delta u(x) = 0, \quad x = (x_1, x_2, x_3) \in \Omega, \tag{A.1}$$

and the boundary condition

$$u(x) = \psi(x), \quad x \in \partial\Omega. \tag{A.2}$$

Using the finite-difference technique, one can derive a system of linear equations whose solution approximates the solution of (A.1). The system depends on the approximation molecule for the Laplace operator.

Let us consider the usual seven-point approximation molecule. The equation which approximates (A.1) in the point $x_i = (i_1h, I_2h, i_3h)$ is

$$\Lambda_1(u_i) - 6u_i = 0, \tag{A.3}$$

where

$$\begin{aligned}\Lambda_1(u_i) = \;& u((i_1+1)h, i_2h, i_3h) + u((i_1-1)h, i_2h, i_3h) + u(i_1h, (i_2+1)h, i_3h) \\ & + u(i_1h, (i_2-1)h, i_3h) + u(i_1h, i_2h, (i_3+1)h) + u(i_1h, i_2h, (i_3-1)h),\end{aligned} \tag{A.4}$$

and h is the mesh size of the discrete domain $\partial\Omega_h$. For brevity, in what follows we assume h to be unity.

Using (A.3) for all terms in (A.4) we obtain

$$u_i = \frac{1}{36}\left[\Lambda_2(u_i) + 2\Lambda_{1,m}(u_i) + 6u_i\right], \tag{A.5}$$

where

$$\begin{aligned}\Lambda_{1,m}(u_i) = \;& u(i_1+1, i_2, i_3+1) + u(i_1-1, i_2, i_3+1) + u(i_1-1, i_2, i_3-1) \\ & + u(i_1+1, i_2, i_3-1) + u(i_1+1, i_2+1, i_3) + u(i_1, i_2+1, i_3+1) \\ & + u(i_1-1, i_2+1, i_3) + u(i_1, i_2+1, i_3-1) + u(i_1+1, i_2-1, i_3) \\ & + u(i_1, i_2-1, i_3+1) + u(i_1-1, i_2-1, i_3) + u(i_1, i_2-1, i_3-1),\end{aligned} \tag{A.6}$$

and $\Lambda_2(u)$ is obtained from the formula

$$\begin{aligned}\Lambda_k(u_i) &= u(i_1+k, i_2, i_3) + u(i_1-k, i_2, i_3) + u(i_1, i_2+k, i_3) \\ &+ u(i_1, i_2-k, i_3) + u(i_1, i_2+k, i_3+k) + u(i_1, i_2, i_3-k),\end{aligned}$$

when $k = 2$.

If we use the Taylor formula for terms in $\Lambda_{1,m}(u_i)$ it is easy to construct another approximation molecule for (A.1) which leads to

$$\Lambda_{1,m}(u_i) = 12u_i. \tag{A.7}$$

Then (A.5) becomes

$$u_i = \frac{1}{6}\Lambda_2(u_i), \tag{A.8}$$

which is of the same type as (A.3) but the step in the approximation molecule is 2. Application of the algorithm described above leads to the following theorem.

Theorem Appendix A.1 . *Let $x_i = (i_1, i_2, i_3)$ be an arbitrary point in Ω_h and k be the radius of the largest sphere in Ω_h with the centre in x_i. Then the following equation holds:*

$$u_i = \frac{1}{6}\Lambda_k(u_i). \tag{A.9}$$

To prove this theorem some preliminary statements are needed.

Lemma Appendix A.1 . *For each integer k the following formula holds:*

$$\Lambda_k(u_i) = \frac{1}{6}\left[\Lambda_{k+1}(u_i) + \Lambda_{k-1}(u_i) + \tilde{\Lambda}_k(u_i)\right], \tag{A.10}$$

where

$$\begin{aligned}
\tilde{\Lambda}_k(u_i) \;&=\; u(i_1+k, i_2+1, i_3) + u(i_1+k, i_2, i_3+1) + u(i_1+k, i_2-1, i_3) \\
&+\; u(i_1+k, i_2, i_3-1) + u(i_1-k, i_2+1, i_3) + u(i_1-k, i_2, i_3+1) \\
&+\; u(i_1-k, i_2-1, i_3) + u(i_1-k, i_2, i_3-1) + u(i_1+1, i_2+k, i_3) \\
&+\; u(i_1, i_2+k, i_3+1) + u(i_1-1, i_2+k, i_3) + u(i_1, i_2+k, i_3-1) \\
&+\; u(i_1+1, i_2-k, i_3) + u(i_1, i_2-k, i_3+1) + u(i_1-1k, i_2-k, i_3) \\
&+\; u(i_1, i_2-k, i_3-1) + u(i_1+1, i_2, i_3+k) + u(i_1, i_2+1, i_3+k) \\
&+\; u(i_1-1, i_2, i_3+k) + u(i_1, i_2-1, i_3+k) + u(i_1+1, i_2, i_3-k) \\
&+\; u(i_1, i_2+1, i_3-k) + u(i_1-1, i_2, i_3-k) + u(i_1, i_2-1, i_3-k).
\end{aligned}$$

The prof of Lemma 1 follows from (A.3) and (A.6).

Lemma Appendix A.2 *For an arbitrary integer k it follows that*

$$\tilde{\Lambda}_k(u_i) = \begin{cases} \displaystyle\sum_{l=0}^{(k-3)/2} (-1)^l \left(12\Lambda_{k-2l-1}(u_i) - \overline{\Lambda}_{k-2l-1}(u_i)\right) + (-1)^{[k/2]}\tilde{\Lambda}_1(u_i), \\ \hfill k \quad \textit{odd}, \\ \displaystyle\sum_{l=0}^{(k-2)/2} (-1)^l \left(12\Lambda_{k-2l-1}(u_i) - \overline{\Lambda}_{k-2l-1}(u_i)\right) + (-1)^{[k/2]}\tilde{\Lambda}_0(u_i), \\ \hfill k \quad \textit{even}, \end{cases} \tag{A.11}$$

where [t] means the integer part of t, and

$$\begin{aligned}
\overline{\Lambda}_k(u_i) &= u(i_1+k, i_2+1, i_3-1) + u(i_1+k, i_2+1, i_3+1) + u(i_1+k, i_2-1, i_3+1) \\
&+ u(i_1+k, i_2-1, i_3-1) + u(i_1-k, i_2+1, i_3-1) + u(i_1-k, i_2+1, i_3+1) \\
&+ u(i_1-k, i_2-1, i_3+1) + u(i_1-k, i_2-1, i_3-1) + u(i_1+1, i_2+k, i_3-1) \\
&+ u(i_1-1, i_2+k, i_3-1) + u(i_1-1, i_2+k, i_3+1) + u(i_1+1, i_2+k, i_3+1) \\
&+ u(i_1+1, i_2-k, i_3-1) + u(i_1-1, i_2-k, i_3+1) + u(i_1-1, i_2-k, i_3-1) \\
&+ u(i_1+1, i_2-k, i_3+1) + u(i_1+1, i_2-1, i_3+k) + u(i_1+1, i_2+1, i_3+k) \\
&+ u(i_1-1, i_2+1, i_3+k) + u(i_1-1, i_2-1, i_3+k) + u(i_1+1, i_2-1, i_3-k) \\
&+ u(i_1+1, i_2+1, i_3-k) + u(i_1-1, i_2+1, i_3-k) + u(i_1-1, i_2-1, i_3-k).
\end{aligned}$$

P r o o f. Using formula (A.7) for each term in $\Lambda_{k-1}(u_i)$, we obtain

$$12\Lambda_{k-1}(u_i) = \tilde{\Lambda}_k(u_i) + \overline{\Lambda}_{k-1}(u_i) + \tilde{\Lambda}_{k-2}(u_i)$$

or

$$\tilde{\Lambda}_k(u_i) = \tilde{\Lambda}_{k-2}(u_i) + \overline{\Lambda}_{k-1}(u_i) + 12\Lambda_{k-1}(u_i), \tag{A.12}$$

and applying it recursively yields (A.11). $\diamondsuit$

Lemma Appendix A.3 *For an arbitrary integer k the following formula holds:*

$$\overline{\Lambda}_k(u_i) = \begin{cases} 8\displaystyle\sum_{l=0}^{(k-3)/2} (-1)^l \Lambda_{k-2l-1}(u_i) + (-1)^{[k-2]}\overline{\Lambda}_1(u_i), & k \quad odd, \\ 8\displaystyle\sum_{l=0}^{(k-2)/2} (-1)^l \Lambda_{k-2l-1}(u_i) + (-1)^{[k-2]}\overline{\Lambda}_0(u_i), & k \quad even, \end{cases} \tag{A.13}$$

P r o o f. Using the Taylor formula one can derive the approximation formula

$$\Lambda_{1,e}(u_i) = 8u_i, \tag{A.14}$$

where $\Lambda_{1,e}(u_i) = \frac{1}{3}\overline{\Lambda}_1(u_i)$.

Then applying this formula for all terms in $\Lambda_{k-1}(u_i)$, we obtain

$$8\Lambda_{k-1}(u_i) = \tilde{\Lambda}_k(u_i) + \overline{\Lambda}_{k-2}(u_i)$$

or

$$\overline{\Lambda}_k(u_i) = -8\Lambda_{k-1}(u_i) + \overline{\Lambda}_{k-2}(u_i), \tag{A.15}$$

which leads to (A.13). $\diamondsuit$

P r o o f. of Theorem 1. When k is unity, (A.9) becomes the usual approximation formula (A.3).

We will prove (A.9) when $k = n$, provided it holds for all $k = 2, 3, \ldots, n-1$.

From Lemma 1 it follows that

$$\Lambda_n(u_i) = \frac{1}{6}\left[\Lambda_{n+1}(u_i) + \Lambda_{n-1}(u_i) + \overline{\Lambda}_n(u_i)\right],$$

and according to the above assumption

$$\Lambda_n(u_i) = \frac{1}{6}\left[\Lambda_{n+1}(u_i) + 6u_i + \overline{\Lambda}_n(u_i)\right],$$

and so for $k = n$, (A.9) becomes

$$u_i = \frac{1}{36}\left[\Lambda_{n+1}(u_i) + 6u_i + \tilde{\Lambda}_n(u_i)\right]. \tag{A.16}$$

Without any restrictions of generality we assume that $n = 2m - 1$. So using Lemmas 2 and 3 we obtain

$$\tilde{\Lambda}_{2m-1}(u_i) = \sum_{l=0}^{m-2}(-1)^l\left[12\Lambda_{2(m-l-1)}(u_i) - \overline{\Lambda}_{2(m-l-1)}(u_i)\right] + (-1)^m\tilde{\Lambda}_1(u_i)$$

$$= \sum_{l=0}^{m-2}(-1)^l[12\Lambda_{2(m-l-1)}(u_i) - 8\sum_{s=0}^{m-l-2}(-1)^s\Lambda_{2(m-2l)-3}(u_i).$$

$$-(-1)^{m-l-1}\overline{\Lambda}_0(u_i)\Big] + (-1)^m\tilde{\Lambda}_1(u_i).$$

From the definitions follows that

$$\overline{\Lambda}_0(u_i) = 24u_i \text{ and } \tilde{\Lambda}_1(u_i) = 24u_i,$$

and from the assumption that

$$\Lambda_j(u_i) = 6u_i,$$

when $j < n$. Then

$$\tilde{\Lambda}_{2m-1}(u_i) = \sum_{l=0}^{m-2}(-1)^l\left[72u_i - 8\sum_{s=0}^{m-l-2}(-1)^s 6u_i - (-1)^{m-l-1}24(u_i)\right] + (-1)^m 24u_i$$

$$= 72u_i\sum_{l=0}^{m-2}(-1)^l - 48u_i\sum_{l=0}^{m-2}(-1)^l\sum_{s=0}^{m-l-2}(-1)^s - \sum_{l=0}^{m-2}(-1)^{m-1}24u_i$$

$$+(-1)^m 24u_i = 24u_i,$$

and (A.16) becomes

$$u_i = \frac{1}{36}\left[\Lambda_{n+1}(u_i) + 30u_i\right] \quad \text{or} \quad u_i = \frac{1}{6}\Lambda_{n+1}(u_i). \tag{A.17}$$

The case when $k = 2m$ is similar. $\diamondsuit$

Theorem 1 is used to construct a Monte Carlo method for finding the inner product of a given vector **g** with the solution of the system (A.3).

The algorithm is as follows.

- **(i)** The start point x_0 is selected according to a density permissible for $\mathbf{g}$.
- **(ii)** Determine the mesh distance $d_h(x_0)$ from the selected point x_0 to the boundary; the next point is selected from among the neighbours on the seven-point approximation molecule with step $d_h(x_0)$;
- – if the point is on the boundary, the the process terminates;
- – otherwise the process continues with (ii).

Appendix B

Here all the results for the values of interest are summarized.

B.1. Algorithm $\mathcal{A}$ $\quad (f(x) \equiv 0, \quad p = (c_{0.5}\sigma(\theta)/\epsilon)^2)$

$$ET_1(\mathcal{A}) = \tau\left((k+1+\gamma_A)l_A) + (n+1+\gamma_L)l_L\right)\frac{1}{4\epsilon}\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$ET_{pipe}(\mathcal{A}) = \tau(s+k+l_A+\gamma_A+(n+1+\gamma_L)l_L)\frac{1}{4\epsilon}\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$ET_p(\mathcal{A}) = \tau\left((k+1+\gamma_A)l_A + (n+1+\gamma_L)l_L\right)\frac{1}{4\epsilon},$$

$$ET_{2np}(\mathcal{A}) = \tau\left((k+1+\gamma_A)l_A + 3l_L\right)\frac{1}{4\epsilon},$$

$$S_{pipe}(\mathcal{A}) = \left(1+\frac{k+1+\gamma_A}{n+1+\gamma_L}\frac{l_A}{l_L}\right)\bigg/\left(1+\frac{s+k+l_A+\gamma_A}{n+1+\gamma_L}\frac{1}{l_L}\right),$$

$$S_p(\mathcal{A}) = p,$$

$$S_{2np}(\mathcal{A}) = p\left(1+\frac{n+1+\gamma_L}{k+1+\gamma_A}\frac{l_L}{l_A}\right)\bigg/\left(1+\frac{3}{k+1+\gamma_A}\frac{l_L}{l_A}\right),$$

$$E_p(\mathcal{A}) = 1,$$

$$E_{2np}(\mathcal{A}) = \frac{1}{2n}\left(1+\frac{n+1+\gamma_L}{k+1+\gamma_A}\frac{l_L}{l_A}\right)\bigg/\left(1+\frac{3}{k+1+\gamma_A}\frac{l_L}{l_A}\right).$$

B.2. Algorithm $\mathcal{B}$ $\quad (f(x) \equiv 0, \quad n = 3, \quad p = (c_{0.5}\sigma(\theta)/\epsilon)^2)$

$$ET_1(\mathcal{B}) = 6\tau\left((k+1+\gamma_A)l_A + 9l_L\right)\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$ET_{pipe}(\mathcal{B}) = 6\tau(s+k+l_A+\gamma_A+9l_L)\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$ET_p(\mathcal{B}) = 6\tau\left((k+1+\gamma_A)l_A + 9l_L\right),$$

$$ET_{6p}(\mathcal{B}) = 6\tau\left((k+1+\gamma_A)l_A + 5l_L\right),$$

$$S_{pipe}(\mathcal{B}) = \left(1 + \frac{1}{9}(k+1+\gamma_A)\frac{l_A}{l_L}\right) \Big/ \left(1 + \frac{1}{9}(s+k+l_A+\gamma_A)\frac{1}{l_L}\right),$$

$$S_p(\mathcal{B}) = p,$$

$$S_{6p}(\mathcal{B}) = p\left(1 + \frac{9}{k+1+\gamma_A}\frac{l_L}{l_A}\right) \Big/ \left(1 + \frac{5}{k+1+\gamma_A}\frac{l_L}{l_A}\right),$$

$$E_p(\mathcal{B}) = 1,$$

$$E_{6p}(\mathcal{B}) = \frac{1}{6}\left(1 + \frac{9}{k+1+\gamma_A}\frac{l_L}{l_A}\right) \Big/ \left(1 + \frac{5}{k+1+\gamma_A}\frac{l_L}{l_A}\right).$$

B.3. Algorithm $\mathcal{C}$ $\quad (f(x) \equiv 0, \quad n = 3, \quad p = (c_{0.5}\sigma(\theta)/\epsilon)^2)$

$$ET_1(\mathcal{C}) = \tau\left((2k+\gamma_A+q_A)l_A) + (\gamma_L+1)l_L\right) c \ln\epsilon \left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$ET_{pipe}(\mathcal{C}) = \tau\left(s+2k+\gamma_A+q_A+l_A-1+(\gamma_L+1)l_L\right) c \ln\epsilon \left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$ET_p(\mathcal{C}) = \tau\left((k+\gamma_A+q_A)l_A + (\gamma_L+1)l_L\right) c \ln\epsilon,$$

$$ET_{6p}(\mathcal{C}) = \tau\left((2k+\gamma_A+q_A)l_A + (\gamma_L+1)l_L\right) c \ln\epsilon,$$

$$S_{pipe}(\mathcal{C}) = \left(1 + \frac{2k+\gamma_A+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right) \Big/ \left(1 + \frac{s+2k+\gamma_A+q_A+l_A-1}{\gamma_L+1}\frac{1}{l_L}\right),$$

$$S_p(\mathcal{C}) = p,$$

$$S_{6p}(\mathcal{C}) = p\left(1 + \frac{2k+\gamma_L+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right) \Big/ \left(1 + \frac{k+\gamma_A+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right),$$

$$E_p(\mathcal{C}) = 1,$$

$$E_{6p}(\mathcal{C}) = \frac{1}{6}\left(1 + \frac{2k+\gamma_L+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right) \Big/ \left(1 + \frac{k+\gamma_A+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right).$$

Appendix C

Grid Monte Carlo Algorithm for Parix-C

```
#define p 15
struct task {double x0,y0,h;
             int n;
             unsigned int iy;} tsk;
LinkCB_t *(to[3]);
int pnr;

void buildBinTree(void)
{ /* establish complete binary tree */
  pnr = GET_ROOT() -> ProcRoot -> MyProcID;
  if (pnr != 0)
    to[0] = ConnectLink((pnr-1)/2, 0,NULL);
  if (pnr < p/2) {
    to[1] = ConnectLink(2*pnr+1, 0, NULL);
    to[2] = ConnectLink(2*(pnr+1),0,NULL);}
}

void broadcast(struct task (*tsk))
{ if (pnr != 0)
    RecvLink(to[0], (char *) tsk,
             sizeof(struct task));
  if (pnr < p/2){
    SendLink(to[1], (char *) tsk,
             sizeof(struct task));
    SendLink(to[2], (char *) tsk,
             sizeof(struct task));}
}

void collect(double (*result))
{ double r1, r2;
  if (pnr < p/2){
    RecvLink(to[1], (char *) &r1,
             sizeof(double));
    RecvLink(to[2], (char *) &r2,
             sizeof(double));
    (*result) += r1 + r2;}
  if (pnr != 0)
    SendLink(to[0], (char *) result,
             sizeof(double));
}
```

```
int main(int args, char * arg[])
{ double x0,y0,h,x,y,s;
  int n,i,running;
  unsigned int time;

  buildBinTree();
  printf("%d nBT\n",pnr);

  if (pnr == 0){
    ... input tsk.x0,y0,h,n...
    ... and initial random number tsk.iy ...}

  /* broadcast task */
  broadcast(&tsk);
  x0 = tsk.x0; y0 = tsk.y0;
  h = tsk.h; n = tsk.n;

  /* compute n/p trajectories */
  srand(tsk.iy+pnr);    s=0;
  for(i=1;i<=n/p;i++){
    x=x0;  y=y0;
    do
      switch ((int) (((double)rand())/
                      RAND_MAX*4)){
        case 0: y=y+h; running=y<1; break;
        case 1: x=x+h; running=x<1; break;
        case 2: y=y-h; running=y>0; break;
        default:  x=x-h; running=x>0;}
    while (running);
    s += x+y;}

  /* collect results */
  collect(&s);
  if (pnr == 0)
    printf("u(x0,y0)=%f\n", s/n);
}
```

Bibliography

[Ak56] H. Akaike, *Monte Carlo method applied to the solution of simultaneous linear equations*, **Ann. Inst. Statist. Math. Tokyo, Vol. 7** (1956), pp. 107–113.

[Ak56a] H. Akaike, *On optimum character of von Neumann's Monte Carlo model*, **Ann. Inst. Statist. Math. Tokyo, Vol. 7** (1956), pp. 183–193.

[Ba59] N.S. Bachvalov, *On the approximate computation of multiple integrals*, **Vestnik Moscow State University, Ser. Mat., Mech., Vol. 4** (1959), pp. 3–18.

[Ba61] N.S. Bachvalov, *Average Estimation of the Remainder Therm of Quadrature Formulas*, **USSR Comput. Math. and Math. Phys., Vol. 1(1)** (1961), pp. 64–77.

[Ba64] N.S. Bachvalov, *On the optimal estimations of convergence of the quadrature processes and integration methods*, **Numerical methods for solving differential and integral equations** Nauka, Moscow, 1964, pp. 5–63.

[BT89] Bertsekas D., P. Tsitsiklis , *Parallel and Distributed Computation* , **Prentice Hall**, 1989.

[Bi82] A.V. Bitzadze, *Equations of the Mathematical Physics*, **Nauka**, Moscow, 1982.

[BF80] P. Bradley, B. Fox, *Implementing Sobol's Quasi Random Sequence Generator*, **ACM Trans. Math. Software, Vol. 14(1)**, pp. 88–100.

[CR90] Chauvin B., Rounault A., *A Stochastic Simulation for Solving Scalar Reaction-Diffusion Equations,* **Adv. Appl. Prob.,** Vol. 22, 1990, pp. 80–100.

[Cu49] J.H. Curtiss, *Sampling methods applied to differential and difference equations*, **Proc. seminar on Sci. Comput. IBM**, New York, 1949.

[Cu54] Curtiss J.H., *Monte Carlo methods for the iteration of linear operators.* **J. Math. Phys., Vol. 32**, No 4 (1954), pp. 209 –232.

[Cu56] Curtiss J.H. *A theoretical comparison of the efficiencies of two classical methods and a Monte Carlo method for computing one component of the solution of a set of linear algebraic equations.*, **Proc. Symposium on Monte Carlo Methods** , **John Wiley and Sons**, 1956, pp. 191–233.

[Cut51] R.E. Cutkosky, *A Monte Carlo method for solving a class of integral equations*, **J. research NBS, 47** (1951), No. 2, pp. 113–115.

[Da93] J. Darlington, *Parallel Programming Using Skeleton Functions*, Proceedings of **PARLE '93**, LNCS 694 (1993), pp. 146–160.

[DB85] De Boor, *Practical guide for splines*, **Radio i svjaz**, Moscow, 1985.

[D88] Delosme J.M., *A parallel algorithm for the algebraic path problem*, **Parallel and Distributed Algorithms**, (Cosnard et all. Eds.) Bonas, France, 1988, pp. 67–78.

[DS87] Doroshkevich, A, I. Sobol, *Monte Carlo evaluation of integrals encountered in nonlinear theory of gravitational instability*, **USSR Comput. Math. and Math. Phys., Vol. 27(10)**(1987), pp. 1577–1580.

[Di85] Dimov I.T., *"A Random walk on the mesh squares" - method for the Poisson equation*, Numerical methods and applications, Sofia, 1985, pp. 258-264.

[Di86] Dimov I.T., *Minimization of the probable error for Monte Carlo methods.* **Application of Mathematics in Technology. Differential equations and applications.** Varna 1986, Sofia 1987, pp. 161–164.

[Di89] Dimov I.T., *Efficiency estimator for the Monte Carlo algorithms.* **Numerical Methods and Applications**, Proc. **II Intern. Conf. on Numerical Methods and Appl.** , Sofia, Publ. House of the Bulg. Acad. Sci., Sofia, 1989, pp. 111–115.

[DT89] Dimov I.T., Tonev O.I., *Monte Carlo numerical methods with overconvergent probable error.* **Numerical Methods and Applications**, Proc. **II Intern. Conf. on Numerical Methods and Appl.** , Sofia, Publ. House of the Bulg. Acad. Sci., Sofia, 1989, pp. 116–120.

[DT90] I. Dimov, O. Tonev, *Performance Analysis of Monte Carlo Algorithms for Some Models of Computer Architectures*, International Youth Workshop on **Monte Carlo Methods and Parallel Algorithms** - Primorsko (Bl. Sendov, I. Dimov, Eds.), **World Scientific**, Singapore, 1990, pp. 91–95.

[Di91] Dimov I.T., *Minimization of the probable error for some Monte Carlo methods* - Proc. of the Summer School on **Mathematical Modelling and Scientific Computations**, 23-28. 09. 1990, Albena, Bulgaria, Sofia, Publ. House of the Bulg. Acad. Sci., 1991, pp. 159–170.

[DT92] Dimov I., Tonev O., *Criteria Estimators for Speed-up and Efficiency of Some Parallel Monte Carlo Algorithms for Different Computer Architectures*, Proc. **WP & DP'91** (Sofia, April 16-19) , **North-Holland Publ. Co.**, Amsterdam, 1992, pp. 243–262.

[Di93] I. Dimov , *Efficient and Overconvergent Monte Carlo Methods*, **Adv. in Parallel Algorithms, IOS Press**, Amsterdam, 1993, pp. 100–111.

[Di93a] I. Dimov, *A Monte Carlo Method for Air Pollution Problem*, **Scientific Computation and Mathematical Modelling**, (S. Markov, Edt.), DATECS Publishing, Sofia, 1993, pp. 59–62.

[DG93] I. Dimov, T. Gurov, *Parallel Monte Carlo Algorithms for Calculation Integrals*, **Proc. WP&DP**, (K. Boyanov, Edt.), 1993, Sofia, pp. 426–434.

[DT93] I. Dimov, O. Tonev, *Random walk on distant mesh points Monte Carlo methods*, **J. of Statistical Physics, Vol. 70**(5/6), 1993, pp. 1333–1342.

[DT93a] I. Dimov, O. Tonev, *Monte Carlo algorithms: performance analysis for some computer architectures.* **J. of Computational and Applied Mathematics, Vol. 48** (1993), pp. 253–277.

[DK94] I. Dimov, A. Karaivanova, *Overconvergent Monte Carlo Methods for Density-function Modelling Using B-Splines*, **Adv. in Numerical Methods and Appl.**, **World Scientific**, 1994, pp. 85–93.

[DK94a] I. Dimov, A. Karaivanova, *Monte Carlo Parallel Algorithms*, Proc. III Intern. Conf. on **Applications of Supercomputers in Engineering - ASE/93, Comp. Mech. Publ., Elsevier Appl. Sci.**, London, New York, 1993, pp. 479–495.

[DKKS96] I. Dimov, A. Karaivanova, H. Kuchen, H. Stoltze, *Monte Carlo Algorithms for Elliptic Differential Equations. Data Parallel Functional Approach*, **Journal of Parallel Algorithms and Applications, Vol. 9** (1996), pp. 39–65.

[Di94] I Dimov, *Efficient and Overconvergent Monte Carlo Methods. Parallel algorithms.*, **Advances in Parallel Algorithms**, (I. Dimov, O. Tonev, Eds.), Amsterdam, **IOS Press** (1994), pp. 100–111.

[DK96a] Ivan T. Dimov, Aneta N. Karaivanova, *A Fast Monte Karlo Method for Matrix Computations*, in **Iterative Methods in Linear Algebra II, IMACS Series in Computational and Applied Mathematics** (S. Margenov and P.S. Vassilevski Eds.), 1996, pp. 204–213.

[DK96] I.T.Dimov, A.N.Karaivanova, *Iterative Monte Carlo algorithms for linear algebra problems*, First Workshop on Numerical Analysis and Applications, Rousse, Bulgaria, June 24-27, 1996, in : **Numerical Analysis and Its Applications, Springer Lecture Notes in Computer Science, ser. 1196**, pp. 150–160.

[DKY96] I. Dimov, A. Karaivanova and P. Yordanova, *Monte Carlo Algorithms for calculating eigenvalues*, Second International Conference on Monte Carlo and Quasi-Monte Carlo methods in scientific computing, University of Salzburg, 8-12 July, 1996, (to appear in: **Proceedings of MC & QMC 96**, *Springer Notes in Statistics* (H. Niederreiter, P. Hellekalek, G. Larcher and P. Zinterhof, Eds)), 1996.

[DJV96] I. Dimov, U. Jaekel, H. Vereecken, *A Numerical Approach for Determination of Sources in Transport Equations*, **J. Computers and Mathematics with Applications, Vol. 32, No. 5** (1996), pp. 31–42.

[Di88] Dittrich P., *A Stochastic Model of a Chemical Reaction with Diffusion,* **Probab. Th. Rel. Fields, vol. 79** (1988), pp. 115–128.

[Du56] V. Dupach, *Stochasticke pocetni metody,* **Cas. Pro. Pest. Mat., 81**, No 1 (1956), pp. 55–68.

[EA89] El-Amawy A., *A Systolic Architecture for Fast Dense Matrix Inversion*, **IEEE Transactions on Computers, Vol.C-38**, No3 (1989), pp. 449–455.

[Er67] Ermakov S.M., *On the admissibleness of Monte Carlo procedures*, **Dokl. Acad. Nauk SSSR, Vol. 172(2)**, pp. 262– 264.

[Er75] Ermakov S.M., *Monte Carlo Methods and Mixed Problems*, **Nauka**, Moscow, 1985.

[Er84] Ermakov S.M., *On summation of series connected with integral equation.* **Vestnik Leningrad Univ. Math. Vol. 16** (1984) pp. 57–63.

[EM82] S.M. Ermakov, G.A. Mikhailov, *Statistical Modeling*, **Nauka**, Moscow, 1982.

[ENS84] S.M.Ermakov, V.V.Nekrutkin, A.S.Sipin, *Random processes for solving classical equations of mathematical physics*, **Nauka**, Moscow, 1984.

[EZ60] S.M. Ermakov, V.G. Zolotukhin, *Polynomial approximations in the Monte Carlo method*, in Russian: **Teoria Veroyatn. i Yeye Primen., 4** (1960), No. 4, pp. 473–476.

[Fr90] W. Frensley, **Rev. Mod. Phys. Vol. 62**, 3 (1990).

[GWKD89] M. W. Gery, G. Z. Whitten, J. P. Killus, M. C. Dodge, *A photochemical kinetics mechanism for urban and regional modeling,* **J. Geophys. Res., Vol. 94** (1989), pp. 12925–12956.

[Go91] S.K. Godunov, *Spectral portraits of matrices and criteria of spectrum dichotomy,* **International symposium on computer arithmetic and scientific computation** (J. Herzberger and L. Atanasova, eds.), Oldenburg, Germany, North-Holland (1991).

[GK58] G. Goertzel, M.H. Kalos, *Monte Carlo methods in transport problems*, **Progress in nuclear energy, Ser. 1, Phys. and Math., 2** (1958), pp. 315–369.

[GV83] G. H. Golub, C.F. Van Loon, *Matrix computations*, **The Johns Hopkins Univ. Press**, Baltimore, 1983.

[Gu94] Gurov T., *Monte Carlo Methods for Nonlinear Equations.* Advances in Num. Methods and Appl., **World Scientific**, 1994, pp. 127–135

[Ha66] S.Haber, *A modified Monte Carlo quadrature*, **Math. of Comput., 20**, No 95 (1966), pp. 361–368 ; **Vol. 21**, No 99 (1967), pp. 388–397.

[HH57] J.H. Halton, D.C. Handscome, *A method for increasing the efficiency of Monte Carlo integrations*, **J. Assoc. comput. machinery, Vol. 4** (1957), No. 3, pp. 329–340.

[HH64] J.M. Hammersley, D.C. Handscomb, *Monte Carlo methods*, **John Wiley & Sons, inc.**, New York, London, Sydney, Methuen, 1964.

[HK94] H.Haug, S.W.Koch, *Quantum Theory of the Optical and Electronic Properties of Semiconductors*, **World Scientific**, Singapore, 1994 (3rd ed.).

[HPFF93] High Performance Fortran Forum, *High Performance Fortran Language Specification*, **Scientific Programming, Vol. 2(1)**, 1993.

[HKL92] G. Hogen, A. Kindler, R. Loogen, *Automatic Parallelization of Lazy Functional Programs*, Proceedings of **ESOP'92**, LNCS 582, pp. 254–268, 1992.

[HPW92] P. Hudak, S. Peyton Jones, P. Wadler (Eds.), *Report on the Programming Language Haskell*, **A Non-Strict Purely Functional Language**, SIGPLAN Notices 27(5), 1992.

[Hu91] P. Hudak, *Para-functional Programming in Haskell*, (B.K. Szymanski Edr.) **Parallel Functional Languages and Compilers**, Addison-Wesley, 1991.

[Hu93] P. Hudak, *Mutable Abstract Datatypes or How to Have Your State and Munge It Too*, Yale Research Report YALEU/DCS/RR-914, 1993.

[HB85] P. Hudak, A. Bloss, *The Aggregate Update Problem in Functional Programming Systems*, ACM Symp. on **Principles of Programming Languages** (1985), pp. 300–314.

[JL89] C.Jacoboni, P. Lugli, *The Monte Carlo method for semiconductor device simulation, Springer-Verlag*, 1989.

[JPR88] C.Jacoboni, P.Poli, L.Rota, **Solid State Electronics, Vol. 31** (1988), p. 523.

[JR83] C. Jacoboni, L. Reggiani, *Rev. Mod. Phys. 55*, 1983, p. 645.

[Ka50] H. Kahn, Random sampling (Monte Carlo) techniques in neutron attenuation problems, **Nucleonics ,Vol. 6** No 5 (1950), pp. 27–33 ; **Vol. 6**, No 6 (1950), pp. 60–65.

[Ka59] M.H. Kalos, *Importance sampling in Monte Carlo calculations of thick shield penetration*, **Nuclear Sci. and Eng., 2** (1959), No. 1, pp. 34–35.

[KA77] L.V. Kantorovich, G.P. Akilov, *Functional analysis*, **Nauka**, Moscow, 1977.

[KLB91] H. Kingdon, D. Lester, G.L. Burn, *The HDG-machine: a highly distributed graph-reducer for a transputer network*, **Computer Journal, Vol. 34(4)**(1991), pp. 290–301.

[KV78] Kloek, T., van Dijk, *Bayesian estimates of equation system parameter. An application of integration by Monte Carlo*, **Econometrica, Vol. 46** (1978), pp. 1–19.

[Ko57] N.M. Korobov, *Approximate computation of multiple integrals with the aid of methods of the theory of numbers*, in Russian: **Dokl. Acad. Nauk SSSR, Vol. 115** (1957), No. 6, pp. 1062– 1065.

[Ko59] N.M. Korobov, *On the approximate computation of multiple integrals*, in Russian: **Vestnik Moskow. State University, Vol. 4** (1959).

[KPS94] H. Kuchen, R. Plasmeijer, H. Stoltze, *Efficient Distributed Memory Implementation of a Data Parallel Functional Language*, **PARLE'94**, LNCS 817(1994), Springer Verlag, pp. 464–477.

[Ku93] H. Kuchen, *Distributed Memory Implementation of a Data Parallel Functional Language*, PMG report # 76, Chalmers Univ. of Technology, Sweden, 1993.

[Ku94] H. Kuchen, *Datenparallele Programmierung von MIMD-Rechnern mit verteiltem Speicher*, RWTH Aachen, 1994.

[KR92] T.Kuhn, F.Rossi, **Physical Review B, Vol. 46**, 12 (1992).

[KL78] Kung H.T., C.E.Leiserson, *Systolic Arrays for VLSI*, **Sparse Matrix Proceeding 1978, SIAM**, 1979, pp. 256–282.

[Ku84] Kung S.Y., *On Supercomputing with systolic wavefront array processors* , **IEEE Proc. Vol. 72**, pp. 867–884, 1984.

[KLL87] Kung S.Y., S.C. Lo, P.S. Lewis, *Optimal Systolic Design for the Transitive Closure and shortest path problems*, **IEEE Transactions on Computers**.

[Ku66] *J. Phys. Soc. Japan, Supl., 24*, 1966, p. 424.

[LP71] P. Lebwohl, P. Price, *Appl. Phys. Lett.*, **Vol 14**, 1971, p. 530.

[LMRS88] Y. Levitan, N. Markovich, S. Rozin, I. Sobol, *On Quasi-random Sequences for Numerical Computations*, **USSR Comput. math. and Math. Phys., Vol. 28(5)**, pp. 755–759.

[Li89] Lichoded, H., *Processing of Monte Carlo estimations for Continual Integrals*, Moscow, **Nauka**, 1989.

[LKID89] R. Loogen, H. Kuchen, K. Indermark, W. Damm, *Distributed Implementation of Programmed Graph Reduction*, **PARLE '89**, LNCS 365(1989), pp. 136–157.

[MP84] O.A. Mahotkin, M.I. Pirimkulov, *Application of splines for some problems of statistical modelling*, **Theory and Practice of Statistical Modelling** (Mikhailov, Edr.), Novosibirsk, Computing Center(1984), pp. 43–53.

[Ma82] G.I. Marchuk, *Mathematical modelling in the problem of environment*, **Nauka**, Moscow, 1982.

[Ma85] G. I. Marchuk, *Mathematical modeling for the problem of the environment*, **Studies in Mathematics and Applications, No. 16**, North-Holland, Amsterdam, 1985.

[MGS84] G. J. McRae, W. R. Goodin, J. H. Seinfeld, *Numerical solution of the atmospheric diffusion equations for chemically reacting flows*, **J. Computational Physics, Vol. 45,** 1984, pp. 1–42.

[M92] Megson G.M., *A Fast Faddeev Array*, **IEEE Transactions on Computers, Vol. 41**, No 12, December 1992, pp. 1594–1600.

[MAD94] G. Megson, V. Aleksandrov, I. Dimov, *Systolic Matrix Inversion Using a Monte Carlo Method*, **J. of Parallel Algorithms and Appl., Vol. 3, No 3/4** (1994), pp. 311–330.

[MAD94a] G. Megson, V. Aleksandrov, I. Dimov, *Systolic Matrix Inversion by Monte Carlo Method*, **Proc. 14-rd IMACS World Congress on Computational and Appl. Math.**, July 11–15, 1994, **Atlanta, USA, Vol. 3.**, pp. 1371–1373.

[MAD94b] G.M. Megson, V.N. Aleksandrov, I.T. Dimov, *A Fixed Sized Regular Array for Matrix Inversion by Monte Carlo Method*, **Adv. in Numerical Methods and Appl.**, **World Scientific**, 1994, pp. 255–264.

[MU49] N. Metropolis, S. Ulam, *The Monte Carlo Method*, **J. of Amer. Statistical Assoc., 44**, (1949), No. 247, pp. 335–341.

[Mi87] G.A. Mikhailov, *Optimization of Wieght Monte Carlo methods*, **Nauka**, Moscow, 1987.

[Mi70] G.A. Mikhailov, *A new Monte Carlo algorithm for estimating the maximum eigenvalue of an integral operator*, **Docl. Acad. Nauk SSSR, 191**, No 5 (1970), pp. 993–996.

[Mi83] Mikhailov V.P., *Partial differential equations,* Moskow, **Nauka**, 1983.

[Mi55] C. Miranda, *Equasioni alle dirivate parziali di tipo ellittico*, **Springer-Verlag**, Berlin, 1955.

[Mo57] K.W. Morton, *A generalization of the antithetic variate technique for evaluating integrals*, **J. Math. Phys., 36** (1957), No. 3, pp. 289–293.

[Mu56] , *Some continuous Monte Carlo methods for the Dirichlet problem*, **Ann. Math. Statistics, 27** (1956), No. 3, pp. 569–589.

[NDRJ96] M. Nedjalkov, I. Dimov, F. Rossi, C. Jacoboni, **Journal of Mathematical and Computer Modeling, Vol. 23**, N 8/9, (1996).

[NV89] M.Nedjalkov, P.Vitanov, **Solid State Electronics, Vol. 32** (1989), pp. 10.

[NV91] M.Nedjalkov, P.Vitanov , **COMPEL, Vol. 10**, N 4 (1991).

[Ni87] H. Niederreiter, *Point Sets and Sequences with Small Discrepancy*, **Monatsh. Math., Vol. 104** (1987), pp. 273–337.

[Ni92] H. Niederreiter, *Random number generation and Quasi-Monte Carlo Methods.* Number **63** in **CBMS-NSF Series in Applied Mathematics. SIAM**, Philadelphia, 1992.

[Ni88] S.M. Nikolski, *Quadrature formulas* **Nauka**, Moscow, 1988.

[Pa92] Parsytec GmbH, *PARIX Documentation*, Aachen, 1992.

[Pe91] Perihelion Software Ltd. *The Helios Parallel Operating System*, **Prentice Hall**, 1991.

[Pl94] B. Plateau, *APACHE: Algorithmique Parallele et pArtagede CHargE*, Raport APACHE, Institut IMAG, Grenoble, ♯1, 1994, pp. 28.

[Qu87] M.J. Quinn, *Designing Efficient Algorithms for Parallel Computers*, **McGraw-Hill**, 1987.

[RJN94] F. Rossi, C. Jacoboni, M. Nedjalkov, **Semicond. Sci. Technol. 9**, pp. 934 (1994).

[RPJ92] F. Rossi, P. Poli, C. Jacoboni, **Semicond. Sci. Technol. Vol. 7**, pp. 1017 (1992).

[RJP89] L. Rota, C. Jacoboni, P. Poli, **Solid State Electronics, Vol. 31** (1989), p. 523.

[RJP89a] L. Rota, C. Jacoboni, P. Poli, **Solid State Electronics, Vol. 32** (1989), p. 1417.

[Sa89] K. Sabelfeld, *Algorithms of the Method Monte Carlo for Solving Boundary Value Problems*, **Nauka**, Moscow, 1989.

[SCA93] A.V.S. Sastry, W. Clinger, Z. Ariola, *Order-of-evaluation Analysis for Destructive Updates in Strict Functional Languages with Flat Aggregates*, **FPCA'93**, ACM, pp. 266-275, 1993.

[SP88] Bl. Sendov, V. Popov, *Averaged modui of smoothness*, **Mir**, Moscow, 1988.

[SAK94] Bl. Sendov, A. Andreev, N. Kjurkchiev, Numerical Solution of Polynomial Equations, *Handbook of Numerical Analysis (General Editors: P.G. Ciarlet and J. L. Lions), Vol. 3)*, Solution of Equations in R^n (Part 2), (North-Holland, Amsterdam, N.Y.,1994).

[SKM94] J. Schilp, T. Kuhn, G. Mahler, **Physical Review B, Vol. 50**, N8, (1994).

[Sh88] J. Shaw, *Aspects of Numerical Integration and Summarization*, **Bayesian Statistics, Vol. 3** (1988), pp. 411–428.

[Sh64] Shreider Yu.A., *Method of Statistical Testing. Monte Carlo method.* **Elsevierm Publishing Co.,** 335 Jan Van Galenstraat, P.O. Box 211, Amsterdam (Netherlands), 1964.

[Sh66] Shreider Yu.A., *The Monte Carlo method. Method of Statistical Testing.* **Pergamon Press Ltd.,** Ozford, London, Edinburgh, New York, Paris, Frankfurt, 1966.

[SZ97] S. Skelboe, Z. Zlatev, *Exploiting the natural partitioning in the numerical solution of ODE systems arising in atmospheric chemistry*, In: **"Numerical Analysis and Its Applications"** (L. Vulkov, J. Wasniewski and P. Yalamov, Eds.), pp. 458–465, Springer, Berlin, 1997.

[SB93] J. Smetsers, E. Barendsen, *Conventional and uniqueness typing in graph rewrite systems*, 13th Conf. on **Foundations of Software Technology and TCS**, LNCS, 1993.

[So73] I.M. Sobol, *Monte Carlo numerical methods*, **Nauka**, Moscow, 1973.

[So79] I.M. Sobol, *On the Systematic Search in a Hypercube*, **SIAM J. Numerical Analysis, Vol. 16** (1979), pp. 790–793.

[So89] I.M. Sobol, *On Quadratic Formulas for Functions of Several Variables Satisfying a General Lipschitz Condition*, **USSR Comput. Math. and Math. Phys., Vol. 29(6)** (1989), pp. 936 – 941.

[So91] I.M. Sobol, *Quasi - Monte Carlo Methods*, International Youth Workshop on **Monte Carlo Methods and Parallel Algorithms** - Primorsko (Bl. Sendov, I. Dimov, Eds.), **World Scientific**, Singapore, 1990, pp. 75–81.

[St83] Stewart, L., *Bayesian analysis using Monte Carlo integration - a powerful methodology for handling some difficult problems*, **Statistician, Vol. 32** (1983), pp. 195–200.

[St85] Stewart, L., *Multiparameter Bayesian inference using Monte Carlo integration - some techniques for bivariate analysis*, in: J.M.Bernardo, M.H. de Groot, D.V. Lindley and A.F. Smith, eds, **Bayesian Statistics, Vol. 2** (North-Holland, Amsterdam).

[St87] Stewart, L., *Hieralhical Bayesian analysis using Monte Carlo integration computing posterior distributions when there are many possible models*, **Statistician, Vol. 36** (1987), pp. 211–219.

[SD86] Stewart, L., W. Davis, *Bayesian posterior distribution over sets of possible models with inferences computed by Monte Carlo integration*, **Statistician, Vol. 35** (1986), pp. 175–182.

[St94] H. Stoltze, *Implementierung einer parallelen funktionalen Sprache mit algorithmischen Skeletten zur Lösung mathematisch-technischer Probleme*, Dissertation, RWTH Aachen, 1994.

[Ta92] M.D. Takev, On Probable Error of the Monte Carlo Method for Numerical Integration, *Mathematica Balkanica (New Series), Vol. 6* (1992), pp. 231–235.

[Ta83] W. Tatarskii, **Sov. Journal Uspehi Phys. Sci. Vol. 139**, 4 (1983).

[TN90] Tonev O., Nicolov P., Sabev V., Dimov I., *Realization of Monte Carlo algorithms on transputer systems*, International Youth Workshop on **Monte Carlo methods and Parallel Algorithms** - Primorsko, 1989 (Bl.Sendov, I.Dimov, Eds.), **World Scientific**, Singapore, 1990, pp. 91–95.

[TS77] Tikchonov A.N., Samarskii A.A., *Equations of the Mathematical Physics.*, Moskow, **Nauka**, 1977.

[Tr91] L.N. Trefethen, *Pseudospectra of matrices*, **14th Dundee Biennal Conference on Numerical Analysis** (D.F. Griffiths and G.A. Watson, Eds.) (1991).

[VHL87] Van Dijk, H., J. Hop, A. Louter, *An algorithm for the computation of posterior moments and densities using simple importance sampling*, **Statistician, Vol. 37** (1987), pp. 83–90.

[VK83] Van Dijk, H., T. Kloek, *Monte Carlo analysis of skew posterior distributions: An illustrative econometric example*, **Statistician, Vol. 32** (1983), pp. 216–223.

[VK85] Van Dijk, H., T. Kloek, *Experiments with some alternatives for simple importance sampling in Monte Carlo integration*, in: J. Bernardo, M.H. de Groot, D.V. Lindley and A.F. Smith, eds, **Bayesian Statistics, Vol. 2** (North-Holland, Amsterdam).

[VNJRA94] P. Vitanov, M. Nedjalkov, C. Jacoboni, F. Rossi, A. Abramo, in: *Advances in Parallel Algorithms*, **IOS Press**, Amsterdam, p. 117 (1994).

[VN91] P. Vitanov, M. Nedjalkov, *COMPEL*, **Vol. 10**, No 4, 1991.

[Vl56] V.S.Vladimirov, *On the application of the Monte Carlo method to the finding of the least eigenvalue, and the corresponding eigenfunction*, of a linear integral equation, in Russian: **Teoriya Veroyatn i Yeye Primenenie, 1**, No 1 (1956), pp. 113–130.

[Vl60] V.S. Vladimirov, *On the approximate computation of Wiener integrals*, in Russian: **Uspechi Mathem. Nauk, 15**, No. 4 (1960), pp. 129–135.

[VS58] V.S. Vladimirov, I.M. Sobol *Computation of the least eigenvalue of Peiers' equation by the Monte Carlo method*, in Russian: **Vichislit. Mathematika, 3** (1958), pp. 130–137.

[Wa90] P. Wadler, *Comprehending monads*, Symp. on **LISP and Functional Programming**, pp. 61–78, ACM, 1990.

[Wa87] W.Wagner, *Unbased Monte Carlo Evaluation of Certain Functional Integrals*, **J. Comput. Phys., Vol. 71**, No 1 (1987), pp. 21–33.

[Wa51] W. Wasow, *Random walks and the eigenvalues of elliptic difference equations*, **J. Research NBS, Vol. 46** (1951), pp. 65–73.

[Wa51a] W. Wasow, *On the mean duration of random walks*, **J. Research NBS, 46** (1951), pp. 462–472.

[Wa51b] W. Wasow, *On the duration of random walks*, **Ann. Math. Statistics, Vol. 22** (1951), pp. 199–216.

[Wa56] W. Wasow, *A note on the inversion of matrices by random walks*, **MTAC, 5** (1956), No. 38, pp. 78–81.

[W68] J.R. Westlake, *A Handbook of Numerical matrix Inversion and Solution of Linear Equations*, **John Wiley & Sons, inc.**, New York, London, Sydney, 1968.

[We16] H. Weyl, *Ueber die Gleichverteilung von Zahlen mod Eins*, **Math. Ann., Vol. 77 (3)**, pp. 313–352.

[ZKM80] Zavjalov, Y., Kvasov, B., Miroshnichenko, V., *Methods of spline-functions,* **Nauka**, Moscow, 1980.

[ZCM91] Z. Zlatev, J.Christensen, J, Moth, J. Wasniewski, *Vectorizing codes for studying long-range transport of air pollutants*, **Math. Comput. Modelling, Vol. 15**, No 8, (1991) pp. 37–48.

[Zl95] Z. Zlatev, *Computer treatment of large air pollution models*, **Kluwer Academic Publishers**, Dordrecht-Boston-London, 1995.

[ZDG96] Z. Zlatev, I. Dimov, K. Georgiev, **Three-dimensional version of the Danish Eulerian Model**, **Zeitschrift für Angewandte Mathematik und Mechanik**, **Vol. 76,** (1996) S4, pp. 473–476.

SYMBOL TABLE

- $x = (x_{(1)}, x_{(2)}, \ldots, x_{(d)}) \in \Omega \subset I\!\!R^d$ - point in $I\!\!R^d$ (d-dimensional vector)
- $I\!\!R^d$ - d-dimensional Euclidean space
- Ω - domains in the Euclidean space
- $\partial\Omega$ - boundary of the domain Ω
- $dist(x, D)$ - distance between the point x and the set D
- $\mu = (\omega, \omega')$ - cos of angle between directions ω and ω'
- $t \in [0, T]$ - time
- $\delta(x)$ - Dirac's function
- $\mathbf{X}$ - a Banach space of functions
- $u^*, \mathbf{X}^*$ - conjugate function, dual Banach space
- $C(\Omega)$ - space of functions continuous on Ω
- $C^{(k)}(\Omega)$ - space of functions u for which $u^{(k)} \in C(\Omega)$
- $H^\alpha(M, \Omega)$ - space of functions for which $|f(x) - f(x')| \leq M|x - x'|^\alpha$
- $\| f \|_{\mathbf{L}_q} = (\int_\Omega f^q(x)p(x)dx)^{1/q}$ - $\mathbf{L}_q$-norm
- $\mathbf{W}^r(M; \Omega)$ - a class of functions $f(x)$, continuous on Ω with partially continuous rth derivatives, such that
$$|D^r f(x)| \leq M,$$
where
$$D^r = D_1^{r_1} \ldots D_d^{r_d}$$
is the rth derivative, $r = (r_1, r_2, \ldots, r_d), |r| = r_1 + r_2 + \ldots + r_d$, and $D_i = \frac{\partial}{\partial x_{(i)}}$
- I - value of the integral
- $J(u)$ - linear functional
- $(h, u) = \int_\Omega h(x)u(x)dx$ - inner product of functions $h(x)$ and $u(x)$
- $Lu(x) = \int_\Omega l(x, x')u(x')dx'$ integral transformation (L - integral operator)
- Δ - the Laplace operator

- $A \in I\!\!R^{m\times m}$ or $L \in I\!\!R^{m\times m}$ - $m \times m$ matrices

- $Au = f$ - one linear system of equations
- a_{ij} or l_{ij} - element in the ith row and jth column of the matrix A or L
- x^T - transposed vector
- $(h, u) = \sum_{i=1}^{m} h_i u_i$ - inner product of vectors $h = (h_1, h_2, \ldots, h_m)^T$ and $u = (u_1, u_2, \ldots, u_m)^T$
- L^k - kth iterate of the matrix L
- δ_j^i - Kronneker's symbol

- $\xi \in \Omega$ - random point in Ω
- $\theta(\xi)$ - random variable (r.v.)
- $E(\theta)$ - mathematical expectation of r.v. θ
- $D(\theta)$ - variance of r.v. θ
- $\sigma(\theta)$ - standard deviation of r.v. θ
- $Pr\{\varepsilon_k = i\}$ - probability that $\varepsilon_k = i$
- γ - random number (uniformly distributed r.v. in $[0, 1]$ with $E(\gamma) = 1/2$ and $D(\gamma) = 1/12$)
- $\xi_i (i = 1, 2, \ldots, n)$ - realizations (values) of the random point ξ
- $\theta_i (i = 1, 2, \ldots, n)$ - realizations (values) of the random variable θ
- $p(x)$ - density (frequency) function
- $p(x, y)$ - transition density function
- $F(x)$ - distribution function
- $T_i = (\xi_0 \to \xi_1 \to \ldots \to \xi_i)$ - random trajectory
- $\overline{\xi}_n = \frac{1}{n} \sum_{i=1}^{n} \xi_i$ - mean value of n realizations of the r.v. ξ
- r_n - probable error defined as a value r_n for which

$$Pr\left\{|\overline{\xi}_n - J| \leq r_n\right\} = \frac{1}{2} = Pr\left\{|\overline{\xi}_n - J| \geq r_n\right\}$$