MONTE CARLO METHODS FOR APPLIED SCIENTISTS

Ivan T Dimov

World Scientific

MONTE CARLO METHODS FOR APPLIED SCIENTISTS

World Scientific

NEW JERSEY • LONDON • SINGAPORE • BEIJING • SHANGHAI • HONG KONG • TAIPEI • CHENNAI

This page intentionally left blank

MONTE CARLO METHODS FOR APPLIED SCIENTISTS

Ivan T Dimov

University of Reading, Reading, UK &
Bulgarian Academy of Sciences, Sofia, Bulgaria

World Scientific

NEW JERSEY • LONDON • SINGAPORE • BEIJING • SHANGHAI • HONG KONG • TAIPEI • CHENNAI

Published by

World Scientific Publishing Co. Pte. Ltd.

5 Toh Tuck Link, Singapore 596224

USA office: 27 Warren Street, Suite 401-402, Hackensack, NJ 07601

UK office: 57 Shelton Street, Covent Garden, London WC2H 9HE

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library.

MONTE CARLO METHODS FOR APPLIED SCIENTISTS

ISBN-13 978-981-02-2329-8
ISBN-10 981-02-2329-3

Printed in Singapore.

To my wife Milena for her support

This page intentionally left blank

Preface

The primary motivation for new developments in Monte Carlo methods is the fact that they are widely used for those problems for which deterministic algorithms hopelessly break down. An important reason for this is that Monte Carlo methods do not require additional regularity of the problem's initial data. The main problem with the deterministic algorithms is that they normally need some additional approximation procedure, requiring additional regularity. Monte Carlo algorithms do not need such procedures. Nevertheless, if one can exploit the existing smoothness of the input data then normally Monte Carlo methods have a better convergence rate than the deterministic methods. At the same time, dealing with Monte Carlo algorithms one has to accept the fact that the result of the computation can be close to the true value only with a certain probability. Such a setting may not be acceptable if one needs a guaranteed accuracy or strictly reliable results. But in most cases it is reasonable to accept an error estimate with a probability smaller than 1. In fact, we shall see that this is a price paid by Monte Carlo methods to increase their convergence rate. The better convergence rate for Monte Carlo algorithms is reached with a given probability, so the advantage of Monte Carlo algorithms is a matter of definition of the probability error.

The second important motivation is that Monte Carlo is efficient in dealing with large and very large computational problems: multidimensional integration, very large linear algebra problems, integro-differential equations of high dimensions, boundary-value problems for differential equations in domains with complicated boundaries, simulation of turbulent flows, studying of chaotic structures, etc.. At the same time it is important to study applicability and acceleration analysis of Monte Carlo algorithms both theoretically and experimentally. Obviously the performance analysis of al-

gorithms for people dealing with large-scale problems is a very important issue.

The third motivation for new developments of Monte Carlo methods is that they are very efficient when parallel processors or parallel computers are available. The reason for this is because Monte Carlo algorithms are inherently parallel and have minimum dependency. In addition, they are also naturally vectorizable when powerful vector processors are used. At the same time, the problem of parallelization of the Monte Carlo algorithms is not a trivial task because different kinds of parallelization can be used. To find the most efficient parallelization in order to obtain high speed-up of the algorithm is an extremely important practical problem in scientific computing. Another very important issue is the *scalability* of the algorithms. Scalability is a desirable property of algorithms that indicates their ability to handle growing amounts of computational work on systems with increasing number of processing nodes. With the latest developments of distributed and Grid computing during last ten years there is a serious motivation to prepare scalable large-scale Monte Carlo computational models as Grid applications.

One of most surprising findings is the absence of a systematic guide to Monte Carlo methods for applied scientists. This book differs from other existing Monte Carlo books by focusing on performance analysis of the algorithms and demonstrating some existing large scale applications in semiconductor device modeling. The reader will find description of some basic Monte Carlo algorithms as well as numerical results of implementations. Nevertheless, I found it important to give some known fundamental facts about Monte Carlo methods to help readers who want to know a little bit more about the theory. I also decided to include some exercises specially created for students participating in the course of *"Stochastic Methods and Algorithms for Computational Science"* at the University of Reading, UK. My experience working with students studying stochastic methods shows that the exercises chosen help in understanding the nature of Monte Carlo methods.

This book is primary addressed to applied scientists and students from Computer Science, Physics, Biology and Environmental Sciences dealing with Monte Carlo methods.

I. Dimov

Acknowledgements

Most Monte Carlo numerical methods and results discussed in this book have been implemented after many discussions with specialists: from

- the Centre for Advanced Computing and Emerging Technologies (ACET), the University of Reading (Vassil Alexandrov, Simon Branford, and Christian Weihrauch);
- the Institute for Parallel Processing, Bulgarian Academy of Sciences (Todor Gurov, Mihail Nedjalkov, Aneta Karaivanova, Emanouil Atanassov, Rayna Georgieva, Mariya Durchova, Rumyana Papancheva, and Tzvetan Ostromsky);
- the National Environmental Research Institute (Zahari Zlatev, Ruwim Bertkowicz, and Jesper Christensen);
- the University of Mainz (Kurt Binder);
- the University of North Carolina at Chapel Hill (John H. Halton);
- The Florida State University (Michael Mascagni and Nikolai Simonov);
- the Brooklyn College, CIS Department (Paula Whitlock);
- the Weierstrass Institute for Applied Analysis and Stochastics (Karl Sabelfeld and Peter Mathé);
- the Russian Academy of Sciences (Ilya M. Sobol', Gennadiy A. Mikhailov (Siberian Branch));
- IRISA / INRIA – Rennes (Bernard Philippe).

The author should like to thank very much each of them.

All parallel Monte Carlo methods were implemented at ACET, The University of Reading, DCSC (the Danish Center for Scientific Computing), EPCC, The University of Edinburgh, UK, HPC- Europe, IRISA / INRIA-Rennes, and IPP-BAS. The specialists from these institutions helped me to

run the algorithms or some modules of the algorithms on different vector and parallel computer systems. I should like to thank very much all of these specialists for their great help in different runs of the algorithms on high performance computers; especially Christian Weihrauch, Simon Branford, Todor Gurov, Emanouil Atanassov, Rayna Georgieva, Sofiya Ivanovska, Rumyana Papancheva, Jerzy Wasniewski and Zahari Zlatev.

The author should like to thank very much all of these scientists for the helpful discussions related to different issues from the field of large-scale computations.

I had many very helpful discussions with Sean McKee from the University of Strathclyde, Glasgow. These discussions helped me a lot to improve the manuscript of the book. I am also very much obligated to Simon Branford from the University of Reading, UK, who read the manuscript and made a number of suggestions for improvement.

The research on many of the topics discussed in this book was a part of many different projects funded by

- the European Commission (EC),
- the North Atlantic Treaty Organization (NATO),
- the National Science Foundation of Bulgaria,
- the Bulgarian Academy of Sciences,
- the Danish Centre for Scientific Computing (DCSC),
- the Centre for Advanced Computing and Emerging Technologies, the University of Reading.

The authors should like to thank very much all of these institutions for the financial support.

During 2004/2005, 2005/2006 and 2006/2007 academic years the author gave a course of lectures *"Stochastic Methods and Algorithms for Computational Science" (SEMS13)* as a part of *Advanced European MSC Program in Network Centered Computing* at the University of Reading, UK. I had many useful discussions with my students. The exercises given in the book are part of exercises prepared for students. This book is based on the lectures given in Reading.

Contents

This page intentionally left blank

Chapter 1

Introduction

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

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

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

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

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

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

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

$$P = Pr\left\{|\overline{\xi}_N - J| \leq R_N\right\}, \tag{1.1}$$

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

The probable error is the value r_N for which:

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

which completely characterizes the given chain T_k.

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

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

$$T = \alpha_{t_0} \to \alpha_{t_1} \to \alpha_{t_2} \to \ldots \to \alpha_{t_i}, \tag{1.2}$$

where α_{t_q}, $(q = 1, \ldots, i)$ is one of the absorbing states. This determines the value of some function $F(T) = J(u)$, which depends on the sequence (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 α_{t_0} and the transitions are begun anew. A number of N independent runs are performed through the Markov chain starting from the state s_{t_0} to any of the absorbing states. The average

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

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

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

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

Here the following important problems arise:

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

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

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

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

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

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

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

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

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

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

where

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

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

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

Definition 1.7. The $\mathbf{W}_q^r$-semi-norm is defined as:
$\| f \|_{\mathbf{W}_q}^r = \left[\int_\Omega (D^r f(x))^q p(x)dx\right]^{1/q}$.

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

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

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

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

We will be also interested in *computational complexity.*

Definition 1.9. Computational complexity is defined by

$$C_N = tN,$$

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

Suppose there exists different Monte Carlo algorithms to solve a given problem. It is easy to show that the computational effort for the achievement of a preset probable error is proportional to $t\sigma^2(\theta)$, where t is the expectation of the time required to calculate one realization of the random variable θ. Indeed, let a Monte Carlo algorithm A_1 deal with the r.v. θ_1, such that $E\theta_1 = J$ (where J is the exact solution of the problem). Alternatively there is another algorithm A_2 dealing with another r.v. θ_2, such that $E\theta_2 = J$. Using N_1 realizations of θ_1 the first algorithm will produce an approximation to the exact solution J with a probable error $\varepsilon = c\sigma(\theta_1)N_1^{-1/2}$. The second algorithm A_2 can produce an approximation to the solution with the same probable error ε using another number of realizations, namely N_2. So that $\varepsilon = c\sigma(\theta_2)N_2^{-1/2}$. If the time (or number of operations) is t_1 for A_1 and t_2 for A_2, then we have

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

and

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

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

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

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

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

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

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

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

This page intentionally left blank

Chapter 2

Basic Results of Monte Carlo Integration

In this chapter some basic facts about Monte Carlo integration are considered. Almost all facts presented here are well known to scientists dealing with the theory of Monte Carlo methods. Nevertheless, this exposition facilitates the understanding of methods and algorithms discussed in this book. The results presented here are mainly extracted from the basic works on the theory of Monte Carlo methods [Curtiss (1954, 1956); Dupach (1956); Ermakov and Mikhailov (1982); Hammersley and Handscomb (1964); Kahn (1950a,b); Kalos and Whitlock (1986); Lepage (1978); Metropolis and Ulam (1949); Mikhailov (1987); Pitman (1993); Press and Farrar (1990); Rubinstein (1981); Sabelfeld (1989); Sobol (1973, 1994); Spanier and Gelbard (1969)]. Some more recent results, which further develop the basic ideas of Monte Carlo integration, are also presented.

2.1 Convergence and Error Analysis of Monte Carlo Methods

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

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

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

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

$$\overline{\xi}_N = \frac{1}{N}\sum_{i=1}^{N}\xi_i. \tag{2.1}$$

For a sequence of uniformly distributed independent random variables, for which the mathematical expectation exists, the theorem of J. Bernoulli (who proved for the first time the Law of Large Numbers Theorem) [Grimmett and Stirzaker (1992)] is valid. This means that the arithmetic mean of these variables converges to the mathematical expectation:

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

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

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

Thus, when N is sufficiently large

$$\overline{\xi}_N \approx I \tag{2.2}$$

and (2.1) may be used to approximate the value of I whenever $E\xi = I$ exists.

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

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

It is well known that the sequences of the uniformly distributed independent random variables with finite variances satisfy the *central limit theorem* (see [Hammersley and Handscomb (1964)], or [Tijms (2004)]). This means that for arbitrary x_1 and x_2

$$\lim_{N\to\infty} P\left\{x_1 < \frac{1}{\sqrt{ND\xi}}\sum_{i=1}^{N}(\xi_i - I) < x_2\right\} = \frac{1}{\sqrt{2\pi}}\int_{x_1}^{x_2} e^{-\frac{t^2}{2}}dt. \tag{2.3}$$

Let $x_2 = -x_1 = x$. Then from (2.3) it follows that

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

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

When N is sufficiently large

$$P\left\{|\overline{\xi}_N - I| < x\sqrt{\frac{D\xi}{N}}\right\} \approx \Phi(x). \tag{2.4}$$

Formula (2.4) 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 (2.4), it follows that the probability of the inequality

$$|\overline{\xi}_N - I| \le x_\beta\sqrt{\frac{D\xi}{N}} \tag{2.5}$$

is approximately equal to β.

The term on the right-hand side of the inequality (2.5) is exactly the *probability error* R_N defined in Chapter 1 (see Definition 1.2). Indeed, since R_N is the least possible real number, for which $P = Pr\left\{|\overline{\xi}_N - I| \le R_N\right\}$, $R_N = x_\beta\sqrt{\frac{D\xi}{N}}$. Taking into account that the *probable error* is the value r_N for which: $Pr\left\{|\overline{\xi}_N - I| \le r_N\right\} = \frac{1}{2} = Pr\left\{|\overline{\xi}_N - I| \ge r_N\right\}$ one can determine r_N:

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

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

2.2 Integral Evaluation

In this section we consider two basic Monte Carlo algorithms for multidimensional integration: the *Plain Monte Carlo* and the *Geometric Monte Carlo*.

2.2.1 *Plain (Crude) Monte Carlo Algorithm*

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

Let Ω be an arbitrary domain (bounded or unbounded, connected or unconnected) and let $x \in \Omega \subset I\!R^d$ 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 a density function if $\int_\Omega p(x)dx = 1$. Note that every integral $\int_\Omega f(x)dx$, when Ω is a bounded domain, is an integral of the kind (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 ξ be a random point with probability density function $p(x)$. Introducing the random variable

$$\theta = f(\xi) \tag{2.8}$$

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

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

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

$$\overline{\theta}_N = \frac{1}{N}\sum_{i=1}^{N}\theta_i. \tag{2.9}$$

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

A generalization of the Crude Monte Carlo for the case of infinite variance is considered in [Torn (1966)].

2.2.2 *Geometric Monte Carlo Algorithm*

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

$$0 \le f(x) \le c \quad \text{for} \quad x \in \Omega, \tag{2.10}$$

where c is a generic constant. Consider the cylindrical domain

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

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

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

Let $\tilde{\xi_1}, \ldots, \tilde{\xi_N}$ be independent realizations of the random point $\tilde{\xi}$. Introduce the random variable $\tilde{\theta}$, whose dependency on $\tilde{\xi}$ is clear,

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

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

$$\begin{aligned} E\tilde{\theta} &= cPr\{\xi_{(3)} < f(\xi)\} \\ &= \int_{\Omega} dx_{(1)} dx_{(2)} \int_0^{f(x_{(1)}, x_{(2)})} \tilde{p}(x_{(1)}, x_{(2)}, x_{(3)}) dx_{(3)} = I. \end{aligned}$$

The absolute convergence of the integral follows from (2.10). 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 $\tilde{\theta}_N \xrightarrow{p} I$.

The algorithm consists of generating a sequence of random points $\tilde{\xi} \equiv (\xi_{(1)}, \xi_{(2)}, \xi_{(3)})$ uniformly distributed in the third direction $(\xi_{(3)})$ and accepting points if they are under the graph of the function $f(x)$ and rejecting other points. This is the reason to call this *Geometric* algorithm an *acceptance-rejection* technique.

2.2.3 *Computational Complexity of Monte Carlo Algorithms*

Let us compare the accuracy of the *Geometric* and the *Plain* Monte Carlo algorithm.

Let $f \in \mathbf{L}_2(\Omega, p)$. 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 *Geometric* 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

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

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

The last inequality shows that the *Plain* Monte Carlo algorithm is more accurate than the *Geometric* one (except for the case when the function f is a constant). Nevertheless, the *Geometric* algorithm is often preferred, from the algorithmic point of view, because its computational complexity $C(G) = \tilde{t}D^2\tilde{\theta}$ may be less than the computational complexity of the *plain* algorithm $C(P) = tD^2\theta$. In practical computations it may happen that $\tilde{t} << t$, so that $C(G) < C(P)$ [Sobol (1973)]. The problem of the computational complexity of different Monte Carlo algorithms is considered in detail in [Dimov and Tonev (1993a)].

2.3 Monte Carlo Methods 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*. The techniques used to achieve such a reduction are called *variance-reduction* techniques. Let us consider several classical algorithms of this kind.

2.3.1 *Separation of Principal Part*

Consider again the integral

$$I = \int_\Omega f(x)p(x)dx, \tag{2.11}$$

where $f \in \mathbf{L}_2(\Omega, p),\ x \in \Omega \subset I\!R^d$.

Let the function $h(x) \in \mathbf{L}_2(\Omega, p)$ be *close* to $f(x)$ with respect to its $\mathbf{L}_2$ norm; i.e. $\|f - h\|_{\mathbf{L}_2} \leq \varepsilon$. Let us also suppose that the value of the integral

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

is known.

The random variable $\theta' = f(\xi) - h(\xi) + I'$ generates the following estimate for the integral (2.11)

$$\theta'_N = I' + \frac{1}{N}\sum_{i=1}^{N}[f(\xi_i) - h(\xi_i)]. \tag{2.12}$$

Obviously $E\theta'_N = I$ and (2.12) defines a Monte Carlo algorithm, which is called the *Separation of Principal Part* algorithm. 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 will be quite small, if the function $h(x)$ is such that the integral I' can be calculated analytically. The function $h(x)$ is often chosen to be piece-wise linear function in order to compute the value of I' easily.

2.3.2 *Integration on a 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 (2.11) can be represented as

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

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

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

$$\theta' = I' + (1 - c)f(\xi'). \tag{2.13}$$

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

$$\theta'_N = c + \frac{1}{N}(1 + c)\sum_{i=1}^{N} f(\xi'_i),$$

where ξ_i' are independent realizations of the d-dimensional random point ξ'. The latter presentation defines the *Integration on Subdomain* Monte Carlo algorithm.

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

Theorem 2.1. *([Sobol (1973)]). If the variance $D\theta$ exists then*

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

Proof. Let us calculate the variances of both random variables θ (defined by (2.8) and θ' (defined by (2.13)):

$$D\theta = \int_\Omega f^2 p\ dx - I^2 = \int_{\Omega_1} f^2 p\ dx + \int_{\Omega'} f^2 p\ dx - I^2; \tag{2.14}$$

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

$$= (1-c)\int_{\Omega_1} f^2 p\ dx - \left(\int_{\Omega_1} fp\ dx\right)^2. \tag{2.15}$$

Multiplying both sides of (2.14) by $(1-c)$ and subtracting the result from (2.15) 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. □

2.3.3 *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 point ξ (uniformly distributed in this interval) and the random variable $\theta = (b-a)f(\xi)$. 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. Consider also the random variable θ' defined as $\theta' = (b-a)f_1(\xi)$.

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 2.2. *([Sobol (1973)]). If the partially continuous function f is monotonic in the interval $a \le x \le b$, then*

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

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

$$D\theta = (b-a)\int_a^b f^2(x)dx - I_0^2, \tag{2.16}$$

$$2D\theta' = (b-a)\int_a^b f^2 dx + (b-a)\int_a^b f(x)f(a+b-x)dx - I_0^2. \tag{2.17}$$

From (2.16) and (2.17) it follows that the assertion of the theorem is equivalent to establishing the inequality

$$(b-a)\int_a^b f(x)f(a+b-x)dx \le I_0^2. \tag{2.18}$$

Without loss of generality, suppose that f is non-decreasing, $f(b) > f(a)$, 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, namely

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

is monotonic and since

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

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

$$\int_a^b v(x) f'(x) dx \geq 0. \tag{2.19}$$

Thus integrating (2.19) by parts, one obtains

$$\int_a^b f(x) v'(x) dx \leq 0. \tag{2.20}$$

Now (2.18) follows by replacing the expression for $v'(x)$ in (2.20). (The case of a non-increasing function f can be treated analogously.) □

2.3.4 *Importance Sampling Algorithm*

Consider the problem of computing the integral

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

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

Definition 2.1. Define the probability density function $p(x)$ to be *tolerant to* $f(x)$, if $p(x) > 0$ for $x \in \Omega_+$ and $p(x) \geq 0$ for $x \in \Omega_0$.

For an arbitrary tolerant probability density function $p(x)$ for $f(x)$ in Ω let us define the random variable θ_0 in the following way:

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

It is interesting to consider the problem of finding a tolerant density, $p(x)$, which minimizes the variance of θ_0. The existence of such a density means that the optimal Monte Carlo algorithm with minimal probability error exists.

Theorem 2.3. *([Kahn (1950a,b)]). The probability density function* $c|f(x)|$ *minimizes* $D\theta_0$ *and the value of the minimum variance is*

$$D\hat{\theta}_0 = \left[\int_\Omega |f(x)| dx \right]^2 - I_0^2. \tag{2.21}$$

Proof. 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 normalization of probability density must be satisfied. At the same time

$$D\theta_0 = \int_{\Omega_+} \frac{f^2(x)}{p(x)}dx - I_0^2 = D\hat{\theta}_0. \tag{2.22}$$

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

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

Applying the Cauchy-Schwarz inequality to the last expression one gets

$$\left[\int_{\Omega} |f(x)|dx\right]^2 \leq \int_{\Omega_+} f^2p^{-1}dx \int_{\Omega_+} p\ dx \leq \int_{\Omega_+} \frac{f^2}{p}dx. \tag{2.23}$$

□

Corollary 2.1. *If f does not change sign in Ω, then $D\hat{\theta}_0 = 0$.*

Proof. The corollary is obvious and follows from the inequality (2.21). Indeed, let us assume that the integrand $f(x)$ is non-negative. Then $\left[\int_{\Omega} |f(x)|dx\right] = I_0$ and according to (2.21) the variance $D\hat{\theta}_0$ is zero. If $f(x)$ is non-positive, i.e., $f(x) \leq 0$, then again $\left[\int_{\Omega} |f(x)|dx\right] = I_0$ and $D\hat{\theta}_0$. □

For practical algorithms this assertion allows random variables with 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.

2.3.5 *Weight Functions Approach*

If the integrand contains a weight function an approach proposed in [Shaw (1988)] can be applied. In [Shaw (1988)] 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 (1988)]:

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

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

$$\hat{S}(g; m) = \sum_{i=1}^{N} w_i f(\theta_i) g(\theta_i). \tag{2.24}$$

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

$$S(g; m) = \int g(\theta)m(\theta)d\theta = \int g(\theta)w(\theta)f(\theta)d\theta \tag{2.25}$$

$$= \int g(\theta)f(\theta)dW(\theta) = Ew(g(\theta)f(\theta)). \tag{2.26}$$

This suggests a Monte Carlo approach to numerical integration ([Shaw (1988)]): generate nodes $\theta_1, \ldots, \theta_N$ independently from the distribution w and estimate $S(g; m)$ by $\hat{S}(g; m)$ in (2.24) 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 (2.3.3)). Such an approach is efficient if one deals with a set of integrals with different weight functions.

In [Shaw (1988)] 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); van Dijk *et al.* (1987); Stewart (1983, 1985, 1987); Stewart and Davis (1986); Kloek and van Dijk (1978)].

2.4 Superconvergent Monte Carlo Algorithms

As was shown earlier, the probability error usually has the form of (2.6): $R_N = cN^{-1/2}$. The speed of convergence can be increased if an algorithm

with a probability error $R_N = cN^{-1/2-\psi(d)}$ can be constructed, where c is a constant, $\psi(d) > 0$ and d is the dimension of the space. As will be shown later such algorithms exist. This class of Monte Carlo algorithms exploit the existing smoothness of the integrand. Often the exploiting of smoothness is combined with subdividing the domain of integration into a number of non-overlapping sub-domains. Each sub-domain is called *stratum*. This is the reason to call the techniques leading to superconvergent Monte Carlo algorithms *Stratified sampling*, or *Latin hypercube sampling*. Let us note that the *Plain* Monte Carlo, as well as algorithms based on variance reduction techniques, do not exploit any smoothness (regularity) of the integrand. In this section we will show how one can exploit the regularity to increase the convergence of the algorithm.

Definition 2.2. Monte Carlo algorithms with a probability error

$$R_N = cN^{-1/2-\psi(d)} \tag{2.27}$$

(c is a constant, $\psi(d) > 0$) are called Monte Carlo algorithms with a *superconvergent probable error*.

2.4.1 *Error Analysis*

Let us consider the problem of computing the integral

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

where $\Omega \in I\!R^d$, $f \in \mathbf{L}_2(\Omega; p) \cap \mathbf{W}^1(\alpha; \Omega)$ and p is a probability density function, i.e. $p(x) \geq 0$ and $\int_\Omega p(x)dx = 1$. The class $\mathbf{W}^1(\alpha; \Omega)$ contains function $f(x)$ with continuous and bounded derivatives ($\left|\frac{\partial f}{\partial x_{(k)}}\right| \leq \alpha$ for every $k = 1, 2, \ldots, d$).

Let $\Omega \equiv \mathbf{E}^d$ be the unit cube

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

Let $p(x) \equiv 1$ and consider the partition of Ω into the subdomains (stratums) $\Omega_j, j = 1, 2, \ldots, m$, 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:

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

$$p_j = \int_{\Omega_j} p(x)dx \leq \frac{c_1}{N}, \tag{2.28}$$

and

$$d_j = \sup_{x_1, x_2 \in \Omega_j} |x_1 - x_2| \leq \frac{c_2}{N^{1/d}}, \tag{2.29}$$

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 2.4. *([Dupach (1956); Haber (1966)])*

Let $N_j = 1$ for $j = 1, \ldots, m$ (so that $m = N$). The function f has continuous and bounded derivatives ($\left|\frac{\partial f}{\partial x_{(k)}}\right| \leq \alpha$ for every $k = 1, 2, \ldots, d$) and let there exist constants c_1, c_2 such that conditions (2.28) and (2.29) hold.

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

$$D\theta_N^* = (dc_1c_2\alpha)^2 N^{-1-2/N}.$$

Using the Tchebychev's inequality (see, [Sobol (1973)]) it is possible to obtain

$$R_N = \sqrt{2}dc_1c_2\alpha N^{-1/2-1/d}. \tag{2.30}$$

The Monte Carlo algorithm constructed above has a superconvergent probability error. Comparing (2.30) with (2.27) one can conclude that $\psi(d) = \frac{1}{d}$. We can try to relax a little bit the conditions of the last theorem because they are too strong. So, the following problem may be considered: *Is it possible to obtain the same result for the convergence of the algorithm but for functions that are only continuous?*

Let us consider the problem in $\mathbb{R}$. 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. *([Dimov and Tonev (1989)]).*

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 $j = 1,2,\ldots,N$. Then

$$r_N \leq 4\sqrt{2}\frac{c_1c_2}{c_3}\tau\left(f;\frac{3}{2}d\right)_{L_2} N^{-3/2},$$

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

$$\tau(f;\delta)_{\mathbf{L}_2} = \| \omega(f,\bullet;\delta) \|_{\mathbf{L}_2} = \left(\int_a^b (\omega(f,x;\delta))^q dx\right)^{1/q}, \ 1 \leq q \leq \infty,$$

$\delta \in [0,(b-a)]$ 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 $\mathbb{R}^d$ the following theorem holds:

Theorem 2.6. *([Dimov and Tonev (1989)]).*

Let f be continuous in $\Omega \subset \mathbb{R}^d$ 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

$$r_N \leq 4\sqrt{2}\frac{c_1c_2}{c_3}\tau(f;d)_{\mathbf{L}_2} N^{-1/2-1/d}.$$

Let us note that the best quadrature formula with fixed nodes in $\mathbb{R}$ in the sense of [Nikolskiy (1988)] for the class of functions $\mathbf{W}^{(1)}(l;[a,b])$ 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 2.6 when $N_j = 1$ the rate of convergence is improved by an order of $1/2$. This is an essential improvement. At the same time, the estimate given in Theorem 2.5 for the rate of convergence attains the lower bound estimate obtained by Bakhvalov ([Bakhvalov (1959, 1961, 1964)]) for the error of an arbitrary random quadrature formula for the class of continuous functions in an interval $[a,b]$. Some further developments in this direction will be presented in Chapter 3.

2.4.2 *A Simple Example*

Let us consider a simple example of evaluating multi-dimensional integrals in functional classes which demonstrates the power of the Monte Carlo algorithms. This is a case of practical computations showing high efficiency of the Monte Carlo approach versus the deterministic one. Consider the classical problem of integral evaluation in functional classes with bounded derivatives. Suppose $f(x)$ is a continuous function and let a quadrature formula of Newton or Gauss type be used for calculating the integrals. Consider an example with $d = 30$ (this is a typical number for some applications in control theory and mathematical economics). In order to apply such formulae, we generate a grid in the d-dimensional domain and take the sum (with the respective coefficients according to the chosen formula) of the function values at the grid points. Let a grid be chosen with 10 nodes on the each of the coordinate axes in the d-dimensional cube $\mathbf{E}^d = [0,1]^d$. 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 (remember that 1 year $= 31536 \times 10^3 s$, and that there has been less than $9 \times 10^{10} s$ since the birth of Pythagoras). Suppose the calculations have been done for a function $f(x) \in \mathbf{W^{(2)}}(\alpha, [\mathbf{0},\mathbf{1}]^{\mathbf{d}})$. If the formula of rectangles (or some similar formula) is applied then the error in the approximate integral calculation is

$$\varepsilon \leq cMh^3, \quad (h = 0.1), \tag{2.31}$$

where h is the mesh-size and c is a constant independent of h. More precisely, if $f(x) \in C^2$ then the error is $\frac{1}{12} f^{(2)}(c) h^3$, where c is a point inside the domain of integration [Krommer and Ueberhuber (1998); Sendov *et al.* (1994)].

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

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

$$\varepsilon \leq 0.6745\sigma(\theta)\frac{1}{\sqrt{N}}, \tag{2.32}$$

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

We can estimate the probable error using (2.32) and the variance properties:

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

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

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

The computational complexity of this commonly-used Monte Carlo algorithm will now be estimated. From (2.32), we may conclude:

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

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

$$N \approx 10^6;$$

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

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

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

The plain (or crude) Monte Carlo integration is often considered as the best algorithm for evaluating multidimensional integrals in case of very high dimensions (see, [Davis and Rabinowitz (1984); Kalos and Whitlock (1986); Krommer and Ueberhuber (1998)]). In the case of low dimensions (less than 3) Gauss product rules seem to be the most efficient choice. Between these two bound cases algorithms that exploit the regularity of

the integrand should be used. An example of such an approach can be found in Chapter 3 (see, also [Atanassov and Dimov (1999)]). In fact, we use a variance reduction technique to build Monte Carlo algorithms with an increased rate of convergence. The proposed algorithms are based on piecewise interpolation polynomials and on the use of the control variate method. It allowed us to build two algorithms reaching the optimal rate of convergence for smooth functions. Monte Carlo algorithms with increased rate of convergence are presented in [Maire (2003b,a); Morton (1957); Press and Farrar (1990); Stewart (1987); Stewart and Davis (1986)].

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 [Bakhvalov (1964); Dimov (1994); Dimov and Gurov (1993); Dimov *et al.* (1996); Dimov and Tonev (1989, 1993b); Nikolskiy (1988); Sendov *et al.* (1994)]. For example, as shown in Subsection 2.4.1 if $f(x) \in \mathbf{W^1}(\alpha; [\mathbf{0}, \mathbf{1}]^{\mathbf{d}})$, then instead of (2.32) we have the following estimate of the probability error:

$$R_N = \varepsilon \leq c_1 \alpha \frac{1}{N^{1/2+1/d}}, \tag{2.33}$$

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

Let us stress on the fact that we compare different approaches assuming that the integrand belongs to a certain functional class. It means that we do not know the particular integrand and our consideration is for the *worst* function from a given class. Another approach is considered in [Schurer (2003)], where Monte Carlo and Quasi–Monte Carlo quadratures are compared with adaptive and interpolation deterministic type quadrature. To be able to apply adaptive or interpolation quadrature one should know the integrand, so this problem is much easier for consideration. It is also shown in [Schurer (2003)] that some dimensions and test–functions adaptive type quadrature outperform the crude Monte Carlo. Such a result is not surprising since such type of deterministic quadrature should be compared with interpolation and adaptive Monte Carlo quadrature [Sobol (1973); Dimov (1994)].

2.5 Adaptive Monte Carlo Algorithms for Practical Computations

In this section a *Superconvergent Adaptive algorithm* for practical computations will be presented. 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 2.4 we defined a Superconvergent Monte Carlo algorithm for which $r_N = cN^{-1/2-\psi(d)}$, where c is a constant and $\psi(d) > 0$ [Sobol (1973); Dimov and Tonev (1989)]. The idea of the algorithm consists in separating the domain Ω into stratums Ω_j that are uniformly small according both to the probability and to the sizes.

Another degree of quality of Monte Carlo algorithms would be the variance reduction. 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 finiteness of the variance $D\hat{\theta}$ be fulfilled. As it was shown in Subsection 2.3.4 an algorithm for which $D\hat{\theta} < D\theta$ is called an *efficient Monte Carlo algorithm* [Kahn (1950a,b); Mikhailov (1987); Dimov (1991); Dimov and Tonev (1993b)]. An algorithm of this type is proposed by Kahn, for evaluation of integrals ([Kahn (1950a,b)]); and by Mikhailov and Dimov, for evaluation of integral equations ([Mikhailov (1987); Dimov (1991)]).

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 [Mikhailov (1987); Dimov (1991)] 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 2.5.1 contains error analysis results concerning SAMC algorithms. In fact, the proposed algorithm is an superconvergent Monte Carlo algorithm with the probable error of type $c \times N^{-1/2-\psi(d)}$, but

the constant c in front of the main term ($N^{-1/2-\psi(d)}$) is smaller than the constant of the usual Superconvergent Monte Carlo algorithm.

2.5.1 *Superconvergent Adaptive Monte Carlo Algorithm and Error Estimates*

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

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

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

where the positive function $f(x) \in \mathbf{W}^1(\alpha;[0,1])$ 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 ($m \le N$):

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

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

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

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

The scheme (2.34) defines 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

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

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

Let $N_i = 1$ (so, $m = N$). Since $f(x) \in \mathbf{W}^1(L; [0,1]$, there exist constants α_i, such that

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

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

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

and

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

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 an importance separation.

Theorem 2.7. *Let $f(x) \in \mathbf{W}^1(\alpha; [0,1])$ and $m = n$. Then for the importance separation of Ω*

$$r_N \leq \sqrt{2}\left[\frac{1}{N}\sum_{j=1}^{N}(\alpha_j c_1^{(j)} c_2^{(j)})^2\right]^{1/2} N^{-3/2}.$$

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

$$\begin{aligned} Df(\xi^{(j)}) &\leq Ef^2(\xi^{(j)}) \\ &\leq \alpha_j^2 E(\xi^{(j)} - s^{(j)})^2 \\ &\leq \alpha_j^2 \sup_{x_1^{(i)}, x_2^{(i)} \in \Omega_j} |x_1^{(j)} - x_2^{(j)}|^2 \\ &\leq \frac{\alpha_j^2 (c_2^{(j)})^2}{N^2}. \end{aligned}$$

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

$$\begin{aligned} D\theta_N^* &= \sum_{j=1}^{N} p_j^2 Df(\xi^{(j)}) \\ &\leq \sum_{j=1}^{N} ((c_1^{(j)})^2 N^{-2} \alpha_j^2 (c_2^{(j)})^2 N^{-2}) \\ &= \frac{1}{N} \sum_{j=1}^{N} (\alpha_j c_1^{(j)} c_2^{(j)})^2 N^{-3}. \end{aligned}$$

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

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

where $h > 0$.

Let

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

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

$$r_N \leq \sqrt{2} (D\theta_N^*)^{1/2} = \sqrt{2} \left(\frac{1}{N} \sum_{j=1}^{N} \left(\alpha_j c_1^{(j)} c_2^{(j)} \right)^2 \right)^{1/2} N^{-3/2}. \tag{2.38}$$

□

This result presents a Superconvergent Adaptive Monte Carlo algorithm. Moreover, the constant $\sqrt{2}\left(\frac{1}{N}\sum_{j=1}^{N}(\alpha_j c_1^{(j)} c_2^{(j)})^2\right)^{1/2}$ in (2.38) is smaller than the constant in the algorithms of the Dupach type [Dupach (1956); Dimov and Tonev (1989)] presented in Section 2.4. In fact, $\alpha_i \leq \alpha$ for any $i = 1, \ldots, m = N$. Obviously,

$$\frac{1}{N}\sum_{j=1}^{N}\alpha_j c_1^{(j)} c_2^{(j)} \leq \alpha c_1 c_2.$$

Now consider multidimensional integrals:

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

where the positive function $f(x)$ belongs to $\mathbf{W}^1(\alpha;\Omega)$.

Let

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

Let there exist vectors

$$\alpha_i = (\alpha_{i_1}, \ldots, \alpha_{i_d}),\quad i = 1, \ldots, N,$$

such that

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

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

$$c_2^{(i)} = \left(c_{2(1)}^{(i)}, c_{2(2)}^{(i)}, \ldots, c_{2(d)}^{(i)}\right), \tag{2.41}$$

such that,

$$p_i = \int_{\Omega_i} p(x)\,dx \leq \frac{c_1^{(i)}}{N},\ i = 1, \ldots, N \tag{2.42}$$

and

$$d_{il} = \sup_{x_{1(l)}^{(i)}, x_{2(l)}^{(i)} \in \Omega_i} \left|x_{1(l)}^{(i)} - x_{2(l)}^{(i)}\right| \leq \frac{c_{2(l)}^{(i)}}{N^{1/d}};\quad i = 1, \ldots, N,\quad l = 1, \ldots, d. \tag{2.43}$$

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

$$r_N \leq \sqrt{2}d \left[\frac{1}{N} \sum_{i=1}^{N} \sum_{l=1}^{d} \left(\alpha_{il} c_1^{(i)} c_{2(l)}^{(i)} \right)^2 \right]^{1/2} N^{-1/2-1/d}.$$

The proof of this theorem follows the same techniques as the proof of Theorem 2.7.

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

$$\alpha_j = \max_l \alpha_{jl}, \quad j = 1, \ldots, N \tag{2.44}$$

and let

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

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

$$r_N \leq \sqrt{2}d \left[\frac{1}{N} \sum_{j=1}^{N} (c_1^{(j)} c_2^{(j)} \alpha_j)^2 \right]^{1/2} N^{-1/2-1/d},$$

where $\alpha_j = \max_l \alpha_{jl}$.

The proof is obvious since,

$$\sum_{l=1}^{d} \alpha_{jl} \leq d \cdot \max_l \alpha_{jl} = d\alpha_j.$$

2.5.2 *Implementation of Adaptive Monte Carlo Algorithms. Numerical Tests*

We run two Monte Carlo algorithms.

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 $x_i \in \mathbf{E} \equiv [0,1]^d$, $i = 1, \ldots, N$ into d- dimensional cube $\mathbf{E}^d$. 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 ($d = 1, \ldots, 5$) 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.

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

Example 1.

$$d = 4$$
$$I_1 = \int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{4x_1x_3^2e^{2x_1x_3}}{(1+x_2+x_4)^2}dx_1dx_2dx_3dx_4 \approx 0.57536.$$

Example 2.

$$d = 5$$
$$I_2 = \int_0^1 \int_0^1 \int_0^1 \int_0^1 \int_0^1 \frac{4x_1 x_3^2 e^{2x_1 x_3} e^{x_5}}{(1+x_2+x_4)^2} dx_1 \dots dx_5 \approx 0.98864.$$

Example 3.

$$d = 10$$
$$I_3 = \int_0^1 \dots \int_0^1 \frac{4x_1 x_3^2 e^{2x_1 x_3}}{(1+x_2+x_4)^2} e^{x_5+\dots+x_{10}}\ dx_1 \dots dx_{10} \approx 14.80844.$$

Example 4.

$$d = 25$$
$$I_3 = \int_0^1 \dots \int_0^1 \frac{4x_1 x_3^2 e^{2x_1 x_3}}{(1+x_2+x_4)^2} e^{x_5+\dots+x_{20}} \times x_{21} \dots x_{25}\ dx_1 \dots dx_{25} \approx 103.82529.$$

Example 5.

$$d = 30$$
$$I_4 = \int_0^1 \dots \int_0^1 \frac{4x_1 x_3^2 e^{2x_1 x_3}}{(1+x_2+x_4)^2} e^{x_5+\dots+x_{20}} \times x_{21} x_{22} \dots x_{30}\ dx_1 \dots dx_{30} \approx 3.24454. \tag{2.46}$$

Some results are presented in Table 2.1.

Table 2.1 contains information about the dimension of the integral, the exact solution, the applied Monte Carlo approach, calculated solution, CP-time, as a measure of the computational cost, *a posteriori* estimated error by the code, the real relative error, and number of random points.

Some of numerical results are presented in Figures 2.1 to 2.4. Figure 2.1 shows the results of implementation of SAMC algorithm for solving the 5-dimensional integral. The dependence of the calculated values and the error from the logarithm of the number of random points ($logN$) 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-dimensional integral for different numbers of random points (here again on the x-coordinate

Table 2.1 Results of numerical experiments performed by SAMC and AMC for integration problems of different dimensions.

Dim. (d)	Exact sol.	Algor.	Calcul. sol.	CP-time, (s)	Estim. error	Rel. error	Num. of points
4	0.5753	AMC	0.5763	0.184	0.025	0.0017	2×10^4
4	0.5753	SAMC	0.5755	0.025	0.0082	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

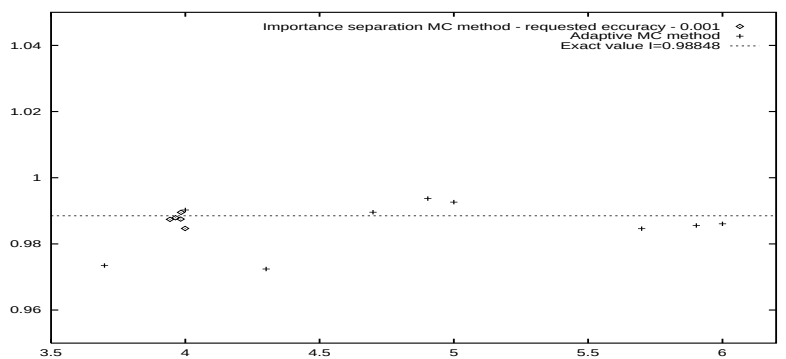


Fig. 2.1 Numerical results for *Superconvergent Adaptive* MC integration and *Plain Adaptive* MC for Example 2, $d = 5$ (depending on $\log N$).

the values of $logN$ are presented). Figure 2.3 expresses the results for the solution as well as for the estimated error obtained for 25-dimensional integral. Some numerical results for the implementation of AMC algorithm for 30-dimensional 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 considered in control theory.

2.5.3 *Discussion*

- For low dimensions the SAMC algorithm gives better results than the AMC algorithm. For example, for $d = 5$, the SAMC algorithm needs 9000 realizations to reach an error of 0.03%, while the AMC algorithm needs 20000 realizations to reach an error of 0.17%. The

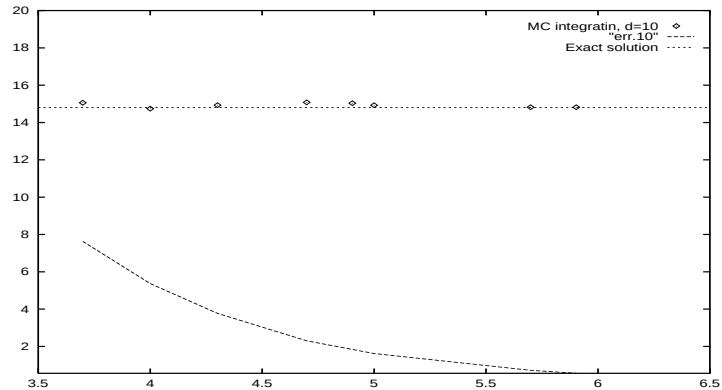


Fig. 2.2 Superconvergent Adaptive MC integration for Example 3, $d = 10$ and dependance of the error function of $\log N$.

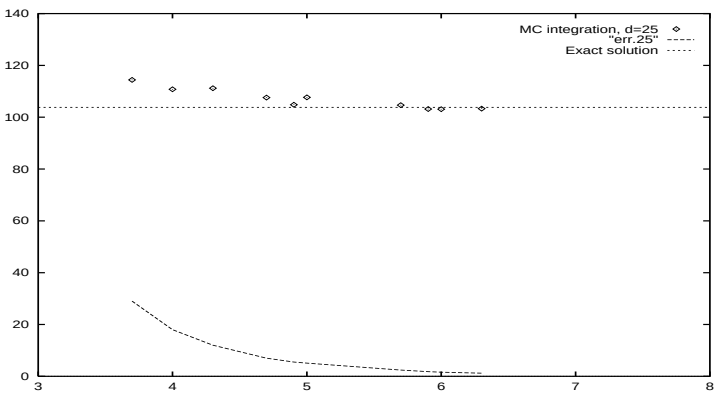


Fig. 2.3 Adaptive MC integration for Example 4, $d = 25$.

CP-time of the SAMC algorithm is about 7 times less than the corresponding CP-time of the AMC algorithm.

- When the dimension d is high the AMC algorithm gives better results than the SAMC algorithm. It is explained by 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.
- The Monte Carlo algorithms are important as they permit one 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

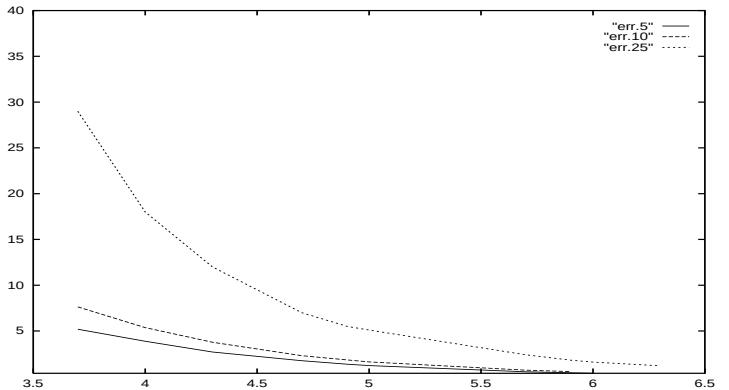


Fig. 2.4 Error functions for Adaptive MC integration and different dimensions of $\log N$.

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 a modern computer is used.

2.6 Random Interpolation Quadratures

A quadrature is called *interpolation* for a given class of functions if it is exact for any linear combination of functions from this class. In this section we show how a random interpolation quadrature can be defined. These algorithms have zero theoretical probability error. In practical computations, since one can only perform a sample of r.v., the probability error is not zero, but it is very small. So, the random interpolation quadrature are high quality quadrature. The problem is that they have a restricted area of application: one should be sure that each integrand belongs to a given class of function presented by a linear combination of a system of orthonormal function. These quadratures could be effective for solving problems in some areas of modern physics, where people are interested to compute a large number of multidimensional integrals with similar integrands. Assume that

the quadrature formula for computing the integral

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

is denoted by the expression

$$I \approx \sum_{j=1}^{N} c_j f(x_j), \tag{2.47}$$

where $x_1, \ldots, x_N \in \Omega$ are nodes and $c_1, \ldots, c_N$ are weights. Then the random quadrature formula can be written in the following form:

$$I \approx \sum_{j=1}^{N} \varkappa_j f(\xi_j), \tag{2.48}$$

where $\xi_1, \ldots, \xi_N \in \Omega$ are random nodes and $\varkappa_1, \ldots, \varkappa_N$ are random weights.

All functions considered in this section are supposed to be partially continuous and belong to the space $\mathbf{L}_2(\Omega)$.

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

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

where δ_{kj} the Kronecker funciton.

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

$$I = \int_\Omega f(x)\varphi_0(x)dx \tag{2.49}$$

can be found using a quadrature formula of the type (2.47).

Let us fix arbitrary nodes $x_0, x_1, \ldots, x_m \quad (x_i \neq x_j \text{ when } i \neq j)$ and choose the weights $c_0, c_1, \ldots, c_m$ such that (2.47) is exact for the system of orthonormal functions $\varphi_0, \varphi_1, \ldots, \varphi_m$. In this case it is convenient to represent the quadrature formula (2.47) as a ratio of two determinants

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

where

$$W_g(x_0, x_1, \ldots, x_m) = \begin{vmatrix} g(x_0) & \varphi_1(x_0) & \ldots & \varphi_m(x_0) \\ g(x_1) & \varphi_1(x_1) & \ldots & \varphi_m(x_1) \\ \vdots & \vdots & \ddots & \vdots \\ g(x_m) & \varphi_1(x_m) & \ldots & \varphi_m(x_m) \end{vmatrix}. \tag{2.50}$$

It is easy to check that if $W_{\varphi_0} \neq 0$ then the formula (2.49) is exact for every linear combination of the following form:

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

Theorem 2.9. *([Sobol (1973)]).*

Let $\varphi_0, \varphi_1, \ldots, \varphi_m$ be an arbitrary set of orthonormal functions in Ω. Then

$$\int_\Omega \ldots \int_\Omega W_{\varphi_0}^2 dx_0 \ldots dx_m = (m+1)!.$$

Theorem 2.10. *([Sobol (1973)]).*

Let $\varphi_0, \varphi_1, \ldots, \varphi_m, \psi$ be an arbitrary set of orthonormal functions in Ω. Then

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

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

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

so that $dt = dx_0 \ldots dx_m$; 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 $\xi_0, \ldots, \xi_m$ in Ω and consider the random variable

$$\hat{\theta}[f] = \begin{cases} \frac{W_f(\xi_0, \xi_1, \ldots, \xi_m)}{W_{\varphi_0}(\xi_0, \xi_1, \ldots, \xi_m)}, & \text{if } (\xi_0, \xi_1, \ldots, \xi_m) \in B_+, \\ 0, & \text{if } (\xi_0, \xi_1, \ldots, \xi_m) \in B_0. \end{cases} \tag{2.51}$$

as an approximate value for (2.49).

Theorem 2.11. *([Ermakov and Zolotukhin (1960); Ermakov (1967)])*

If the joint probability density function of the random points $\xi_0, \xi_1, \ldots, \xi_m$ in B is

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

then, for every function $f \in \mathbf{L}_2(D)$, the following is true:

$$E\hat{\theta}[f] = \int_\Omega f(x)\varphi_0(x)dx, \tag{2.52}$$

$$D\hat{\theta}[f] \leq \int_\Omega f^2(x)dx - \sum_{j=0}^{m} \left[\int_\Omega f(x)\varphi_j(x)dx \right]. \tag{2.53}$$

Proof. Let us denote by c_j the Fourier coefficients of the function $f \in \mathbf{L}_2(\Omega)$

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

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

$$f(x) = \sum_{j=0}^{m} c_j\varphi_j(x) + a\psi(x). \tag{2.54}$$

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 (2.54) still holds, because every normed function, which is orthogonal to every φ_j, can be chosen as ψ. Replacing (2.54) in (2.50) one can deduce the following result:

$$W_f = \sum_{j=0}^{m} c_j W_{\varphi_j} + aW_\psi = c_0 W_{\psi_0} + aW_\psi. \tag{2.55}$$

Using (2.55), it is easy to calculate the mathematical expectation of (2.51)

$$E\hat{\theta}[f] = \int_{B_+} \frac{W_f}{W_{\varphi_0}} p\ dt = \frac{1}{(m+1)!} \int_{B_+} W_f W_{\varphi_0} dt = \frac{1}{(m+1)!} \int_B W_f W_{\varphi_0} dt$$

$$= \frac{c_0}{(m+1)!} \int_B W_{\varphi_0}^2 dt + \frac{a}{(m+1)!} \int_B W_{\varphi_0} W_\psi dt.$$

The first of the last two integrals is equal to $(m+1)!$ by Theorem 2.9; the second integral is equal to 0 according to Theorem 2.10. Therefore $E\hat{\theta}[f] = c_0$, which is equivalent to (2.52). This proves the statement (2.52) of the theorem.

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

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

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

To prove the statement (2.53) we have to show that $D\hat{\theta}[f] \le a^2$. Using Theorem 2.9 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

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

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

□

From the proof it is obvious that the inequality (2.53) 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 Methods

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 (1991)]). Quasi-random numbers are connected with a certain class of algorithms and their applicability is more restricted than that of pseudo-random numbers ([Sobol (1991); Niederreiter (1987, 1992)]). 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 $N^{-1+\psi}$, where $\psi > 0$ is arbitrarily small.

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

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

Denote by $S_N(\Omega)$ the number of points with $1 \leq i \leq N$ that lie in Ω, where $\Omega \subset \mathbf{E}^d$. The sequence $x_1, x_2, \ldots$ 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 2.12. *([Weyl (1916); Sobol (1991)])*
The relation

$$\lim_{N\to\infty} \frac{1}{N} \sum_{i=1}^{N} f(\xi_j) = \int_{\mathbf{E}^d} f(x)dx \tag{2.56}$$

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

Comparing (2.9) with (2.56) 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 $(x^{(i)}_{(1)}, \ldots, x^{(i)}_{(d)})$ of the point x_i, rather than the random numbers $\gamma_1, \ldots, \gamma_N$. For practical purposes a u.d.s. must be found that satisfied three additional requirements ([Sobol (1973)], [Sobol (1989)]):

(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 [Sobol (1989)] 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 $j = 1, 2, \ldots, 2^m, m = 0, 1, \ldots$ (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 $\mathbf{E}^d$, 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 [Sobol (1979)] and [Bradley and Fox (1980)]. More details are contained in [Levitan *et al.* (1988)].

The problem of determining an error estimate for

$$\frac{1}{N} \sum_{i=1}^{N} f(\xi_j) \approx \int_{\mathbf{E}^d} f(x)dx$$

arises.

If one uses the points $x_0, x_1, \ldots$ the answer is quite simple. Finite nets $x_0, \ldots, x_{N-1}$, $N = 2^m$ (with m a positive integer), have good uniform properties ([Sobol (1991)]).

Non-random estimates of the error

$$\delta_N(f) = \frac{1}{N} \sum_{i=1}^{N-1} f(x_i) - \int_{\mathbf{E}^d} f(x)dx \tag{2.57}$$

are known ([Sobol (1991, 1989)]). Let us mention two results. First, assume that all partial derivatives of the function $f(x)$, $x \in \mathbf{E}^d \subset I\!R^d$, that include at most one differentiation with respect to each variable,

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

are continuous. It follows from [Sobol (1991)] that for $N = 2^m$

$$|\delta_N(f)| \le A(f) N^{-1} (\log N)^{d-1}. \tag{2.58}$$

If in (2.57) arbitrary values of N are used then in (2.58) $(\log N)^{d-1}$ must be replaced by $(\log N)^d$ ([Sobol (1991)]).

Consider a class $\mathbf{W}^1$ of functions $f(x)$, $x \in \mathbf{E}^d \subset I\!R^d$, with continuous first partial derivatives. Denote

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

It follows from the Sobol's result ([Sobol (1989)]) that if $N = 2^m$, then

$$|\delta_N(f)| \le \max \left(s! \prod_{k=1}^{s} \alpha_{i_k} / N \right)^{\frac{1}{s}},$$

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

If, for example, some of the α_i are very small, then 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 (1987)] a five-dimensional improper integral is evaluated by using the $\prod_\tau$-approximation and the *Plain* Monte Carlo algorithm.

2.8 Exercises

(1) **Random Variables**

(a) Find the mathematical expectation and the variance of the r.v. γ^3, where γ is a continuous uniformly distributed r.v. (c.u.d.r.v.) in $[0,1]$.

(b) Consider an algorithm for generating a discrete r.v. ξ:

$$\xi = \begin{pmatrix} x_1 & x_2 & \dots & x_n \\ p_1 & p_2 & \dots & p_n \end{pmatrix},$$

where $p_i = P\{\xi = x_i\} = \frac{1}{n}$ using a random number γ (γ is a c.u.d.r.v. in $[0,1]$).

(2) **Plain Monte Carlo Algorithms**

Consider a *Plain* Monte Carlo algorithm for computing

$$I = \int_0^1 \sqrt[5]{x} dx$$

using a r.v. θ with a constant density function $p(x) = const$. Show that the variance $D\theta$ is much smaller than I^2.

(3) **Importance Sampling Monte Carlo Algorithms**

Consider a *Importance sampling Monte Carlo algorithm* for

$$I = \int_0^1 e^x dx.$$

Show that the variance of the algorithm $D\theta_0$ is zero.

(4) **Symmetrization of the Integrand**

For calculating the integral

$$I = \int_0^1 e^x dx$$

apply Monte Carlo with a r.v. $\theta' = f_1(x)$ using *symmetrization of the integrand*: $f_1(x) = \frac{1}{2}[f(x) + f(a+b-x)]$. Show that $D\theta' << I^2$.

(5) **Separation of the principle part**

Consider the integral

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

where $f(x) = e^x$ and $p(x) = 1, \quad x \in [0, 1]$.
a) Find the value of I.
b) Consider a Monte Carlo algorithm using *separation of principle part* for $h(x) = x$ and show that $E\theta' = I$.
c) Show that $D\theta' << I^2$.

(6) **Integration on Subdomain Monte Carlo Algorithm**

Consider the integral

$$I = \int\int_{\Omega} (2 - y)dxdy,$$

where $\Omega \equiv \{(x, y) : 0 \leq x \leq 1; \ 0 \leq y \leq 1 + \frac{x}{4}\}$. Consider an *integration on subdomain Monte Carlo algorithm* assuming that

$$I' = \int\int_{\Omega'} (2 - y)dxdy \quad \text{and} \quad c = \int\int_{\Omega'} p(x, y)dxdy,$$

where $\Omega' \equiv \{(x, y) : 0 \leq x \leq 1; \quad 0 \leq y \leq 1\}$ using a r.v. θ' with a constant density function $p(x, y)$ (such that $\int\int_{\Omega} p(x, y)dxdy = 1$).
Show that:
a) $D\theta' << I^2$;
b) $D\theta' \leq (1 - c)D\theta$, where θ is the corresponding r.v. for the *plain* Monte Carlo integration.

This page intentionally left blank

Chapter 3

Optimal Monte Carlo Method for Multidimensional Integrals of Smooth Functions

An optimal Monte Carlo method for numerical integration of multidimensional integrals is proposed and studied. It is know that the best possible order of the mean square error of a Monte Carlo integration method over the class of the k times differentiable functions of d variables is $O\left(N^{-\frac{1}{2}-\frac{k}{d}}\right)$. In this chapter we present two algorithms implementing the method under consideration.

Estimates for the computational complexity are obtained. Numerical tests showing the efficiency of the algorithms are also given.

3.1 Introduction

In this chapter a Monte Carlo method for calculating multidimensional integrals of smooth functions is considered. Let d and k be integers, and $d, k \geq 1$. We consider the class $\mathbf{W}^k(\|f\|; \mathbf{E}^d)$ (sometimes abbreviated to $\mathbf{W}^k$) of real functions f defined over the unit cube $\mathbf{E}^d = [0,1)^d$, possessing all the partial derivatives

$$\frac{\partial^r f(x)}{\partial x_1^{\alpha_1} \dots \partial x_d^{\alpha_d}}, \;\; \alpha_1 + \dots + \alpha_d = r \leq k,$$

which are continuous when $r < k$ and bounded in sup norm when $r = k$. The semi-norm $\|\cdot\|$ on $\mathbf{W}^k$ is defined as

$$\|f\| = \sup \left\{ \left| \frac{\partial^r f(x)}{\partial x_1^{\alpha_1} \dots \partial x_d^{\alpha_d}} \right| \;\; |\alpha_1 + \dots + \alpha_d = k, \;\; x \equiv (x_1, ..., x_d) \in \mathbf{E}^d \right\}.$$

There are two classes of methods for numerical integration of such functions over $\mathbf{E}^d$– *deterministic* and *stochastic* or *Monte Carlo methods*. We

consider the following quadrature formula

$$I(f) = \sum_{i=1}^{N} c_i f(x^{(i)}), \tag{3.1}$$

where $x^{(i)} \equiv \left(x_1^{(i)}, \ldots, x_d^{(i)}\right) \in \mathbf{E}^d$, $i = 1, \ldots, N$, are the nodes and c_i, $i = 1, \ldots, N$ are the weights. If $x^{(i)}$ and c_i are real values, the formula (3.1) defines a deterministic quadrature formula. If $x^{(i)}$, $i = 1, \ldots, N$, are random points defined in $\mathbf{E}^d$ and c_i are random variables defined in $\mathbb{R}$ then (3.1) defines a Monte Carlo quadrature formula.

The following results of Bahvalov establish lower bounds for the integration error in both cases:

Theorem 3.1. *([Bakhvalov (1959, 1961)]) There exists a constant $c(d,k)$ such that for every quadrature formula $I(f)$ that is fully deterministic and uses the function values at N points there exists a function $f \in \mathbf{W}^k$ such that*

$$\left| \int_{\mathbf{E}^d} f(x)dx - I(f) \right| \geq c(d,k) \, \|f\| \, N^{-\frac{k}{d}}.$$

Theorem 3.2. *([Bakhvalov (1959, 1961)]) There exists a constant $c(d,k)$ such that for every quadrature formula $I(f)$ that involves random variables and uses the function values at N points there exists a function $f \in \mathbf{W}^k$ such that*

$$\left\{ \mathrm{E} \left[\int_{\mathbf{E}^d} f(x)dx - I(f) \right]^2 \right\}^{1/2} \geq c(d,k) \, \|f\| \, N^{-\frac{1}{2}-\frac{k}{d}}.$$

When d is sufficiently large it is evident that methods involving the use of random variables will outperform the deterministic methods.

Monte Carlo methods, that achieve the order $O\left(N^{-\frac{1}{2}-\frac{k}{d}}\right)$ are called *optimal*. In fact methods of this kind are superconvergent following Definition 2.2 given in Section 2.4, hey have a unimprovable rate of convergence. It is not an easy task to construct a unified method with such rate of convergence for any dimension d and any value of k. Various methods for Monte Carlo integration that achieve the order $O\left(N^{-\frac{1}{2}-\frac{k}{d}}\right)$ are known. While in the case of $k = 1$ and $k = 2$ these methods are fairly simple and are widely used (see, for example, [Sobol (1989); Sendov *et al.* (1994); Tanaka and Nagata (1974); Novak and Ritter (1996)]), when $k \geq 3$ such methods become much more sophisticated. The first optimal stochastic method was

proposed by Mrs. Dupach [Dupach (1956)] for $k = 1$. This method uses the idea of separation of the domain of integration into uniformly small (according both to the probability and to the sizes) disjoint subdomains and generating one or small number of points in each subdomain. This idea was largely used [Sendov *et al.* (1994)] for creation Monte Carlo methods with high rate of convergence.

There exist also the so-called *adaptive* Monte Carlo methods proposed by Lautrup (see, pp. 391-392 in the book [Davis and Rabinowitz (1984)]), which use *a priori* and/or *a posteriori* information obtained during calculations. The main idea of this approach is to adapt the Monte Carlo quadrature formula to the element of integration. In Section 2.5 (see also [Karaivanova and Dimov (1998)]) the idea of separation of the domain into *uniformly small* subdomains is combined with the idea of adaptivity to obtain an optimal Monte Carlo quadrature for the case $k = 1$.

In this chapter we also combine both ideas - separation and adaptivity and present an optimal Monte Carlo quadrature for any k. We separate the domain of integration into disjoint subdomains. Since we consider the cube $\mathbf{E}^d$ we divide it into $N = n^d$ disjoint cubes $K_j, j = 1, \ldots, N$. In each cube K_j we calculate the coordinates of $\binom{d+k-1}{d}$ points $y^{(r)}$. We select m uniformly distributed and mutually independent random points from each cube K_j and consider the Lagrange interpolation polynomial of the function f at the point z, which uses the information from the function values at the points $y^{(r)}$. After that we approximate the integral in the cube K_j using the values of the function and the Lagrange polynomial at the m selected random points. Then we sum these estimates over all cubes $K_j,\ j = 1, \ldots, N$. The adaptivity is used when we consider the Lagrange interpolation polynomial. The estimates for the probable error and for the mean square error are proven. It is shown that the presented method has the best possible rate of convergence, i.e. it is an optimal method.

In this chapter we present two algorithms for Monte Carlo integration that achieve such order of the integration error, along with estimates of their *computational complexity*. It is known that the *computational complexity* is defined as number of operations needed to perform the algorithm on the sequential (von Neumann) model of computer architecture. It is important to be able to compare different Monte Carlo algorithms for solving the same problem with the same accuracy (with the same probable or mean square error). In the Introduction (Chapter 1) we showed that the computational complexity could be estimated as a product of $t\sigma^2(\theta)$, where t is the time

(number of operations) needed to calculate one value of the random variable θ, whose mathematical expectation is equal to the exact value of the integral under consideration and $\sigma^2(\theta)$ is the variance (it is shown in Subsection 2.2.3; see, also Definition 1.9 in the Introduction and [Sobol (1973)]). Here we do not use this presentation and, instead, estimate the computational complexity directly as number of floating point operations (flops) used to calculate the approximate value of the integral.

One can also use some other estimators of the quality of the algorithm (if parallel machines are available), such as *speedup* and *parallel efficiency* [Dimov and Tonev (1993b)]. It is easy to see that the speed-up of our algorithms is linear and the parallel efficiency is close to 1 due to the relatively small communication costs. The numerical tests performed on 2-processor and 4-processor machines confirm this.

The chapter is organized as follows. Section 3.2 contains the description of the method. Here the estimates for the probable error and for mean square error are obtained, showing that the method is optimal. In Section 3.3 two algorithms implementing the described Monte Carlo method are presented. Estimates of the computational complexity are proven. Section 3.4 presents some numerical tests confirming the high computational efficiency of the algorithms. The chapter ends with some concluding remarks about applicability of the algorithms under consideration.

3.2 Description of the Method and Theoretical Estimates

Definition 3.1. Given a Monte Carlo integration formula for the functions in the space $\mathbf{W}^k$ by $err(f, I)$ we denote the integration error

$$\int_{\mathbf{E}^d} f(x)dx - I(f),$$

by $\varepsilon(f)$ the probable error meaning that $\varepsilon(f)$ is the least possible real number with

$$P\left(|err(f, I)| < \varepsilon(f)\right) \geq \frac{1}{2}$$

and by $r(f)$ the mean square error

$$r(f) = \left\{ E\left[\int_{\mathbf{E}^d} f(x)dx - I(f) \right]^2 \right\}^{1/2}.$$

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

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

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

then $P \equiv 0$. Let $N = n^d$, $n \geq 1$. We divide the unit cube $\mathbf{E}^d$ into n^d disjoint cubes

$$\mathbf{E}^d = \bigcup_{j=1}^{n^d} K_j, \text{ where } K_j = \prod_{i=1}^{d} [a_i^j, b_i^j),$$

with $b_i^j - a_i^j = \frac{1}{n}$ for all $i = 1, \ldots, d$. Now in each cube K_j we calculate the coordinates of $\binom{d+k-1}{d}$ points $y^{(r)}$, defined by

$$y_i^{(r)} = a_i^r + \frac{1}{n} x_i^{(r)}.$$

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

For all polynomials P of degree at most $k-1$ we have $L_k(p,z) \equiv z$.

We approximate

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

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

$$I(f) \approx \frac{1}{mn^d} \sum_{j=1}^{N} \sum_{s=1}^{m} [f(\xi(j,s)) - L_k(f, \xi(j,s))] + \sum_{j=1}^{N} \int_{K_j} L_k(f,x)dx.$$

We prove the following

Theorem 3.3. *The quadrature formula constructed above satisfies*

$$\varepsilon(f,k,d,m) \leq c'_{d,k} \frac{1}{m} \|f\| N^{-\frac{1}{2} - \frac{k}{d}}$$

and

$$r(f,k,d,m) \leq c''_{d,k} \frac{1}{m} \|f\| N^{-\frac{1}{2} - \frac{k}{d}},$$

where the constants $c'_{d,k}$ and $c''_{d,k}$ depend implicitly on the points $x^{(r)}$, but not on N.

Proof. One can see that

$$E\left\{\frac{1}{mn^d}\sum_{s=1}^{m}\left[f(\xi(j,s))-L_k(f,\xi(j,s))\right]+\int_{K_j}L_k(f,x)dx\right\}=\int_{K_j}f(x)dx$$

and

$$D\left\{\frac{1}{mn^d}\sum_{s=1}^{m}\left[f(\xi(j,s))-L_k(f,\xi(j,s))\right]+\int_{K_j}L_k(f,x)dx\right\}$$
$$=\frac{1}{m}D\left\{\frac{1}{n^d}\left[f(\xi(j,1))-L_k(f,\xi(j,1))\right]\right\}.$$

Note that

$$\int_{K_j}L_k(f,x)dx=\frac{1}{n^d}\sum_{0\le i_1+\ldots+i_d\le k-1}A(r)f(y^{(r)}),$$

where the coefficients $A(r)$ are the same for all cubes K_j and depend only on $\left\{x^{(r)}\right\}$. Using Taylor series expansion of f over the center of the cube K_j, one can see that

$$\left|f(\xi(s,t)-L_k(f,\xi(j,s)\right|\le c_{d,k}n^{-k}\left\|f\right\|,$$

and therefore

$$D\left\{\frac{1}{n^d}\left[f(\xi(j,s))-L_k(f,\xi(j,s))\right]\right\}\le c_{d,k}^{'}n^{-2d}n^{-2k}\left\|f\right\|^2.$$

Taking into account that the $\xi(j,s)$ are independent, we obtain that

$$D\left\{\sum_{j=1}^{n^d}\frac{1}{mn^d}\sum_{t=1}^{m}\left[f(\xi(j,s))-L_k(f,\xi(j,s))\right]+\int_{K_j}L_k(f,x)dx\right\}$$
$$\le n^d\frac{1}{m^2}mc_{d,k}^{'}n^{-2k}n^{-2d}\left\|f\right\|^2=\frac{1}{m}c_{d,k}^{'}n^{-d}n^{-2k}\left\|f\right\|^2$$

and therefore ($N=n^d$)

$$r(f,k,d,m)\ge\frac{1}{\sqrt{m}}c(d,k)N^{-\frac{1}{2}-\frac{k}{d}}\left\|f\right\|.$$

The application of the Tchebychev's inequality yields

$$\varepsilon(f,k,d,m)\le\frac{1}{\sqrt{m}}c_{d,k}^{'}\left\|f\right\|N^{-\frac{1}{2}-\frac{k}{d}}$$

for the probable error ε, where $c'(d,k)=\sqrt{2}c(d,k)$, which concludes the proof. □

One can replace the Lagrange approximation polynomial with other approximation schemes that use the function values at some fixed points, provided they are exact for all polynomials of degree less than k. In [Maire and DeLuigi (2006)] for Quasi-Monte Carlo integration of smooth functions an approach with Tchebychev polynomial approximation is developed. In [Dimov and Atanassov (2007)] we show that the proposed technique allows one to formulate optimal algorithms in the Hölder class of functions $\mathbf{H}_{\lambda}^{k}(\alpha, \mathbf{E}^{d}), (0 < \lambda \leq 1)$. The class $\mathbf{H}_{\lambda}^{k}(\alpha, \mathbf{E}^{d}), (0 < \lambda \leq 1)$ is defined as functions from $\mathbf{C}^{k}$, which derivatives of order k satisfy Hölder condition with a parameter λ:

$$\mathbf{H}_{\lambda}^{k}(\alpha, \mathbf{E}^{d}) \equiv \Big\{ f \in \mathbf{C}^{k} : \big| D^{k} f(y_1, \ldots, y_d) - D^{k} f(z_1, \ldots, z_d) \big| \leq \alpha \sum_{j=1}^{d} |y_j - z_j|^{\lambda} \Big\}. \tag{3.2}$$

In [Dimov and Atanassov (2007)], for the class $\mathbf{H}_{\lambda}^{k}(\alpha, \mathbf{E}^{d})$ we prove the following theorem:

Theorem 3.4. *The cubature formula constructed above satisfies*

$$r_N(f, k+\lambda, d, m) \leq c^{'}(d, k+\lambda) \frac{1}{m} \alpha N^{-\frac{1}{2} - \frac{k+\lambda}{d}}$$

and

$$\left(E \left(\int_{\mathbf{E}^d} f(x)dx - I(f) \right)^2 \right)^{1/2} \leq c^{''}(d, k+\lambda) \frac{1}{m} \alpha N^{-\frac{1}{2} - \frac{k+\lambda}{d}},$$

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

The above theorem shows that the convergence of the method can be improved by adding a parameter λ to the factor of smoothness k if the integrand belongs to the Hölder class of functions $\mathbf{H}_{\lambda}^{k}(\alpha, \mathbf{E}^{d}), (0 < \lambda \leq 1)$.

3.3 Estimates of the Computational Complexity

In this section two algorithms implementing our method are given. Estimates of the computational complexity of both algorithms are presented.

Algorithm 1 (A.1)

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

$$a_{ij}=\mu_i(x^{(j)})$$

is the smallest one.

Once it is selected for fixed k and d, the same set of points will be used for integrating every functions from the space $\mathbf{W}^k$. We do not need to store the co-ordinates of the points $x^{(j)}$, if we can record the state of the generator just before it produces the *best* set of $x^{(j)}$. Since the calculations of the elements of the matrix A and its inverse, as well as the coefficients of the interpolation type quadrature formula are made only once if we know the state of the pseudo-random number generator that will produce the set of points $x^{(j)}$, they count as $O(1)$ in our estimates for the number of flops used to calculate the integral of a certain function from $\mathbf{W}^k$. These calculations could be considered as *preprocessing*. We prove the following

Theorem 3.5. *The computational complexity of the numerical integration of a function from* $\mathbf{W}^k$ *using Algorithm A.1 is estimated by:*

$$N_{fp}\le N\left[m+\binom{d+k-1}{d}\right]a_f+mN\left[d\,(b_r+2)+1\right]+ \tag{3.3}$$
$$N\binom{d+k-1}{d}\left[2m+1+2d+2\binom{d+k-1}{d}+1\right]+c(d,k)$$

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

Proof. As it was pointed out above, the calculation of the coordinates of the points $x^{(r)}$, the elements of A and A^{-1}, and the coefficients of the interpolation type quadrature formula can be considered to be done with $c_1(d,k)$ flops, if we know the initial state of the pseudo-random number generator that produces the set of points $x^{(r)}$. Then in

$$2dN\binom{d+k-1}{d}$$

operations we calculate the coordinates of the points $y^{(r)}$ in each cube and in

$$mdN(b_r + 2)$$

flops we obtain the uniformly distributed random points we shall use. The calculation of the values of f at all these points takes

$$N\left[m + \binom{d+k-1}{d}\right] a_f$$

flops.

Next we apply the interpolation type quadrature formula with the previously calculated coefficients (reordering the terms) using

$$(N+1)\binom{d+k-1}{d}$$

operations. About the contribution of the Lagrange interpolation polynomials note that

$$S = \sum_{j=1}^{N}\sum_{s=1}^{m} L_k(f, \xi(j,s)) = \sum_{j=1}^{N}\sum_{s=1}^{m}\sum_{i=1}^{\binom{d+k-1}{d}} f\left(y^{(i)}\right) \sum_{r=1}^{\binom{d+k-1}{d}} t_{ir}\mu_r\left(\xi(j,s)\right),$$

where μ_r are all the monomials of degree less than k. Reordering the terms we obtain

$$S = \sum_{j=1}^{N}\sum_{i=1}^{\binom{d+k-1}{d}} f\left(y^{(i)}\right) \sum_{r=1}^{\binom{d+k-1}{d}} t_{ir} \sum_{s=1}^{m} \mu_r\left(\xi(j,s)\right).$$

Using the fact the the value of each monomial can be obtained using only one multiplication, once we know all the values of the monomials of lesser degree at the same point, we see that S can be calculated with less than

$$(2m-1)N\binom{d+k-1}{d} + \left(2\binom{d+k-1}{d} + 2\right)\binom{d+k-1}{d}N$$

flops. The summing of all function values at the random points $\xi(s,k)$ takes $mN-1$ flops. Now we sum all these estimates to obtain

$$\begin{aligned}
N_{fp} &\le 2dN\binom{d+k-1}{d} + mdN\left(b_r+2\right) \\
&\quad + N\left[m + \binom{d+k-1}{d}\right] a_f + (2m+1)\,N\binom{d+k-1}{d} \\
&\quad + 2N\binom{d+k-1}{d}^2 + (N+1)\binom{d+k-1}{d} + mN + c_1(d,k) \\
&= N\left[m + \binom{d+k-1}{d}\right] a_f + mN\left(d\left(b_r+2\right)+1\right) \\
&\quad + N\binom{d+k-1}{d}\left[2m+1+2d+2\binom{d+k-1}{d}+1\right] + c(d,k).
\end{aligned}$$

The theorem is proven. □

Algorithm 2 (A.2)

The second algorithm is a variation of the first one, when first some k different points $z_i^{(j)} \in (0,1)$ are selected in each dimension, and then the points $x^{(r)}$ have coordinates

$$\left\{(z_1^{(j_1)}, \ldots, z_d^{(j_d)}) : (j_1 + \cdots + j_d < k)\right\}.$$

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

$$L_k(f,\xi) = \sum_{j_1+\ldots+j_d<k} R(j_1, ..., j_d, 0, ..., 0) \prod_{i=1}^{d} \left(\xi_i(j,s) - w_i^{(1)}\right) \ldots$$
$$\ldots \left(\xi_i(j,s) - w_i^{(j_i-1)}\right),$$

where

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

and

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

if $j_i > l_i$.

In this modification we have the following

Theorem 3.6. *The computational complexity of the numerical integration of a function from* $\mathbf{W}^k$ *using Algorithm A.2 is estimated by:*

$$N_{fp} \le N \left[m + \binom{d+k-1}{d}\right] a_f + Nm\left[d(b_r + 2 + k) + 1\right]$$
$$+ N\binom{d+k-1}{d}(2d + 1 + 2m + 1) + c(d,k),$$

where a_f *and* b_m *are as above.*

Proof. One can see that for the calculation of the divided differences in d dimensions required for the Lagrange - Newton approximation we need to apply (3.4) exactly

$$dN\binom{d+k-1}{d+1}$$

times, which is less than

$$Nk\binom{d+k-1}{d}.$$

For the calculation of the sum

$$S=\sum_{j=1}^{N}\sum_{s=1}^{m}L_k\left(f,\xi\left(j,s\right)\right)=\sum_{j=1}^{N}\sum_{s=1}^{m}\sum_{j_1+\ldots+j_d<k}R(j_1,...,j_d,0,...,0)$$
$$\times\prod_{i=1}^{d}\left(\xi_i\left(j,s\right)-w_i^{(1)}\right)...\left(\xi_i\left(j,s\right)-w_i^{(j_i-1)}\right)$$

we make use of the fact that each term

$$\prod_{i=1}^{d}\left(\xi_i\left(j,s\right)-w_i^{(1)}\right)...\left(\xi_i\left(j,s\right)-w_i^{(j_i-1)}\right)$$

can be obtained from a previously computed one through one multiplication, provided we have computed all the dk differences $\xi_i\left(j,s\right)-w_i^{(r)}$. Thus, we are able to compute the sum S with less than

$$(2m+1)\binom{d+k-1}{d}N+dkmN$$

flops.

The other estimates are done in the same way as in Theorem 3.5 to obtain

$$\begin{aligned}N_{fp}&\leq 2dN\binom{d+k-1}{d}+mdN(b_r+2)+N\left[m+\binom{d+k-1}{d}\right]a_f\\&+(N+1)\binom{d+k-1}{d}+(2m+1)\binom{d+k-1}{d}N+dkmN\\&+2Nk\binom{d+k-1}{d}+mN+c_1(d,k)\\&=N\left[m+\binom{d+k-1}{d}\right]a_f+Nm\left(d(b_r+2+k)+1\right)\\&+N\binom{d+k-1}{d}\left(2d+1+2m+1\right)+c\left(d,k\right),\end{aligned}$$

which proves the theorem. □

3.4 Numerical Tests

In this section some numerical tests, showing the computational efficiency of the algorithms under consideration are given.

Here we present results for the following integrals:

$$I_1 = \int_{\mathbf{E}^4} \frac{e^{(x_1+2x_2)}\cos(x_3)}{1+x_2+x_3+x_4}\, dx_1dx_2dx_3dx_4;$$

$$I_2 = \int_{\mathbf{E}^4} x_1x_2^2e^{x_1x_2}\sin x_3\cos x_4\; dx_1dx_2dx_3dx_4;$$

$$I_3 = \int_{\mathbf{E}^4} e^{x_1}\sin x_2\cos x_3\log(1+x_4)\; dx_1dx_2dx_3dx_4;$$

$$I_4 = \int_{\mathbf{E}^4} e^{x_1+x_2+x_3+x_4}\; dx_1dx_2dx_3dx_4;$$

In the Tables 3.1 to 3.4 the results of some numerical experiments in the case when $k = d = 4$ are presented. They are performed on ORIGIN-2000 machine using only one CPU.

Some results are presented on Figures 3.1–3.4. The dependencies of the error on the number of points where the function values are taken when

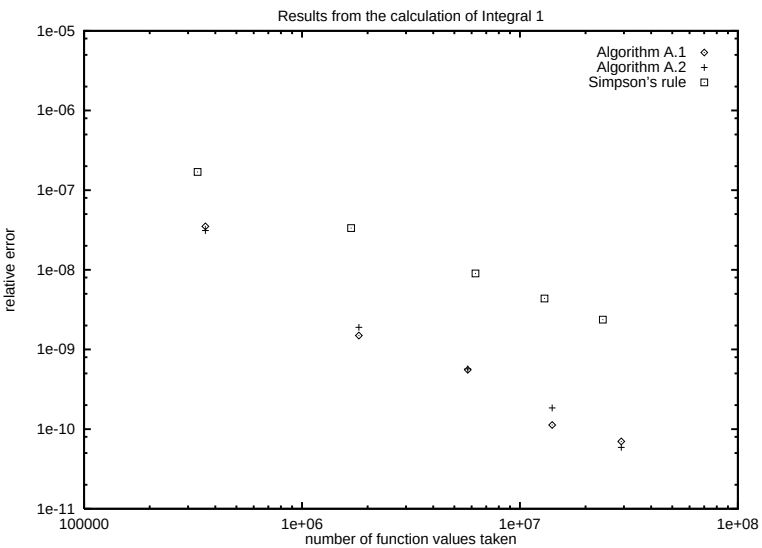


Fig. 3.1 Errors for the integral I_1.

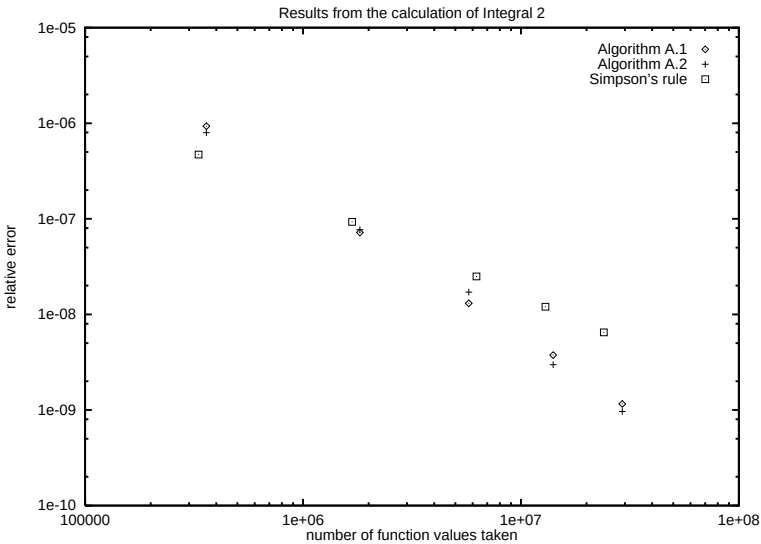


Fig. 3.2 Errors for the integral I_2.

Table 3.1 Results of MC numerical integration performed on ORIGIN-2000 for I_1 (Exact value - 1.8369031187...)

n	Algorithm	Error	$err_{rel} * N^6$	CPU time, s.
10	A.1	$3.51\ 10^{-8}$	0.04	0.68
10	A.2	$3.11\ 10^{-8}$	0.03	0.53
15	A.1	$1.49\ 10^{-9}$	0.02	3.40
15	A.2	$1.90\ 10^{-9}$	0.02	2.68
20	A.1	$5.56\ 10^{-10}$	0.04	10.71
20	A.2	$5.71\ 10^{-10}$	0.04	8.47
25	A.1	$1.12\ 10^{-10}$	0.03	26.12
25	A.2	$1.84\ 10^{-10}$	0.05	20.67
30	A.1	$7.01\ 10^{-11}$	0.05	54.15
30	A.2	$5.90\ 10^{-11}$	0.04	42.86

both algorithms are applied to the integral I_1 are presented on Figure 3.1. The same dependencies for the integral I_2 are presented on Figure 3.2. The results from the application of the iterated Simpson's method are provided for comparison.

The Figure 3.3 shows how increases the computational time for the calculation of integral I_2 when $n = N^{1/d}$ increases. On Figure 3.4 we

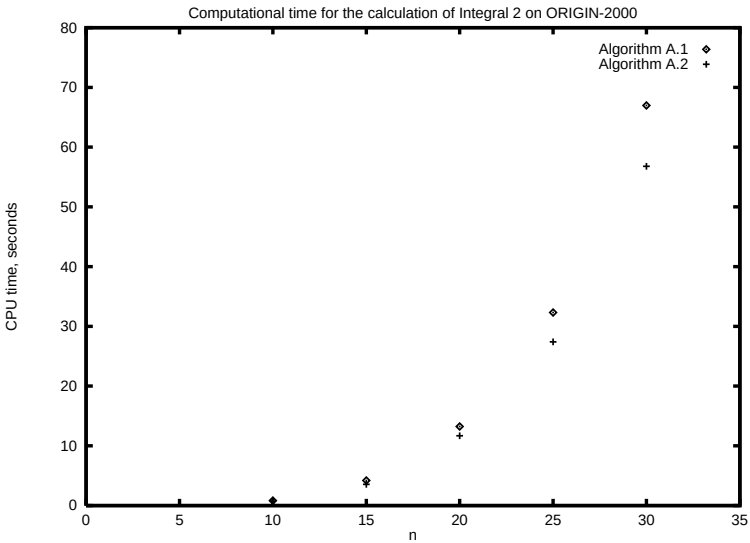


Fig. 3.3 Computational time for I_2.

Table 3.2 Results of MC numerical integration performed on ORIGIN-2000 for I_2 (Exact value - 0.1089748630...)

n	Algorithm	Error	$err_{rel} * N^6$	CPU time, s.
10	A.1	$9.31\ 10^{-7}$	0.93	0.84
10	A.2	$7.96\ 10^{-7}$	0.80	0.71
15	A.1	$7.20\ 10^{-8}$	0.81	4.20
15	A.2	$7.69\ 10^{-8}$	0.87	3.55
20	A.1	$1.31\ 10^{-8}$	0.83	13.24
20	A.2	$1.71\ 10^{-8}$	1.09	11.69
25	A.1	$3.75\ 10^{-9}$	0.92	31.49
25	A.2	$2.97\ 10^{-9}$	0.73	27.40
30	A.1	$1.16\ 10^{-9}$	0.84	66.97
30	A.2	$9.66\ 10^{-10}$	0.70	56.80

compare the CPU time and relative error of the calculation of integral I_2 using algorithm A.2 with two different values of the parameter m - 1 and 16. One can see that setting $m = 16$ yields roughly twice better results for the same amount of CPU time.

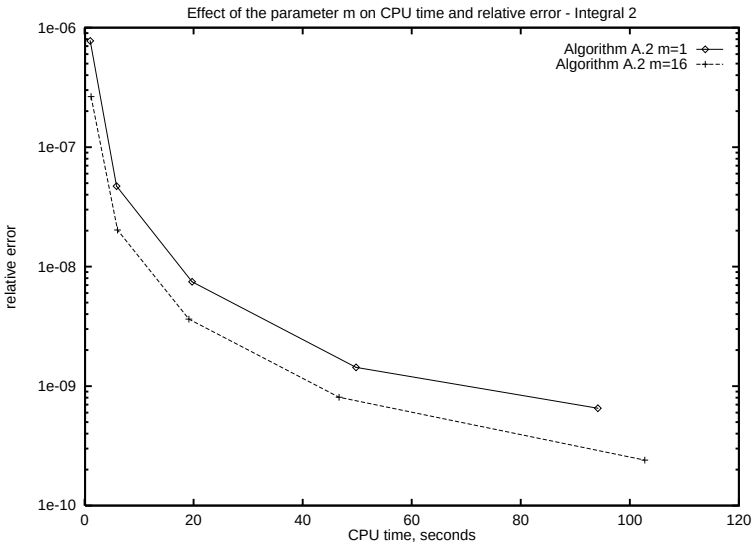


Fig. 3.4 CPU-time and relative error for I_2 (A.2).

Table 3.3 Results of MC numerical integration performed on ORIGIN-2000 for I_3 (Exact value - 0.2567581493).

n	Algorithm	Error	$err_{rel} * N^6$	CPU time, s.
10	A.1	$6.28\ 10^{-8}$	0.06	1.05
10	A.2	$8.22\ 10^{-8}$	0.08	0.92
15	A.1	$7.80\ 10^{-9}$	0.09	5.28
15	A.2	$5.55\ 10^{-9}$	0.06	4.65
20	A.1	$1.39\ 10^{-9}$	0.09	16.67
20	A.1	$1.39\ 10^{-9}$	0.09	14.69
25	A.1	$2.60\ 10^{-10}$	0.06	40.66
25	A.2	$4.77\ 10^{-10}$	0.11	35.82
30	A.1	$1.20\ 10^{-10}$	0.09	84.27
30	A.2	$1.98\ 10^{-10}$	0.14	74.24

3.5 Concluding Remarks

- A Monte Carlo method for calculating multidimensional integrals of smooth functions is presented and studied. It is proven that the method has the highest possible rate of convergence, i.e. it is an optimal super-

Table 3.4 Results of MC numerical integration performed on ORIGIN-2000 for I_4 (Exact value - 8.7172116201).

n	Algorithm	Error	$err_{rel} * N^6$	CPU time, s.
10	A.1	$7.00\ 10^{-8}$	0.06	0.46
10	A.2	$9.69\ 10^{-8}$	0.10	0.33
15	A.1	$3.80\ 10^{-9}$	0.04	2.27
15	A.2	$4.62\ 10^{-7}$	0.05	1.63
20	A.1	$7.77\ 10^{-10}$	0.05	7.15
20	A.2	$1.24\ 10^{-9}$	0.08	5.13
25	A.1	$3.11\ 10^{-10}$	0.08	17.44
25	A.2	$3.91\ 10^{-10}$	0.10	12.51
30	A.1	$5.99\ 10^{-11}$	0.04	36.13
30	A.2	$9.56\ 10^{-11}$	0.07	25.93

convergent method.

- Two algorithms implementing the method are described. Estimates for the computational complexity of both algorithms (A.1 and A.2) are presented.
- The numerical examples show that both algorithms give comparable results for the same number of points where the function values are taken.
- In all our examples Algorithm A.2 is quicker. It offers more possibilities for use in high dimensions and for functions with high order smoothness. We demonstrated how one can achieve better results for the same computational time using carefully chosen $m > 1$.
- Both algorithms are easy to implement on parallel machines, because the calculations performed for each cube are independent from that for any other. The fine granularity of the tasks and the low communication costs allow for efficient parallelization.
- It is important to know how expensive are the most important parts of the algorithms. Our measurements in the case of integral I_2 yield the following results :

  – Algorithm A.1:
    – 2% to obtain the uniformly distributed random points;
    – 73% to calculate all the values of f at all points;
    – 1% to apply the interpolation type quadrature formula;
    – 18% to calculate the Lagrange interpolation polynomial at the random points.

  - 6% other tasks
- Algorithm A.2:
  - 2% to obtain the uniformly distributed random points;
  - 86% to calculate the values of f at all points;
  - 1% to apply the interpolation type quadrature formula;
  - 5% to calculate the Lagrange-Newton approximation for the function f at all random points;
  - 6% other tasks.

This page intentionally left blank

Chapter 4

Iterative Monte Carlo Methods for 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 (see Chapters 2 and 3 as well as [Hammersley and Handscomb (1964); Sobol (1973)]). 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 - *systematic* and *stochastic*. 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.

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 k^{th} 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 analyse both systematic and stochastic errors. Sometimes the *iterative stationary linear Monte Carlo algorithms* are called *Power* Monte Carlo algorithms. The reason is that these algorithms find an approximation of a functional of *powers* of linear operators. In literature this class of algorithms is also known as Markov chain Monte Carlo since the statistical estimates can be considered as weights of Markov chains [Andrieu *et al.* (2003); Berg (2004); Gelfand and Smith (1990); Robert and Casella (2004)]. In Section 4.1 we consider how such weights can be defined. We are focusing on two kind of linear operators: integral operators and matrices.

4.1 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 $\mathbb{R}^d$ and $L = L(u)$ be a linear operator defined on $\mathbf{X}$.

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

$$u_k = L(u_{k-1}) + f, \;\; k = 1, 2, \ldots \tag{4.1}$$

The formal solution of (4.1) is the truncated Neumann series

$$u_k = f + L(f) + \cdots + L^{k-1}(f) + L^k(u_0), \qquad k > 0, \tag{4.2}$$

where L^k means the k^{th} iterate of L.

As an example consider the integral iterations.

Let $u(x) \in \mathbf{X}$, $x \in \Omega \subset \mathbb{R}^d$ 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

$$u = L(u) + f. \tag{4.3}$$

The truncation error of (4.2) 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

$$\mathbf{T}_{i+1} = \underbrace{I\!R^d \times I\!R^d \times \cdots \times I\!R^d}_{i \text{ times}}, \qquad i = 1, 2, \ldots, k, \tag{4.4}$$

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

$$J(u_k) \approx \frac{1}{N} \sum_{s=1}^{N} \{\theta_k\}_s, \tag{4.5}$$

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

The probable error r_N of (4.5) (see Definition 1.2 given in Introduction, as well as [Ermakov and Mikhailov (1982)]) 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 (4.3) becomes

$$u(x) = \int_\Omega l(x,y)u(y)dy + f(x) \text{ or } u = Lu + f. \tag{4.6}$$

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

$$J(u) = \int_\Omega h(x)u(x)dx = (u,h). \tag{4.7}$$

In fact, equation (4.7) defines an inner product of a given function $h(x) \in \mathbf{X}$ with the solution of the integral equation (4.3).

Sometimes, the adjoint equation

$$v = L^*v + h \tag{4.8}$$

will be used.

In (4.8) $v, h \in \mathbf{X}^*$, $L^* \in [\mathbf{X}^* \to \mathbf{X}^*]$, where $\mathbf{X}^*$ is the dual functional space to $\mathbf{X}$ and L^* is an adjoint operator.

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

$$\| f \|_{\mathbf{L}_1} = \int_\Omega | f(x) | dx;$$

$$\| L \|_{\mathbf{L}_1} \leq \sup_x \int_\Omega | l(x,x') | dx'. \tag{4.9}$$

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

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

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 (4.7) 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 also 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 $L^2u(x), \ldots, L^iu(x), \ldots$ also belong to $\mathbf{L}_2(\Omega)$.

Obviously, if $u \in \mathbf{L}_1$ and $h \in \mathbf{L}_\infty$ the inner product (4.7) 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 [Sabelfeld (1989)], it is possible to consider 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 the 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 (4.6) by v and (4.8) by u and integrate. We obtain

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

Since

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

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

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 (4.7). 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 ξ_i $(i = 1, 2, \ldots, N)$. 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, Definition 1.3 given in Introduction). Markov chain Monte Carlo is also known as the Metropolis or the Metropolis-Hastings method [Metropolis *et al.* (1953); Green and Mira (2001); Haario *et al.* (2001); Roberts and Rosenthal (2001)]. Here this method is considered as a special class of iterative stochastic methods, since such a consideration helps to obtain some error analysis results.

An approximate value of the linear functional J, defined by (4.7), 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

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

where $W_0 = 1$; $W_j = W_{j-1} \dfrac{l(\xi_{j-1}, \xi_j)}{p(\xi_{j-1}, \xi_j)}, j = 1, 2, \ldots,$ 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 (4.2) can be written in the following form:

$$u_k = L^k u_0 + L^{k-1} f + \cdots + Lf + f = (I - L^k)(I - L)^{-1} f + L^k u_0, \quad (4.10)$$

where I is the unit (identity) matrix; $L = (l_{ij})_{i,j=1}^{n}$; $u_0 = (u_1^0, \ldots, u_n^0)^T$ 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 then there exists a vector u such that

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

which satisfies the equation

$$u = Lu + f \quad (4.11)$$

(see, for example, [Golub and Van-Loan (1983)]).

Now consider the problem of evaluating the inner product

$$J(u) = (h, u) = \sum_{i=1}^{n} h_i u_i \,, \tag{4.12}$$

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

To define a random variable whose mathematical expectation coincides with the functional (4.12) for the system (4.14) first consider the integral equation (4.6) for which $\Omega = [0, n)$ is an one-dimensional interval divided into equal subintervals $\Omega_i = [i-1, i)$, $i = 1, 2, \ldots n$ such that

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

Then the integral equation (4.6) becomes

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

for $u_i \in \Omega_i$. Denote

$$u_j = \int_{\Omega_j} u(y) dy \tag{4.13}$$

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

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

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

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

or in a matrix form

$$u = Lu + f \;, \tag{4.14}$$

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

The above presentation permits the consideration of the following random variable

$$\theta[h] = \frac{h_{\alpha_0}}{p_0} \sum_{\nu=0}^{\infty} W_\nu f_{\alpha_\nu} \;, \tag{4.15}$$

where

$$W_0 = 1; \qquad W_\nu = W_{\nu-1} \frac{l_{\alpha_{\nu-1},\alpha_\nu}}{p_{\alpha_{\nu-1},\alpha_\nu}}, \qquad \nu = 1, 2, \ldots \tag{4.16}$$

and $\alpha_0, \alpha_1, \ldots$ is a Markov chain on elements of the matrix L created by using an initial probability p_0 and a transition probability $p_{\alpha_{\nu-1},\alpha_\nu}$ for choosing the element $l_{\alpha_{\nu-1},\alpha_\nu}$ of the matrix L.

4.2 Solving Linear Systems and Matrix Inversion

Consider a matrix L:

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

and a vector

$$f = (f_1, \ldots, f_n)^T \in I\!R^{n\times 1}$$

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

$$Lf \in I\!R^{n\times 1} \tag{4.17}$$

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

Since iterative Monte Carlo algorithms using the transformation (4.17) will be considered, the linear transformation (4.17) will be called an *iteration*. The algebraic transformation (4.17) 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

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

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

$$Au = b,$$

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

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

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 (4.15), (4.16) become a *stationary linear iterative Monte Carlo algorithm.*

As a result the convergence of the Monte Carlo algorithm depends on truncation error of (4.10).

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

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

where $A \in I\!R^{n\times n}$ is a given real matrix. The matrix inversion is not often used in practical computations since this operation is computationally expensive. Nevertheless, for some problem (like approximate finding of good matrix preconditioners) algorithms for matrix inversion with relatively low accuracy could be very important as a part of some numerical solvers.

Assume 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 (4.10).

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

$$L = I - A$$

can be considered.

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

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

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

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

Definition 4.1. The initial density vector $p = \{p_i\}_{i=1}^n$ is called *permissible* to the vector $h = \{h_i\}_{i=1}^n \in I\!R^n$, i.e. $p \in \mathcal{P}_h$, if

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

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

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

Note that the set of *permissible* densities is a subset of *tolerant* densities, defined with respect to density functions in Section 2.3.4 (see Definition 2.1).

Consider the following Markov chain:

$$T_i = \alpha_0 \to \alpha_1 \to \ldots \to \alpha_i, \tag{4.18}$$

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

The rules for defining the chain (4.18) are:

$$Pr(\alpha_0 = \alpha) = p_\alpha, \quad Pr(\alpha_j = \beta | \alpha_{j-1} = \alpha) = p_{\alpha\beta}. \tag{4.19}$$

Assume that

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

Now define the random variables W_ν using the formula (4.16). One can see, that the random variables W_ν, where $\nu = 1, \ldots, i$, can also be considered as weights on the Markov chain (4.19).

From all possible *permissible* densities we choose the following

$$p = \{p_\alpha\}_{\alpha=1}^n \in \mathcal{P}_h, \quad p_\alpha = \frac{|h_\alpha|}{\sum_{\alpha=1}^n |h_\alpha|}; \tag{4.20}$$

$$P = \{p_{\alpha\beta}\}_{\alpha,\beta=1}^n \in \mathcal{P}_L, \quad p_{\alpha\beta} = \frac{|l_{\alpha\beta}|}{\sum_{\beta=1}^n |l_{\alpha\beta}|}, \alpha = 1, \ldots, n. \tag{4.21}$$

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 $p = \{p_\alpha\}_{\alpha=1}^n$ is called *almost optimal initial density vector* and the transition density matrix $P = \{p_{\alpha\beta}\}_{\alpha,\beta=1}^n$ is called *almost optimal density matrix* [Dimov (1991)]. Such density distributions lead to almost optimal algorithms in the sense that for a class of matrices A and vectors h such a choice coincides with optimal weighted algorithms defined in [Ermakov and Mikhailov (1982)] and studied in [Mikhailov and Sabelfeld (1992)] (for more details see [Dimov (1991)]). The reason to use MAO instead of Uniform Monte Carlo is that MAO normally gives much smaller variances. On the other hand, the truly optimal weighted algorithms are very time consuming, since to define the optimal densities one needs to solve an additional integral equation with a quadratic kernel. This procedure makes the optimal algorithms very expensive. A significant progress in approximating optimal densities using approximate solution of adjoint integral equation is in [Medvedev and Mikhailov (2007)]. One should take into

account that the procedure proposed in [Medvedev and Mikhailov (2007)], called *partial value* modeling, is applied for a specific class of problems.

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 $\delta_q (q = 1, 2, \dots)$. Assume $\delta_q = 0$ if the trajectory is broken (absorbed) and $\delta_q = 1$ in other cases. Let

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

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:

$$E\{W_i f_{\alpha_i}\} = (h, L^i f), \quad i = 1, 2, \dots;$$

$$E\left\{\sum_{i=0}^{n} W_i f_{\alpha_i}\right\} = (h, u), \quad (P1),$$

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

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

4.3 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 converges slowly.

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

$$u - \lambda L u = f, \tag{4.22}$$

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 $\lambda_1, \lambda_2, \dots$ be the eigenvalues of (4.22), where it is supposed that

$$|\lambda_1| \geq |\lambda_2| \geq \dots$$

Monte Carlo algorithms are based on the representation

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

where

$$R_\lambda = L + \lambda L^2 + \dots , \tag{4.23}$$

The systematic error of (4.23), when m terms are used, is

$$r_s = O[(|\lambda|/|\lambda_1|)^{m+1} m^{\rho-1}], \tag{4.24}$$

where ρ is the multiplicity of the root λ_1.

From (4.24) is follows that when λ is approximately equal to λ_1 the sequence (4.23) and the corresponding Monte Carlo algorithm converges slowly. When $\lambda \geq \lambda_1$ the algorithm does not converge.

Obviously, the representation (4.23) 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 defined when the sequence (4.23) does not converge.

To answer these questions we apply a *mapping* of the spectral parameter λ in (4.22).

The algorithm under consideration follows an approach which is similar to the resolvent analytical continuation method used in functional analysis [Kantorovich and Krylov (1964); Kantorovich and Akilov (1982)] and in integral equations [Sabelfeld (1989)]. In [Kantorovich and Akilov (1982)] the mapping approach is used for solving problems of numerical analysis.

To extend these results it is necessary to show that the mapping approach can be applied for any linear operators (including matrices). Consider the problem of constructing the solution of (4.22) for $\lambda \in \Omega$ and $\lambda \neq \lambda_k, k = 1, 2, \dots$, 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

$$R_{\psi(\alpha)} f = \sum_{j=0}^{\infty} b_j \alpha^j \ , \tag{4.25}$$

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 (4.25) converges; hence the solution of the functional equation (4.22) 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 (4.23) 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 ($\lambda = \lambda_* = 1$):

$$\lambda = \psi(\alpha) = \frac{4a\alpha}{(1-\alpha)^2}. \tag{4.26}$$

The sequence $R_{\psi(\alpha)} f$ for the mapping (4.26) converges absolutely and uniformly [Kantorovich and Akilov (1982)].

In Monte Carlo calculations we cut the sequence in (4.25) after m terms

$$R_{\lambda_*} f \approx \sum_{k=1}^{m} b_k \alpha_k^k = \sum_{k=1}^{m} \alpha_*^k \sum_{i=1}^{k} d_i^{(k)} c_i = \sum_{k=1}^{m} g_k^{(m)} c_k, \tag{4.27}$$

where

$$g_k^{(m)} = \sum_{j=k}^{m} d_k^{(j)} \alpha_*^j. \tag{4.28}$$

Table 4.1 Table of the coefficients $q_{k,j} = (4a)^{-k} d_k^{(j)}$ 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

The coefficients

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

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

The coefficients $d_k^{(j)}$ for the mapping (4.27) are calculated and presented in Table 4.1 (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 (4.26) creates the following Monte Carlo iteration process

$$\begin{aligned} u_0 &= f, \\ u_1 &= 4aLu_0, \\ u_2 &= 4aLu_1 + 2u_1, \\ u_3 &= 4aLu_2 + 2u_2 - u_1, \\ u_j &= 4aLu_{j-1} + 2u_{j-1} - u_{j-2}, \ \ j \geq 3. \end{aligned} \tag{4.29}$$

and from (4.29) we have

$$u^{(k)} = 4a\alpha Lu^{(k-1)} + 2\alpha u^{(k-1)} - \alpha^2 u^{(k-2)} + f(1-\alpha^2), \ \ k \geq 3.$$

4.4 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 α_0 was defined in Section 4.2 as follows

$$T_i = \alpha_0 \to \alpha_1 \to \cdots \to \alpha_j \to \cdots \to \alpha_i,$$

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

Assume that

$$P(\alpha_0 = \alpha) = p_\alpha, \quad P(\alpha_j = \beta | \alpha_{j-1} = \alpha) = p_{\alpha\beta},$$

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 $P \in I\!R^{n\times n}$. We require that

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

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 (4.12) $J(u) = (h, u) = \sum_{\alpha=1}^{n} h_\alpha u_\alpha$ of a given vector h with the vector solution of the system (4.14).

Define the random variable $\theta_m^*[h]$

$$\theta_m^*[h] = \frac{h_{\alpha_0}}{p_0} \sum_{\nu=0}^{m} g_\nu^{(m)} W_\nu f_{\alpha_\nu}, \tag{4.30}$$

where $W_0 = 1$, $g_0^{(m)} = 1$ and

$$W_\nu = W_{\nu-1} \frac{l_{\alpha_{\nu-1},\alpha_\nu}}{p_{\alpha_{\nu-1},\alpha_\nu}}, \qquad \nu = 1, 2, \ldots,$$

($\alpha_0, \alpha_1, \alpha_2, \ldots$ is a Markov chain with initial density function p_{α_0} and transition density function $p_{\alpha_{\nu-1},\alpha_\nu}$) and coefficients $g_j^{(m)}$ are defined by (4.28) for $j \geq 1$.

The following theorem holds:

Theorem 4.1. *Consider matrix L, whose Neumann series (4.23) does not necessarily converge. Let (4.26) be the required mapping, so that the presentation (4.27) exists. Then*

$$E\left\{ \lim_{m\to\infty} \frac{h_{\alpha_0}}{p_0} \sum_{\nu=0}^{m} g_\nu^{(m)} W_\nu f_{\alpha_\nu} \right\} = (h, u).$$

Proof.

First consider the density of the Markov chain $\alpha_0 \to \alpha_1 \to \ldots \to \alpha_i$ as a point in $m(i+1)$-dimensional Euclidian space $\mathbf{T_{i+1}} = \underbrace{I\!R^n \times \ldots \times I\!R^n}_{\mathbf{i+1}}$:

$$P\{\alpha_0 = t_0, \alpha_1 = t_1, \ldots, \alpha_i = t_i\} = p_0 p_{t_0 t_1} p_{t_1 t_2} \ldots p_{t_{i-1} t_i}.$$

Now calculate the mathematical expectation of the random variable

$$\frac{h_{\alpha_0}}{p_0} g_\nu^{(m)} W_\nu f_{\alpha_\nu}.$$

From the definition of the mathematical expectation it follows that:

$$\begin{aligned} E\left\{\frac{h_{\alpha_0}}{p_0} g_\nu^{(m)} W_\nu f_{k\nu}\right\} &= \sum_{t_0,\ldots,t_\nu=1}^{m} \frac{h_{t_0}}{p_0} g_\nu^{(m)} W_\nu f_{t_\nu} p_0 p_{t_0 t_1} \ldots p_{t_{\nu-1} t_\nu} \\ &= \sum_{t_0,\ldots,t_\nu=1}^{m} h_{t_0} l_{t_0 t_1} l_{t_1 t_2} \ldots l_{t_\nu - 1} t_\nu f_{t\nu} = (h, L^\nu f). \end{aligned}$$

The existence and convergence of the sequence (4.28) ensures the following representations:

$$\begin{aligned} &\sum_{\nu=0}^{m} E\left|\frac{h_{\alpha_0}}{p_0} g_\nu^{(m)} W_\nu f_{k\nu}\right| = \sum_{\nu=0}^{m} (|h|, |L|^\nu |f|) = \left(|h|, \sum_{\nu=0}^{m} |L|^\nu |f|\right), \\ &E\left\{\lim_{m\to\infty} \frac{h_{\alpha_0}}{p_0} \sum_{\nu=0}^{m} g_\nu^{(m)} W_\nu f_{\alpha_\nu}\right\} \\ &= \sum_{\nu=0}^{\infty} E\left\{\frac{h_{\alpha_0}}{p_0} g_\nu^{(m)} W_\nu f_{\alpha_\nu}\right\} = \sum_{\nu=0}^{\infty} (h, L^\nu f) = (h, u). \end{aligned}$$

□

This theorem permits the use of the random variable $\theta_m^*[h]$ for calculating the inner product (4.12).

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

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

where the one is in the "r^{th}" position and "T" means transposition. It follows that

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

and the corresponding Monte Carlo algorithm is given by

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

where N is the number of chains and

$$\theta_m^*[e(r)]_s = \sum_{\nu=0}^{m} g_\nu^{(m)} W_\nu f_{\alpha_\nu};$$

$$W_\nu = \frac{l_{r\alpha_1} l_{\alpha_1\alpha_2}\ldots l_{\alpha_{\nu-1}\alpha_\nu}}{p_{r\alpha_1} p_{\alpha_1\alpha_2}\ldots p_{\alpha_{\nu-1}p_\nu}}.$$

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

$$L = I - A, \tag{4.31}$$

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 (4.22) be the following unit vector

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

Theorem 4.2. *Consider matrix L, whose Neumann series (4.23) does not necessarily converge. Let (4.26) be the required mapping, so that representation (4.27) exists. Then*

$$E\left\{\lim_{m\to\infty}\sum_{\nu=0}^{m} g_\nu^{(m)} \frac{l_{r\alpha_1} l_{\alpha_1\alpha_2}\ldots l_{\alpha_{\nu-1}\alpha_\nu}}{p_{r\alpha_1} p_{\alpha_1\alpha_2}\ldots p_{\alpha_{\nu-1}p_\nu}} f_{r'}\right\} = c_{rr'}.$$

Proof. The proof is similar to the proof of Theorem 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 $f_{k\nu}$:

$$E\left\{\frac{e(r)}{1} g_\nu^{(m)} W_\nu f_{\alpha_\nu}\right\} = (e(r), L^\nu f) = (L^\nu f)_r\,.$$

So, in this case the "r^{th}" component of the solution is estimated:

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

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

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

that is:

$$E\left\{\lim_{m\to\infty}\sum_{\nu=0}^{m} g_\nu^{(m)} \frac{l_{r\alpha_1} l_{\alpha_1\alpha_2}\ldots l_{\alpha_{\nu-1}\alpha_\nu}}{p_{r\alpha_1} p_{\alpha_1\alpha_2}\ldots p_{\alpha_{\nu-1}\alpha_\nu}} e(r')\right\}$$
$$= \lim_{m\to\infty}\sum_{\nu=0}^{\infty} E\left\{g_\nu^{(m)} \frac{l_{r\alpha_1} l_{\alpha_1\alpha_2}\ldots l_{\alpha_{\nu-1}\alpha_\nu}}{p_{r\alpha_1} p_{\alpha_1\alpha_2}\ldots p_{\alpha_{\nu-1}\alpha_\nu}} e(r')\right\} = \sum_{\nu=0}^{\infty}(e(r), L^\nu e(r'))$$
$$= \left(e(r), \sum_{\nu=0}^{\infty} L^\nu e(r')\right) = \sum_{i=1}^{n} c_{ri} e(r') = c_{rr'}.$$

□

Theorem 4.2 permits the use of the following Monte Carlo algorithm for calculating elements of the inverse matrix C:

$$c_{rr'} \approx \frac{1}{N}\sum_{s=1}^{N}\left[\sum_{(\nu|\alpha_\nu = r')}^{m} g_\nu^{(m)} \frac{l_{r\alpha_1} l_{\alpha_1\alpha_2}\ldots l_{\alpha_{\nu-1}\alpha_\nu}}{p_{r\alpha_1} p_{\alpha_1\alpha_2}\ldots p_{\alpha_{\nu-1}p_\nu}}\right]_s,$$

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

$$W_\nu^{(m)} = g_\nu^{(m)} \frac{l_{r\alpha_1} l_{\alpha_1\alpha_2}\ldots l_{\alpha_{\nu-1}\alpha_\nu}}{p_{r\alpha_1} p_{\alpha_1\alpha_2}\ldots p_{\alpha_{\nu-1}p_\nu}}$$

for which $\alpha_\nu = r'$ are included in the sum (4.31).

Observe that since $W_\nu^{(m)}$ is only contained in the corresponding sum for $r' = 1, 2, \ldots, n$ 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.

4.5 Balancing of Errors

As it was mentioned in the introduction of Chapter 4 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 m, i.e., $T = E(m)$.

Let us consider the case when the algorithm is applied to **Problem 1**. Using the mapping procedure and a random variable, defined by (4.30), we accelerate the convergence of the algorithm proposed in [Curtiss (1954, 1956)]. This means that for a fixed number of steps m

$$r_s(m) < r_s^{(C)}(m), \tag{4.32}$$

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

$$\sigma(\theta^*) < \sigma(\theta) \tag{4.33}$$

and thus

$$r_N(\sigma(\theta^*)) < r_N^{(C)}(\sigma(\theta)), \tag{4.34}$$

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

Next consider the general error

$$R = r_N(\sigma) + r_s(m)$$

for matrix inversion by our Monte Carlo approach. Let R be fixed. Obviously from (4.32) and (4.33) it follows that there exist constants $c_s > 1$ and $c_N > 1$, such that

$$r_s^{(C)}(m) = c_s r_s,$$

$$r_N^{(C)}(\sigma) = c_N r_N.$$

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

$$R = R^{(C)} = r_N^{(C)}(\sigma) + r_s^{(C)}(m) = c_N r_N(\sigma) + c_s r_s(m).$$

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

$$\frac{c\sigma(\theta)}{N_c^{1/2}} + r_s^{(C)}(m) = \frac{cc_N\sigma(\theta^*)}{N_c^{1/2}} + c_s r_s(m)$$

$$= \frac{c\sigma(\theta^*)}{N^{1/2}} + r_s(m),$$

or

$$\frac{c\sigma(\theta^*)}{N^{1/2}} = \frac{cc_N\sigma(\theta^*)}{N_c^{1/2}} + (c_s - 1)r_s(m)$$

and

$$\frac{1}{N^{1/2}} = \frac{c_N}{N_c^{1/2}} + \frac{(c_s - 1)r_s(m)}{c\sigma(\theta^*)}, \tag{4.35}$$

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

Denote by b the following strictly positive variable

$$b = \frac{(c_s - 1)r_s(m)}{c\sigma(\theta^*)} > 0. \tag{4.36}$$

From (4.35) and (4.36) we obtain:

$$N = \frac{N_c}{\left(c_N + bN_c^{1/2}\right)^2}. \tag{4.37}$$

The result (4.37) 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 (4.36) 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 Monte Carlo iterative process [Megson *et al.* (1994)]. In section 4.6 we explore some of these possibilities and develop some comparisons.

4.6 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 ([Dimov (1991)]) $p_{\alpha\beta}$ is chosen proportional to the $|l_{\alpha\beta}|$ (see, (4.21)). 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 $|W_\nu^{(m)}| < \delta$;
- **(ii)** when a chain enters an absorbing state (see, Definition 1.4 in the Introduction).

In the case of a Monte Carlo algorithm without any absorbing states ($f_{\alpha_j} = \delta_{\alpha_j \beta}$ 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

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

where the matrix norm $\|L\|$ is defined as $\|L\| = \|L\|_1 = \max_j \sum_{i=1}^{n} |l_{ij}|$. Consider the Monte Carlo algorithms with absorbing states (see, Section 4.2) 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 $D\hat{\theta}[h]$ ([Curtiss (1954, 1956)]) if the chain starts in state $r = \alpha$ and $p_{\alpha\beta} = |l_{\alpha\beta}|$, for $\alpha, \beta = 1, 2, ..., n$ 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 $N >> n$ 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 [Dimov and Tonev (1993b)] show that $T \approx \sqrt{N}$, for sufficiently large N.

4.7 A Refined Iterative Monte Carlo Approach for Linear Systems and Matrix Inversion Problem

In this Section a refined 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 control of 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.

4.7.1 *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:

$$Au = b, \tag{4.38}$$

where

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

The inverse matrix problem is equivalent to solving m-times the problem (4.38), i.e.

$$Ac_j = b_j, \quad j = 1, \ldots, n \tag{4.39}$$

where

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

and

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

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

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

$$L = I - DA, \tag{4.40}$$

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

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

The system (4.38) can be presented in the following form:

$$u = Lu + f, \tag{4.41}$$

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 4.7.2. Obviously, if A is a diagonally dominant matrix, then the elements of the matrix L must satisfy the following condition:

$$\sum_{j=1}^{n} |l_{ij}| \leq 1 \qquad i = 1, \ldots, n. \tag{4.42}$$

4.7.2 *Refined Iterative Monte Carlo Algorithms*

Here refined iterative Monte Carlo algorithms are considered. The first algorithm evaluates every component of the solution u of the following linear algebraic system (4.38).

Algorithm 4.1.

1. **Input** initial data: the matrix A, the vector $\mathbf{b}$, the constants ε , γ and N.
2. Preliminary calculations (preprocessing):

   2.1. **Compute** the matrix L using the parameter $\gamma \in (0, 1]$:

   $$\{l_{ij}\}_{i,j=1}^{n} = \begin{cases} 1 - \gamma & \text{when} \quad i = j \\ -\gamma \frac{a_{ij}}{a_{ii}} & \text{when} \quad i \neq j \,. \end{cases}$$

   2.2. **Compute** the vector $lsum$:

   $$lsum(i) = \sum_{j=1}^{n} |l_{ij}| \quad \text{for} \quad i = 1, 2, \ldots, n.$$

   2.3. **Compute** the transition probability matrix $P = \{p_{ij}\}_{i,j=1}^{n}$, where

   $$p_{ij} = \frac{|l_{ij}|}{lsum(i)}, \quad i = 1, 2, \ldots, n \quad j = 1, 2, \ldots, n \,.$$

3. **For** $i_0 := 1$ **to** n **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** $W := Wsign(l_{i_0 j}) \times lsum(i_0)$;
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 Algorithm 4.1.

Algorithm 4.1 describes the evaluation of every component of the solution of the problem (4.38), which is, in fact, linear algebraic system. Algorithm 4.1 is considered separately, since it (or some of its steps) will be used in algorithms. For finding the corresponding "i"th component of the solution the following functional is used

$$J(u) = (v, u),$$

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

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 4.1.

Algorithm 4.2.

1. **Input** initial data: the matrix A, the constant ε, N and the vector $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_l) \in (0, 1]^l$.
2. **For** $j_0 := 1$ **to** n **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** Algorithm 4.1 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 $\hat{c}^{\alpha}_{j_0} = (c^k_{1j_0}, \ldots, c^k_{nj_0})$.

2.3. **Compute** the l_2-norm of the column - vector $\hat{c}^k_{j_0}$:

$$r^k_{j_0} = \sum_{j=1}^{n} \left\{ \sum_{i=1}^{n} a_{ji} c^k_{ij_0} - \delta_{jj_0} \right\}^2 .$$

2.4. **If** $(r^{\alpha}_{j_0} < r)$ **then**

$\hat{c}_{j_0} := c^k_{j_0}$;

$r := r^k_{j_0}$.

3. **End** of Algorithm 4.2.

Algorithm 4.2 is based on Algorithm 4.1 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:

$$E_j = A\hat{C}_j - I_j^T, \quad j = 1, 2, \ldots, n, \tag{4.43}$$

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 4.3.

1. **Input** initial data: the matrix A, the constant N and the vectors $\gamma = (\gamma_1, \gamma_2, \ldots, \gamma_l) \in (0, 1]^l$, $\varepsilon = (\varepsilon_1, \varepsilon_2, \ldots, \varepsilon_n)$ and $r = (r_1, r_2, \ldots, r_N)$.

2. **While** $(k \le l)$ **do**

   2.1. Step 2 of Algorithm 4.1 for $\gamma := \gamma_k$.

   2.2. **While** $(i_0 \le n)$ and $(j_0 \le n)$ **do** step 4 and step 5 of Algorithm 4.1 to compute the elements $\hat{c}^{\alpha}_{i_0 j_0}$ 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** n **do**

      2.3.1. **Calculate**

$$r^{\alpha}_{i_0} = \max_{i \in \{1,2,\ldots,n\}} \left| \sum_{j=1}^{n} c_{i_0 j} a_{ji} - \delta_{i_0 i} \right| .$$

      2.3.2. **If** $(r^{\alpha}_{i_0} < r_{i_0})$ **then**

$$\hat{c}_{i_0} := c^{k}_{i_0};$$
$$r_{i_0} := r^{k}_{i_0}.$$

3. **End** of Algorithm 4.3.

The difference between the last two algorithms is that Algorithm 4.3 can not be applied in traditional (non-stochastic) iterative algorithms. The traditional algorithms allow one to evaluate the columns of the inverse matrix in parallel, but they do not allow one 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 one 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 4.3 also depends on "how ill-conditioned" is the given row of the matrix A. The given row A_i of the matrix A is "ill-conditioned", when the condition

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

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

Algorithm 4.3 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:

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

The possibility to treat non-diagonally dominant 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 a parameter ε. For *a posteriori* criteria we use the minimization of the C-norm of the following row-vector

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

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

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

$$E = \hat{C}A - I. \tag{4.44}$$

The above mentioned procedure allows the minimization of 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) to have a good balance between the stochastic and systematic error (i.e., the truncation error). The problem of balancing of the systematic and stochastic errors is very important when Monte Carlo algorithms are used. It is clear that in order to obtain good results 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 an increase in 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 than in the case of lower convergence. Our approach permits use of different stop-criteria for different random trajectories, which allows the optimization of the algorithm in the sense of balancing errors.

4.7.3 *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

$$\left| \begin{array}{l} -\mathrm{div}(a(x)\underline{\nabla} p) = f(x), \text{ in } \Omega \\ p = 0, \text{ on } \partial\Omega, \end{array} \right. \tag{4.45}$$

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

$$\underline{u} \equiv (u_1, u_2) = a(x)\underline{\nabla} p, \quad \alpha_i(x) = a_i(x)^{-1}, i = 1, 2.$$

Let us consider the spaces $\underline{V}$ and W defined by

$$\underline{V} = \underline{H}(div; \Omega) = \{\underline{v} \in L^2(\Omega)^2 \ : \ div\ \underline{v} \in L^2(\Omega)\},$$
$$W = L^2(\Omega)$$

provided with the norms

$$\|\underline{v}\|_{\underline{V}} \equiv \|\underline{v}\|_{\underline{H}(div;\Omega)} \ = \ (\ \|\underline{v}\|^2_{0,\Omega} \ + \ \|div\ \underline{v}\|^2_{0,\Omega})^{1/2} \ \text{ and}$$
$$\|w\|_W = \|w\|_{L^2(\Omega)} \ = \ \|w\|_{0,\Omega}$$

respectively.

Then the mixed variational formulation of the problem (4.45) is given by characterizing the pair $(\underline{u}, p)$, as the solution of

$$\left| \begin{array}{ll} a(\underline{u}, \underline{v}) \ + \ b(\underline{v}, p) \ = \ 0, & \forall \underline{v} \in \underline{V}; \\ b(\underline{u}, w) \ = \ -\,(f, w), & \forall w \in W, \end{array} \right. \tag{4.46}$$

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 (4.46) in the Raviart-Thomas spaces leads to the following linear algebraic system:

$$Ku = \begin{pmatrix} A_1 & 0 & B_1 \\ 0 & A_2 & B_2 \\ B_1^T & B_2^T & 0 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ p \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ -f \end{pmatrix}, \tag{4.47}$$

where A_i are $n \times n$ matrices, B_i are $n \times n_1$ matrices $(n_1 < n)$, $u_i \in I\!R^n$ and $p, f \in I\!R^{n_1}$, $i = 1, 2$.

If $A_i^{-1}\,(i = 1, 2)$ is obtained then the system (4.47) 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 $A = A_i$, $i = 1, 2$ Algorithm 4.2 is applied. Numerical examples for a matrix $A \in I\!R^{16\times 16}$, 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 4.2.

Table 4.2 Connection between ε and the parameter γ. Here $m = 16$, $n = 24$.

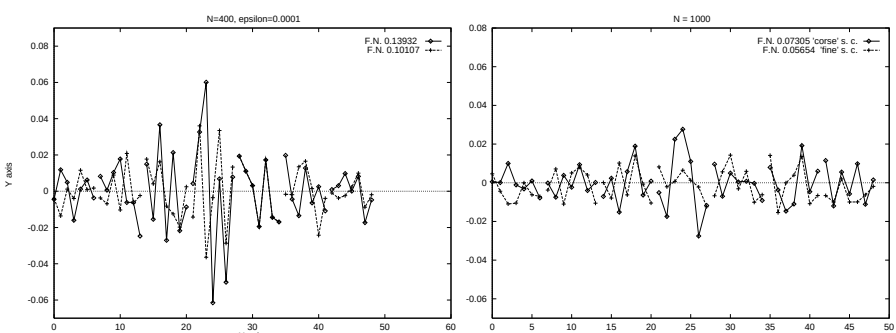
column		l_1 norm				C norm			
number	$\varepsilon =$	0.05	0.01	0.005	0.001	0.05	0.01	0.005	0.001
1		1	0.9	0.6	0.2	0.5	0.1	1	1
2		0.4	0.8	0.9	0.8	0.5	0.9	0.6	1
3		1	0.8	0.8	0.9	0.5	1	0.3	0.1
4		0.5	1	0.7	0.7	0.3	0.3	1	0.9
5		0.9	0.5	0.9	0.9	1	1	0.9	0.8
6		0.8	0.1	0.6	0.6	1	0.8	0.9	0.8
7		0.5	0.1	0.9	0.9	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

As a basic test example for applying Algorithm 4.3 a matrix of of size 7 is used. The size of the matrix is relatively small, because our aim was only to demonstrate how Algorithm 4.3 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 from [Dimov and Karaivanova (1996)] 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

$$\| A \|_F^2 = \sum_{i=1}^{n} \sum_{j=1}^{n} a_{ij}^2.$$

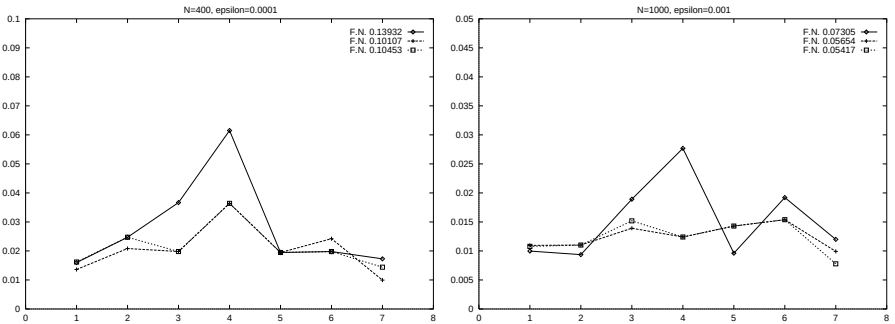
Some of the numerical results performed are shown in Figures 4.1 to 4.4, and also provided by Table 4.2. In all figures the value of the Frobenius norm is denoted by F.N., the number of realizations of the random variable (i.e., the number of random trajectories) is denoted by N and the value of the stop-criteria is denoted by ε. In **Algorithm 4.3** we use n values of ε (in our case $n = 7$).



(a) (b)

Fig. 4.1 Frobenius norm: (a) non-balanced case; (b) balanced case.

Figure 4.1 presents the values of the error matrix (4.44) in the both cases under consideration – coarse stop criteria (a) and fine stop criteria (b). 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 (Figure 4.1 (a)) $\varepsilon = 0.0001$.



(a) (b)

Fig. 4.2 Balancing: (a) Non-controlled balance; (b) Controlled balance.

When the fine stop criteria is used (Figure 4.1 (b)) 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 4.2). The values of the Frobenius norm for both cases when the number of realizations N is equal to 400 are also given. For such number of realizations the stochastic error is relatively large in comparison with the systematic one. So, the results on Figure 4.1(a) 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 4.1(b). One can see, that

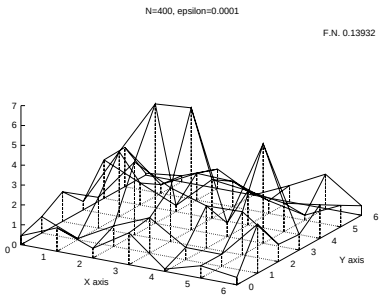
- ε is 10 times large then in the previous case, but the Frobenius norm is about two times smaller, because the number of realizations is larger.

The results presented in Figure 4.1(a) and Figure 4.1(b) show the statistical convergence of the algorithm, i.e. the error decreases when N increases (even in the case when the parameter ε increases).

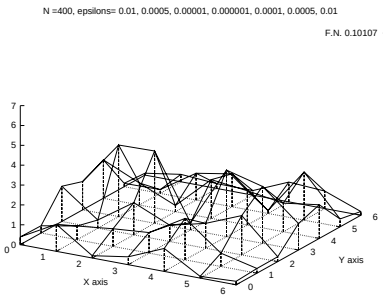
These results show how important is to have a good balancing between the stochastic and systematic error. The computational effort for the cases presented in Figure 4.1(a) and Figure 4.1(b) is approximately equal, but the results in the case of Figure 4.1(b), when we have a good balancing are almost 2 times better.

Let us discuss the results presented in Figures 4.2(a) and 4.2(b). 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 $\varepsilon = \varepsilon_i,\ i = 1, \ldots, n$ is denoted by R_f we consider the case when

$$R_c \geq R_f = 1.$$

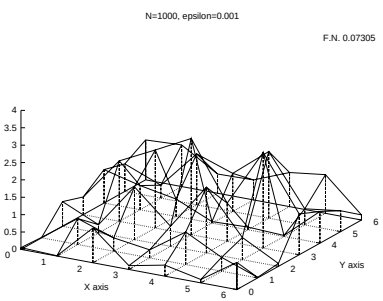


(a)

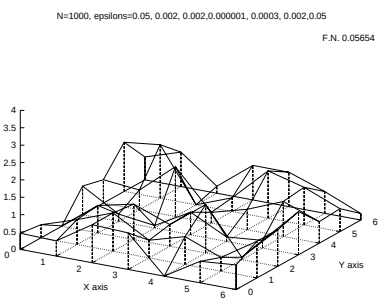


(b)

Fig. 4.3 Use of different fine stop-criteria: (a) Coarse stop-criteria; (b) Fine stop-criteria.



(a)



(b)

Fig. 4.4 Controlled balancing: (a) Coarse stop-criteria; (b) Fine stop-criteria.

The results presented in Figures 4.2(a) and 4.2(b) show that apart from the smaller computational complexity R_f of the fine stop criteria algorithm it also gives better results than the coarse stop criteria algorithm with complexity R_c. This fact is observed in both cases - balanced (Figure 4.2(a)) and non-balanced (Figure 4.2(b)):

- 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.

Figures 4.3(a) and 4.3(b) present test results for the modulo of every element of the error matrix (4.44) 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 4.4(a) and 4.4(b) show the corresponding results as in Figures 4.3(a) and 4.3(b) 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.

Table 4.3 Computational complexity

	non 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

Results presented in Table 4.3 show how the computational complexity depends on the parameter γ for the balanced and non balanced cases. One can see that the application of fine stoping criteria makes the balanced algorithm about 3.5% more efficient than the algorithm in which the course stoping criteria is used. At the same time for the non balanced case the fine stoping criteria algorithm is approximately 8.5% more efficient than the course stoping criteria algorithm.

4.7.4 *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 (Multi Instruction – Multi data)-machines.

Chapter 5

Markov Chain Monte Carlo Methods for Eigenvalue Problems

In this chapter a Monte Carlo approach for evaluating the eigenvalues of real symmetric matrices shall be studied. Algorithms for both dominant and smallest by magnitude eigenvalues will be considered. It is known that the problem of calculating the smallest by magnitude eigenvalue of a matrix A is more difficult from a 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 by magnitude eigenvalue, since this usually defines the most stable state of the system described by the considered matrix. Therefore we shall consider algorithms for the smallest by magnitude eigenvalue.

There are, also, many problems in which it is important to have an efficient algorithm that 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 [Godunov (1991); Trefethen (1991, 1999)] 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 $z \to \|(zI - B)^{-1}\|_2$ for $z \in \mathbf{C}$. It is clear that the latter mapping 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.

In this chapter we also analyse applicability and robustness of Markov chain Monte Carlo algorithms for eigenvalue problem.

Monte Carlo Almost Optimal (MAO) algorithms for solving eigenvalue problems is used. The consideration is focussed on the *Power* MC algorithm for computing the dominant eigenvalue and on the *Resolvent* MC algorithm for computing both the dominant eigenvalue and the eigenvalue of minimum modulus. Special attention is payed to bilinear forms of matrix powers, since they form the Krylov subspaces [Arnoldi (1951); Beattie *et al.* (2004); Arioli *et al.* (1998); Bai *et al.* (2000); Ton and Trefethen (1996)]. Results for the structure of both systematic and probability errors are presented. It is shown that the values of both errors can be controlled independently by different algorithmic parameters. The results present how the systematic error depends on the matrix spectrum. The analysis of the probability error shows that as close (in some sense) the matrix under consideration is to the stochastic matrix the smaller is this error. Sufficient conditions for constructing robust and interpolation Monte Carlo algorithms are obtained. For stochastic matrices an interpolation Monte Carlo algorithm is considered.

A number of numerical tests for large symmetric dense matrices are performed in order to study experimentally the dependence of the systematic error on the structure of matrix spectrum. It will also be studied how the probability error depends on the balancing of the matrix.

Many scientific and engineering applications are based on the problems of finding extremal (dominant) eigenvalues of real $n \times n$ matrices. The computation time for very large problems, or for finding solutions in real-time, can be prohibitive and this prevents the use of many established algorithms. Several authors have presented work on the estimation of computational complexity of linear algebra problems [Dimov (2000); Dimov *et al.* (2001, 1997); Dimov and Tonev (1993b)]. In this chapter we consider bilinear forms of matrix powers, which are used to formulate a solution for the eigenvalue problem. We consider our Monte Carlo approach for computing extremal eigenvalues of real symmetric matrices as a special case of Markov chain stochastic method for computing bilinear forms of matrix

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 where $A = (zI^* - B)(zI - B)$ is a symmetric matrix.

polynomials.

5.1 Formulation of the Problems

Consider the following problem of evaluating eigenvalues $\lambda(A)$:

$$Ax = \lambda(A)x. \tag{5.1}$$

It is assumed that

$$\begin{cases} 1.\ A \text{ is a given symmetric matrix, i.e. } a_{ij} = a_{ji} \text{ for all } i, j = 1, \ldots, n; \\ 2.\ \lambda_{min} = \lambda_n < \lambda_{n-1} \leq \lambda_{n-2} \leq \ldots \leq \lambda_2 < \lambda_1 = \lambda_{max}. \end{cases}$$

We use the following presentation of matrices:

$$A = \{a_{ij}\}_{i,j=1}^{n} = (a_1, \ldots, a_i, \ldots, a_n)^T,$$

where $a_i = (a_{i1}, \ldots, a_{in})$, $i = 1, \ldots, n$ and the symbol T means *transposition*.

The following norms of vectors (l_1-norm):

$$\|h\| = \|h\|_1 = \sum_{i=1}^{n} |h_i|, \quad \|a_i\| = \|a_i\|_1 = \sum_{j=1}^{n} |a_{ij}|$$

and matrices

$$\|A\| = \|A\|_1 = \max_j \sum_{i=1}^{n} |a_{ij}|$$

are used.

Let us note that in general $\|A\| \neq \max_i \|a_i\|$.

By $\bar{A}$ we denote the matrix containing the absolute values of elements of a given matrix A:

$$\bar{A} = \{|a_{ij}|\}_{i,j=1}^{n}.$$

By

$$p_k(A) = \sum_{i=0}^{k} c_i A^i, \quad c_i \in I\!R$$

we denote matrix polynomial of degree k.

As usual, $(v, h) = \sum_{i=1}^{n} v_i h_i$ denotes the inner product of vectors v and h.

We will be interested in computing inner products of the following type:

$$(v, p_k(A)h)\,.$$

As mentioned in the Introduction, by $E\{\xi\}$ we denote the *mathematical expectation* of the random variable ξ. The random variable ξ could be a randomly chosen component h_{α_k} of a given vector h. In this case the meaning of $E\{h_{\alpha_k}\}$ is *mathematical expectation of the value of randomly chosen element of* h.

By

$$D\{\xi\} = \sigma^2\{\xi\} = E\{\xi^2\} - (E\{\xi\})^2$$

we denote the variance of the random variable ξ ($\sigma\{\xi\}$ is the standard deviation).

Basically, we are interested in evaluation of forms:

$$(v, p_k(A)h)\,. \tag{5.2}$$

5.1.1 *Bilinear Form of Matrix Powers*

In a special case of $p_k(A) = A^k$ the form (5.2) becomes

$$(v, A^k h), \quad k \geq 1.$$

5.1.2 *Eigenvalues of Matrices*

Suppose that a real symmetric matrix A is diagonalisable, i.e.,

$$x^{-1}Ax = diag(\lambda_1, \ldots, \lambda_n).$$

If A is a symmetric matrix, then the values λ are real numbers, i.e., $\lambda \in I\!R$.

The well-known Power method [Monahan (2001); Golub and Van-Loan (1996); Isaacsons and Keller (1994)] gives an estimate for the dominant eigenvalue λ_1. This estimate uses the so-called *Rayleigh quotient* $\mu_k = \frac{(v,A^k h)}{(v,A^{k-1}h)}$:

$$\lambda_1 = \lim_{k\to\infty} \frac{(v, A^k h)}{(v, A^{k-1}h)},$$

where $v, h \in I\!R^n$ are arbitrary vectors. The Rayleigh quotient is used to obtain an approximation to λ_1:

$$\lambda_1 \approx \frac{(v, A^k h)}{(v, A^{k-1}h)}, \tag{5.3}$$

where k is an arbitrary large natural number.

To construct an algorithm for evaluating the eigenvalue of minimum modulus λ_n, one has to consider the following matrix polynomial:

$$p_i(A) = \sum_{k=0}^{i} q^k C_{m+k-1}^k A^k, \tag{5.4}$$

where C_{m+k-1}^k are binomial coefficients, and the characteristic parameter q is used as acceleration parameter of the algorithm (see Section 4.3 and [Dimov *et al.* (2001); Dimov and Karaivanova (1998, 1999)] for more details).

If $|q|\|A\| < 1$ and $i \to \infty$, then the polynomial (5.4) becomes the resolvent matrix [Dimov and Alexandrov (1998); Dimov and Karaivanova (1998)]:

$$p_\infty(A) = p(A) = \sum_{k=0}^{\infty} q^k C_{m+k-1}^k A^k = [I - qA]^{-m} = R_q^m,$$

where $R_q = [I - qA]^{-1}$ is the resolvent matrix of the equation

$$x = qAx + h. \tag{5.5}$$

Values $q_1, q_2, \ldots$ ($|q_1| \leq |q_2| \leq \ldots$) for which the equation (5.5) is fulfilled are called characteristic values of the equation (5.5). The resolvent operator

$$R_q = [I - qA]^{-1} = I + A + qA^2 + \ldots \tag{5.6}$$

exists if the sequence (5.6) converges. The systematic error of the presentation (5.6) when m terms are used is

$$R_s = O\left[(|q|/|q_1|)^{m+1}\, m^{\rho-1}\right], \tag{5.7}$$

where ρ is multiplicity of the root q_1. Estimation (5.7) is analogue of (4.24) from Chapter 4 and shows that the MC algorithm converges if $|q| < |q_1|$. When $|q| \geq |q_1|$ the algorithm does not converge for $q = q_* = 1$, but the solution of (5.5) exists (and moreover, the solution is unique). In this case one may apply a mapping of the spectral parameter q described in Section 4 (see also [Dimov and Alexandrov (1998)]).

One can consider the ratio:

$$\lambda = \frac{(v, Ap(A)h)}{(v, p(A)h)} = \frac{(v, AR_q^m h)}{(v, R_q^m h)},$$

where $R_q^m h = \sum_{k=1}^m g_k^{(m)} c_k$ and $g_k^{(m)}$ are computed according to (4.28). If $q < 0$, then

$$\frac{(v, AR_q^m h)}{(v, R_q^m h)} \approx \frac{1}{q}\left(1 - \frac{1}{\mu^{(k)}}\right) \approx \lambda_n, \tag{5.8}$$

where $\lambda_n = \lambda_{min}$ is the minimal by modulo eigenvalue, and $\mu^{(k)}$ is the approximation to the dominant eigenvalue of R_q.

If $|q| > 0$, then

$$\frac{(v, AR_q^m h)}{(v, R_q^m h)} \approx \lambda_1, \tag{5.9}$$

where $\lambda_1 = \lambda_{max}$ is the dominant eigenvalue.

The approximate equations (5.3), (5.8), (5.9) can be used to formulate efficient Monte Carlo algorithms for evaluating both the dominant and the eigenvalue of minimum modulus of real symmetric matrices.

The two problems formulated in this section rely on the bilinear form $(v, p_k(A)h)$. The latter fact allows to concentrate our study on MC algorithms for computing

$$\left(v, A^k h\right), \quad k \geq 1. \tag{5.10}$$

In the next section we are going to consider Almost Optimal Markov Chain Monte Carlo algorithms for computing both bilinear forms of matrix powers $(v, A^k h)$ (Subsection 5.2.1) and extremal eigenvalues of real symmetric matrices (Subsection 5.2.2). It is easy to see that since presentations (5.4) and (5.8) exist the developed technique can be used to compute the eigenvalue of minimum modulus λ_n.

5.2 Almost Optimal Markov Chain Monte Carlo

We use the algorithm presented in Section 4.2 (see also [Alexandrov *et al.* (2004); Dimov (1991, 2000); Dimov *et al.* (1997); Dimov and Alexandrov (1998)]).

The choice of permissible density distributions is according to (4.20) and (4.21). In our case p_i and p_{ij} are defined as follows:

$$p_i = \frac{|v_i|}{\|v\|}, \quad p_{ij} = \frac{|a_{ij}|}{\|a_i\|}. \tag{5.11}$$

5.2.1 *MC Algorithm for Computing Bilinear Forms of Matrix Powers* $(v, A^k h)$

The pair of density distributions (5.11) defines a finite chain of vector and matrix entries:

$$v_{\alpha_0} \to a_{\alpha_0\alpha_1} \to \ldots \to a_{\alpha_{k-1}\alpha_k}. \tag{5.12}$$

The chain induces the following product of matrix/vector entries and norms:

$$A_v^k = v_{\alpha_0} \prod_{s=1}^{k} a_{\alpha_{s-1}\alpha_s}$$

$$\|A_v^k\| = \|v\| \times \prod_{s=1}^{k} \|a_{\alpha_{s-1}}\|.$$

Note, that the product of norms $\|A_v^k\|$ is not a norm of A_v^k. The rule for creating the value of $\|A_v^k\|$ is following: the norm of the initial vector v, as well as norms of all row-vectors of matrix A visited by the chain (5.12) defined by densities (5.11), are included. For such a choice of densities p_i and p_{ij} we can prove the following Lemma.

Lemma 5.1.

$$E\{h_{\alpha_k}\} = \frac{sign\{A_v^k\}}{\|A_v^k\|} \left(v, A^k h\right).$$

Proof. Consider the value $\theta^{(k)} = sign\{A_v^k\}\|A_v^k\|h_{\alpha_k}$ for $k \geq 1$. We have

$$\begin{aligned}
\theta^{(k)} &= sign\{A_v^k\}\|A_v^k\|h_{\alpha_k} \\
&= sign\{v_{\alpha_0} \prod_{s=1}^{k} a_{\alpha_{s-1}\alpha_s}\}\|v\| \prod_{s=1}^{k} \|a_{\alpha_{s-1}}\|h_{\alpha_k} \\
&= sign\{v_{\alpha_0} \prod_{s=1}^{k} a_{\alpha_{s-1}\alpha_s}\}\|v\|\|a_{\alpha_0}\| \ldots \|a_{\alpha_{k-1}}\|h_{\alpha_k} \\
&= \frac{v_{\alpha_0}}{|v_{\alpha_0}|} \frac{a_{\alpha_0\alpha_1} \ldots a_{\alpha_{k-1}\alpha_k}}{|a_{\alpha_0\alpha_1}| \ldots |a_{\alpha_{k-1}\alpha_k}|}\|v\|\|a_{\alpha_0}\| \ldots \|a_{\alpha_{k-1}}\|h_{\alpha_k}.
\end{aligned} \tag{5.13}$$

Let us stress that among elements $v_{\alpha_0}, a_{\alpha_0\alpha_1}, \ldots, a_{\alpha_{k-1}\alpha_k}$ there are no elements equal to zero because of the special choice of acceptable distributions

p_i and p_{ij} defined by (5.11). The rules (5.11) ensure that the Markov chain *visits* non-zero elements only. From (5.13) and taking into account (5.11) one can get:

$$
\begin{aligned}
E\left\{\theta^{(k)}\right\} &= E\left\{\frac{v_{\alpha_0}}{|v_{\alpha_0}|}\frac{a_{\alpha_0\alpha_1}\dots a_{\alpha_{k-1}\alpha_k}}{|a_{\alpha_0\alpha_1}|\dots|a_{\alpha_{k-1}\alpha_k}|}\|v\|\|a_{\alpha_0}\|\dots\|a_{\alpha_{k-1}}\|h_{\alpha_k}\right\} \\
&= E\left\{\frac{v_{\alpha_0}}{p_{\alpha_0}}\frac{a_{\alpha_0\alpha_1}\dots a_{\alpha_{k-1}\alpha_k}}{p_{\alpha_0\alpha_1}\cdots p_{\alpha_{k-1}\alpha_k}}h_{\alpha_k}\right\} \\
&= \sum_{\alpha_0,\dots\alpha_k=1}^{n}\frac{v_{\alpha_0}}{p_{\alpha_0}}\frac{a_{\alpha_0\alpha_1}\dots a_{\alpha_{k-1}\alpha_k}}{p_{\alpha_0\alpha_1}\cdots p_{\alpha_{k-1}\alpha_k}}h_{\alpha_k}p_{\alpha_0}p_{\alpha_0\alpha_1}\cdots p_{\alpha_{k-1}\alpha_k} \\
&= \sum_{\alpha_0=1}^{n}v_{\alpha_0}\sum_{\alpha_1=1}^{n}a_{\alpha_0\alpha_1}\dots\sum_{\alpha_{k-1}=1}^{n}a_{\alpha_{k-2}\alpha_{k-1}}\sum_{\alpha_k=1}^{n}a_{\alpha_{k-1}\alpha_k}h_{\alpha_k} \\
&= \sum_{\alpha_0=1}^{n}v_{\alpha_0}\sum_{\alpha_1=1}^{n}a_{\alpha_0\alpha_1}\dots\sum_{\alpha_{k-1}=1}^{n}a_{\alpha_{k-2}\alpha_{k-1}}\left(Ah\right)_{\alpha_{k-1}} \\
&= \sum_{\alpha_0=1}^{n}v_{\alpha_0}\left(A^kh\right)_{\alpha_0} \\
&= \left(v, A^kh\right)
\end{aligned}
$$

□

Obviously, the standard deviation $\sigma\{h_{\alpha_k}\}$ is finite. Since we proved that the random variable $\theta^{(k)} = sign\{A_v^k\}\|A_v^k\|h_{\alpha_k}$ is a unbiased estimate of the form (v, A^kh), Lemma 5.1 can be used to construct a MC algorithm.

Let us consider N realizations of the Markov chain T_k defined by the pair of density distributions (5.11). Denote by $\theta_i^{(k)}$ the i^{th} realization of the random variable $\theta^{(k)}$. Then the value

$$\bar{\theta}^{(k)} = \frac{1}{N}\sum_{i=1}^{N}\theta_i^{(k)} = \frac{1}{N}\sum_{i=1}^{N}\{sign(A_v^k)\|A_v^k\|h_{\alpha_k}\}_i, \quad k \geq 1 \tag{5.14}$$

can be considered as a MC approximation of the form (v, A^kh). The probability error of this approximation can be presented in the following form:

$$R_N^{(k)} = \left|(v, A^kh) - \bar{\theta}^{(k)}\right| = c_p\sigma\{\theta^{(k)}\}N^{-\frac{1}{2}}, \tag{5.15}$$

where the constant c_p only depends on the probability P used in Definition 1.2 and does not depend on N and on $\theta^{(k)}$. Because of the finiteness of the standard deviation the probability error is always finite.

In fact, (5.14) together with the rules (5.11) defines a MC algorithm. The expression (5.14) gives a MC approximation of the form $(v, A^k h)$ with a probability error $R_N^{(k)}$. Obviously, the quality of the MC algorithm depends on the behavior of the standard deviation $\sigma\{\theta^{(k)}\}$. So, there is a reason to consider a special class of *robust MC algorithms.*

5.2.2 *MC Algorithm for Computing Extremal Eigenvalues*

Now consider again the pair of density distributions (5.11) defining a finite chain of vector and matrix entries (5.12). For such a choice of densities p_i and p_{ij} we can prove the following theorem.

Theorem 5.1. *Consider a real symmetric matrix A and the chain of vector and matrix entries (5.12) defined by MAO density distributions (5.11).*

Then

$$\lambda_1 = \lim_{k\to\infty} sign\{a_{\alpha_{k-1}\alpha_k}\}\|a_{\alpha_{k-1}}\|\frac{E\{h_{\alpha_k}\}}{E\{h_{\alpha_{k-1}}\}}.$$

Proof.

First consider the density of the Markov chain T_k of length k $\alpha_0 \to \alpha_1 \to \ldots \to \alpha_k$ as a point in $n(k+1)$-dimensional Euclidian space $T_{k+1} = \underbrace{I\!R^n \times \ldots \times I\!R^n}_{k+1}$:

$$P\{\alpha_0 = t_0, \alpha_1 = t_1, \ldots, \alpha_k = t_k\} = p_{t_0} p_{t_0 t_1} p_{t_1 t_2} \ldots p_{t_{k-1} t_k}.$$

To prove the theorem we have to show that $sign\{a_{\alpha_{k-1}\alpha_k}\}\|a_{\alpha_{k-1}}\|\frac{E\{h_{\alpha_k}\}}{E\{h_{\alpha_{k-1}}\}} = \mu_k = \frac{(v,A^k h)}{(v,A^{k-1}h)}$.

From the definition of *sign* function and MAO distributions (5.11) one can write:

$$sign\{a_{\alpha_{k-1}\alpha_k}\}\|a_{\alpha_{k-1}}\|\frac{E\{h_{\alpha_k}\}}{E\{h_{\alpha_{k-1}}\}}$$

$$= \frac{a_{\alpha_{k-1}\alpha_k}}{|a_{\alpha_{k-1}\alpha_k}|}\|a_{\alpha_{k-1}}\|\frac{E\{h_{\alpha_k}\}}{E\{h_{\alpha_{k-1}}\}}$$

$$= \frac{\frac{v_{\alpha_0}}{|v_{\alpha_0|}}\frac{a_{\alpha_0\alpha_1}\ldots a_{\alpha_{k-2}\alpha_{k-1}}a_{\alpha_{k-1}\alpha_k}}{|a_{\alpha_0\alpha_1}|\ldots|a_{\alpha_{k-2}\alpha_{k-1}}||a_{\alpha_{k-1}\alpha_k}|}\|v\|\|a_{\alpha_0}\|\ldots\|a_{\alpha_{k-2}}\|\|a_{\alpha_{k-1}}\|}{\frac{v_{\alpha_0}}{|v_{\alpha_0|}}\frac{a_{\alpha_0\alpha_1}\ldots a_{\alpha_{k-2}\alpha_{k-1}}}{|a_{\alpha_0\alpha_1}|\ldots|a_{\alpha_{k-2}\alpha_{k-1}}|}\|v\|\|a_{\alpha_0}\|\ldots\|a_{\alpha_{k-2}}\|}$$
$$\times \frac{E\{h_{\alpha_k}\}}{E\{h_{\alpha_{k-1}}\}}$$

$$= \frac{sign\{A_v^k\}\|A_v^k\|E\{h_{\alpha_k}\}}{sign\{A_v^{k-1}\}\|A_v^{k-1}\|E\{h_{\alpha_{k-1}}\}}.$$

According to Lemma 5.1 we have:

$$\frac{sign\{A_v^k\}\|A_v^k\|E\{h_{\alpha_k}\}}{sign\{A_v^{k-1}\}\|A_v^{k-1}\|E\{h_{\alpha_{k-1}}\}} = \frac{(v, A^k h)}{(v, A^{k-1}h)} = \mu_k.\diamondsuit$$

Obviously, since the standard deviation $\sigma\{h_{\alpha_k}\}$ is finite Theorem 5.1 allows to define a biased estimate of the extremal eigenvalue λ_1. Since, according to Theorem 5.1 for large enough values of k

$$\lambda_1 \approx \mu_k = \frac{E\{sign(a_{\alpha_{k-1}\alpha_k})\|a_{\alpha_{k-1}}\|h_{\alpha_k}\}}{E\{h_{\alpha_{k-1}}\}}$$

and the computational formula of the algorithm can be presented in the following form:

$$\lambda_1 \approx \{\mu_k\}_N := \frac{1}{\sum_{i=1}^N h_{\alpha_{k-1}}^{(i)}}\sum_{i=1}^N sign(a_{\alpha_{k-1}\alpha_k}^{(i)})\|a_{\alpha_{k-1}}^{(i)}\|h_{\alpha_k}^{(i)},$$

where the upper subscript (i) denotes the i^{th} realisation of the Markov chain, so that $h_{\alpha_{k-1}}^{(i)}$ is the value of the corresponding element of vector h after the $(k-1)^{st}$ jump in the i^{th} Markov chain, $h_{\alpha_k}^{(i)}$ is the value of the corresponding element of vector h after the k^{th} jump in the i^{th} Markov chain, $\|a_{\alpha_{k-1}}^{(i)}\|$ is the corresponding vector norm of the row which element

is last visited by the Markov chain number i after the k^{th} jump, and N is the total number of Markov chains performed.

In this subsection we presented an Almost Optimal Markov Chain Monte Carlo algorithms for computing extremal eigenvalues of real symmetric matrices. As it was mentioned before, the developed technique can easily be applied to compute the eigenvalue λ_n of minimum modulus. For computing λ_n one needs to consider bilinear forms of polynomials (5.4) (see, also (5.8)) instead of just bilinear forms of matrix powers.

5.2.3 *Robust MC Algorithms*

Definition 5.1. An MC algorithm for which the standard deviation does not increase with increasing of matrix power k is called a robust MC algorithm.

We can prove the following lemma.

Lemma 5.2. *If a MC algorithm is robust, then there exists a constant M such that*

$$\lim_{k\to\infty} \sigma\{\theta^{(k)}\} \leq M.$$

Proof. Let us choose the constant M as:

$$M = \|v\| \times \|a_{\alpha_0}\| \times \sigma\{h_{\alpha_1}\}.$$

Consider the equality (5.15):

$$R_N^{(k)} = c_p \sigma\{\theta^{(k)}\} N^{-\frac{1}{2}}.$$

If a MC algorithm is robust, then for any fixed pair (N, P) (number of realizations N and probability P) we have:

$$\sigma\{\theta^{(k)}\} \leq \sigma\{\theta^{(k-1)}\}.$$

Since the smallest possible value of k is 1 (according to our requirement (5.10))

$$\begin{aligned}\sigma\{\theta^{(k)}\} &\leq \sigma\{\theta^{(k-1)}\} \leq \ldots \leq \sigma\{\theta^{(1)}\} \\ &= \sigma\{sign\{v_{\alpha_0} a_{\alpha_0\alpha_1}\} \times \|v\| \times \|a_{\alpha_0}\| \times h_{\alpha_1}\} \\ &= \|v\| \times \|a_{\alpha_0}\| \times \sigma\{h_{\alpha_1}\} = M. \end{aligned} \tag{5.16}$$

□

It is interesting to answer the question:

- *How small could the probability error be? and*
- *Is it possible to formulate MC algorithms with zero probability error?*

To answer the first question one has to analyze the structure of the variance. Then it will be possible to answer the second question concerning the existence of algorithms with zero probability error.

5.2.4 *Interpolation MC Algorithms*

Definition 5.2. An MC algorithm for which the probability error is zero is called an interpolation MC algorithm.

The next theorem gives the structure of the variance for MAO algorithm.

Theorem 5.2. *Let*

$$\hat{h} = \{h_i^2\}_{i=1}^n, \quad \bar{v} = \{|v_i|\}_{i=1}^n, \quad \bar{A} = \{|a_{ij}|\}_{i,j=1}^n.$$

Then

$$D\{\theta^{(k)}\} = \|A_v^k\| \left(\bar{v}, \bar{A}^k \hat{h}\right) - (v, A^k h)^2.$$

Proof. Taking into account MAO density distributions (5.11) the random variable $\theta^{(k)}$ can be presented in the following form:

$$\begin{aligned}
\theta^{(k)} &= sign\{A_v^k\}\|A_v^k\|h_{\alpha_k} \\
&= \frac{v_{\alpha_0}}{|v_{\alpha_0}|} \frac{a_{\alpha_0\alpha_1} \dots a_{\alpha_{k-1}\alpha_k}}{|a_{\alpha_0\alpha_1}| \dots |a_{\alpha_{k-1}\alpha_k}|} \|v\| \|a_{\alpha_0}\| \dots \|a_{\alpha_{k-1}}\| h_{\alpha_k} \\
&= \frac{v_{\alpha_0}}{p_{\alpha_0}} \frac{a_{\alpha_0\alpha_1} \dots a_{\alpha_{k-1}\alpha_k}}{p_{\alpha_0\alpha_1} \dots p_{\alpha_{k-1}\alpha_k}} h_{\alpha_k}
\end{aligned}$$

We deal with the variance

$$D\{\theta^{(k)}\} = E\{(\theta^{(k)})^2\} - (E\{\theta^{(k)}\})^2. \tag{5.17}$$

Consider the first term of (5.17).

$$
\begin{aligned}
E\{(\theta^{(k)})^2\} &= E\left\{\frac{v_{\alpha_0}^2}{p_{\alpha_0}^2}\frac{a_{\alpha_0\alpha_1}^2\dots a_{\alpha_{k-1}\alpha_k}^2}{p_{\alpha_0\alpha_1}^2\dots p_{\alpha_{k-1}\alpha_k}^2}h_{\alpha_k}^2\right\}\\
&= \sum_{\alpha_0\dots\alpha_k=1}^{n}\frac{v_{\alpha_0}^2}{p_{\alpha_0}^2}\frac{a_{\alpha_0\alpha_1}^2\dots a_{\alpha_{k-1}\alpha_k}^2}{p_{\alpha_0\alpha_1}^2\dots p_{\alpha_{k-1}\alpha_k}^2}h_{\alpha_k}^2 p_{\alpha_0}p_{\alpha_0\alpha_1}\dots p_{\alpha_{k-1}\alpha_k}\\
&= \sum_{\alpha_0\dots\alpha_k=1}^{n}\frac{v_{\alpha_0}^2}{p_{\alpha_0}}\frac{a_{\alpha_0\alpha_1}^2\dots a_{\alpha_{k-1}\alpha_k}^2}{p_{\alpha_0\alpha_1}\dots p_{\alpha_{k-1}\alpha_k}}h_{\alpha_k}^2\\
&= \sum_{\alpha_0\dots\alpha_k=1}^{n}\frac{v_{\alpha_0}^2}{|v_{\alpha_0}|}\|v\|\frac{a_{\alpha_0\alpha_1}^2\dots a_{\alpha_{k-1}\alpha_k}^2}{|a_{\alpha_0\alpha_1}|\dots|a_{\alpha_{k-1}\alpha_k}|}\|a_{\alpha_0}\|\dots\\
&\dots\|a_{\alpha_{k-1}}\|h_{\alpha_k}^2\\
&= \|A_v^k\|\sum_{\alpha_0\dots\alpha_k=1}^{n}|v_{\alpha_0}||a_{\alpha_0\alpha_1}|\dots|a_{\alpha_{k-1}\alpha_k}|h_{\alpha_k}^2\\
&= \|A_v^k\|\sum_{\alpha_0=1}^{n}|v_{\alpha_0}|\sum_{\alpha_1=1}^{n}|a_{\alpha_0\alpha_1}|\dots\\
&\dots\sum_{\alpha_{k-1}=1}^{n}|a_{\alpha_{k-2}\alpha_{k-1}}|\sum_{\alpha_k=1}^{n}|a_{\alpha_{k-1}\alpha_k}|h_{\alpha_k}^2\\
&= \|A_v^k\|\sum_{\alpha_0=1}^{n}|v_{\alpha_0}|\sum_{\alpha_1=1}^{n}|a_{\alpha_0\alpha_1}|\cdots\sum_{\alpha_{k-1}=1}^{n}|a_{\alpha_{k-2}\alpha_{k-1}}|(\bar{A}\hat{h})_{\alpha_{k-1}}\\
&= \|A_v^k\|\sum_{\alpha_0=1}^{n}|v_{\alpha_0}|(\bar{A}^k\hat{h})_{\alpha_0} = \|A_v^k\|(\bar{v},\bar{A}^k\hat{h}).
\end{aligned}
$$

According to Lemma 5.1 the second term of (5.17) is equal to $(v, A^k h)^2$.

Thus,

$$D\{\theta^{(k)}\} = \|A_v^k\|\left(\bar{v},\bar{A}^k\hat{h}\right) - (v, A^k h)^2.$$

□

Now we can formulate an important corollary that gives a sufficient condition for constructing an interpolation MC algorithm.

Corollary 5.2.1. *Consider vectors* $h = (1,\dots,1)^T$, $v = (\frac{1}{n},\dots,\frac{1}{n})$ *and a perfectly balanced singular stochastic matrix* $A = hv = \begin{pmatrix} \frac{1}{n} & \cdots & \frac{1}{n} \\ \vdots & \ddots & \vdots \\ \frac{1}{n} & \cdots & \frac{1}{n} \end{pmatrix}$. *Then*

the MC algorithm defined by density distributions (5.11) is an interpolation MC algorithm.

Proof. To prove the corollary it is sufficient to show that the variance $D\{\theta^{(k)}\}$ is zero. Obviously,

$$\|v\| = 1 \quad \text{and} \quad \|a_i\| = 1 \quad \text{for any} \quad i = 1, \ldots, n.$$

Thus,

$$\|A_v^k\| = \|v\| \|a_{\alpha_0}\| \ldots \|a_{\alpha_{k-1}}\| = 1. \tag{5.18}$$

The following equalities are true:

$$Ah = \begin{pmatrix} \frac{1}{n} & \ldots & \frac{1}{n} \\ \vdots & \ddots & \vdots \\ \frac{1}{n} & \ldots & \frac{1}{n} \end{pmatrix} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}.$$

Obviously,

$$A^k h = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix},$$

and

$$(v, A^k h) = \left(\frac{1}{n}, \ldots, \frac{1}{n}\right) \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = 1.$$

Since

$$\bar{A} = \begin{pmatrix} \left|\frac{1}{n}\right| & \ldots & \left|\frac{1}{n}\right| \\ \vdots & \ddots & \vdots \\ \left|\frac{1}{n}\right| & \ldots & \left|\frac{1}{n}\right| \end{pmatrix} = A$$

and

$$\hat{h} = \{|h_i^2|\}_{i=1}^n = h,$$

and taking into account (5.18) we have:

$$\|A_v^k\|(\bar{v}, \bar{A}^k \hat{h}) = 1.$$

Thus, we proved that $D\{\theta^{(k)}\} = 0$. □

This corollary does not have theoretical importance. It will help us to prepare computational experiments with artificially generated large random matrices. Obviously the closer, in some sense, the random matrix is to the perfectly balanced stochastic matrix the smaller the variance of the method.

5.3 Computational Complexity

It is very important to have an estimation of the computational complexity (or number of operations) of MAO algorithms. Such estimates are important when there are more than one algorithm for solving the problem. We consider a MAO algorithm for computing bilinear forms of matrix powers, which can be also used to formulate the solution for the dominant eigenvalue problem. Assume, we considering the set, $\mathcal{A}$, of algorithms, A, for calculating bilinear forms of matrix powers $(v, A^k h)$ with a probability error $R_{k,N}$ less than a given constant ε:

$$\mathcal{A} = \{A : Pr(R_{k,N} \leq \varepsilon) \geq c\}\,. \tag{5.19}$$

There is the following problem: which algorithm in the set $\mathcal{A}$ has the smallest computational cost? In this chapter we are not going to analyze the performance of different Monte Carlo algorithms. We can only mention that the MAO algorithm has a performance close to the best one (for more details we refer to [Dimov (1991, 2000); Dimov *et al.* (2001); Dimov and Karaivanova (1999)]).

We assume that the probability error $R_{k,N}$ is fixed by the value of ε and the probability $c < 1$ in (5.19) is also fixed. Obviously, for fixed ε and $c < 1$ the computational cost depends linearly on the number of iterations k and on the number of Markov chains N.

Definition 5.3. Computational cost of the Markov chain MAO algorithm A is defined by

$$\tau(A) = NE(q)t,$$

where N is the number of Markov chains, $E(q) = k$ is the mathematical expectation of the number of transitions in a single Markov chain and t is the mean time (or number of operations) needed to compute the value of the random variable.

Two types of errors, systematic and stochastic (probability), can occur in Monte Carlo algorithms, and achieving a balance between these two types of error is necessary. Clearly, to obtain good results the stochastic (probability) error $R_{k,N}$ must be approximately equal to the systematic one $R_{k,s}$ and so

$$R_{k,N} \approx R_{k,s}.$$

The problem of error balancing 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 of the number of steps in each random trajectory (number of iterations) k.

5.3.1 *Method for Choosing the Number of Iterations k*

Assume that we wish to estimate the value of the bilinear form $(v, A^k h)$, so that with a given probability $P < 1$ the error is smaller than a given positive ε:

$$\left| (v, A^k h) - \frac{1}{N} \sum_{i=1}^{N} \theta_i^{(k)} \right| \leq \varepsilon.$$

We consider the case of balanced errors, i.e.,

$$R_{k,N} = R_{k,s} = \frac{\varepsilon}{2}.$$

When a mapping procedure is applied one may assume that there exists a positive constant $\alpha < 1$ such that

$$\alpha \geq \left| g_i^{(k)} \right| \times \|A\| \quad \text{for any } i \text{ and } k. \tag{5.20}$$

Then

$$\frac{\varepsilon}{2} \leq \frac{\left(\left| g_i^{(k)} \right| \|A\| \right)^{k+1} \|h\|}{1 - \left| g_i^{(k)} \right| \|A\|} \leq \frac{\alpha^{k+1} \|h\|}{1 - \alpha}$$

and for k should be chosen the smallest natural number for which

$$k \geq \frac{|log\delta|}{|log\alpha|} - 1, \quad \delta = \frac{\varepsilon(1-\alpha)}{2\|h\|}. \tag{5.21}$$

If a mapping procedure is not applied, i.e., the corresponding Neumann series converges fast enough, then one assumes that a positive constant α, such that $\alpha \geq \|A\|$ exists. Then the number of iterations k should be chosen according to (5.21).

We should also mention here that there are other possibilities to estimate the number of needed iterations k if a mapping procedure is applied. An example follows. Assume that the multiplicity of the characteristic value q_1 of the equation (5.5) is $\rho = 1$. Assume also that there exists a positive constant q_*, such that

$$q_* \leq |q_1|.$$

Then we have

$$R_s = \frac{\varepsilon}{2} = c_s \left| \frac{q}{q_*} \right|^{k+1},$$

$$k \geq \frac{\left| \log\left(\frac{\varepsilon}{2c_s} \right) \right|}{\left| \log \left| \frac{q}{q_*} \right| \right|} - 1.$$

The choice of the method of estimation of k depends on the available *a priori* information, which comes from the concrete scientific application. One may use the first or the second approach depending on the information available.

5.3.2 *Method for Choosing the Number of Chains*

To estimate the computational cost $\tau(A)$ we should estimate the number N of realizations of the random variable $\theta^{(k)}$. To be able to do this we assume that there exists a constant σ such that

$$\sigma \geq \sigma(\theta^{(k)}). \tag{5.22}$$

Then we have

$$\varepsilon = 2R_N^{(k)} = 2c_p \sigma(\theta^{(k)}) N^{-\frac{1}{2}} \geq 2c_p \sigma N^{-\frac{1}{2}},$$

and

$$N \geq \left\{ \frac{2c_p \sigma}{\varepsilon} \right\}^2. \tag{5.23}$$

Taking into account relations (5.21) and (5.23) one can get estimates of the computational cost of biased MC algorithms. Let us stress that to obtain relations (5.21) and (5.23) we needed some *a priori* information in form of (5.20) and (5.22). We do not assume that one needs to calculate $\|A\|$ in order to have a good estimate for α, as well as to compute $\sigma(\theta^{(k)})$ in order to get an estimate for σ. In real-life applications very often parameters like α and σ are known as *a priori* information. That information may come from physics, biology or any concrete scientific knowledge. This remark is important, because we do not include computation of $\|A\|$ or $\sigma(\theta^{(k)})$ into the computational cost of our algorithm.

Let us also note that if *a priori* information of type (5.22) is not available one may use *a posteriori* information. To compute *a posteriori* estimation

of $\sigma(\theta^{(k)})$ one needs to use the mean value of $\theta^{(k)} - \frac{1}{N}\sum_{i=1}^{N}\theta_i^{(k)}$ as well as the mean value of $(\theta^{(k)})^2 - \frac{1}{N}\sum_{i=1}^{N}(\theta_i^{(k)})^2$. Since the first mean value should be computed as a MC approximation to the solution, one needs just to add one or two rows into the code to get *a posteriori* estimation for $\sigma(\theta^{(k)})$.

5.4 Applicability and Acceleration Analysis

In this section we discuss applicability and acceleration analysis of MAO algorithm. Summarizing results from previous sections we can present Monte Carlo computational formulas for various linear algebra problems.

Power MC algorithm for computing the dominant eigenvalue

The corresponding matrix polynomial is $p_k(A) = A^k$, so that

$$(v, A^k h) = E\{\theta^{(k)}\} \approx \bar{\theta}^{(k)} = \frac{1}{N}\sum_{i=1}^{N}\{sign(A_v^k)\|A_v^k\|h_{\alpha_k}\}_i, \tag{5.24}$$

and the computational formula is:

$$\lambda_{max} \approx \frac{\bar{\theta}^{(k)}}{\bar{\theta}^{(k-1)}} = \frac{1}{\sum_{i=1}^{N} h_{\alpha_{k-1}}^{(i)}}\sum_{i=1}^{N} sign(a_{\alpha_{k-1}\alpha_k}^{(i)})\|a_{\alpha_{k-1}}^{(i)}\|h_{\alpha_k}^{(i)}.$$

Resolvent MC algorithm for eigenvalue problems

For the Inverse shifted (Resolvent) MC the matrix polynomial is $p(A) = \sum_{i=0}^{\infty} q^i C_{m+i-1}^i A^i$. If $|qA| < 1$, then $p(A) = \sum_{i=0}^{\infty} q^i C_{m+i-1}^i A^i = [I - qA]^{-m} = R_q^m$ (R is the resolvent matrix) and

$$\lambda = \frac{(v, Ap(A)h)}{(v, p(A)h)} = \frac{(v, AR_q^m h)}{(v, R_q^m h)}.$$

If $q < 0$, then $\frac{(v,AR_q^m h)}{(v,R_q^m h)} \approx \frac{1}{q}\left(1 - \frac{1}{\mu^{(m)}}\right) \approx \lambda_{min}$ ($\mu^{(m)}$ is the approximation to the dominant eigenvalue of the resolvent matrix R).

For a positive q ($q > 0$): $\frac{(h,AR_q^m f)}{(h,R_q^m f)} \approx \lambda_{max}$.

Thus the extremal eigenvalue is

$$\lambda \approx \frac{E\sum_{i=0}^{k} q^i C_{i+m-1}^i \theta^{(i+1)}}{E\sum_{i=0}^{k} q^i C_{i+m-1}^i \theta^{(i)}}, \tag{5.25}$$

where $\theta^{(0)} = \frac{v_{\alpha_0}}{p_{\alpha_0}}$ and the r.v. $\theta^{(i)}$ are defined according to (5.14). The value v_{α_0} is the entrance α_0 of the arbitrary vector v chosen according to the initial distribution p_{α_0}. For a positive value of the acceleration parameter q one cam compute an approximation to $\lambda_1 = \lambda_{max}$ while for a negative value of q, $\lambda_n = \lambda_{min}$ can be evaluated.

Since the initial vector v can be any vector $v \in I\!R^{n\times 1}$ the following formula for calculating λ is used

$$\lambda \approx \frac{E\{\theta^{(1)} + qC_m^1\theta^{(2)} + q^2C_{m+1}^2\theta^{(3)} + \ldots + q^kC_{k+m-1}^k\theta^{(k+1)}\}}{E\{1 + qC_m^1\theta^{(1)} + q^2C_{m+1}^2\theta^{(2)} + \ldots + q^kC_{k+m-1}^k\theta^{(k)}\}},$$

that is

$$\lambda \approx \frac{\frac{1}{N}\sum_{s=1}^{N}\{\sum_{i=0}^{k} q^i C_{i+m-1}^i \theta^{(i+1)}\}_s}{\frac{1}{N}\sum_{s=1}^{N}\{\sum_{i=0}^{k} q^i C_{i+m-1}^i \theta(i)\}_s} \tag{5.26}$$

If $q > 0$ the formula (5.26) evaluates λ_1, if $q < 0$, then it evaluates λ_n.

To analyse the applicability of the MAO algorithm in the Power method with MC iterations we consider two matrices: the original matrix A with eigenvalues λ_i and the iterative matrix R with eigenvalues μ_i. Thus values λ_i and μ_i can be considered as solutions of the problems:

$$Ax = \lambda x \quad \text{and} \quad Rx = \mu x.$$

We assume that $|\lambda_1| > |\lambda_2| \geq \ldots \geq |\lambda_{n-1}| > |\lambda_n|$ as well as $|\mu_1| > |\mu_2| \geq \ldots \geq |\mu_{n-1}| > |\mu_n|$.

The systematic error that appears from the Power method is:

$$O\left(\left|\frac{\mu_2}{\mu_1}\right|^k\right),$$

where $\mu = \lambda$ if $R = A$ (Plain Power method), $\mu = \frac{1}{\lambda}$ if $R = A^{-1}$(Inverse Power method), $\mu = \lambda - q$ if $R_q = A - qI$ (Shifted Power method), $\mu = \frac{1}{1-q\lambda}$ if $R_q = (I - qA)^{-1}$ (Resolvent Power method). For the Resolvent Power method in case of negative q the eigenvalues of matrices A and R_q are connected through the equality:

$$\mu_i = \frac{1}{1 + |q|\lambda_{n-i+1}}.$$

The stochastic error that appears because we calculate mathematical expectations approximately is $O(\sigma(\theta^{(k)})N^{-1/2})$.

The choice of the parameter q is very important since it controls the convergence. When we are interested in evaluating the smallest eigenvalue applying iterative matrix $R_q = (I - qA)^{-1}$ the parameter $q < 0$ has to be chosen so that to minimize the following expression

$$J(q, A) = \frac{1 + |q|\lambda_1}{1 + |q|\lambda_2}, \tag{5.27}$$

or if

$$q = -\frac{\alpha}{||A||},$$

then

$$J(q, A) = \frac{\lambda_1 + \alpha\lambda_n}{\lambda_1 + \alpha\lambda_{n-1}}.$$

In practical computations we chose $\alpha \in [0.5, 0.9]$. In case of $\alpha = 0.5$ we have

$$q = -\frac{1}{2\|A\|}. \tag{5.28}$$

The *systematic error* in this case is

$$O\left[\left(\frac{2\lambda_1 + \lambda_n}{2\lambda_1 + \lambda_{n-1}}\right)^m\right], \tag{5.29}$$

where m is the power of the resolvent matrix R_q (or the number of iterations by R_q).

The convergence in this case can not be better than $O[(1/3)^m]$. Such a convergence is almost reached for matrices with λ_n having opposite sign than all other eigenvalues $\lambda_1, \ldots \lambda_{n-1}$. The best convergence in case when all eigenvalues are positive or all are negative is $O[(2/3)^m]$ for $\alpha = \frac{1}{2}$. In case of $\alpha = \frac{9}{10}$ the convergence can not be better than $O[(1/19)^m]$.

Let us consider some examples in order to demonstrate applicability and acceleration analysis of our approach. We use two randomly generated matrices A_1 and A_2 with controlled spectra and three different values of $\alpha \in [0.5, 0.9]$ (see Table 5.1). The last column of Table 5.1 contains results of the convergence (the ratio μ_2/μ_1 characterizes the rate of convergence). Obviously, the smaller the ratio μ_2/μ_1 the faster is the convergence.

The results presented in Table 5.1 show that for some classes of matrices the convergence is relatively slow, but for some other classes it is very fast.

Table 5.1 Illustration of the convergence of the resolvent MC algorithm.

Matrix	α	$\lambda_1(A)$	$\lambda_{n-1}(A)$	$\lambda_n(A)$	$\mu_1(R_q)$	$\mu_2(R_q)$	$\mu_n(R_q)$	$\frac{\mu_2}{\mu_1}$
A_1	1/2	0.5	0.22	0.05	0.952	0.820	0.667	0.860
A_2	1/2	1.0	0.95	-0.94	1.887	0.678	0.667	0.359
A_2	4/5	1.0	0.95	-0.94	4.032	0.568	0.556	0.141
A_2	9/10	1.0	0.95	-0.94	6.493	0.539	0.526	0.083

If one has *a priori* information about the spectrum of the matrix A the acceleration parameter q (respectively α) can be chosen so that to reach a very fast convergence of order $O[(0.0830)^m]$ (see the last row of Table 5.1). Such an analysis is important because it helps to choose the power m of the resolvent matrix R_q. If the matrix has a spectrum like A_1 and $q = -\dfrac{1}{2||A||}$ then m has to be chosen $m \approx 30$ in order to get results with a relative error 0.01. If the matrix has spectrum like A_2 and $q = -\dfrac{9}{10||A||}$ than after just 4 iterations the relative error in computing λ_n is smaller than 5×10^{-5}. One should try to find appropriate values for q (respectively for α) in order to get satisfactory accuracy for relatively small values of number of iterations. It is quite important for non-balanced matrices since the imbalance leads to increasing stochastic errors with increasing of number of iterations. In illustration of this fact is the next example. We consider a random symmetric non-balanced matrix A_3 with $\lambda_1 = 64$ and $\lambda_n = 1$, $n = 128$. We apply Plain Power MC algorithm for computing the dominant eigenvalue λ_1 .

We compute the first 10 iterations by Monte Carlo and by simple matrix-vector multiplication with double precision assuming that the obtained results are "exact" (they still contain some roundoff errors that are relatively small). The results of computations are presented in Figure 5.1. The first impression is that the results are good.

But more precise consideration shows that the error increases with increasing of matrix power (see Figure 5.2). Since we consider values of matrix powers $\dfrac{v^T A^k v}{\|v\|}$ we eliminate the systematic error (the Monte Carlo estimate is a unbiased estimate) in this special case. So that considering the results from Figure 5.2 we can see how stochastic error propagates with the matrix power.

One can see that for the first seven iterations the error is less than 1% while for the ten iterations it is almost 3%. This is an expected result, since

the applied MC algorithm is not *robust* for such a non-balanced matrix. It means that with increasing of number of iterations the standard deviation (respectively the probability error) also increases (see Subsection 5.2.3). This consideration shows that one has to pay a special attention to the problem of robustness.

To study experimentally this phenomenon we generated matrices and vectors with *a priori* given properties. Matrices A were generated of order 100, 1000 and 5000. The vector h was filled with ones and v was filled with $\frac{1}{n}$. The matrices A were filled with elements of size $\frac{1}{n}$ and then perturbed by $2, 5, 10, 50$ and 90%. After such perturbations the matrices are not stochastic. The bigger the perturbation applied the further away the matrix is from its stochastic form. The norm of such matrices is around 1. For comparison random non-balanced matrices were generated too. Our aim was to compute matrix powers in a form of $\frac{(v, A^k h)}{(v, h)}$ since in this case there is no systematic error. In such a way we can study how the stochastic error propagates with the number of Monte Carlo iterations for different classes of symmetric matrices. Note also that in our experiments the choice of vectors v and h ensures the equality $(v, h) = 1$.

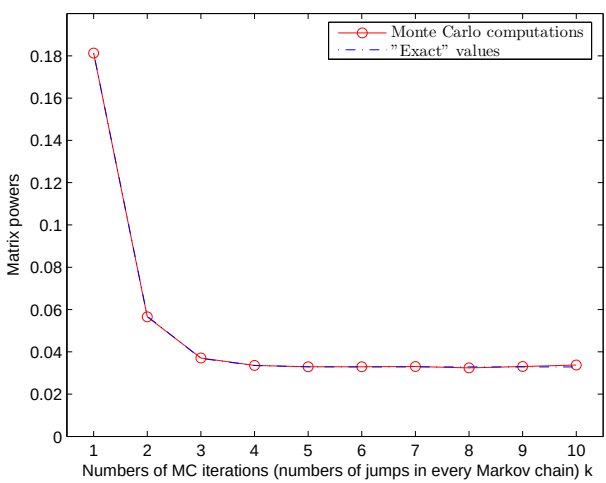


Fig. 5.1 Monte Carlo and "Exact" values $\frac{v^T A^k v}{\|v\|}$ for a random non-balanced matrix A_3 of size 128×128. In all experiments the number N of Markov chains is 10^6.

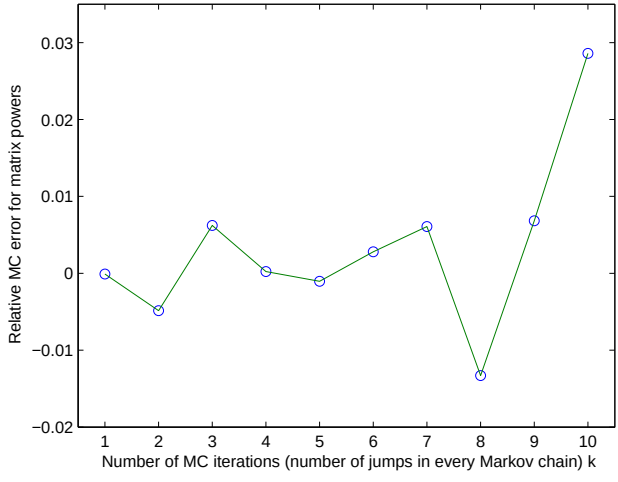


Fig. 5.2 Relative MC error for values $\frac{v^T A^k v}{\|v\|}$ for a random non-balanced matrix of size 128×128. In all experiments the number N of Markov chains is 10^6.

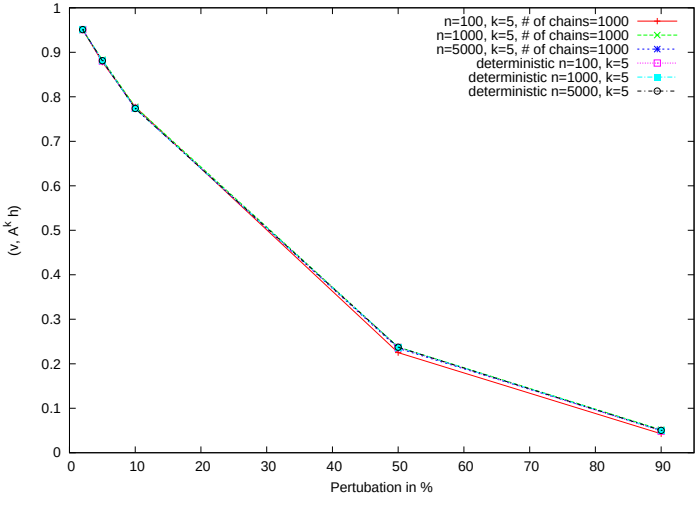


Fig. 5.3 The dependence of MC results for matrix powers $(v, A^k h)$ on perturbation of the entries of the matrix A.

In Figure 5.3 we present results for different perturbations for the Monte Carlo algorithm and for the deterministic code that gives roundoff error

only. Since the deterministic computations were performed with a double precision we accept the results obtained as *"exact results"* and use them to analyse the accuracy of the results produced by our Monte Carlo code. Our numerical experiments show that the results are very close for perturbations of up to 10% whereas the results for 50% and 90% differ up to 2% for matrices of size 1000 and 5000 and differ up to 14% for a matrix of size 100 (it is hard to see on Figure 5.3 since the results look very close to each other).

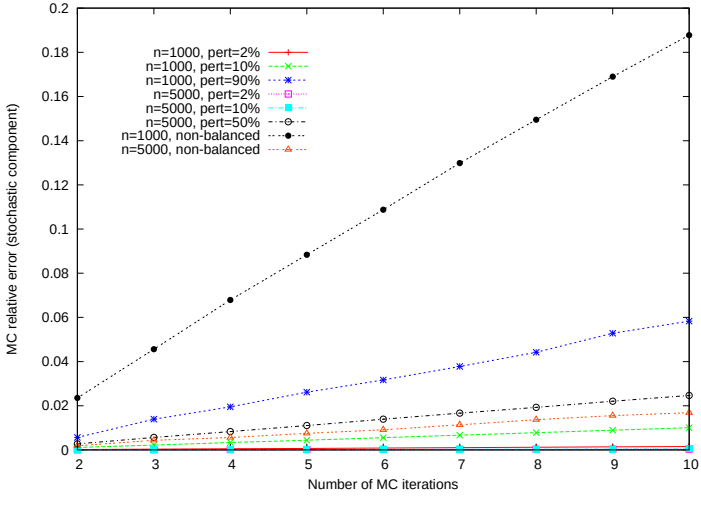


Fig. 5.4 The dependence of MC relative error (stochastic component) on power of the matrix A.

In Figure 5.4 the relative error of the results for Monte Carlo algorithm is shown. The Monte Carlo probability error $R_N^{(k)}$ and the Relative Monte Carlo probability error $Rel_N^{(k)}$ was computed in the following way:

$$R_N^{(k)} = \left| \frac{1}{N} \sum_{i=1}^{N} \theta_i^{(k)} - \frac{(v, A^k h)}{(v, h)} \right|, \quad Rel_N^{(k)} = \frac{(v, h)}{(v, A^k h)} R_N^{(k)}.$$

From Figure 5.4 we can see that the error increases linearly as k increases. The larger the matrix is, the smaller the influence of the perturbation. For comparison, the results for non-balanced matrices were included.

The variance of the results for the different perturbations are shown in Figure 5.5. In this figure we compare results for different sizes of the matrix for a fixed (but relatively small) number of Markov chains. Again it is obvious that the influence of the perturbation is a lot bigger for the

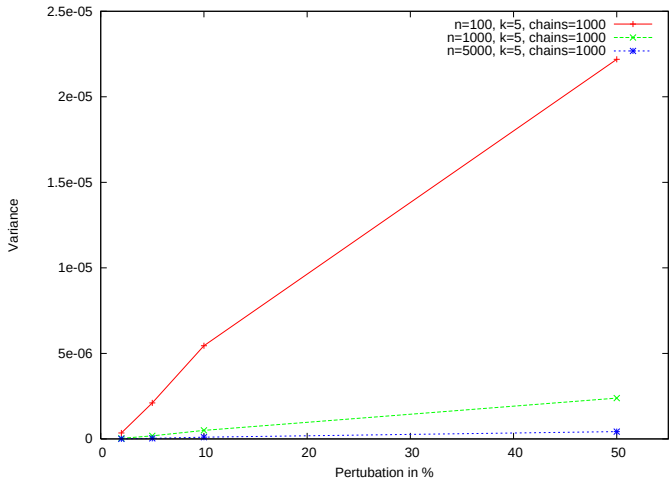


Fig. 5.5 The dependence of variance of the r.v. on perturbation of of the matrix entries.

smaller matrix (of size $n = 100$). But over all a variance of 10^{-5} is a good result and shows that the Monte Carlo algorithm works well with this kind of balanced matrices. Nevertheless, the algorithm is still not robust since the variance (and the probability error $R_N^{(k)}$) increases with increasing of k (see results shown on Figure 5.4). It is because the norms of iterative matrices A are large. Such matrices should be scaled in order to get a robust algorithm.

To test the robustness of the Monte Carlo algorithm, a re-run of the experiments was done with matrices of norm $\|A\| \approx 0.1$. In fact we used the same randomly generated matrices scaled by a factor of 1/10. The results for these experiments are shown in Figure 5.6 and Figure 5.7.

In Figure 5.6 the Monte Carlo errors for matrices of size $n = 1000$ and $n = 5000$ and number of Markov chains $N = 1000$ are shown. The number of Markov chains in these experiments is relatively small because the variance of $\theta^{(k)}$ is small (as one can see from the experimental results). One can also see that the Monte Carlo algorithm is very robust in this case because with an increasing k the error is decreasing enormously.

The results shown in Figure 5.7 illustrate the fact that even with a very small number of Markov chains ($N = 100$) one can obtain quite accurate results. The variance shown in Figure 5.7 is 10^{10} smaller than the variance shown in Figure 5.5. It increases with the increasing of the perturbation because matrices become more and more unbalanced. The last results

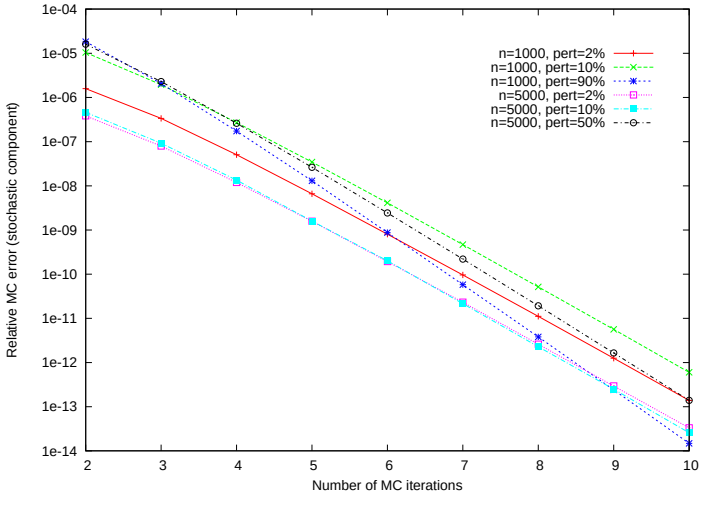


Fig. 5.6 The dependence of MC error on power of matrices k with "small" spectral norms ($\|A\| \approx 0.1$).

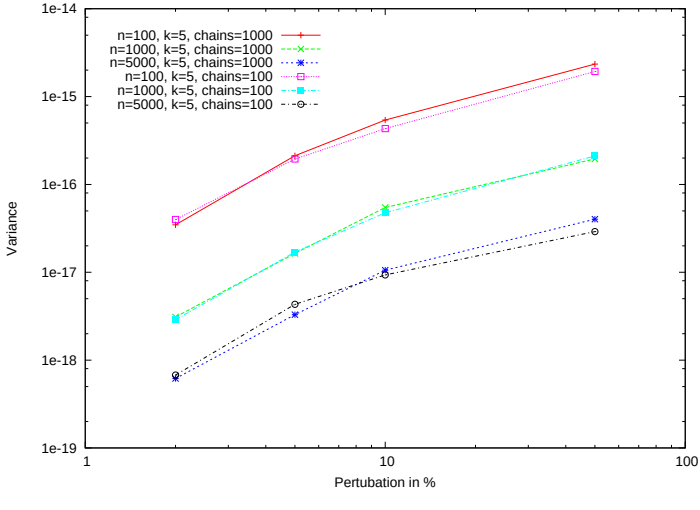


Fig. 5.7 The dependence of variance of the r.v. on perturbation of matrices with "small" spectral norms ($\|A\| \approx 0.1$).

show how one may control robustness (and the stochastic error of MC algorithms). It seems that it's very important to have a special balancing procedure as a pre-processing before running the Monte Carlo code. Such a balancing procedure ensures robustness of the algorithm and therefore

relatively small values for the stochastic component of the error.

If one is interested in computing dominant or the smallest by modulo eigenvalue the balancing procedure should be done together with choosing appropriate values for the acceleration parameter q (or α) if Resolvent MC is used. If all these procedures are properly done, then one can have robust high quality Monte Carlo algorithm with nice parallel properties.

Parallel properties of the algorithms is another very important issue of the acceleration analysis. It is known that Monte Carlo algorithms are inherently parallel [Dimov (2000); Dimov *et al.* (2001); Dimov and Karaivanova (1998)]. Nevertheless, it is not a trivial task to chose the parallelization scheme in order to get appropriate load balancing of all processors (computational nodes). In our experiments we distribute the job dividing the number of Markov chains between nodes. Such a method of parallelization seems to be one of the most efficient from the point of view of good load balancing [Dimov and Karaivanova (1998)]. Experiments are performed using Message Passing Interface (MPI). Some important parameters of the random matrices used in our experiments are presented in Table 5.2.

Table 5.2 Matrices for testing parallel properties

Matrix name	Size n	# of non-zero elements per row	λ_n	λ_1
A_3	128	52	1.0000	64.0000
A_4	512	178	1.0000	64.0000
A_5	1000	39	1.0000	-1.9000
A_6	1024	56	1.0000	64.0000
A_7	1024	322	1.0000	64.0000
A_8	2000	56	1.0000	64.0000

The test matrices are produced using a specially created generator of symmetric matrices called *MATGEN*. This generator allows to generate random 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 Givens rotations are used such that the angle of rotation and the place of appearing the non-zero entries are randomly chosen. The test matrices used in our numerical experiments are of size 128 to 2000 and have different number of non-zero elements. Using

MATGEN we are 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.

The numerical tests are performed on an Shared Memory Machine (SMM). The shared memory machine 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 shared memory machine on which our experiments are performed consists of a mesh-grid of $16 \times 4 = 64$ nodes. Data needed for processing by the shared memory machine must be shared between processors by MPI. There are a number of libraries available for message passing on the shared memory machine system.

Before we analyse the parallel properties of the Resolvent MC we looked at the convergence of the algorithm with increasing of the number of Markov chains N. According to our analysis the convergence depends very much on the balancing of matrices. We have to take into account that matrices A_3 and A_6 are very badly balanced, matrices A_4 and A_5 are well balanced, and matrices A_7 and A_8 are not well balanced. Convergence results are presented in Table 5.3.

Table 5.3 Convergence of the Resolvent MC algorithm with the number of Markov chains N.

N	Matrix A_3 $n = 128$	Matrix A_4 $n = 512$	Matrix A_5 $n = 1000$	Matrix A_6 $n = 1024$	Matrix A_7 $n = 1024$	Matrix A_8 $n = 2000$
10^3	64.13	1.0278	-1.9068	64.35	0.9268	63.99
10^4	63.33	0.9958	-1.9011	64.19	0.9865	64.00
10^5	63.18	0.99998	-1.9002	64.17	0.9935	64.02
10^6	63.19	1.00001	-1.9000	64.16	0.9959	64.01

As one can expect for well balanced matrices A_4 and A_5 the algorithm converges well. For not well balanced matrices A_7 and A_8 the convergence is good enough. For the matrix A_6 the algorithm still converges, but for A_3 it does not converge since the variance of the r.v. increases dramatically, so that the large number of Markov chains N even makes the situation worst.

The results of runs on a parallel system are given in Tables 5.4 to 5.7. The computational cost τ of the MAO algorithm is measured in milliseconds.

Experimental results presented on Tables 5.4 and 5.5 show that the

Table 5.4 Resolvent Monte Carlo Method (RMC) for matrix A_4 ($\lambda_{min} = 1.00000$). Implementation on SMM: computational time, speed-up and efficiency for $N = 10^6$. Calculated $\lambda_{max} = -1.900057$

Number of nodes	1	2	3	4	5	6	7	8	9	10
Time (ms)	70.75	35.97	24.9	18.3	14.78	12.96	11.49	9.63	8.67	7.68
Speed-up 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 5.5 Resolvent Monte Carlo Method (RMC) for matrix A_5 ($\lambda_{max} = -1.9000$). Implementation on SMM for $N = 10^6$. Calculated $\lambda_{max} = -1.900057$.

Number of nodes	1	2	3	4	5	6	7	8	9	10
Time (ms)	26.72	14.05	10.05	7.97	6.75	5.02	4.51	3.75	3.36	3.30
Speed-up 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

speed-up and parallel efficiency is very good. The results for matrix A_7 presented in Table 5.6 are not as good as for matrices A_4 and A_5. The reason is that the computational task in this case is much smaller. The number of Markov chain used is 10^5 instead of 10^6. It should be remembered that for all matrices the same method of parallelization is used. Thus, to be able to exploit well the parallel behaviours of the Resolvent MC one should have a large enough computational task (in comparison with communications).

Some information about the computational complexity, speed-up and parallel efficiency of the algorithm is presented in Figures 5.8 and 5.9. From Figure 5.9 it could be seen that when the computational task is small with comparison with the communications the speed-up increases not as rapidly as in the case of large computational tasks.

The main observations from the computational experiments are the following.

- The systematic error depends very much on the spectrum of the iter-

Table 5.6 Resolvent Monte Carlo Method (RMC) for matrix A_7 ($\lambda_{min} = 1.0$). Implementation on SMM (Number of trajectories $N = 10^5$).

Number of nodes	λ_{min}	Time (ms)	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 5.7 Computational cost τ (in millisecond) of MAO algorithm Implementation of the Resolvent Monte Carlo Algorithm for evaluation of λ_1 using MPI (number of Markov chains $N = 10^5$; $q > 0$ for all experiments).

Number of nodes	1	2	3	4	5
Matrix A_3 n=128	18	9	6	4	3
Matrix A_4 n=1024	30	15	10	7	6
Matrix A_5 n=2000	21	11	7	5	4

ation matrix. For a reasonable choice of the acceleration parameter q (or respectively α) the convergence of Resolvent MC can be increased significantly (in comparison with the Plain Power MC).

- The experimental analysis of the stochastic component of the error shows that the variance can be reduced significantly if a pre-processing balancing procedure is applied.
- *The computational cost (time)* τ is almost independent from the size of the matrix. It depends linearly from the mathematical expectation of the number of non-zero elements per row.
- There is a linear dependence of the computational cost from the number of Markov chains N.

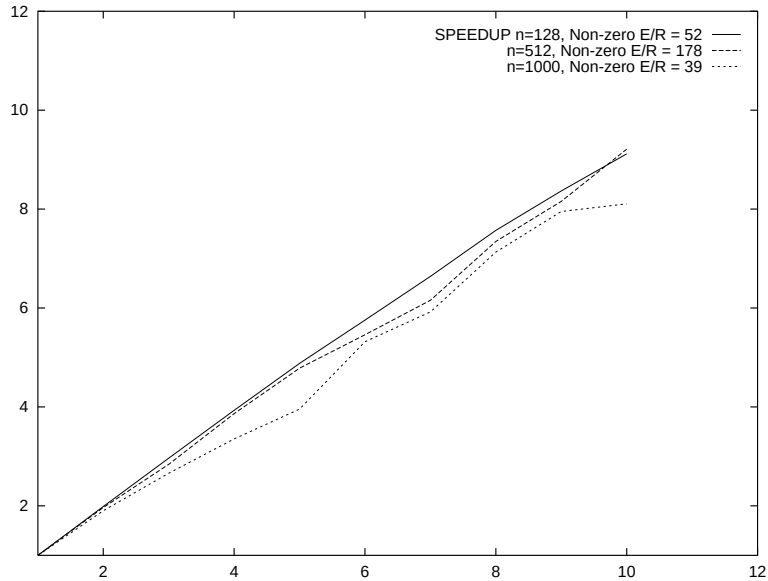


Fig. 5.8 Dependence of the speed-up on the number of processors for different matrices (A_3, A_4, and A_5). The computational time is relatively large in comparison with communicational time for all matrices; the number of realizations N for all matrices is relatively large.

- When MAO algorithm is run on parallel systems the speed-up is almost linear when the computational cost τ for every processor is not too small.
- A very important observation is that the implemented Power and Resolvent MC algorithms scale very well with increasing number of computational nodes. Our experience also shows that the code is easy portable from one computational system to another.

All observations are expected; they confirm the theoretical analysis of MAO algorithm.

5.5 Conclusion

In this chapter we have analysed the robustness and applicability of the Almost Optimal Monte Carlo algorithm for solving a class of linear algebra problems based on bilinear form of matrix powers $(v, A^k h)$. We have shown how one has to choose the acceleration parameter q (or α) in case of using

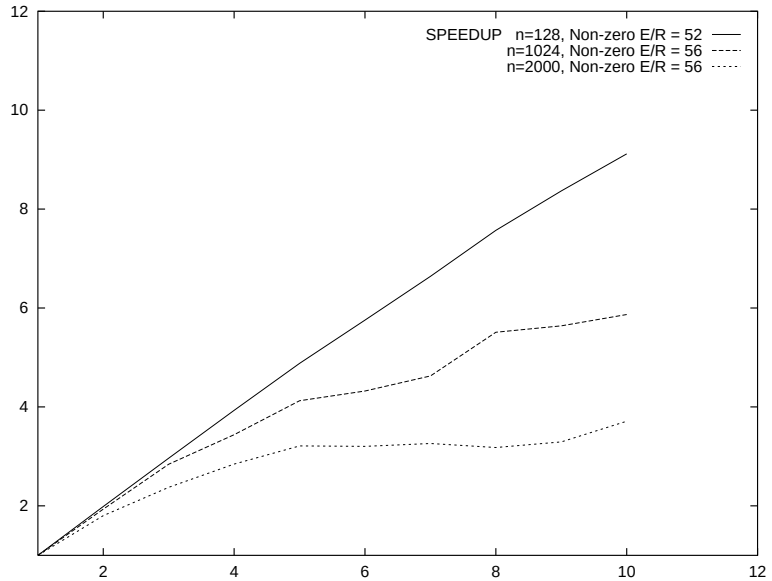


Fig. 5.9 Dependence of the speed-up from the number of processors for different matrices (A_3, A_6, and A_8). The computational time is relatively small in comparison with communicational time for matrices A_6 ($n = 1024$) and A_8 ($n = 2000$) since the number of realizations N is 10 times smaller than in the case of matrix A_3).

Resolvent Power MC. We analysed the systematic error and showed that the convergence can not be better that $O\left(\frac{1+|q|\lambda_n}{1+|q|\lambda_{n-1}}\right)^m$. We have analysed theoretically and experimentally the robustness. We have shown that with increasing the perturbations of entries of perfectly balanced matrices the error and the variance are increasing too. Especially small matrices have a high variance. For a rising power of A an increase of the relative error can be observed. The robustness of the Monte Carlo algorithm with balanced matrices with matrix norms much smaller than 1 has been demonstrated. In these cases the variance has improved a lot compared to cases were matrices have norms close to 1. We can conclude that the balancing of the input matrix is very important for MC computations. A balancing procedure should be performed as an initial (preprocessing) step in order to improve the quality of Monte Carlo algorithms. For matrices that are "close" in some sense to the stochastic matrices the accuracy of the MC algorithm is fairly high.

Chapter 6

Monte Carlo Methods for Boundary-Value Problems (BVP)

6.1 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 chapter 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 $|\alpha| = \alpha_1 + \alpha_2 + \cdots + \alpha_d$ 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 $\alpha : |\alpha| \le k$.

We consider the linear boundary value problem

$$Lu \equiv \sum_{|\alpha| \le 2m} a_\alpha(x) D^\alpha u(x) = -f(x), \qquad x \in \Omega \tag{6.1}$$

$$u(x) = \varphi(x), \qquad x \in \partial\Omega\,, \tag{6.2}$$

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 6.1. The equation

$$\sum_{|\alpha| \le 2m} a_\alpha(x) D^\alpha u(x) = -f(x)$$

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 (6.1, 6.2) exists and is unique [Jost (2002); Miranda (1955)].

We shall study Monte Carlo algorithms for calculating linear functionals of the solution of the problem (6.1, 6.2)

$$J(u) = (h, u) = \int_\Omega u(x)h(x)dx, \tag{6.3}$$

where $h \in \mathbf{X}^*(\Omega)$ ($\mathbf{X}^*(\Omega)$ is the dual functional space to $\mathbf{X}(\Omega)$).

For many applications $\mathbf{X} = \mathbf{L}_1$ and thus $\mathbf{X}^* = \mathbf{L}^\infty$, or $\mathbf{X} = \mathbf{L}_2$, $\mathbf{X}^* = \mathbf{L}_2$.

There are two approaches for calculating (6.3). The first approach uses a discretization of the problem (6.1, 6.2) on a mesh and solves the resulting linear algebraic system, which approximates the original problem (6.1, 6.2). This approach leads to the so-called *grid Monte Carlo algorithm*, or *grid walk algorithm*. The second approach - the *grid-free approach* - uses an integral representation for the problem (6.1, 6.2).

6.2 Grid Monte Carlo Algorithm

Consider a regular mesh (lattice) with step-size h in $\mathbb{R}^d$. 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 $\mathbb{R}^d \setminus \bar{\Omega}$) 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:

$$(L_h u_h)_i = \sum_{x_j \in P_k(x_i)} a_h(x_i, x_j)u_h(x_j) \ , \tag{6.4}$$

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 discretization of (6.4), the following system of linear equations arises

$$Au = b, \tag{6.5}$$

where $b = (b_1, \ldots, b_n)^T \in I\!R^{n\times 1}$ is an n-dimensional vector and $A \in I\!R^{n\times n}$ is an $n \times n$-dimensional matrix. For solving the system (6.5) one can use MC methods described in Chapter 4.

6.3 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 an example of the following linear elliptic BVP:

$$\Delta u(x) - c^2 u(x) = -\varphi(x),\ x \in \Omega \tag{6.6}$$

$$u(x) = \psi(x),\ x \in \partial\Omega, \tag{6.7}$$

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 (6.6, 6.7).

From the theory of fundamental solutions it follows that the solution of the problem (6.6, 6.7) can be represented as the integral equation (4.6) (see, Section 4.1) [Bitzadze (1982); Ermakov and Mikhailov (1982)], where

$$k(x,y) = \begin{cases} \dfrac{cd(x)}{\sinh[cd(x)]}\delta(y-x) & \text{, when } x \in \Omega \setminus \partial\Omega \\ 0 & \text{, when } x \in \partial\Omega \end{cases}$$

$$f(x) = \begin{cases} \dfrac{1}{4\pi}\displaystyle\int \frac{\sinh((d(x)-|y-x|)c}{|y-x|\sinh[cd(x)]}\varphi(y)dy & \text{, when } x \in \Omega \setminus \partial\Omega \\ \psi(x) & \text{, when } x \in \partial\Omega \end{cases}$$

and $d = d(x)$ is the distance from x to the boundary $\partial\Omega$.

It will it be necessary to calculate the functional (6.3), where u is the solution of the problem (6.6, 6.7) and h is a given function.

This representation permits the use of a random variable for calculating the functional (6.3). 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:

$$Mu = -\phi(x),\ x \in \Omega,\ \Omega \subset I\!R^3 \tag{6.8}$$

$$u = \psi(x),\ x \in \partial\Omega, \tag{6.9}$$

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 6.2. 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 $\sigma^{(k)}(z_1, z_2) \in \mathbf{H}^{\lambda}(\alpha; \partial\Omega)$, i.e.

$$|\sigma^{(k)}(y) - \sigma^{(k)}(y')| \leq \alpha |y - y'|^{\lambda},$$

where the vectors $y \equiv (z_1, z_2)$ and $y' \equiv (z_1', z_2')$ are 2-dimensional vectors, α is a 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 $b_j,\ c(x) \in \mathbf{H}^{\lambda}(\alpha; \bar{\Omega}),\ c(x) \leq 0$ and $\phi \in \mathbf{H}^{\lambda}(\alpha; \Omega) \cap \mathbf{C}(\bar{\Omega}),\ \psi \in \mathbf{C}(\partial\Omega)$, the problem (6.8, 6.9) has a 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 ([Jost (2002)], p. 179 and [Bitzadze (1982)], 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 (6.3).

6.3.1 *Local Integral Representation*

We use the grid-free Monte Carlo approach to estimate the functional (6.3). This approach is based on the use of a local integral representation of the solution $u(x)$ in the problem (6.8, 6.9). 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:

$$u(x) = \int_{B(x)} k(x,y)u(y)dy + f(x) \tag{6.10}$$

assuming that such a representation exists.

The iterative Monte Carlo process converges when the condition

$$\| K(u) \|_{L_1} = \max_{x\in\Omega} \int_{\Omega} | k(x,y) | dy \leq q < 1 \tag{6.11}$$

holds.

For the existence of the integral representation, (6.10) might be obtained using the result of C. Miranda [Miranda (1955)] 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 6.1. *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

$$\int_{\Omega} v(x)Mu(x)dx = \int_{\Omega} u(x)M^*v(x)dx. \tag{6.12}$$

To prove (6.12) we use Green's formulas:

$$\int_\Omega u(x)\sum_{i=1}^{3}\frac{\partial v(x)}{\partial x_{(i)}}dx = -\int_\Omega v(x)\sum_{i=1}^{3}\frac{\partial u(x)}{\partial x_{(i)}}dx + \int_{\partial\Omega}\sum_{i=1}^{3}u(x)v(x)n_i d_x S \tag{6.13}$$

and

$$\int_\Omega u(x)\Delta v(x)dx = -\int_\Omega \mathrm{grad}\, u(x)\mathrm{grad}\, v(x)dx + \int_{\partial\Omega} u(x)\sum_{i=1}^{3} n_i\frac{\partial v(x)}{\partial x_{(i)}}d_x S,$$

where

$$\Delta = \sum_{i=1}^{3}\frac{\partial^2}{\partial x_{(i)}^2},\ \ \mathrm{div}\,\mathbf{b}(x) = \sum_{i=1}^{3}\frac{\partial b_i(x)}{\partial x_{(i)}},$$
$$\mathrm{grad}\, u(x) \equiv \left(\frac{\partial u(x)}{\partial x_{(1)}}, \frac{\partial u(x)}{\partial x_{(2)}}, \frac{\partial u(x)}{\partial x_{(3)}}\right),$$

and $\mathbf{n} \equiv (n_1, n_2, n_3)$ is the exterior normal for the boundary $\partial\Omega$.

Taking into consideration that

$$\mathrm{div}\,\mathbf{b}(x) = 0 \ \text{ and } \ \frac{\partial v(x)}{\partial x_{(i)}} = v(x) = 0 \ \textit{ for any } \ x \in \partial\Omega,\ i = 1, 2, 3,$$

we have

$$\int_\Omega v(x)Mu(x)dx = \int_\Omega v(x)\left(\Delta u(x) + \mathbf{b}(x)\mathrm{grad}\, u(x) + c(x)u(x)\right)dx$$
$$= -\int_\Omega \mathrm{grad}\, v(x)\mathrm{grad}\, u(x)dx + \int_{\partial\Omega} v(x)\sum_{i=1}^{3} n_i\frac{\partial u(x)}{\partial x_{(i)}}d_x S$$
$$+\int_\Omega v(x)\mathbf{b}(x)\mathrm{grad}\, u(x)dx + \int_\Omega v(x)c(x)u(x)dx$$
$$= -\int_\Omega \mathrm{grad}\, v(x)\mathrm{grad}\, u(x)dx + \int_\Omega v(x)\sum_{i=1}^{3} b_i(x)\frac{\partial u(x)}{\partial x_{(i)}}dx + \int_\Omega v(x)c(x)u(x)dx.$$

On the other hand

$$\int_\Omega u(x)M^*v(x)dx = \int_\Omega u(x)\left[\Delta v(x) - \mathbf{b}(x)\mathrm{grad}\, v(x) + c(x)v(x)\right]dx$$

$$= -\int_\Omega \mathrm{grad}\, u(x)\mathrm{grad}\, v(x)dx + \int_{\partial\Omega} u(x)\sum_{i=1}^{3} n_i \frac{\partial v(x)}{\partial x_{(i)}} d_x S$$

$$-\int_\Omega u(x)\mathbf{b}(x)\mathrm{grad}\, v(x)dx + \int_\Omega u(x)c(x)v(x)dx$$

$$= -\int_\Omega \mathrm{grad}\, u(x)\mathrm{grad}\, v(x)dx - \int_\Omega u(x)\sum_{i=1}^{3} b_i(x)\frac{\partial v(x)}{\partial x_{(i)}}dx$$

$$+\int_\Omega u(x)c(x)v(x)dx$$

$$= -\int_\Omega \mathrm{grad}\, u(x)\mathrm{grad}\, v(x)dx + \int_\Omega v(x)\sum_{i=1}^{3}\frac{\partial(u(x)b_i(x))}{\partial x_{(i)}}dx$$

$$-\int_{\partial\Omega}\sum_{i=1}^{3} n_i b_i(x)u(x)v(x)dx + \int_\Omega v(x)c(x)u(x)dx$$

$$= -\int_\Omega \mathrm{grad}\, u(x)\mathrm{grad}\, v(x)dx + \int_\Omega v(x)\sum_{i=1}^{3} b_i(x)\frac{\partial u(x)}{\partial x_{(i)}}dx$$

$$+\int_\Omega v(x)c(x)u(x)dx.$$

From the last result there follows the proof of the lemma. □

The Levy's function for the problem (6.8, 6.9) is

$$L_p(y,x) = \mu_p(R)\int_r^R (1/r - 1/\rho)p(\rho)d\rho, \quad r \le R, \tag{6.14}$$

where the following notations are used:

$p(\rho)$ is a density function;

$$r = |x-y| = \left(\sum_{i=1}^{3}(x_{(i)} - y_{(i)})^2\right)^{1/2};$$

$$\mu_p(R) = \left[4\pi q_p(R)\right]^{-1};$$

$$q_p(R) = \int_0^R p(\rho)d\rho.$$

It is clear that the 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 (6.14) defines a family of functions.

We seek a choice of $p(\rho)$ which leads to a representation of type (6.10). Moreover, the kernel of the integral transform should be a transition density function, i.e. $k(x,y) \geq 0$.

From an algorithmic point of view the domain $B(x)$ must be chosen in such a way that the coordinates of the boundary points $y \in \partial B(x)$ can be easily calculated.

Denote by $B(x)$ the ball:

$$B(x) = B_R(x) = \{y : r =\mid y - x \mid \leq R(x)\}, \tag{6.15}$$

where $R(x)$ is the radius of the ball.

For the Levy's function $L_p(y,x)$ the following representation holds (see, [Miranda (1955)]):

$$\begin{aligned} u(x) = &\int_{B(x)} \left(u(y) M_y^* L_p(y,x) + L_p(y,x)\phi(y)\right) dy \\ &+ \int_{\partial B(x)} \sum_{i=1}^{3} n_i \left[\left(\frac{L_p(y,x)\partial u(y)}{\partial y_{(i)}} - \frac{u(y)\partial L_p(y,x)}{\partial y_{(i)}} \right) \right. \\ &- \left. b_i(y) u(y) L_p(y,x) \right] d_y S, \end{aligned} \tag{6.16}$$

where $\mathbf{n} \equiv (n_1, n_2, n_3)$ is the exterior normal to the boundary $\partial T(x)$.

Formula (6.16) 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 (6.16) holds.

Now we express the solution $u(x)$ by the Green's function $G(x,y)$. It is known, that the Green's function is a solution of the problem:

$$\begin{aligned} M_y^* G(x,y) &= -\delta(x-y), \ y \in \Omega \setminus \partial\Omega \setminus \{x\}, \\ G(x,y) &= 0, \quad y \in \partial\Omega, \ x \in \Omega \setminus \partial\Omega. \end{aligned}$$

The Green's function is the Levy's function, $L_p(y,x)$, for which (6.8, 6.9) hold.

Under the condition $L_p(y,x) = G(x,y)$ from (6.16) it is possible to get the integral representation:

$$u(x) = \int_{B(x)} G(x,y) f(y) dy - \int_{\partial B(x)} \sum_{i=1}^{3} n_i \frac{\partial G(x,y)}{\partial y_{(i)}} u(y) d_y S. \tag{6.17}$$

Representation (6.17) is the basis for the Monte Carlo method.

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.

$$L_p(y,x) = \partial L_p(y,x)/\partial y_i = 0 \quad \text{for } y \in \partial B(x), \quad i = 1,2,3.$$

Lemma 6.2. *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

$$k \geq \max_{x\in\Omega} \mid \mathbf{b}(x) \mid + R \max_{x\in\Omega} \mid c(x) \mid \tag{6.18}$$

and R is the radius of the maximal ball $B(x) \subset \bar{\Omega}$.

Proof. The condition

$$L_p(y,x) = 0, \quad \text{for any} \ \ y \in \partial B(x)$$

obviously holds. It follows from (6.14), (6.15), since if $y \in \partial B(x)$, then $r = R$ and $L_p(y,x) = 0$.

The condition

$$\partial L_p(y,x)/\partial y_{(i)} = 0, \quad \text{for any} \ \ y \in \partial B(x), \quad i = 1,2,3$$

can be checked immediately. Indeed,

$$\begin{aligned}
\frac{\partial L_p(y,x)}{\partial y_{(i)}} &= \frac{\partial L_p}{\partial r}\frac{\partial r}{\partial y_{(i)}} = \mu_p(R)\frac{\partial}{\partial r}\left(\int_r^R (1/r - 1/\rho)p(\rho)d\rho\right)\frac{\partial r}{\partial y_{(i)}} \\
&= \mu_p(R)\frac{\partial}{\partial r}\left(\frac{1}{r}\int_r^R p(\rho)d\rho - \int_r^R \frac{1}{\rho}p(\rho)d\rho\right)\frac{\partial r}{\partial y_{(i)}} \\
&= \mu_p(R)\left[-\frac{1}{r^2}\int_r^R p(\rho)d\rho + \frac{1}{r}\left(-p(r)\right) - \left(-\frac{1}{r}p(r)\right)\right]\frac{\partial r}{\partial y_{(i)}} \\
&= \mu_p(R)\left(-\frac{1}{r^2}\int_r^R p(\rho)d\rho\right)\frac{\partial r}{\partial y_{(i)}}.
\end{aligned}$$

Taking into consideration that $\frac{\partial r}{\partial y_{(i)}} = \frac{-(x_{(i)}-y_{(i)})}{r}$ one can get

$$\frac{\partial L_p(y,x)}{\partial y_{(i)}} = \mu_p(R)\frac{(x_{(i)}-y_{(i)})}{r^3}\int_r^R p(\rho)d\rho. \tag{6.19}$$

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:

$$\begin{aligned} M_y^* L_p(y,x) = &\sum_{i=1}^{3}\left(\frac{\partial^2 L_p(y,x)}{\partial y_{(i)}^2}\right) \\ &- \sum_{i=1}^{3}\left(b_i(y)\frac{\partial L_p(y,x)}{\partial y_{(i)}}\right) + c(y)L_p(y,x). \end{aligned} \tag{6.20}$$

The second term of (6.20) is calculated using (6.19), i.e.

$$\sum_{i=1}^{3} b_i(y)\frac{\partial L_p(y,x)}{\partial y_{(i)}} = \mu_p(R)\sum_{i=1}^{3} b_i(y)\frac{(x_{(i)}-y_{(i)})}{r^3}\int_r^R p(\rho)d\rho. \tag{6.21}$$

Calculate the first term in (6.20). 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 ([Tikchonov and Samarskii (1977)], 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, (6.14)). Thus,

$$\begin{aligned}\Delta_y L_p(y,x) = \Delta_{r,\theta,\varphi} L_p(r) &= \frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2 \frac{\partial L_p(r)}{\partial r}\right) \\ &= \mu_p(R)\frac{1}{r^2}\frac{\partial}{\partial r} r^2 \frac{\partial}{\partial r}\left(\int_r^R (1/r - 1/\rho) p(\rho) d\rho\right) \\ &= \mu_p(R)\frac{1}{r^2}\frac{\partial}{\partial r}\left(r^2\left(-\frac{1}{r^2}\right)\int_r^R p(\rho)d\rho\right) \\ &= \mu_p(R)\left(-\frac{1}{r^2}\right)\frac{\partial}{\partial r}\int_r^R p(\rho)d\rho = \mu_p(R)\frac{p(r)}{r^2}. \qquad (6.22)\end{aligned}$$

Taking into consideration (6.20), (6.21) we obtain:

$$\begin{aligned}M_y^* L_p(y,x) = \mu_p(R)\frac{p(r)}{r^2} &- \mu_p(R)c(y)\int_r^R \frac{p(\rho)}{\rho}d\rho \\ &+ \frac{\mu_p(R)}{r^2}\left[c(y)r + \sum_{i=1}^3 b_i(y)\frac{y_{(i)} - x_{(i)}}{r}\right]\int_r^R p(\rho)d\rho.\end{aligned}$$

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

$$\begin{aligned}\Gamma_p(y,x) = p(r) &+ c(y)r\left(\int_r^R p(\rho)d\rho - \int_r^R \frac{p(\rho)r}{\rho}d\rho\right) \\ &+ \sum_{i=1}^3 b_i(y)\frac{y_{(i)} - x_{(i)}}{r}\int_r^R p(\rho)d\rho.\end{aligned}$$

It is necessary to show that for all $y \in B(x)$ the function $\Gamma_p(y,x)$ is non-negative. From the condition $c(y) \le 0$ it follows that

$$\begin{aligned}\Gamma_p(y,x) = p(r) &- \left|c(y)r\left(\int_r^R p(\rho)d\rho - \int_r^R \frac{p(\rho)r}{\rho}d\rho\right)\right| \\ &+ \sum_{i=1}^3 b_i(y)\frac{y_{(i)} - x_{(i)}}{r}\int_r^R p(\rho)d\rho \ge 0. \qquad (6.23)\end{aligned}$$

So, it is necessary to prove (6.23). For $p(r) = e^{-kr}$ we have

$$p(r) \ge e^{-kr} - e^{-kR} = k\int_r^R p(\rho)d\rho.$$

Choosing

$$k \geq \max_{x\in\Omega} \mid \mathbf{b}(x) \mid + R \max_{x\in\Omega} \mid c(x) \mid$$

one can obtain

$$\begin{aligned} p(r) &\geq \left(\max_{x\in\Omega} \mid \mathbf{b}(x) \mid + R \max_{x\in\Omega} \mid c(x) \mid \right) \int_r^R p(\rho) d\rho \\ &\geq \mid c(y) \mid r \int_r^R p(\rho) d\rho + \mid \mathbf{b}(y) \mid \int_r^R p(\rho) d\rho \\ &\geq \mid c(y) \mid r \left(\int_r^R p(\rho) d\rho - \int_r^R \frac{p(\rho) r}{\rho} d\rho \right) \\ &+ \left| \sum_{i=1}^{3} b_i(y) \frac{y_{(i)} - x_{(i)}}{r} \right| \int_r^R p(\rho) d\rho. \end{aligned} \tag{6.24}$$

One can see that (6.23) follows from (6.24). □

Now the representation (6.10) can be written in the form:

$$u(x) = \int_{B(x)} M_y^* L_p(y, x) u(y) dy + \int_{B(x)} L_p(y, x) \phi(y) dy. \tag{6.25}$$

The last representation enables the formulation of a unbiased estimate for the solution of the problem under consideration.

6.3.2 *Monte Carlo Algorithms*

Some simple numerical examples for performing *grid* and *grid-free* Monte Carlo algorithms will now be considered.

Let the operator L in the equation (6.1) be the Laplacian:

$$L = \Delta.$$

Using a regular discretisation with a step–size h equation (6.1) is approximated by the following difference equation

$$\Delta_h^{(d)} u = -f_h. \tag{6.26}$$

Assume that (6.26) is solved for the i^{th} point $i = (i_1, \ldots, i_d)$:

$$u_i = L_h u + \frac{h^2}{2d} f_i,$$

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 (6.26) becomes

$$u_{ij} = \frac{1}{4}\Lambda_1(i,j) + \frac{h^2}{4} f_{i,j}. \tag{6.27}$$

The matrix form of equation (6.27) has only $2d$ non-zero elements in each row and they all are equal to $\frac{1}{2d}$.

The *grid* Monte Carlo algorithm for solving (6.27) consists in simulating a Markov chain with initial density p_0 which is *permissible* to the vector h (see, Definition 4.1 given in Section 4 and formulas (4.20) and (4.21)). The probability $p_{\alpha\beta}$ for the transition from the point α to the next point β in our case 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 $\partial\Omega_h$

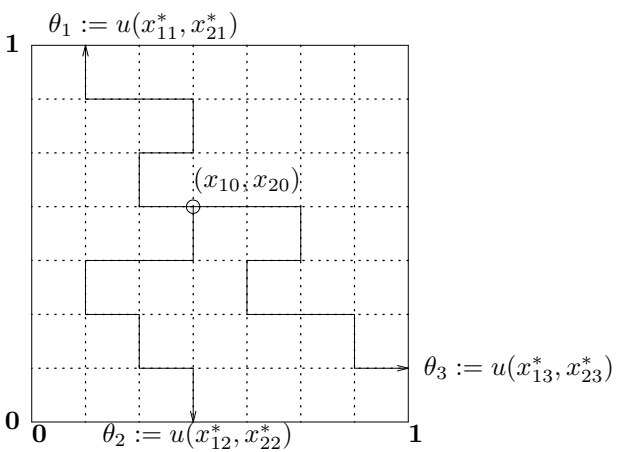


Fig. 6.1 *Grid* Monte Carlo algorithm (N trajectories from the initial point (x_{10}, x_{20}) to the boundary are constructed and the mean value of the encountered boundary values is computed).

The well known *grid* algorithm (see, for example, [Shreider (1964)]) can be described in pseudo-code notation (see Figure 6.1):

Algorithm 6.1.

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.

Now consider the *grid-free Monte Carlo algorithm* based on the local integral representation of the problem.

First, let us describe the *selection algorithm* (due to John von Neumann [von Neumann (1951)]) in general case. This approach is also commonly called the *acceptance-rejection* method or *accept-reject algorithm* [Robert and Casella (2004); von Neumann (1951)]. Suppose $v_1(x)$ 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 $\gamma v_2(x) \le v_1(x)$. If the last inequality holds, ξ is the needed realization. Otherwise, the process have to be repeated. This means that, with enough replicates, the algorithm generates a sample from the desired distribution $v_2(x)/V_2$. There are a number of extensions to this algorithm, such as the *Metropolis (or Metropolis-Hastings)* algorithm [Metropolis *et al.* (1953); Berg (2004); Hastings (1970); Chib and Greenberg (1995)]. The efficiency of the *selection algorithm* is measured by $E = V_1/V_2$.

A local integral representation (6.25) for the boundary value problem (6.8, 6.9) is obtained. Comparing (6.10) with (6.25) 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

$$p(x,y) = k(x,y) = M_y^* L_p(y,x) \geq 0. \tag{6.28}$$

This transition density function defines a Markov chain $\xi_1, \xi_2, \ldots, \xi_i$ such that every point ξ_j, $j = 1, \ldots, i-1$ is chosen on the maximal ball $B(x_{j-1})$, lying in Ω in accordance with the density (6.28). 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

$$Q_0 = 1 \tag{6.29}$$

$$Q_j = Q_{j-1} M_y^* L_p(\xi_j, \xi_{j-1}) / p(\xi_{j-1}, \xi_j), j = 1, 2, \ldots, i, \tag{6.30}$$

$\varphi(\xi_i)$ 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

$$u(\xi_0) = E\theta[\xi_0]. \tag{6.31}$$

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 (6.28) 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 (6.30) 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 (6.28). Thus, the problem of calculating $u(\xi_0)$ is reduced to estimating the expectation (6.31). As the 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 (6.28).

As it was shown in Section 4.1, 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 *rejection sampling 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.

$$\int_{B(x)} p_0(x,y)dy = 1. \tag{6.32}$$

Indeed,

$$\int_{B(x)} p_0(x,y)dy = \int_{B(x)} M_y^* L_p(y,x)\Big|_{c(y)\equiv 0} dy$$

$$= \int_{B(x)} \sum_{i=1}^{3} \frac{\partial^2 L_p(y,x)}{\partial y_{(i)}^2} dy - \int_{B(x)} \sum_{i=1}^{3} \left(b_i(y) \frac{\partial L_p(y,x)}{\partial y_{(i)}} \right) dy.$$

Apply Green's formula (6.13) to 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 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 (6.32).
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 one can 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 [Ermakov *et al.* (1984)] 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 [Dimov (1989)] 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 rejection sampling *grid-free* Monte Carlo algorithm was proved:

$$E \geq \frac{1 + \alpha}{2 + \alpha}, \tag{6.33}$$

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 6.1. *For the efficiency of the rejection sampling grid-free Monte Carlo algorithm the inequality:*

$$E \geq \frac{1 + \alpha}{2 + \alpha - \varepsilon_R}, \;\; 0 < \varepsilon_R = \frac{1}{e^{kR}} < 1,$$

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 $p_0(\mathbf{w}/r)$:

$$p_0(\mathbf{w}/r) = \frac{\sin\theta}{4\pi}\left[1 + \frac{\mid \mathbf{b}(x + r\mathbf{w}) \mid \cos(\mathbf{b}, \mathbf{w})}{p(r)} \int_r^R p(\rho)d\rho\right]$$
$$\leq \frac{\sin\theta}{4\pi}\left[1 + \frac{B}{p(r)} \int_r^R p(\rho)d\rho\right] = h_r(\mathbf{w})$$

where $B = \max_{x\in\Omega} \mid \mathbf{b}(x) \mid$.
On the other hand

$$h_r(\mathbf{w}) = \frac{\sin\theta}{4\pi}\left[1 + \frac{B}{p(r)} \int_r^R p(\rho)d\rho\right] = \frac{\sin\theta}{4\pi}\left[1 + \frac{B}{k}\left(1 - e^{-k(R-r)}\right)\right]$$
$$= \frac{\sin\theta}{4\pi}\left[1 + \frac{B}{k}\left(1 - \frac{e^{kr}}{e^{kR}}\right)\right]$$
$$\leq \frac{\sin\theta}{4\pi}\left[1 + \frac{B}{k}\left(1 - \frac{1}{e^{kR}}\right)\right] = H(\mathbf{w}). \tag{6.34}$$

The functions $h_r(\mathbf{w})$ and $H(\mathbf{w})$ are majorants for the $p_0(\mathbf{w}/r)$. For the efficiency of the rejection sampling 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

$$k = B + R\max_{x\in\Omega} \mid c(x) \mid, \quad \text{(see (6.18))},$$

$$\alpha = \frac{\max_{x\in\Omega} \mid c(x) \mid R}{B} \quad \text{and } \varepsilon_R = \frac{1}{e^{kR}}.$$

Taking into consideration (6.34), one can get

$$E \geq \frac{1+\alpha}{2 + \alpha - \varepsilon_R}, \tag{6.35}$$

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 (6.33) 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:

$$\begin{aligned} p(r,\mathbf{w}) = {} & \frac{\sin\theta}{4\pi q_p(R)} \\ & \times \left[p(r) + \left(\sum_{i=1}^{3} b_i(x+r\mathbf{w})w_i + c(x+r\mathbf{w})r \right) \right. \\ & \left. \times \int_r^R p(\rho)d\rho - c(x+r\mathbf{w})r^2 \int_r^R \frac{p(\rho)}{\rho} d\rho \right]. \end{aligned}$$

The following inequalities hold:

$$\begin{aligned} p(r,\mathbf{w}) \leq p_0(r,\mathbf{w}) & \leq \frac{p(r)}{q_p(R)} h_r(\mathbf{w}) \qquad (6.36) \\ & = \frac{\sin\theta\, p(r)}{4\pi q_p(R)} \left[1 + \frac{\max_{x\in\Omega} \mid \mathbf{b}(x) \mid}{p(r)} \int_r^R p(\rho)d\rho \right] \equiv h(r,\mathbf{w}). \end{aligned}$$

It is easy to prove that in case when $h(r,\mathbf{w})$ is a majorant of the function $p(r,\mathbf{w})$ the efficiency of the rejection sampling algorithm is

$$E \geq V \frac{1+\alpha}{2+\alpha-\varepsilon_R}.$$

This estimation follows from Theorem 6.1 and (6.36). Clearly, the efficiency E depends on the norm of the kernel $k(x,y)$, because $p(x,y) = k(x,y)$.

In the rejection sampling algorithm it is necessary to simulate a random variable η with a density

$$\begin{aligned} \bar{p}_r(\mathbf{w}) = 1 + & \left[\frac{\mid \mathbf{b}(x+r\mathbf{w}) \mid \cos(\mathbf{b},\mathbf{w}) + c(x+r\mathbf{w})r}{p(r)} \right] \times \\ & \times \int_r^R p(\rho)d\rho - \frac{c(x+r\mathbf{w})r^2}{p(r)} \int_r^R \frac{p(\rho)}{\rho} d\rho. \end{aligned}$$

Since

$$\bar{p}_r(\mathbf{w}) \leq 1 + \frac{B}{p(r)} \int_r^R p(\rho)d\rho = h(r)$$

the function $h(r)$ is a majorant for the rejection sampling 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 6.2.

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

$$\frac{p(r)}{q_p(R)} = \frac{ke^{-kr}}{1-e^{-kR}}. \tag{6.37}$$

3. **Calculate** the function

$$h(r) = 1 + \frac{B}{p(r)} \int_r^R p(\rho)d\rho = 1 + \frac{B}{k}\left(1 - e^{-k(R-r)}\right).$$

4. **Simulate** independent realizations $\mathbf{w}_j$ of a unique isotropic vector in $I\!R^3$.

5. **Simulate** independent realizations γ_j of a 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 random trajectory 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 $x = y$.

9. **Perform** N random trajectories repeating steps 2 to 8.

An illustration of the considered *grid-free* algorithm is given on Figure 6.2.

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

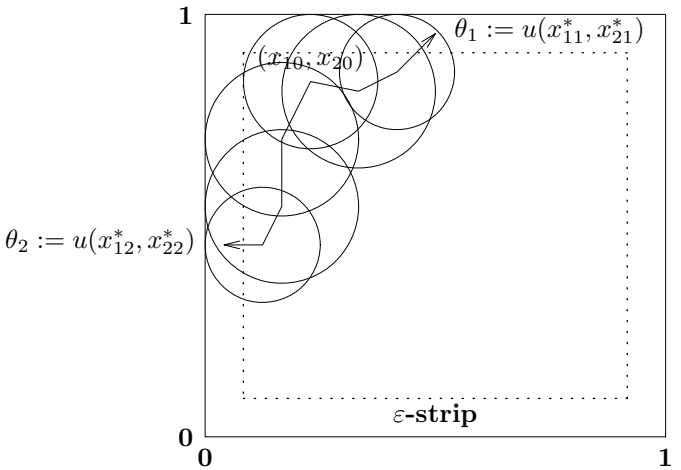


Fig. 6.2 Illustration of the Grid-free Monte Carlo algorithm.

variable τ with a density (6.37). 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 the value of $E\tau$:

$$E\tau = \int_0^R r\frac{p(r)}{q_p(R)}dr = \int_0^R \frac{rke^{-kr}}{1-e^{-kR}}dr = \frac{1}{k} + \frac{R}{1-e^{kR}}.$$

Obviously, the sequential 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).

6.3.3 *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:

- Monte Carlo algorithms are frequently used, within or in conjunction with more complex and large existing codes (usually written in FORTRAN or C), the easiness in programming makes the use of these machines very attractive;
- the peak performance of each processor of these machines is usually not very high, but when a large number of processors is efficiently used a high general computational performance can be reached.

The MIMD computer used for our tests is a IBM SP1 with 32 processors. The aim of our implementation was to demonstrate a high efficiency of a MIMD system in which each processing element is relatively slow. The algorithm has a very good scalability so that it can be easily implemented on modern and future MIMD systems. 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 [Plateau (1994)].

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 example is considered. The example deals with the following problem

$$\sum_{i=1}^{3}\left(\frac{\partial^2 u}{\partial x_{(i)}^2}+b_i(x)\frac{\partial u}{\partial x_{(i)}}\right)+c(x)u=0,\ \text{ in } \Omega=\mathbf{E}^3\equiv[0,1]^3.$$

Note that the cube $\mathbf{E}^3$ 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:

$$u(x_{(1)}, x_{(2)}, x_{(3)}) = e^{a_1 x_{(1)} + a_2 x_{(2)} + a_3 x_{(3)}}\ ,\ (x_{(1)}, x_{(2)}, x_{(3)}) \in \partial\Omega.$$

In our tests

$$b_1(x) = a_2 a_3 (x_{(2)} - x_{(3)}),\ b_2(x) = a_3 a_1 (x_{(3)} - x_{(1)}),\ b_3(x) = a_{(1)} a_2 (x_{(1)} - x_{(2)})$$

(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.

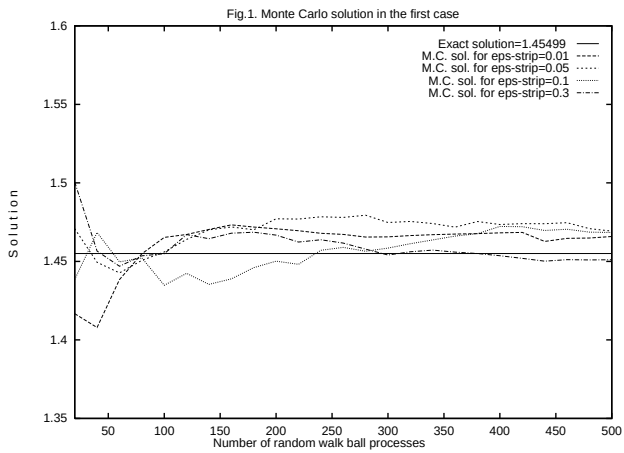


Fig. 6.3 Monte Carlo solution for the first case.

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

$$a_1 = -1\,,\ a_2 = -1\,,\ a_3 = -1\ \text{ and }\ k * R = 1.61603.$$

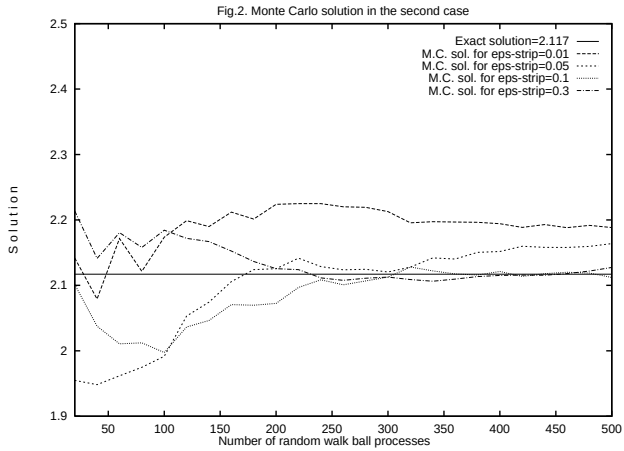


Fig. 6.4 Monte Carlo solution for the second case.

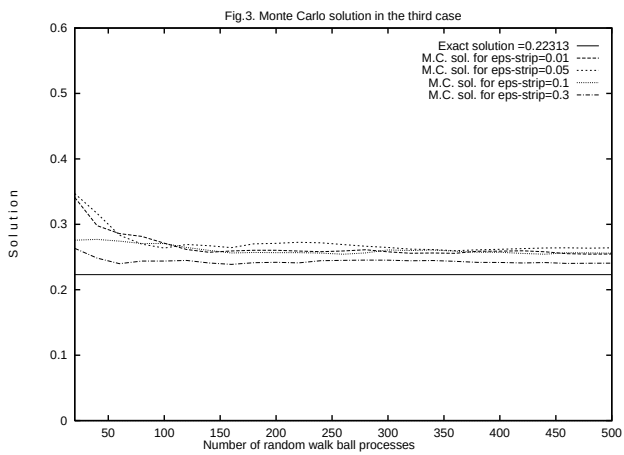


Fig. 6.5 Monte Carlo solution in the third case.

Four different ε-strip are used:

$$\varepsilon = 0.01,\ 0.05,\ 0.1,\ 0.3.$$

The results of evaluating the linear functional (4.7) for the above mentioned parameters and functions are presented in Figures 6.3 - 6.5, the case when

$$h(x) = \delta[(x_{(1)} - 1/2), (x_{(2)} - 1/2), (x_{(3)} - 1/2)].$$

Table 6.1 Selection efficiency and number of the steps to the boundary domain. The number of realizations of the random ball process is 600.

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

The efficiency of the selection *grid-free* Monte Carlo does not depend on the number of trajectories (see, Table 6.1). The result of selection efficiency confirms our corresponding theoretical result.

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:

$$a_1 = 1\,,\ a_2 = -0.5\,,\ a_3 = -0.5 \ \text{ and } \ \varepsilon - \text{strip} = 0.01\,,\ 0.05\,,\ 0.15.$$

For the definition of parallel efficiency and other parallel properties we refer to Chapter 9. The results of parallel implementation showed a very high scalability of the algorithm with increasing the size of the computational job (by increasing the number of random trajectories). The measured parallel efficiency increases with increasing the number of random trajectories and is close to 1.

6.3.4 *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 under consideration can be efficiently implemented on MIMD-machines. The algorithms under consideration are highly scalable.

This page intentionally left blank

Chapter 7

Superconvergent Monte Carlo for Density Function Simulation by B-Splines

In this chapter we consider the possibility of applying spline-functions to create superconvergent Monte Carlo algorithms. The density-function simulation is a very important approach for solving many problems in environmental mathematics, probabilistic theory and physics. For example, the problem of creating efficient algorithms for modeling random variables with a given density-function is of significant interest in air-pollution transport as the parameters of the transport like diffusion coefficient, deposition, drift-coefficients are all random variables [Dimov (1994); Dimov and Karaivanova (1994); Zletev and Dimov (2006)]. Problems of this type arise when it is necessary to estimate a unknown density of a random variable using a given number of its realizations.

The aim of this chapter is to solve this problem using a specially created superconvergent Monte Carlo algorithm for estimating the coefficients in the B-spline approximation of the unknown density.

It might be anticipated that B-splines could be a good tool for densities modeling because they are non-negative in finite intervals and vanish at the beginning and at the end of the intervals. Moreover, the system of B-splines of a given degree for different nodes of the mesh define a linear basis [de Boor (2001); Mahotkin and Pirimkulov (1984)]. The idea of the algorithm is to express the density of the random variable defined in the interval $[a, b]$ as a linear form of B-splines of degree k with defects 1. Error analysis will be carried out.

7.1 Problem Formulation

Consider a set of points

$$w_m = \{a = x_1 < x_2 < \cdots < x_m = b\}, \tag{7.1}$$

in the interval $[a, b]$.

Let $k \geq 0$ be an integer and $\{l_i\}$, $1 \leq l_i \leq k$, $i = 2, 3, \ldots, m$ be a sequence of integers.

Let the set T_n be introduced in the following way:

(i) $2(k+1)$ new nodes are added to the set w_m:

$$T_\nu = \{t_1 \leq t_2 \leq \cdots \leq t_{k+1} = x_1 < x_2 < \cdots < x_m$$

$$= t_{\nu-k} \leq t_{\nu-k+1} \leq \cdots \leq t_\nu\};$$

(ii) for $i = 2, 3, \ldots, m$ $\quad x_i$ is repeated l_i times in the sequence T_ν such that

$$\nu = 2(k+1) + \sum_{i=2}^{m} l_i.$$

Denote by k and l_i the degree and defects of the spline $B_{i,k}(x)$. Define the i^{th} B-spline of k^{th} degree as

$$B_{i,k}(x) = (t_{i+k+1} - t_i)[t_i, \ldots, t_{i+k+1}](. - x)_+^k, \;\; i = 1, 2, \ldots, \nu - k - 1,$$

where $[t_i, \ldots, t_{i+k}](. - x)_+^k$ is the k^{th} divided difference with respect to t of the function

$$(t - x)_+^k = [max\{0, (t - x)\}]^k,$$

where x is a fixed point.

Consider a density-function $f(x)$, defined on the interval $[a, b]$. Suppose

$$f(x) \in \mathbf{C}^{k+1}[a, b].$$

Then, an approximation g_L of f can be represented as [de Boor (2001)]:

$$g_L(x) = \sum_{i=1}^{L} c_i B_{i,k}(x), \;\; x \in [a, b], \;\; L = \nu - k - 1 \tag{7.2}$$

with an error of $O(h^{k+1})$ (c_i are constants). Denote by $B_k(x)$ the B-spline of k^{th} degree with defects $l_i = 1$ at the point $t = 0$ with integer nodes $t_j = j - 1$, $j = 1, 2, \ldots, k+1$. In this case the set T_ν has

$$\nu = 2k + m + 1$$

nodes.

Let $\gamma_j \in [0,1]$ be a random number (continuous uniformly distributed random variable with mathematical expectation $E\gamma_j = 1/2$ and variance $D\gamma_j = \sigma^2\gamma_j = 1/12$).

Let $p_k(x)$ be a density-function of the random variable

$$\xi_k = \sum_{j=1}^{k} \gamma_j.$$

Then, following [Mahotkin and Pirimkulov (1984)] $p_k(x) = B_k(x),\ \ x \in [0,k]$.

Denote by θ_k the following random variable:

$$\theta_k = \sqrt{12/k}(\xi_k - k/2).$$

One can obtain an algorithm for simulation of random variables with densities $B_{i,k}(x)$ on a regular grid with step–size h:

$$\theta_{i,k} = [(i-1) + \theta_k]h.$$

7.2 The Methods

Consider the problem of estimating the error of the density-function $f(x)$ using N realizations $\{\xi_i\}_{i=1}^{N}$ of the r.v.

Consider the approximation $g_L(x)$ of $f(x)$ given by equation (7.2).

The coefficients c_j are obtained using the following representation:

$$\sum_{j=1}^{L} (B_{i,k}(x), B_{j,k}(x))c_j = (f, B_{i,k}(x)), \tag{7.3}$$

where

$$(f,g) = \int_a^b f(x)g(x)\,dx$$

is the inner product. Obviously, the functional $(f, B_{i,k}(x))$ is the mathematical expectation of $B_{i,k}(x)$ with a density-function $f(x)$:

$$(f, B_{i,k}(x)) = \int_a^b B_{i,k}(x)f(x)\,dx = E_\xi B_{i,k}(x), \tag{7.4}$$

where the random point ξ has a density $f(x)$. The inner product $(f, B_{i,k}(x))$ is estimated by the Monte Carlo method:

$$(f, B_{i,k}(x)) \approx \frac{1}{N}\sum_{j=1}^{N} B_{i,k}(\xi_j) = \hat{\theta}_N. \tag{7.5}$$

The constants c_j are calculated and an estimate of the unknown density

$$q_f(x) = \sum_{j=1}^{L} \hat{c}_j B_{j,k}(x)$$

is obtained. The error of the algorithm consists of two components:

$$\|q_f - f\|^2 \le \|f - g_L\|^2 + \|g_L - q_f\|^2 = r_s^2 + r_N^2(L),$$

where

$$\|f\| = \|f\|_{L_2} = \left(\int_a^b f^2(x)\, dx \right)^{1/2}.$$

The first component r_s^2 is the systematic error while the second one $r_N^2(L)$ is the stochastic error. An algorithm is considered good when $r_N(L)$ is approximately equal to r_s.

Estimate the stochastic error:

$$\begin{aligned} r_N^2(L) &= E\left\{ \int_a^b \left[\sum_{j=1}^{L} (\hat{c}_j - c_j) B_{j,k}(x) \right]^2 dx \right\} \\ &= E\left\{ \int_a^b \left[\sum_{i=1}^{L} \sum_{j=1}^{L} (\hat{c}_i - c_i)(\hat{c}_j - c_j) B_{i,k}(x) B_{j,k}(x) \right] dx \right\} \\ &= E\left\{ \sum_{i=1}^{L} \sum_{j=1}^{L} (\hat{c}_i - c_i)(\hat{c}_j - c_j) a_{i,j} \right\}, \end{aligned}$$

where

$$a_{i,j} = \int_a^b B_{i,k}(x) B_{j,k}(x)\, dx.$$

Let $\mathbf{r} = (r_{(1)}, r_{(2)}, \ldots, r_{(L)})$, $r_{(i)} = \hat{c}_i - c_i$, $i = 1, \ldots, L$ and

$$A = \{a_{i,j}\}_{i,j=1}^{L}$$

Obviously , the matrix A is positive definite.

Consider the product

$$A\mathbf{r} = \begin{pmatrix} \sum_{j=1}^{m} (\hat{c}_j - c_j) \int_a^b B_{1,k}(x) B_{j,k}(x)\, dx \\ \sum_{j=1}^{m} (\hat{c}_j - c_j) \int_a^b B_{2,k}(x) B_{j,k}(x)\, dx \\ \vdots \\ \sum_{j=1}^{m} (\hat{c}_j - c_j) \int_a^b B_{L,k}(x) B_{j,k}(x)\, dx \end{pmatrix} = \begin{pmatrix} \varepsilon_1 \\ \varepsilon_2 \\ \vdots \\ \varepsilon_L \end{pmatrix} = \varepsilon.$$

The stochastic error $r_N(L)$ depends on the given sampling of realizations $\{\xi_i\}_{i=1}^n$. Consider the estimate for the mathematical expectation of this error. Since the matrix A is positive definite, it is possible to express the expectation of the stochastic error as follows:

$$E(A\mathbf{r},\mathbf{r}) = E(\varepsilon, A^{-1}\varepsilon) \le \|A^{-1}\| \times E\left(\|\varepsilon\|^2\right).$$

The errors ε_i, $i = 1,\dots,L$ depend on the error of the Monte Carlo algorithm for evaluation of integrals (7.4). This is, because the system (7.3) for finding coefficients c_j can be solved using direct algorithms. (The matrix A is positive definite and diagonally dominant.)

Thus, the problem of increasing the accuracy of the algorithm is reduced to the problem of obtaining a good Monte Carlo algorithm for evaluation of integrals (7.4).

The error is studied for two special cases. The first one is when splines of degree $k \ge 2$ are used for modeling densities.

The second one is when splines of degree $k \ge 3$ are used.

Case 1. When B-spline of degree 2 or more with defects 1 are used it is useful to take into account the fact that the splines $B_{i,2}(x)$ have continuous derivatives. In fact, any spline of degree k with defects l_i has continuous derivatives of degree $r = k - l_i$. In our case $r \ge 1$.

Since the first derivative of $B_{i,k}(x)$ is continuous there exist constants $\alpha_{i,k}$ such that

$$\left|\frac{dB_{i,k}(x)}{dx}\right| \le \alpha_{i,k}, \quad x \in [a,b].$$

In order to create a Monte Carlo algorithm with a high rate of convergence consider a sequence of random points $\{\xi_j\}_{j=1}^N$, $(\xi_1 < \xi_2 < \cdots < \xi_N)$.

Define a set of points $\{y_j\}_{j=0}^N$ such that

$$y_j = \frac{1}{2}(\xi_j + \xi_{j+1}), \quad j = 1,2,\dots,N-1 \text{ and } y_0 = a; \quad y_N = b.$$

The integral (7.5) can be represented as

$$J_{i,k} = \int_a^b B_{i,k}(x) f(x)\,dx = \sum_{j=1}^N \int_{y_{j-1}}^{y_j} B_{i,k}(x) f(x)\,dx.$$

Let

$$J_{i,k}^j = \int_{y_{j-1}}^{y_j} B_{i,k}(x) f(x)\,dx$$

and

$$f^{(j)} = \int_{y_{j-1}}^{y_j} f(x)\,dx. \tag{7.6}$$

One possible choice of $f^{(j)}$ as a measure of the integral (7.6) is the following:

$$f^{(j)} = \frac{y_j - y_{j-1}}{b-a}.$$

Obviously

$$\sum_{j=1}^{N} J_{i,k}^{j} = J_{i,k}, \quad \text{for} \quad k \geq 2. \tag{7.7}$$

and

$$\sum_{j=1}^{N} f^{(j)} = 1, \quad \text{for} \quad k \geq 2.$$

Moreover, there exist constants c_1 and c_2 such that

$$f^{(j)} \leq c_1/N \ \text{ and } \ \Delta x_j = y_j - y_{j-1} \leq c_2/N.$$

For example, one can see that when $f(x) = const$ and $\Delta x_j = \Delta x_{j+1}$ for $j = 1, 2, \ldots, N-1$ both constants c_1 and c_2 are equal to 1. Thus, the r.v. that defines the algorithm is

$$\theta_N = \sum_{j=1}^{N} f^{(j)} B_{i,k}(\xi^{(j)}).$$

The probable error in evaluating integrals is defined in Introduction as the value r_N for which the following equalities hold: $Pr\{|J_{i,k} - \theta_N| \leq r_N\} = Pr\{|J_{i,k} - \theta_N| > r_N\} = \frac{1}{2}$. For the crude Monte Carlo algorithm $r_N = c_{0.5}\sigma(\theta)N^{-1/2}$, where $c_{0.5} \approx 0.6745$ and $\sigma(\theta)$ is the standard deviation (see Subsection 2.2.1). A superconvergent Monte Carlo algorithm is an algorithm for which (eg, see, Section 2.4)

$$r_N = cN^{-1/2-\psi},$$

where c is a constant and $\psi > 0$.

Theorem 7.1. *Let $B_{i,k}(x)$ be a B-spline of degree $k \geq 2$ with defects $l_i = 1$. Then the probable error r_N of the Monte Carlo algorithm (7.7) in evaluating integrals (7.5) is*

$$r_N \leq \sqrt{2} c_1 c_2 \alpha_{i,k} N^{-3/2}.$$

Proof. Consider B-splines $B_{i,k}(x)$, where $k \geq 2$. For a fixed point $s^{(j)} \in \Delta x_j = [y_{j-1}, y_j]$ we have

$$B_{i,k}(x) = B_{i,k}\left(s^{(j)}\right) + B'_{i,k}\left(\eta^{(j)}\right)\left(x - s^{(j)}\right),$$

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

Since $|B'_{i,k}(\eta^{(j)})| \leq \alpha_{i,k}$ we have:

$$DB_{i,k}\left(\xi^{(j)}\right) \leq EB^2_{i,k}\left(\xi^{(j)}\right) \leq \alpha^2_{i,k} E(\xi^{(j)} - s^{(j)})^2$$
$$\leq \alpha^2_{i,k}\left(\sup_{x_1^{(j)}, x_2^{(j)} \in \Delta x_j}\left|x_1^{(j)} - x_2^{(j)}\right|\right)^2 \leq \alpha^2_{i,k} c_2^2 / N^2.$$

Now we can estimate $D\theta^*_N$. Since in our case $\theta_N = \sum_{j=1}^N f^{(j)} B_{i,k}(\xi^{(j)})$ we have

$$D\theta_N = \sum_{j=1}^N f_j^2 DB_{i,k}(\xi^{(j)}) = \sum_{j=1}^N c_1^2 N^{-2} \alpha^2_{i,k} c_2^2 N^{-2}$$
$$= (c_1 c_2 \alpha_{i,k})^2 N^{-3}.$$

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

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

where $h > 0$.

Let us choose h such that $h = 1/\varepsilon (D\theta_N)^{1/2}$, where ε is a positive number. Then

$$\Pr\left\{|\theta_N - J_{i,k}| < \frac{1}{\varepsilon} c_1 c_2 \alpha_{i,k} N^{-3/2}\right\} \geq 1 - \varepsilon^2.$$

For $\varepsilon = 1/\sqrt{2}$ one can obtain:

$$\Pr\{|\theta_N - J_{i,k}| < \sqrt{2} c_1 c_2 \alpha_{i,k} N^{-3/2}\} \geq 1/2.$$

The last inequality proves the theorem. □

This result defines a superconvergent Monte Carlo algorithm, because in this case the rate of convergence is 3/2, i.e., $\psi = 1$.

Case 2. Consider the case when B-splines $B_{i,k}$ are splines of degree $k \geq 3$ with defects 1. In this case $B_{i,k}(x)$ have continuous and bounded second derivatives (since $B_{i,k}(x)$ are defined on a finite interval). In fact,

$r = k - l_i \geq 2$. Since the second derivative of $B_{i,k}(x)$ is bounded there exist constants $\mu_{i,k}$ such that

$$\left| \frac{d^2 B_{i,k}(x)}{dx^2} \right| \leq \mu_{i,k}, \quad x \in [a,b].$$

Consider a sequence of random points $\{\xi_j\}_{j=1}^N$, $(\xi_1 < \xi_2 < \cdots < \xi_N)$.

Define a set of points $\{y_j\}_{j=0}^N$ such that $y_j = 1/2(\xi_j + \xi_{j+1})$, $j = 1, 2, \ldots, N-1$ with $y_0 = a$ and $y_L = b$.

Now consider the following Monte Carlo algorithm for evaluating integrals (7.5) (it is assumed that $\xi_0 = a$ and $\xi_{n+1} = b$):

$$\begin{aligned} \hat{\theta}_N = &- \frac{1}{2} \left[B_{i,k}(\xi_0) + B_{i,k}(\xi_1) \right] \xi_0 \\ &+ \frac{1}{2} \sum_{j=1}^{N} \left[B_{i,k}(\xi_{j-1}) - B_{i,k}(\xi_{j+1}) \right] \xi_j \\ &+ \frac{1}{2} \left[B_{i,k}(\xi_N) + B_{i,k}(\xi_{N+1}) \right] \xi_{N+1}. \end{aligned} \tag{7.8}$$

Obviously, there exist constants c_1 and c_2, such that

$$f^j = \int_{y_{j-1}}^{y_j} f(x)\, dx \leq c_1/N \quad \text{and} \quad \Delta x_j = y_j - y_{j-1} \leq c_2/N.$$

In this case the following theorem holds:

Theorem 7.2. *Let $B_{i,k}(x)$ be a B-spline of degree $k \geq 3$ with defects $l_i = 1$. Then the probable error r_N for the Monte Carlo algorithm (7.8) for evaluating the integrals (7.5) is*

$$r_N \leq \frac{\sqrt{2}}{4} c_1 c_2^2 \mu_{i,k} N^{-5/2}.$$

Proof. For B-splines $B_{i,k}(x)$, where $k \geq 3$ and for a fixed point $s^{(j)} \in \Delta x_j = [y_{j-1}, y_j]$ we have

$$B_{i,k}(x) = B_{i,k}(s^{(j)}) + B'_{i,k}(s^{(j)})(x - s^{(j)}) + \frac{1}{2} B''_{i,k}(\eta^{(j)})(x - s^{(j)})^2,$$

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

Since $B''_{i,k}(\eta^{(j)}) \leq \mu_{i,k}$ it is easy to get

$$D\theta_N = \left(\frac{1}{2} c_1 c_2^2 \mu_{i,k} \right)^2 N^{-5}.$$

Using the Tchebychev's inequality: $\Pr\{|\theta_N - J_{i,k}| < h\} \geq 1 - (D\theta_N / h^2)$, for $h = 1/\varepsilon (D\theta_N)^{1/2}$, we get $r_N \leq \frac{\sqrt{2}}{4} c_1 c_2^2 \mu_{i,k} N^{-5/2}$.

□

In this case we have a superconvergent Monte Carlo algorithm, because the convergence is $N^{-5/2}$, i.e., $\psi = 2$.

7.3 Error Balancing

Here we consider the problem of balancing both systematic and stochastic errors. This problem is closely connected with the problem of obtaining an optimal ratio between the number of realizations of the random variable and the mesh–size on which B-splines are defined.

The systematic error is

$$r_s \leq Ch^k,$$

where h is the mesh–size of w_m (see formula (7.1)) and C is a constant.

For the first Monte Carlo algorithm (case 1) we have:

$$E\left(\|\varepsilon\|^2\right) = C_1^2 N^{-3},$$

where C_1 is a constant.

Thus, $r_N^2(L) \leq \|A^{-1}\| C_1^2 N^{-3}$.

Since for the balancing of errors it is necessary to have

$$C^2 h^{2k} = C_1^2 \|A^{-1}\| N^{-3}$$

one can get the optimal relation between the number of realizations N and the mesh–size h:

$$N = C^{(1)} h^{-\frac{2}{3}k},$$

where

$$C^{(1)} = \left[\left(\frac{C_1}{C}\right)^2 \|A^{-1}\|\right]^{\frac{1}{3}}.$$

It is easy to find the corresponding ratio for the second algorithm (case 2):

$$N = C^{(2)} h^{-\frac{2}{5}k}.$$

Let us note that these results are better than the result obtained in [Mahotkin and Pirimkulov (1984)] for the crude Monte Carlo algorithm, because for the crude algorithm $E\left(\|\varepsilon\|^2\right) = C_0 N^{-1}$ and

$$N = C^{(0)} h^{-2k}.$$

7.4 Concluding Remarks

In this chapter we have shown that when B-splines, with defects 1 of degree $k \geq 2$, are used in density simulations the algorithms are superconvergent. For $k = 2$ the convergence is $N^{-3/2}$ while for $k \geq 3$ the convergence is $N^{-5/2}$.

The idea of balancing of both systematic and stochastic errors is used to obtain an optimal ratio between the number of realizations N and the size of the mesh h on which B-splines are defined.

Chapter 8

Solving Non-Linear Equations

In this chapter algorithms for solving non-linear equations with a polynomial non-linearity are considered. Monte Carlo iterations corresponding to *branching stochastic process* are used to calculate linear functionals of the solution of non-linear integral equations of Fredholm type. In probability theory, a *branching stochastic process* is a Markov process that models a population in which each individual in generation n produces some random number of individuals in generation $n + 1$, according to a fixed probability distribution that does not vary from individual to individual [Harris (1963); Grimmett and Stirzaker (1992)]. The *branching stochastic process* have very important applications in biology (see, [Kimmel and Axelrod (2002)]), as well as in other sciences. At the same time they are powerful tool for solving non-linear integral equations.

8.1 Formulation of the Problems

Consider the problem of evaluation the inner product:

$$J(u) \equiv (g, u) = \int_{\Omega} g(x)u(x)dx, \tag{8.1}$$

on a domain $\Omega \subset I\!R^d$, $x \equiv (x_{(1)}, x_{(2)}, \ldots, x_{(d)}) \in \Omega$ is a point in the Euclidean space $I\!R^d$, $g(x) \in \mathbf{X}$ (where $\mathbf{X}$ is any Banach space), and $u(x) \in \mathbf{X}^*$ (X^* is the dual Banach space adjoint to the Banach space $\mathbf{X}^*$) is a unique solution of the following Fredholm integral equation in an operator form:

$$u = K(u) + f. \tag{8.2}$$

The problem under consideration can be formulated as follows: Given an integral operator $K \in (\mathbf{X}^* \times \mathbf{X}^*)$, a function $f(x) \in \mathbf{X}^*$ and a function

$g(x) \in \mathbf{X}$, compute the functional $J(u)$ (8.1). Of special interest is the case $g = \delta_{x_0}$. In such a case we seek the value of u at the point x_0, ($x_0 \in \Omega$ is fixed).

Consider the space $\mathbf{X}^* = \mathbf{L}_1(\Omega)$. Then, obviously, $\mathbf{X} = \mathbf{L}_\infty(\Omega)$. In the problem under consideration K is an integral transform with polynomial non-linearity

$$K\left(u^{(m)}\right) = \int_\Omega \dots \int_\Omega k(x, y_1, y_2, \dots, y_m) \prod_{i=1}^{m} u(y_i) \prod_{i=1}^{m} dy_i. \tag{8.3}$$

and the equation (8.2) becomes:

$$u(x) = \int_\Omega \dots \int_\Omega k(x, y_1, y_2, \dots, y_m) \prod_{i=1}^{m} u(y_i) \prod_{i=1}^{m} dy_i + f(x), \ \ m \geq 2. \tag{8.4}$$

In this case the problem under consideration can be formulated in the following way:

Knowing $k(x, y_1, y_2, \dots, y_m) \in \mathbf{L}_1 \underbrace{(\Omega \times \dots \times \Omega)}_{m+1}$ and $f(x) \in \mathbf{L}_1(\Omega)$, compute the functional (8.1), where the function $u(x)$ is a solution of an integral equation with polynomial non-linearity.

Suppose, the following iteration process converges:

$$u_{l+1}(x) = \int_\Omega \dots \int_\Omega k(x, y_1, y_2, \dots, y_m) \prod_{i=1}^{m} u_l(y_i) \prod_{i=1}^{m} dy_i + f(x), \ \ l = 1, 2, \dots \tag{8.5}$$

$$u_0(x) = f(x),$$

or

$$u_{l+1} = K u_l^{(m)} + f, \ \ u_l^{(m)} = \prod_{i=1}^{m} u_l(y_i), \ \ u_0 = f.$$

The process (8.5) converges when, for example, the following condition holds:

$$\left\| K\left(u^{(m)}\right) \right\| = \max_{x \in \Omega} \int_\Omega \dots \int_\Omega \mid k(x, y_1, y_2, \dots, y_m) \mid \prod_{i=1}^{m} dy_i < \frac{1}{m}. \tag{8.6}$$

In the next section (Section 8.2) a r.v. with mathematical expectation equal to the functional (8.1) is considered. Branching stochastic processes corresponding to the above iterative non-stationary process are used.

8.2 A Monte Carlo Method for Solving Non-linear Integral Equations of Fredholm Type

In this section we describe a Monte Carlo method, which is applied to the integral equation (8.3).

Consider the following branching stochastic process: Suppose that any particle distributed according to an initial density function $p_0(x) \geq 0$, ($\int p_0(x)dx = 1$) is born in the domain $\Omega \subset I\!R^d$ in the random point x_0. In the next time-moment this particle either dies out with a probability $h(x)$, $(0 \leq h(x) < 1)$ or generates a posterity of m analogical particles in the next random points $x_{00}, x_{01}, \ldots, x_{0m-1}$ with a probability $p_m(x) = 1 - h(x)$ and a transition density function

$$p(x_0, x_{00}, \ldots, x_{0m-1}) \geq 0,$$

where

$$\int \ldots \int p(x_0, x_{00}, x_{01}, \ldots, x_{0m-1}) \prod_{i=0}^{m-1} dx_{0i} = 1$$

for any x_0.

The generated particles behave in the next moment as the initial and the process continues. The traces of such a process is a tree of the type sketched in Figure 8.1.

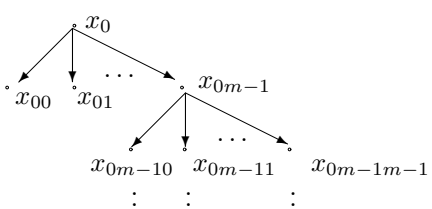


Fig. 8.1 A branching stochastic process used for non-linear integral equations (to the definition of multi-indexes)

The used index numbers in Figure 8.1 are called multi-indices. The particle from the zero generation is enumerated with zero index, i.e. x_0. Its direct inheritors are enumerated with indices $00, 01, \ldots, 0m-1$, i.e. next points $x_{00}, x_{01}, \ldots, x_{0m-1}$ are from the first generation. If a particle from the q^{th} generation of the tree γ has the multi-index $\gamma[q]$, then the multi-index of the 1^{th} inheritor of this particle has the following form:

$$\gamma[q+1] = \gamma[q]j,$$

where the multi-index is a number written in m^{th} numerical system.

Consider a simple case $m = 2$ and the first two iterations of the iterative process (8.5), ([Ermakov and Mikhailov (1982); Dimov (1991)]).

$$
\begin{aligned}
&u_0(x_0) = f(x_0),\\
&u_1(x_0) = f(x_0) + \int\int k(x_0, x_{00}, x_{01}) f(x_{00}) f(x_{01}) dx_{00} dx_{01},\\
&u_2(x_0) = f(x_0) + \int\int k(x_0, x_{00}, x_{01}) f(x_{00}) f(x_{01}) dx_{00} dx_{01}\\
&+ \int\int k(x_0, x_{00}, x_{01}) f(x_{01})\\
&\times \left(\int\int k(x_{00}, x_{000}, x_{001}) f(x_{000}) f(x_{001}) dx_{000} dx_{001} \right) dx_{00} dx_{01}\\
&+ \int\int k(x_0, x_{00}, x_{01}) f(x_{00})\\
&\times \left(\int\int k(x_{01}, x_{010}, x_{011}) f(x_{010}) f(x_{011}) dx_{010} dx_{011} \right) dx_{00} dx_{01}\\
&+ \int\int k(x_0, x_{00}, x_{01}) \left(\int\int k(x_{01}, x_{010}, x_{011}) f(x_{010}) f(x_{011}) dx_{010} dx_{011} \right)\\
&\times \left(\int\int k(x_{00}, x_{000}, x_{001}) f(x_{000}) f(x_{001}) dx_{000} dx_{001} \right) dx_{00} dx_{01}.
\end{aligned}
\tag{8.7}
$$

The branching stochastic process which corresponds to these two Monte Carlo iterations is presented on Figure 8.2.

Obviously the structure of u_1 is linked with all trees which appear after the first generation, (see, Figure 8.2 a Figure 8.2 b).

The structure of u_2 is linked with all trees which appear after the second generation (Figure 8.2).

This similarity allows to define a procedure of a random choice of subtrees of a full tree and to calculate the values of some r.v. This r.v. corresponds to the random choice of the subtree. Thus the arithmetic mean over N independent samples (N is a "large" number) of such r.v. is an estimate of the functional (8.1).

Definition 8.1. A full tree with l generations is called the *tree* Γ_l where the dying out of particles is not visible from zero to the $\{l-1\}^{st}$ generation but all the generated particles of the l^{th} generation die out.

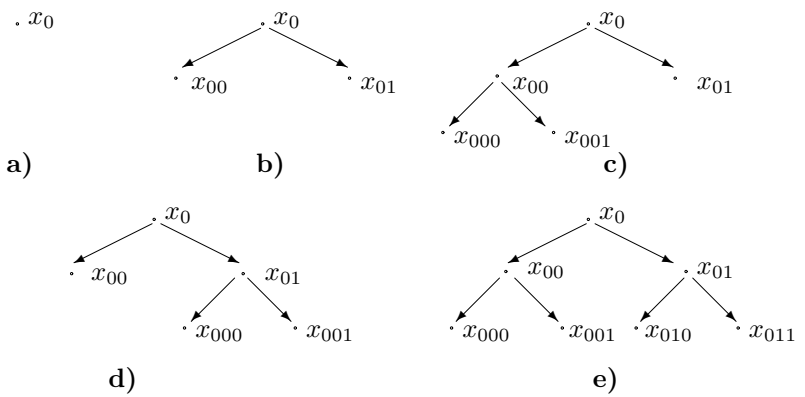


Fig. 8.2 Structure of the branching stochastic process used for non-linear integral equations ($m = 2$). The first two iterations are considered.

Example 8.1. In fact Γ_2 is the tree γ_0 shown on Figure 8.2 *a* and Γ_1 is the tree 8.2 *b*. The next density function corresponds to the tree γ_0 shown on 8.2 *e*:

$$\begin{aligned} p_{\gamma_0} &= p_0(x_0)p_2(x_0)p(x_0, x_{00}, x_{01})p_2(x_{00}) \\ &\times p(x_{00}, x_{000}, x_{001})p_2(x_{01})p(x_{01}, x_{010}, x_{011}) \\ &\times h(x_{000})h(x_{001})h(x_{010})h(x_{011}) \end{aligned} \tag{8.8}$$

Then the r.v. which corresponds to γ_0 is:

$$\begin{aligned} \Theta_{[g]}(\gamma_0) = &\frac{g(x_0)}{p_0(x_0)} \frac{k(x_0, x_{00}, x_{01})}{p_2(x_0)p(x_0, x_{00}, x_{01})} \frac{k(x_{00}, x_{000}, x_{001})}{p_2(x_{00})p(x_{00}, x_{000}, x_{001})} \\ &\times \frac{k(x_{01}, x_{010}, x_{011})}{p_2(x_{01})p(x_{01}, x_{010}, x_{011})} \frac{f(x_{000})f(x_{001})f(x_{010})f(x_{011})}{h(x_{000})h(x_{000})h(x_{000})h(x_{000})}. \end{aligned} \tag{8.9}$$

This r.v. estimates the last of the terms in (8.7) if the condition is that the realization of the random process appears to be a tree of kind γ_0. Thus, r.v. are defined that correspond to trees of another type.

Consider the branching stochastic process in the general case when $m \geq 2$. Denote by A the set of points that generate new points and denote by B the set of points that die out [Dimov (1991); Dimov and Gurov (2000); Gurov (1994)].

The above connection between the branching stochastic process and the iterative process (8.5) allows to define a procedure of a random choice of tree and to calculate the values of some r.v. that corresponds to this tree. When a branching stochastic process is defined we receive arbitrary trees γ and we correspond to each of them the r.v. $\Theta_{[g]}(\gamma/\Gamma_l)$ (sometimes abbreviated to $\Theta_{[g]}(\gamma)$) in the following way:

$$\Theta_{[g]}(\gamma) = \frac{g(x_0)}{p_0(x_0)} \frac{f(x_0)}{h(x_0)}, \tag{8.10}$$

if the tree consists of the initial point only. If the tree contains other points then $\Theta_{[g]}(\gamma)$ is determined simultaneously with the construction of γ. If there is a transition from the random point $x_{\gamma[q]}$ to points $x_{\gamma[q]0}, x_{\gamma[q]1}, \ldots, x_{\gamma[q]m-1}$ then r.v. (8.10) should be multiplied by,

$$\frac{k(x_{\gamma[q]}, x_{\gamma[q]0}, \ldots, x_{\gamma[q]m-1})}{p_m(x_{\gamma[q]}) p(x_{\gamma[q]}, x_{\gamma[q]0}, \ldots, x_{\gamma[q]m-1})},$$

and when the point $x_{\gamma[r]}$ dies out r.v. (8.10) should be multiplied by

$$\frac{f(x_{\gamma[r]})}{h(x_{\gamma[r]})}.$$

Thus the r. v. $\Theta_{[g]}(\gamma)$ which corresponds to γ is equal to the following expression:

$$\Theta_{[g]}(\gamma) = \frac{g(x_0)}{p_0(x_0)} \prod_{x_{\gamma[q]} \in A} \frac{K(x_{\gamma[q]})}{P(x_{\gamma[q]})} \prod_{x_{\gamma[r]} \in B} \frac{f(x_{\gamma[r]})}{h(x_{\gamma[r]})} \tag{8.11}$$

with the density function

$$p_\gamma = p_0(x_0) \prod_{x_{\gamma[q]} \in A} P(x_{\gamma[q]}) \prod_{x_{\gamma[r]} \in B} h(x_{\gamma[r]}), \tag{8.12}$$

$$1 \le q \le l-1, \;\; 1 < r \le l,$$

where

$$K(x_{\gamma[q]}) = k(x_{\gamma[q]}, x_{\gamma[q]0}, \ldots, x_{\gamma[q]m-1})$$

and

$$P(x_{\gamma[q]}) = p_m(x_{\gamma[q]}) p(x_{\gamma[q]}, x_{\gamma[q]0}, \ldots, x_{\gamma[q]m-1}).$$

Thus, we obtain r.v. for arbitrary trees Γ_l which estimate l^{th} iteration, u_l, of the iterative process (8.5).

Theorem 8.1. *The mathematical expectation of the r.v.* $\Theta_{[g]}(\gamma/\Gamma_l)$ *is equal to the functional* $J(u_l)$*, i.e.*

$$E\Theta_{[g]}(\gamma/\Gamma_l) = J(u_l) = (g, u_l).$$

Proof. Suppose that the subtree γ_i is fixed and it has s_1+s_2+1 points, such that $x_0, x_1, \ldots, x_{s_1} \in A$ and $x_{s_1+1}, x_{s_1+2}, \ldots, x_{s_1+s_2} \in B$. It is clear that this can not be considered as a restriction of generality.

Consider the following density function

$$p_{\gamma_i} = p_{\gamma_i}(x_0, x_1, \ldots, x_{s_1}, x_{s_1+1}, \ldots, x_{s_1+s_2}) = p_0(x_0) \prod_{i=0}^{s_1} P(x_i) \prod_{x=1}^{s_2} h(x_{s_1+i})$$

and the probability to obtain the subtree $\gamma = \gamma_i$

$$Pr\{\gamma = \gamma_i\} = \int_G \ldots \int_G p_{\gamma_i} dx_0 \ldots dx_{s_1+s_2}.$$

The r.v.

$$\Theta_{[g]}(\gamma_i) = \frac{g(x_0)}{p_0(x_0)} \prod_{j=0}^{s_1} \frac{K(x_j)}{P(x_j)} \prod_{j=1}^{s_2} \frac{f(x_{s_1+j})}{h(x_{s_1+j})}$$

has the following condition density function

$$p_\gamma(x_0, \ldots, x_{s_1+s_2} | \gamma = \gamma_i) = \frac{p_{\gamma_i}(x_0, x_1, \ldots, x_{s_1}, x_{s_1+1}, \ldots, x_{s_1+s_2})}{Pr\{\gamma = \gamma_i\}},$$

where

$$K(x_j) = k(x_j, x_{j0}, \ldots, x_{jm-1}) \quad \text{and} \quad P(x_j) = p_m(x_j, x_{j0}, \ldots, x_{jm-1}).$$

Let us calculate the mathematical expectation:

$$\begin{aligned} & E\Theta_{[g]}(\gamma_i) = E\{\Theta_{[g]}(\gamma/\Gamma_l) | \gamma = \gamma_i\} \\ &= \int_G \ldots \int_G \Theta_{[g]}(\gamma_i) p_\gamma(x_0, \ldots, x_{s_1+s_2} | \gamma = \gamma_i) dx_0 \ldots dx_{s_1+s_2} \\ &= \int_G \ldots \int_G g(x_0) \prod_{j=0}^{s_1} K(x_j) \prod_{j=1}^{s_2} f(x_{s_1+j}) \frac{dx_0 \ldots dx_{s_1+s_2}}{Pr\{\gamma = \gamma_i\}} \\ &= \frac{(g, u_l^i)}{Pr\{\gamma = \gamma_i\}}, \end{aligned} \tag{8.13}$$

where

$$u_l^i = u_l^i(x_0) = \int_G \ldots \int_G \prod_{j=0}^{s_1} K(x_j) \prod_{j=1}^{s_2} f(x_{s_1+j}) dx_1 \ldots dx_{s_1+s_2}.$$

One can choose N subtrees of the full tree Γ_l. (Note that some of the subtrees may be chosen many times, since N is a large number.) The value of n_{γ_i} is equal to the number of trees that correspond to γ_i. The following expression holds:

$$\frac{n_{\gamma_i}}{N} \approx Pr\{\gamma = \gamma_i\}.$$

On the other hand

$$J(u_l^i) = (g, u_l^i) \approx \frac{Pr\{\gamma = \gamma_i\}}{n_{\gamma_i}} \sum_{j=1}^{n_{\gamma_i}} (\Theta_{[g]}(\gamma_i))_j \approx \frac{1}{N} \sum_{j=1}^{n_{\gamma_i}} (\Theta_{[g]}(\gamma_i))_j. \quad (8.14)$$

Clearly

$$E\Theta_{[g]}(\gamma/\Gamma_l) = \sum_{i=0}^{\tilde{N}} E\{\Theta_{[g]}(\gamma/\Gamma_l)|\gamma = \gamma_i\} Pr\{\gamma = \gamma_i\},$$

where $\tilde{N}$ is the number of all subtrees of Γ_l.

Using (8.13) one can get

$$E\Theta_{[g]}(\gamma/\Gamma_l) = \sum_{i=0}^{\tilde{N}} (g, u_l^i) = (g, u_l) = J(u_l).$$

This completes the proof of the theorem.

□

Note, that if $l \to \infty$, then the mathematical expectation of the r.v. is:

$$\lim_{l\to\infty} E\Theta_{[g]}(\gamma/\Gamma_l) = E\Theta_{[g]}(\gamma/\Gamma_l) = J(u) = \int_G g(x)u(x)dx, \quad (8.15)$$

where $u(x)$ is the solution of (8.5).

It is supposed that all trees have a finite number of generations and the averaged value of the particles which are born in any generation is also finite, i.e. the next inequality for the probability $p_m(x)$ holds:

$$p_m(x) < \frac{1}{m}, \quad \text{where } x \in \Omega.$$

Consider a set of independent values

$$\Theta_{[g]}(\gamma/\Gamma_l)_1, \Theta_{[g]}(\gamma/\Gamma_l)_2, \ldots, \Theta_{[g]}(\gamma/\Gamma_l)_N.$$

From (8.14) we obtain the following Monte Carlo method:

$$J(u_l) = \sum_{i=0}^{\tilde{N}} (g, u_l^i) \approx \frac{1}{N} \sum_{i=0}^{\tilde{N}} \sum_{j=1}^{n_{\gamma_i}} (\Theta_{[g]}(\gamma_i))_j = \frac{1}{N} \sum_{j=1}^{N} (\Theta_{[g]}(\gamma/\Gamma_l))_j. \quad (8.16)$$

Suppose the initial density function and the transition density function are *permissible* to $g(x)$ and $k(x_{\gamma[q]}, x_{\gamma[q]0}, \ldots, x_{\gamma[q]m-1})$, respectively (see Definition 4.1 of *permissible* density functions in Section 4.2).

In this case the probable error is

$$r_N \approx 0.6745\sigma(\Theta_{[g]}(\gamma/\Gamma_l))N^{-1/2},$$

where $\sigma(\Theta_{[g]}(\gamma/\Gamma_l))$ is the standard deviation of the r.v. $\Theta_{[g]}(\gamma/\Gamma_l)$.

The problem of optimization of Monte Carlo algorithms consists in minimization of the standard deviation, i.e. minimization of the second moment $E\Theta^2_{[g]}(\gamma/\Gamma_l)$ of the r.v. $\Theta_{[g]}(\gamma/\Gamma_l)$. This is done by a suitable choice of the density function p_γ. In the next subsection (Subsection 8.3) such a minimization is considered.

8.3 An Efficient Algorithm

To formulate an efficient algorithm one has to optimize the method described in Section 8.2. The problem of optimization of the Monte Carlo algorithms consists in minimization of the standard deviation, i.e. minimization of the second moment $E\Theta^2_{[g]}(\gamma)$ of the r.v. $\Theta_{[g]}(\gamma/\Gamma_l)$. This can be done by a suitable choice of the density function p_γ.

There are various techniques for variance reduction of the standard deviation. We consider one of them.

Let us introduce the functions

$$\Phi(x) = \left(\int \dots \int \frac{k^2(x, y_1, y_2, \dots, y_m)}{p(x, y_1, y_2, \dots, y_m)} \prod_{i=1}^{m} \Phi^2(y_i) \prod_{i=1}^{m} dy_i \right)^{\frac{1}{2}}; \tag{8.17}$$

$$\hat{\Phi}(x) = \int \dots \int \left| k(x, y_1, y_2, \dots, y_m) \prod_{i=1}^{m} \Phi(y_i) \right| \prod_{i=1}^{m} dy_i. \tag{8.18}$$

Suppose that the r.v. has finite variance. The following statements hold:

Lemma 8.1. *The transition density function*

$$p(x, y_1, y_2, \dots, y_m) = \frac{|k(x, y_1, y_2, \dots, y_m) \prod_{i=1}^{m} \Phi(y_i)|}{\int \dots \int |k(x, y_1, y_2, \dots, y_m) \prod_{i=1}^{m} \Phi(y_i)| \prod_{i=1}^{m} dy_i}$$

minimizes $\Phi(x)$ *for any* $x \in \Omega$ *and*

$$\min_p \Phi(x) = \hat{\Phi}(x).$$

Proof. Substitute $p(x, y_1, y_2, \dots, y_m)$ in the expression for $\Phi(x)$:

$$\Phi(x) = \left[\int \dots \int \frac{k^2(x, y_1, y_2, \dots, y_m) \prod_{i=1}^{m} \Phi^2(y_i)}{|k(x, y_1, y_2, \dots, y_m) \prod_{i=1}^{m} \Phi(y_i)|} \times \right.$$

$$\left. \times \left(\int \dots \int |k(x, y_1, y_2, \dots, y_m) \prod_{i=1}^{m} \Phi(y_i)| \prod_{i=1}^{m} dy_i \right) \prod_{i=1}^{m} dy_i \right]^{\frac{1}{2}}$$

$$= \int \dots \int |k(x, y_1, y_2, \dots, y_m) \prod_{i=1}^{m} \Phi(y_i)| \prod_{i=1}^{m} dy_i = \hat{\Phi}(x).$$

Now we must show that for any other transition density function permissible to the kernel $k(x, y_1, y_2, \dots, y_m)$ it holds that:

$$\hat{\Phi}(x) \leq \Phi(x).$$

One can multiply and divide the integrand by $p^{\frac{1}{2}}(x, y_1, y_2, \ldots, y_m) > 0$ and apply the Cauchy-Schwartz inequality. Then we have

$$\begin{aligned}
\hat{\Phi}^2(x) &= \left(\int \ldots \int |k(x, y_1, y_2, \ldots, y_m) \right. \\
&\left. \times \prod_{i=1}^{m} \Phi(y_i) | p^{-\frac{1}{2}}(x, y_1, \ldots, y_m) p^{\frac{1}{2}}(x, y_1, \ldots, y_m) \prod_{i=1}^{m} dy_i \right)^2 \\
&\leq \int \ldots \int k^2(x, y_1, y_2, \ldots, y_m) \prod_{i=1}^{m} \Phi^2(y_i) p^{-1}(x, y_1, y_2, \ldots, y_m) \prod_{i=1}^{m} dy_i \\
&\times \int \ldots \int p(x, y_1, y_2, \ldots, y_m) \prod_{i=1}^{m} dy_i \\
&\leq \int \ldots \int k^2(x, y_1, y_2, \ldots, y_m) \prod_{i=1}^{m} \Phi^2(y_i) p^{-1}(x, y_1, y_2, \ldots, y_m) \prod_{i=1}^{m} dy_i \\
&= \Phi^2(x), \qquad (8.19)
\end{aligned}$$

because

$$\int \ldots \int p(x, y_1, y_2, \ldots, y_m) \prod_{i=1}^{m} dy_i = 1 - h(x) = p_m(x) < 1$$

for any $x \in \Omega$. □

Lemma 8.2. *The initial density function*

$$p_0(x_0) = |g(x_0)\Phi(x_0)| \Big/ \left(\int |g(x_0)\Phi(x_0)| dx_0 \right)$$

minimizes the functional

$$\int g^2(x_0)\Phi^2(x_0) p_0^{-1}(x_0) dx_0.$$

The minimum of this functional is equal to

$$\left(\int |g(x_0)\Phi(x_0)| dx_0 \right)^2.$$

Proof. The proof is similar to the proof of Lemma 8.1:

$$\int g^2(x_0)\Phi^2(x_0)p_0^{-1}(x_0)dx_0$$
$$= \int g^2(x_0)\Phi^2(x_0)|g(x_0)\Phi(x_0)|^{-1}\left(\int |g(x_0)\Phi(x_0)|dx_0\right)dx_0$$
$$= \left(\int |g(x_0)\Phi(x_0)|dx_0\right)^2.$$

It remains to establish that for any another density function the following inequality holds

$$\left(\int |g(x_0)\Phi(x_0)|dx_0\right)^2 \le \int g^2(x_0)\Phi^2(x_0)p_0^{-1}(x_0)dx_0.$$

In fact,

$$\left(\int |g(x_0)\Phi(x_0)|dx_0\right)^2 = \left(\int |g(x_0)\Phi(x_0)|p_0^{-\frac{1}{2}}(x_0)p_0^{\frac{1}{2}}(x_0)dx_0\right)^2$$
$$\le \int g^2(x_0)\Phi^2(x_0)p_0^{-1}(x_0)dx_0 \int p_0(x_0)dx_0 = \int g^2(x_0)\Phi^2(x_0)p_0^{-1}(x_0)dx_0,$$

because $\int p_0(x_0)dx_0 = 1$ for any $x \in \Omega$. □

Consider the r.v. $\Theta_{[g]}(\gamma_0)$ which estimates the last of the terms in (8.11) in private case when $m = 2$. The following theorem can be proven.

Theorem 8.2. *Introduce a constant*

$$c_0 = \left(\int |g(x_0)\hat{\Phi}(x_0)|dx_0\right)^{-1},$$

where $\hat{\Phi}(x_0)$ *is a function* (8.18) *for* $m = 2$ *and the following function*

$$\Phi(x) = \frac{|f(x)|}{(h(x))^{\frac{1}{2}}}.$$

Then the density function

$$\hat{p}_{\gamma_0} = c_0|g(x_0)|\ |k(x_0, x_{00}, x_{01})|\ |k(x_{00}, x_{000}, x_{001})|$$
$$\times\ |k(x_{01}, x_{010}, x_{011})| \tag{8.20}$$
$$\times\ \Phi(x_{000})\Phi(x_{001})\Phi(x_{010})\Phi(x_{011})h(x_{000})h(x_{001})h(x_{010})h(x_{011})$$

minimizes the second moment $E\Theta^2_{[g]}(\gamma_0)$ *of the r.v.* $\Theta_{[g]}(\gamma_0)$ *which corresponds to the tree* γ_0*, i.e.*

$$\min_{p_{\gamma_0}}\left(E\Theta^2_{[g]}(\gamma_0)\right) = E\hat{\Theta}^2_{[g]}(\gamma_0),$$

where $p_{\gamma_0} = \hat{p}_{\gamma_0}$.

Proof. Estimate the second moment $E\Theta^2_{[g]}(\gamma_0)$ of the r.v. $\Theta_{[g]}(\gamma_0)$ with the density function

$$p_{\gamma_0} = p_0(x_0)p(x_0, x_{00}, x_{01})p(x_{00}, x_{000}, x_{001})p(x_{01}, x_{010}, x_{011}) \\ h(x_{000})h(x_{001})h(x_{010})h(x_{011}).$$

We have

$$E\Theta^2_{[g]}(\gamma_0) = \int \dots \int \Theta^2_{[g]}(\gamma_0)p_{\gamma_0}dx_0dx_{00}dx_{01}\dots dx_{011} \tag{8.21}$$

$$= \int \dots \int \frac{g^2(x_0)k^2(x_0, x_{00}, x_{01})k^2(x_{00}, x_{000}, x_{001})k^2(x_{01}, x_{010}, x_{011})}{p_0^2(x_0)p^2(x_0, x_{00}, x_{01})p^2(x_{00}, x_{000}, x_{001})p^2(x_{01}, x_{010}, x_{011})}$$
$$\times \frac{f^2(x_{000})f^2(x_{001})f^2(x_{010})f^2(x_{011})}{h^2(x_{000})h^2(x_{001})h^2(x_{010})h^2(x_{011})}p_0(x_0)p(x_0, x_{00}, x_{01})$$
$$\times p(x_{00}, x_{000}, x_{001})p(x_{01}, x_{010}, x_{011})f(x_{000})f(x_{001})f(x_{010})f(x_{011})$$
$$\times h(x_{000})h(x_{001})h(x_{010})h(x_{011})dx_0dx_{00}\dots dx_{011}$$

$$= \int \dots \int \frac{g^2(x_0)k^2(x_0, x_{00}, x_{01})k^2(x_{00}, x_{000}, x_{001})k^2(x_{01}, x_{010}, x_{011})}{p_0(x_0)p(x_0, x_{00}, x_{01})p(x_{00}, x_{000}, x_{001})p(x_{01}, x_{010}, x_{011})}$$
$$\times \Phi^2(x_{000})\Phi^2(x_{001})\Phi^2(x_{010})\Phi^2(x_{011})dx_0dx_{00}\dots dx_{011}$$

$$= \int \frac{g^2(x_0)}{p_0(x_0)} \left\{ \int\int \frac{k^2(x_0, x_{00}, x_{01})}{p(x_0, x_{00}, x_{01})} \times \right.$$
$$\times \left[\int\int \frac{k^2(x_{00}, x_{000}, x_{001})}{p(x_{00}, x_{000}, x_{001})}\Phi^2(x_{000})\Phi^2(x_{001})dx_{000}dx_{001} \right]$$
$$\left. \times \left[\int\int \frac{k^2(x_{01}, x_{010}, x_{011})}{p(x_{01}, x_{010}, x_{011})}\Phi^2(x_{010})\Phi^2(x_{011})dx_{010}dx_{011} \right] dx_{00}dx_{01} \right\} dx_0.$$

Let $\hat{\Theta}_{[g]}(\gamma_0)$ be the r.v. $\Theta_{[g]}(\gamma_0)$ for which $p_{\gamma_0} = \hat{p}_{\gamma_0}$. Then

$$E\hat{\Theta}^2_{[g]}(\gamma_0) = \int \dots \int \hat{\Theta}^2_{[g]}(\gamma_0)\hat{p}_{\gamma_0} dx_0 dx_{00} dx_{01} \dots dx_{011}$$

$$= \int \dots \int \left[\frac{g(x_0)k(x_0, x_{00}, x_{01})k(x_{00}, x_{000}, x_{001})k(x_{01}, x_{010}, x_{011})}{c_0 \, |g(x_0)| \, |k(x_0, x_{00}, x_{01})| \, |k(x_{00}, x_{000}, x_{001})| \, |k(x_{01}, x_{010}, x_{011})|} \right.$$

$$\times \left. \frac{f(x_{000}) \quad f(x_{001}) \quad f(x_{010}) \quad f(x_{011})}{\Phi(x_{000})\Phi(x_{001})\Phi(x_{010})\Phi(x_{011})h(x_{000})h(x_{001})h(x_{010})h(x_{011})} \right]^2$$

$$\times \; c_0 \, |g(x_0)| \, |k(x_0, x_{00}, x_{01})| \, |k(x_{00}, x_{000}, x_{001})| \, |k(x_{01}, x_{010}, x_{011})|$$

$$\times \; \Phi(x_{000})\Phi(x_{001})\Phi(x_{010})\Phi(x_{011})h(x_{000})h(x_{001})h(x_{010})h(x_{011}) dx_0 \dots dx_{011}$$

$$= c_0^{-1} \int \dots \int |g(x_0)| \, |k(x_0, x_{00}, x_{01})| \, |k(x_{00}, x_{000}, x_{001})| \, |k(x_{01}, x_{010}, x_{011})|$$

$$\times \; \Phi(x_{000})\Phi(x_{001})\Phi(x_{010})\Phi(x_{011}) dx_0 dx_{00} dx_{01} dx_{000} dx_{001} dx_{010} dx_{011}$$

$$= c_0^{-1} \int |g(x_0)|$$

$$\times \left\{ \iint |k(x_0, x_{00}, x_{01})| \left[\iint |k(x_{00}, x_{000}, x_{001})\Phi(x_{000})\Phi(x_{001})| dx_{000} dx_{001} \right] \right.$$

$$\times \left. \left[\iint |k(x_{01}, x_{010}, x_{011})\Phi(x_{010})\Phi(x_{011})| dx_{010} dx_{011} \right] dx_{00} dx_{01} \right\} dx_0. \tag{8.22}$$

The functions $\Phi(x)$ and $\hat{\Phi}(x)$ are non-negative for any $x \in \Omega$ and the density function $\hat{p}_{\gamma_0}$ may be written in the form:

$$
\begin{aligned}
&\hat{p}_{\gamma_0} = c_0|g(x_0)|\ |k(x_0,x_{00},x_{01})|\,|k(x_{00},x_{000},x_{001})|\,|k(x_{01},x_{010},x_{011})|\\
&\times \Phi(x_{000})\Phi(x_{001})\Phi(x_{010})\Phi(x_{011})h(x_{000})h(x_{001})h(x_{010})h(x_{011})\\
&= \frac{|g(x_0)\hat{\Phi}(x_0)|}{c_0^{-1}}\,\frac{|k(x_0,x_{00},x_{01})\hat{\Phi}(x_{00})\hat{\Phi}(x_{01})|}{\hat{\Phi}(x_0)}\\
&\times \frac{|k(x_{00},x_{000},x_{001})\Phi(x_{000})\Phi(x_{001})|}{\hat{\Phi}(x_{00})}\,\frac{|k(x_{01},x_{010},x_{011})\Phi(x_{010})\Phi(x_{011})|}{\hat{\Phi}(x_{01})}\\
&\times h(x_{000})h(x_{001})h(x_{010})h(x_{011})
\end{aligned}
$$

$$
\begin{aligned}
&= \frac{|g(x_0)\hat{\Phi}(x_0)|}{c_0^{-1}}\,\frac{|k(x_0,x_{00},x_{01})\hat{\Phi}(x_{00})\hat{\Phi}(x_{01})|}{\int\int |k(x_0,x_{00},x_{01})\hat{\Phi}(x_{00})\hat{\Phi}(x_{01})|dx_{00}dx_{01}}\\
&\times \frac{|k(x_{00},x_{000},x_{001})\Phi(x_{000})\Phi(x_{001})|}{\int\int |k(x_{00},x_{000},x_{001})\Phi(x_{000})\Phi(x_{001})|dx_{000}dx_{001}}\\
&\times \frac{|k(x_{01},x_{010},x_{011})\Phi(x_{010})\Phi(x_{011})|}{\int\int |k(x_{01},x_{010},x_{011})\Phi(x_{010})\Phi(x_{011})|dx_{010}dx_{011}}\\
&\times h(x_{000})h(x_{001})h(x_{010})h(x_{011})
\end{aligned}
$$

$$
\begin{aligned}
&= \hat{p}_0(x_0)\hat{p}(x_0,x_{00},x_{01})\hat{p}(x_{00},x_{000},x_{001})\hat{p}(x_{01},x_{010},x_{011})\\
&\quad \times h(x_{000})h(x_{001})h(x_{010})h(x_{011}).
\end{aligned}
$$

Now Lemma 8.1 (when $m = 2$) should be applied to the transition density functions $\hat{p}(x_0,x_{00},x_{01})$, $\hat{p}(x_{00},x_{000},x_{001})$ and $\hat{p}(x_{01},x_{010},x_{011})$:

$$
\begin{aligned}
&\int\int \frac{k^2(x,y_1,y_2)\Phi^2(y_1)\Phi^2(y_2)}{p(x,y_1,y_2)}dy_1dy_2\\
&= \int\int \frac{k^2(x,y_1,y_2)\Phi^2(y_1)\Phi^2(y_2)}{\hat{p}(x,y_1,y_2)}dy_1dy_2\\
&= \int\int \frac{k^2(x,y_1,y_2)\Phi^2(y_1)\Phi^2(y_2)}{|k(x,y_1,y_2)\Phi(y_1)\Phi(y_2)|}\left[\int\int |k(x,y_1,y_2)\Phi(y_1)\Phi(y_2)|dy_1dy_2\right]dy_1dy_2\\
&= \int\int |k(x,y_1,y_2)\Phi(y_1)\Phi(y_2)|dy_1dy_2 \int\int |k(x,y_1,y_2)\Phi(y_1)\Phi(y_2)|dy_1dy_2.
\end{aligned}
$$

It should be taken into account that for

$$
p(x,y_1,y_2) = \hat{p}(x,y_1,y_2) = \frac{|k(x,y_1,y_2)\Phi(y_1)\Phi(y_2)|}{\int\int |k(x,y_1,y_2)\Phi(y_1)\Phi(y_2)|dy_1dy_2}
$$

the corresponding functions $\Phi(x)$ and $\hat{\Phi}(x)$ are equal to each generation point x in the branching process, i.e.

$$
\Phi(x) = \hat{\Phi}(x),
$$

because the corresponding density functions minimize $\Phi(x)$ and their minima are equal to the functions $\hat{\Phi}(x)$ for any $x \in \Omega$.

So, for any density function $p_0(x_0)$ which is *permissible* to the function $g(x_0)$

$$\min_{\Phi}\left(\int g^2(x_0)\Phi^2(x_0)p_0^{-1}(x_0)dx_0\right) = \int g^2(x_0)\hat{\Phi}^2(x_0)p_0^{-1}(x_0)dx_0$$

holds and in this case

$$\hat{p}_{\gamma_0} = p_0(x_0)\ |k(x_0, x_{00}, x_{01})|\ |k(x_{00}, x_{000}, x_{001})|\ |k(x_{01}, x_{010}, x_{011})|$$
$$\times\ \Phi(x_{000})\Phi(x_{001})\Phi(x_{010})\Phi(x_{011})h(x_{000})h(x_{001})h(x_{010})h(x_{011}). \quad (8.23)$$

According to Lemma 8.2, the last functional is minimized by the density function (8.4) if

$$p_0(x_0) = \hat{p}_0(x_0) = \frac{|g(x_0)\hat{\Phi}(x_0)|}{\int |g(x_0)\hat{\Phi}(x_0)|dx_0},$$

since

$$\Phi(x_0) = \hat{\Phi}(x_0)$$

under the expression (8.4). This completes the proof. □

Consider the branching process in the general case ($m > 2$). The following theorem holds.

Theorem 8.3. *The density function*

$$\hat{p}_\gamma = c|g(x_0)| \prod_{x_{\gamma[q]} \in \mathbf{A}} |K(x_{\gamma[q]})| \prod_{x_{\gamma[r]} \in \mathbf{B}} \Phi(x_{\gamma[r]})h(x_{\gamma[r]}) \quad (8.24)$$

minimizes the second moment $E\Theta^2_{[g]}(\gamma)$ *of the r.v.* $\Theta_{[g]}(\gamma/\Gamma_l)$ *for any tree* $\gamma \in \Gamma_n$, *i.e.*

$$\min_{p_\gamma} E\Theta^2_{[g]}(\gamma) = E\hat{\Theta}^2_{[g]}(\gamma),$$

where $p_\gamma = \hat{p}_\gamma$ *and* $c = \left(\int |g(x_0)\hat{\Phi}(x_0)|dx_0\right)^{-1}$.

Proof. Introduce the following function

$$F(x_{\gamma[q]}) = \begin{cases} \hat{\Phi}(x_{\gamma[q]}), & \text{if } x_{\gamma[q]} \in \mathbf{A} \\ \Phi(x_{\gamma[q]}), & \text{if } x_{\gamma[q]} \in \mathbf{B}. \end{cases}$$

Let $\hat{\Theta}_{[g]}(\gamma)$ be the r.v. $\Theta_{[g]}(\gamma/\Gamma_l)$ for which $p_\gamma = \hat{p}_\gamma$. Then

$$E\Theta^2_{[g]}(\gamma) = \int \dots \int E\Theta^2_{[g]}(\gamma) p_\gamma \prod_{x_{\gamma[q]} \in \mathbf{A} \bigcup \mathbf{B}} dx_{\gamma[q]} \tag{8.25}$$

$$= \int \dots \int \frac{g^2(x_0)}{p_0(x_0)} \prod_{x_{\gamma[q]} \in \mathbf{A}} \frac{K^2(x_{\gamma[q]})}{P(x_{\gamma[q]})}$$

$$\times \prod_{x_{\gamma[r]} \in \mathbf{B}} F^2(x_{\gamma[r]}) \prod_{x_{\gamma[q]} \in \mathbf{A}} dx_{\gamma[q]} \prod_{x_{\gamma[r]} \in \mathbf{B}} dx_{\gamma[r]}.$$

$$E\hat{\Theta}^2_{[g]}(\gamma) = \int \dots \int E\hat{\Theta}^2_{[g]}(\gamma) \hat{p}_\gamma \prod_{x_{\gamma[q]} \in \mathbf{A} \bigcup \mathbf{B}} dx_{\gamma[q]} \tag{8.26}$$

$$= \left(\int \dots \int |g(x_0)| \prod_{x_{\gamma[q]} \in \mathbf{A}} |K(x_{\gamma[q]})| \right.$$

$$\left. \times \prod_{x_{\gamma[r]} \in \mathbf{B}} F(x_{\gamma[r]}) \prod_{x_{\gamma[q]} \in \mathbf{A}} dx_{\gamma[q]} \prod_{x_{\gamma[r]} \in \mathbf{B}} dx_{\gamma[r]} \right)^2.$$

The functions $\Phi(x)$ and $\hat{\Phi}(x)$ are non-negative for any $x \in \Omega$ and the density function p_γ may be written in the following form:

$$\hat{p}_\gamma = c|g(x_0)| \prod_{x_{\gamma[q]} \in \mathbf{A}} |K(x_{\gamma[q]})| \prod_{x_{\gamma[r]} \in \mathbf{B}} \Phi(x_{\gamma[r]}) h(x_{\gamma[r]})$$

$$= \frac{|g(x_0)F(x_0)|}{c^{-1}}$$

$$\times \prod_{x_{\gamma[q]} \in \mathbf{A}} \frac{|K(x_{\gamma[q]})F(x_{\gamma[q]0})F(x_{\gamma[q]1}) \dots F(x_{\gamma[q]m-1})|}{F(x_{\gamma[q]})} \prod_{x_{\gamma[r]} \in \mathbf{B}} h(x_{\gamma[r]})$$

$$= \frac{|g(x_0)F(x_0)|}{c^{-1}}$$

$$\times \prod_{x_{\gamma[q]} \in \mathbf{A}} \frac{|K(x_{\gamma[q]})F(x_{\gamma[q]0})F(x_{\gamma[q]1}) \dots F(x_{\gamma[q]m-1})|}{\int \dots \int |K(x_{\gamma[q]})F(x_{\gamma[q]0}) \dots F(x_{\gamma[q]m-1})| dx_{\gamma[q]0} \dots dx_{\gamma[q]m-1}}$$

$$\times \prod_{x_{\gamma[r]} \in \mathbf{B}} h(x_{\gamma[r]}) = \hat{p}_0(x_0) \prod_{x_{\gamma[q]} \in \mathbf{A}} \hat{P}(x_{\gamma[q]}) \prod_{x_{\gamma[r]} \in \mathbf{B}} h(x_{\gamma[r]}),$$

since $F(x) \geq 0$.

Let the total number in the set $\mathbf{A}$ be M. Then 8.1 should be applied "M" times to the density function $\hat{p}(x_{\gamma[q]})$ in (8.29) and (8.18) as we begin

with the density functions, which describe the last term and we go to the first generation. The terms in (8.29) are of the following form

$$\int \dots \int \frac{k^2(x, y_1, y_2, \dots, y_m}{p(x, y_1, y_2, \dots, y_m)} \prod_{i=1}^{m} F(y_i) \prod_{i=1}^{m} dy_i,$$

and can be separated depending on the generation for which we consider the corresponding density function $\hat{P}(x_{\gamma[q]})$.

Thus, Lemma 8.1 should be applied "M" times to the density function $p(x, y_1, y_2, \dots, y_m)$ in (8.29), because:

$$\begin{aligned}
&\int \dots \int \frac{k^2(x, y_1, \dots, y_m)}{p(x, y_1, \dots, y_m)} \prod_{i=1}^{m} F(y_i) \prod_{i=1}^{m} dy_i \\
&= \int \dots \int \frac{k^2(x, y_1, \dots, y_m)}{\hat{p}(x, y_1, \dots, y_m)} \prod_{i=1}^{m} F(y_i) \prod_{i=1}^{m} dy_i \\
&= \int \dots \int \frac{k^2(x, y_1, \dots, y_m) \prod_{i=1}^{m} F^2(y_i)}{|k(x, y_1, \dots, y_m) \prod_{i=1}^{m} F(y_i)|} \\
&\quad \left(\int \dots \int |k(x, y_1, \dots, y_m) \prod_{i=1}^{m} F(y_i)| \prod_{i=1}^{m} dy_i \right) \prod_{i=1}^{m} dy_i \\
&= \left(\int \dots \int |k(x, y_1, \dots, y_m) \prod_{i=1}^{m} F(y_i)| \prod_{i=1}^{m} dy_i \right)^2 .
\end{aligned}$$

It should be taken into account that for

$$\begin{aligned}
p(x, y_1, \dots, y_m) &= \hat{p}(x, y_1, \dots, y_m) \\
&= \frac{|k(x, y_1, \dots, y_m) \prod_{i=1}^{m} F(y_i)|}{\int \dots \int |k(x, y_1, \dots, y_m) \prod_{i=1}^{m} F(y_i)| \prod_{i=1}^{m} dy_i}
\end{aligned}$$

the corresponding functions $\Phi(x)$ and $\hat{\Phi}(x)$ are equal to each generation point x in the branching process, i.e.

$$\Phi(x) = \hat{\Phi}(x) = F(x),$$

because the corresponding density functions minimize $\Phi(x)$ and their minima is equal to the function $\hat{\Phi}(x)$ for any $x \in \Omega$.

At the end, it holds for any initial density function $p_0(x_0)$ which is *permissible* to the function $g(x_0)$:

$$\min_{\Phi} \left(\int g^2(x_0) \Phi^2(x_0) p_0^{-1}(x_0) dx_0 \right) = \int g^2(x_0) \hat{\Phi}^2(x_0) p_0^{-1}(x_0) dx_0.$$

In this case

$$p_\gamma = p_0(x_0) \prod_{x_{\gamma[q]} \in \mathbf{A}} |K(x_{\gamma[q]})| \prod_{x_{\gamma[r]} \in \mathbf{B}} \Phi(x_{\gamma[r]}) h(x_{\gamma[r]}). \tag{8.27}$$

According to 8.2 the last functional is minimized by the density function (8.30) if

$$p_0(x_0) = \hat{p}_0(x_0) = \frac{|g(x_0)\hat{\Phi}(x_0)|}{\int |g(x_0)\hat{\Phi}(x_0)|dx_0}.$$

This completes the proof.

□

Since

$$\sigma^2(\Theta_{[g]}(\gamma/\Gamma_l)) = E\Theta^2_{[g]}(\gamma) - (E\Theta_{[g]}(\gamma/\Gamma_l)^2$$

it is easy to show that the density function $\hat{p}_\gamma$ minimizes the standard deviation $\sigma(\Theta_{[g]}(\gamma/\Gamma_l))$, because $E\Theta_{[g]}(\gamma/\Gamma_l)$ is a constant for any permissible density function.

It is possible to show that

$$\overline{p}_\gamma = \overline{c}|g(x_0)| \prod_{x_{\gamma[q]} \in \mathbf{A}} |K(x_{\gamma[q]})| \prod_{x_{\gamma[r]} \in \mathbf{B}} h(x_{\gamma[r]}) \tag{8.28}$$

is *almost optimal*. It is easy to show this in the case when the branching stochastic process has only one generation, that is only one tree γ_0 is considered

$$u(x_0) = \int \dots \int k(x_0, y_1, y_2, \dots, y_m) \prod_{i=1}^{m} f(y_i) \prod_{i=1}^{m} dy_i.$$

The next equalities hold:

$$\Phi(x_0) = \left(\int \dots \int \frac{k^2(x_0, y_1, y_2, \dots, y_m)}{p(x_0, y_1, y_2, \dots, y_m)} \prod_{i=1}^{m} \Phi^2(y_i) \prod_{i=1}^{m} dy_i \right)^{\frac{1}{2}}; \tag{8.29}$$

$$\begin{aligned} \hat{\Phi}(x_0) &= \int \dots \int |k(x_0, y_1, y_2, \dots, y_m)| \prod_{i=1}^{m} \Phi(y_i)| \prod_{i=1}^{m} dy_i \\ &= ||K(x_0)\Phi^{(m)}||; \end{aligned} \tag{8.30}$$

$$\hat{p}_{\gamma_0} = \hat{c}_0|g(x_0)| \ |k(x_0, y_1, \dots, y_m)| \prod_{i=1}^{m} \Phi(y_i)h(y_i); \tag{8.31}$$

$$\overline{p}_{\gamma_0} = \overline{c}_0|g(x_0)| \ |k(x_0, y_1, \dots, y_m)| \prod_{i=1}^{m} h(y_i), \tag{8.32}$$

where $(\hat{c}_0)^{-1} = ||g|| \ ||K(x_0)\Phi^{(m)}||; \quad (\overline{c}_0)^{-1} = ||g|| \ ||K(x_0)||.$

Now, let us represent the density function $\overline{p}_{\gamma_0}$ in the following form:

$$\overline{p}_{\gamma_0} = \overline{c}_0 |g(x_0)| \; |k(x_0, y_1, \ldots, y_m)| \prod_{i=1}^{m} h(y_i)$$
$$= \frac{|g(x_0)|}{||g||} \frac{|k(x_0, y_1, \ldots, y_m)|}{||K(x_0)||} \prod_{i=1}^{m} h(y_i) = \overline{p}_0(x_0) \overline{p}(x_0, y_1, \ldots, y_m) \prod_{i=1}^{m} h(y_i).$$

In fact, by substituting $\overline{p}(x_0, y_1, \ldots, y_m)$ into the expression for $\Phi(x_0)$ we get:

$$\Phi^2(x_0) = \left(\int \ldots \int \frac{k^2(x_0, y_1, y_2, \ldots, y_m)}{|k(x_0, y_1, y_2, \ldots, y_m)|} \|K(x_0)\| \prod_{i=1}^{m} \Phi^2(y_i) \prod_{i=1}^{m} dy_i \right)$$
$$= \|K(x_0)\| \|K(x_0)(\Phi^{(m)})^2\|.$$

A necessary and sufficient condition for validity of the equality

$$\|K(x_0)\| \; \|K(x_0)(\Phi^{(m)})^2\| = \|K(x_0)(\Phi^{(m)})\|^2$$

is the following :

$$\|K(x_0)(\Phi^{(m)})^2\| = c_0^2 \|K(x_0)\|, \tag{8.33}$$

where c_0 is constant. It follows that the density $\overline{p}(x_0, y_1, \ldots, y_m)$ minimizes $\Phi(x_0)$ for any $x_0 \in \Omega$ in the case of $\Phi = (c_0)^{\frac{2}{m}} = const$. The value of this minimum is $\hat{\Phi}(x_0)$. In this case the desired functional is

$$E\Theta^2_{[g]}(\gamma_0) = \int g^2(x_0) \Phi^2(x_0) p_0^{-1}(x_0) dx_0$$
$$= c_0^2 \int |g(x_0)| \, ||K(x_0)||^2 \; ||g|| dx_0 = c_0^2 ||g|| \int |g(x_0)| \; ||K(x_0)||^2 dx_0.$$

Here we use $\overline{p}_0(x_0) = |g(x_0)| \; ||g||^{-1}$ instead of $p_0(x_0)$. It is clear that $\overline{p}_0(x_0)$ minimizes $E\Theta^2_{[g]}(\gamma_0)$ if $||K(x_0)|| = const$.

These arguments show that the density function $\overline{p}_{\gamma_0}$ is *almost optimal*.

Here we chose the transition density function $p(x, y_1, y_2, \ldots, y_m)$ and the initial density function $p_0(x)$ in the following way:

$$p(x, y_1, y_2, \ldots, y_m) = C_x |k(x, y_1, y_2, \ldots, y_m)| \text{ and } p_0(x) = C_1 |g(x)|,$$

where

$$C_x^{-1} = \int_\Omega \ldots \int_\Omega |k(x, y_1, y_2, \ldots, y_m)| \prod_{i=1}^{m} dy_i \text{ and } C_1^{-1} = \int_\Omega |g(x)| dx.$$

In this case the density function p_γ is called *almost optimal* and can be written as

$$\begin{aligned} p_\gamma &\equiv \bar{p}_\gamma \\ &= C_1|g(x_0)| \prod_{x_{\gamma[q]}\in A} p_m(x_{\gamma[q]}) C_{x_{\gamma[q]}} |k(x_{\gamma[q]}, x_{\gamma[q]0}, \ldots, x_{\gamma[q]m-1})| \\ &\times \prod_{x_{\gamma[r]}\in B} h(x_{\gamma[r]}). \end{aligned} \tag{8.34}$$

Let us describe the MAO algorithm using the *almost optimal* density function.

To obtain one value of the r.v. $\Theta_{[g]}(\gamma/\Gamma_l)$ the following algorithm is applied:

Algorithm 8.1. :

1. **Compute** the point $\xi \in \Omega$ as a realization of the r.v. τ with a density $p(x) = C_1|g(x)|$. Here

$$\Theta_{[g]}(\gamma/\Gamma_l) = C_1 \frac{g(\xi)}{|g(\xi)|}.$$

2. **Compute** independent realizations, α, of a uniformly distributed r.v. in the interval $[0, 1]$.

3. **If**

$$\alpha \le p_m(\xi),$$

then execute steps 5, 6 and 7. In the other case execute step 4.

4. **Multiply** the value of the r.v. $\Theta_{[g]}(\gamma/\Gamma_l)$ by

$$\frac{f(\xi)}{h(\xi)}.$$

In this case we say that the point ξ dies out.

5. **Compute** of the points $\xi_1, \xi_2, \ldots, \xi_m$ which are components of the m^{th} dimensional r.v. $Y(y_1, y_2, \ldots, y_m)$ with the chosen density function

$$p(x, y_1, y_2, \ldots, y_m) = C_x|k(x, y_1, y_2, \ldots, y_m)|\,.$$

6. **Multiply** the value of the r.v. $\Theta_{[g]}(\gamma/\Gamma_l)$ by

$$\frac{k(\xi, \xi_1, \xi_2, \ldots, \xi_m)}{p_m(\xi) C_\xi |k(\xi, \xi_1, \xi_2, \ldots, \xi_m)|}.$$

7. **Repeat** steps 2 and 3 for the generated points $\xi_1, \xi_2, \ldots, \xi_m$.

8. **Stop** the algorithm when all points are dead out.

Thus, we calculate one realization of the r.v. $\Theta_{[g]}(\gamma/\Gamma_l)$. The algorithm described above has to be executed N-times to obtain N realizations of the r.v. $\Theta_{[g]}(\gamma/\Gamma_l)$. From an algorithmic point of view it is easy to do when the kernel $k(x, y_1, y_2, \ldots, y_m)$ is non-negative and the probability $p_m(x) = p_m$ is a constant.

8.4 Numerical Examples

We consider the MAO algorithm to estimate the functional (8.1). The function $u(x)$ is a solution of the following integral equation with polynomial nonlinearity ($m = 2$):

$$u(x) = \int_\Omega \int_\Omega k(x, y, z)u(y)u(z)dydz + f(x), \tag{8.35}$$

where $x \in I\!R^1$ and $\Omega = \mathbf{E} \equiv [0, 1]$.

The kernel $k(x, y, z)$ is non-negative and the probability $p_2(x)$ is constant.

In our test

$$k(x, y, z) = \frac{x(a_2y - z)^2}{a_1} \text{ and } f(x) = c - \frac{x}{a_3},$$

where $a_1 > 0$, $a_3 \neq 0$, a_2 and c are constants. Thus,

$$K_2 = \max_{x\in[0,1]} \int_0^1 \int_0^1 |k(x, y, z)|dydz = \frac{2a_2^2 - 3a_2 + 2}{6a_1}.$$

The equation (8.35) has a unique solution $u(x) = c$ when the following condition

$$c = \pm \left(\frac{6a_1}{a_3(2a_2^2 - 3a_2 + 2)} \right)^{1/2}$$

holds.

The transition density function which we use in our tests has the following form:

$$p(y, z) = \frac{6}{2a_2^2 - 3a_2 + 2}(a_2y - z)^2.$$

We consider the results for calculating the linear functional (8.1) for

$$a_1 = 11,\ a_2 = 4,\ a_3 = 12,\ c = 0.5,$$

and

$$g(x) = \delta(x - x_0).$$

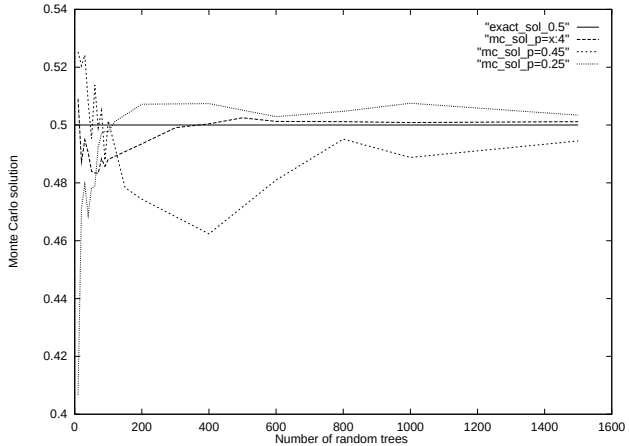


Fig. 8.3 Comparison of the Monte Carlo solutions with the exact solution. Three cases for the probability $p_2(x)$ that generates new points.

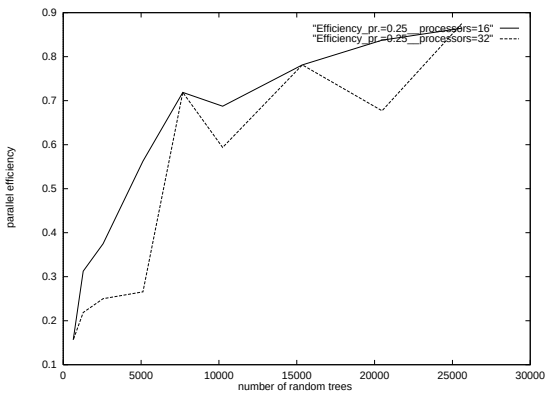


Fig. 8.4 The parallel Monte Carlo efficiency when $p_2(x) = 0.25$. The number of processors is 16 and 32.

Some results are plotted on Figures 8.3 to 8.6. Figure 8.3 shows the dependence of the approximated solution from the number of the random trees. Three different transition density functions are considered $(0.25, 0.45, x/4)$. One can see that there are oscillations of the solution for a small numbers of random trees. The best solution is obtained for the third case $(p(x) = x/4)$. As one can expect with the increasing number of random trees the Monte Carlo solution goes asymptotically to the *true* solution of the problem (which is 0.5 for this particular example).

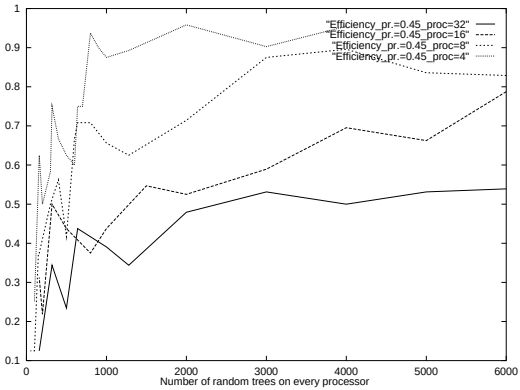


Fig. 8.5 Parallel Monte Carlo efficiency for a number of random trees up to 6000 ($p_2(x) = 0.45$). The number of processors is 4, 8, 16 and 32.

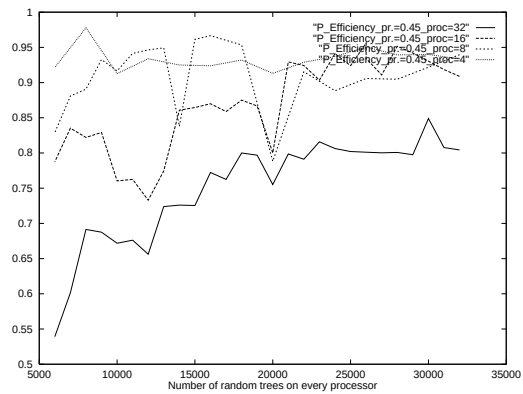


Fig. 8.6 Parallel Monte Carlo efficiency for a number of random trees up to 35000 ($p_2(x) = 0.45$). The number of processors is 4, 8, 16 and 32.

Since the problem under consideration is suitable for parallel implementation we also study the parallel behaviours of the implemented algorithm. On Figure 8.4 the parallel efficiency of the algorithm is shown. The efficiency oscillates around $0.2 - 0.3$ when a small number $(200 - 300)$ of trees is used on every of 16^{th} and 32^{nd} processor nodes used of the computational system. The parallel efficiency grows up when the number of the random trees increase.

Figures 8.5 and 8.6 show how the parallel efficiency of the presented algorithm depends on the number of processor nodes. In these numerical tests all random trees are distributed on all processors. That means that

the system of 4 processors performs the same number of realizations as the system of 32 processors. As a result the system of 4 processors is more efficient (in general) than the system of 32 processors. This result is expected since the method of parallelization gives a better load balancing for a smaller number of processors.

Chapter 9

Algorithmic Efficiency for Different Computer Models

9.1 Parallel Efficiency Criterion

Monte Carlo algorithms are known to be inherently parallel. Already in the first paper on the Monte Carlo method [Metropolis and Ulam (1949)] it was said that "the statistical methods can be applied by many computers working in parallel and independently". Nevertheless, the problem of creating efficient parallel and vectorizable Monte Carlo algorithms is not a trivial problem, because there are different ways for parallelization and vectorization of the algorithms.

In order to estimate how the Monte Carlo algorithms depend on different computer architectures, we consider the following models.

(1) A serial model with time τ required to complete a suboperation (for real computers this time is usually the clock period). The important feature of this model is that the operations have to be performed sequentially and one at a time.

(2) A pipeline model with startup time $s\tau$. Pipelining means an application of assembly-line techniques to improve the performance of the arithmetic operations (see, Figure 9.1).

(3) A multiprocessor configuration consisting of p processors. Every processor of the multiprocessor system performs its own instructions on the data in its own memory (see, Figure 9.2).

(4) A multiprocessor system of p processor arrays. It is a model of the third type but one in which every processor is connected to a processor array of $2d-1$ processing elements.The processor array is a set of processing elements performing the same operations on the data in its own memory,

i.e., SIMD (Single Instruction stream, Multiple Data stream) architecture.

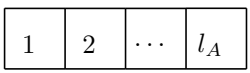


Fig. 9.1 A pipeline model with l_A segments

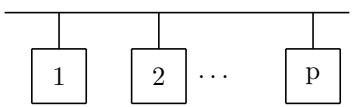


Fig. 9.2 A multiprocessor model with p processors

Considering models instead of real computers allows for the distinction of the typical features of the algorithms on the one hand, and, on the other hand, enables general conclusions about the algorithms to be drawn.

The inherent parallelism of the Monte Carlo algorithms lies in the possibility of calculating each realization of the r.v. θ on a different processor (or computer). There is no need for communication between the processors during the calculation of the realizations - the only need for communication occurs at the end when the averaged value is to be calculated.

To estimate the performance of the Monte Carlo algorithms, we need to modify the usual criterion.

Let a parallel algorithm A for solving the given problem Y be given. If A is a deterministic algorithm then $T_p(A)$ is the time (or the number of suboperations) needed for performing the algorithm A on a system of p processors. An algorithm B is called the best (B) sequential algorithm for the problem Y if the inequality

$$T_1(B) \leq T_1(A)$$

holds for any existing algorithm A.

It is helpful to consider the following two parameters for characterizing

the quality of the algorithm – speed-up and parallel efficiency[1]:

$$S_p(A) = \frac{T_1(A)}{T_p(A)}$$

and

$$E_p(A) = \frac{S_p(A)}{p}.$$

Obviously, $S_1(A) = E_1(A) = 1$ for any A.

If A is a Monte Carlo algorithm it is impossible to obtain an exact theoretical estimate for $T_p(A)$ since every run of the algorithm yields different values for $T_p(A)$. Theoretical estimates are possible only for the mathematical expectation, i.e. for $ET_p(A)$. Therefore the following has to be considered for the speed-up of the Monte Carlo algorithm:

$$S_p(A) = \frac{ET_1(A)}{ET_p(A)}.$$

We shall call the algorithm B *p-the-best* if

$$ET_p(A) \geq ET_p(B).$$

(Obviously, if an algorithm D is a deterministic algorithm, then $ET_p(D) = T_p(D)$, for any $p = 1, 2, \ldots$.)

For almost all existing deterministic algorithms the parallel efficiency goes down rapidly when $p \geq 6$. In the general case the parallel efficiency of the deterministic algorithms strongly depends on the number of processors p. For Monte Carlo algorithms the situation is completely different. The parallel efficiency does not depend on p and may be very close to 1 for a large number of processors.

In the next sections of this chapter we deal with the performance analysis of different Monte Carlo algorithms. We present results for the computational complexity, speed-up and parallel efficiency. In Section 9.2 Markov chain Monte Carlo algorithms for linear algebra problems are considered. In Section 9.3 various Monte Carlo algorithms for solving boundary value problems are studied.

9.2 Markov Chain Algorithms for Linear Algebra Problems

In this section we consider iterative Monte Carlo algorithms that use Markov chains for calculation the required r.v. As it was shown in Chapters 4 and 5 Markov chain Monte Carlo algorithms are used for computing the approximate solution of

[1]The parallel efficiency is the measure of the price to be paid for the speedup achieved.

- bilinear forms of matrix powers (Subsection 5.2.1);
- bilinear forms of the solution of systems of linear algebraic equations (Section 4.2);
- matrix inversion (Section 4.2);
- extremal eigenvalue (or *small* number of eigenvalues) of real symmetric matrices (Subsection 5.2.2).

The basic algorithm for solving the above problems is the *Power* Monte Carlo algorithm. There are modifications of the *Power* Monte Carlo algorithm called *Inverse*, *Shifted* and the *Resolvent* Monte Carlo (see Chapters 4 and 5). For practical computations the *Power* and *Resolvent* Monte Carlo are more frequently used. The only difference is in the matrices used as linear iterative operators for the Monte Carlo iterations. So that the structure of the estimates of algorithmic complexity for all Markov chain algorithms is similar. That's why we analyse the performance of the *Resolvent* Monte Carlo for computing extremal eigenvalues mentioning the difference between the *Resolvent* and the *Power* Monte Carlo algorithm.

The *Resolvent* Monte Carlo algorithm (RMC) for evaluating the extreme eigenvalues of symmetric matrices can be described as follows.

Algorithm 9.1. *Resolvent* Monte Carlo Algorithm

1. Choose *an initial row of the matrix* $A = \{a_{ij}\}_{i,j=1}^{n}$ *as a realization of the density* p_0 *permissible to the vector* v*: divide the interval* $[0,1]$ *in subintervals with lengths proportional of the vector coordinates* $v_1, v_2, \ldots, v_n$*:*

$$\triangle p_i = c|v_i|, \quad i = 1, \ldots, n, \tag{9.1}$$

where c *is a constant. Then generate a random number* γ *in* $[0,1]$.

Suppose the element v_{α_0} *has been chosen.*

2. *Consider the* α_0^{th} *row of the matrix* A. **Choose** *an element of* α_0^{th} *row as a realization of the transition density function* $p_{\alpha_0\alpha_1}$ *in the following way: divide the interval* $[0,1]$ *into subintervals* $\triangle p_i$, $i = 1, \ldots, n$, *proportional to the row elements*

$$\triangle p_i = c_{\alpha_0}|a_{\alpha_0 i}|, \qquad i = 1, 2, \ldots, n, \tag{9.2}$$

where c_{α_0} *is a constant, and generate a random number* γ *in* $[0,1]$. *Let the chosen element be* $a_{\alpha_0\alpha_1}$.

3. *Consider the* α_1^{th} *row of the matrix* A *and repeat the procedure* **2** *using density functions* $p_{\alpha_0\alpha_1}$. **Continue** *this procedure until determining the element* $a_{\alpha_i\alpha_{i+1}}$ *according to the density* $p_{\alpha_i\alpha_{i+1}}$.

4. Compute *the r.v. (for a fixed integer value of* k *and* m*)*

$$\theta[u]_{\nu}^{(1)}[h,A]=\sum_{i=0}^{\nu}q^{i}C_{i+m-1}^{i}\theta^{(i+1)},\quad \nu=k-1,k \tag{9.3}$$

$$\theta[d]_{\nu}^{(1)}[h,A]=\sum_{i=0}^{\nu}q^{i}C_{i+m-1}^{i}\theta^{(i)},\quad \nu=k-1,k \tag{9.4}$$

where

$$\theta^{(i)}=\frac{v_{\alpha_0}}{p_{\alpha_0}}\frac{a_{\alpha_0\alpha_1}a_{\alpha_1\alpha_2}\dots a_{\alpha_{i-1}\alpha_i}}{p_{\alpha_0\alpha_1}p_{\alpha_1\alpha_2}\dots p_{\alpha_{i-1}\alpha_i}}.$$

5. Go to 1 *and repeat steps* **1, 2, 3** *and* **4**. **Perform** $n-1$ *new realizations of the r.v.* $\theta[u]_{\nu}^{(s)}[h,A]$*:* $\theta[u]_{\nu}^{(2)}$, $\theta[u]_{\nu}^{(3)}$, …, $\theta[u]_{\nu}^{(N)}$ *and* $\theta[d]_{\nu}^{(s)}[h,A]$*:* $\theta[d]_{\nu}^{(2)}$, $\theta[d]_{\nu}^{(3)}$, …, $\theta[d]_{\nu}^{(N)}$.

6. Calculate

$$\overline{\theta}[u]_{\nu}[h,A]=\frac{1}{N}\sum_{s=1}^{N}\theta[u]_{\nu}^{(s)}[h,A],\ \text{ for }\ \nu=k-1,k. \tag{9.5}$$

$$\overline{\theta}[d]_{\nu}[h,A]=\frac{1}{N}\sum_{s=1}^{N}\theta[d]_{\nu}^{(s)}[h,A],\ \text{ for }\ \nu=k-1,k. \tag{9.6}$$

7. Compute *the current approximation of the eigenvalue:*

$$\lambda^{(k-1)}=\frac{\overline{\theta}[u]_{k-1}[h,A]}{\overline{\theta}[d]_{k-1}[h,A]}$$

and

$$\lambda^{(k)}=\frac{\overline{\theta}[u]_{k}[h,A]}{\overline{\theta}[d]_{k}[h,A]}$$

8. If

$$|\lambda^{(k)}-\lambda^{(k-1)}|<\varepsilon \tag{9.7}$$

then stop *iterations.* **Else** *continue iterations using steps* **4, 5, 6** *until reaching new values of the parameters* k *and* m *and check the inequality (9.7).*

The process **continues until** *the inequality (9.7) is fulfilled or* **until** *a very large number of iterations* k *or very high power* m *of the resolvent matrix is reached.*

Here the estimates for the mathematical expectation of the time, speed-up and parallel efficiency will be presented. All three parameters define the quality of the parallel and/or vectorizable algorithms.

In fact, all Markov chain algorithms for linear algebra problems could be considered as random processes which consist in jumping on element of the matrix following the schemes described in Chapter 5.

Every move in a Markov chain is done according to the following algorithm:

Algorithm 9.2. *Resolvent* MC Algorithm

- **(i)** generation of a random number (it is usually done in $\varkappa$ floating point operations where $\varkappa = 2$ or 3);
- **(ii)** determination of the next element of the matrix: this step includes a random number of logical operations[2];
- **(iii)** calculating the corresponding r.v.

Since Monte Carlo Almost Optimal (MAO) algorithm which corresponds to permissible densities (see, Section 4.2, Definition 4.1) is used, the random process never visits the zero-elements of the matrix A. (This is one of the reasons why MAO algorithm has high algorithmic efficiency for sparse matrices.)

Let d_i be the number of non-zero elements of i^{th} row of the matrix A. Let us estimate the number of logical operations γ_L in every move of the Markov chain in the worst case. It can be estimated using the following expression

$$E\gamma_L \approx \frac{1}{2}\frac{1}{n}\sum_{i=1}^{n} d_i = \frac{1}{2}d, \tag{9.8}$$

where d is the mean value of the number of non-zero elements of every row of the matrix A. Note that if the binary search method [Knuth (1997)] is used to select the next element in the Markov chain, then $(\log d)$ operations are needed (for more details see [Alexandrov *et al.* (2004)]).

Since γ_L is a r.v. we need an estimate of the probable error of (9.8). It depends on the balance of the matrix. For matrices which are not very unbalanced and of not very small-size ($n = 2, 3$), the probable error of (9.8) is negligible small in comparison with γ_L.

[2]Here again the logical operation deals with testing the inequality "$a < b$".

The number of the arithmetic operations, excluding the number of arithmetic operations $\varkappa$ for generating the random number is γ_A (in our code-realization of the algorithm $\gamma_A = 6$: that is

- 2 operations for obtaining the next value of W_i (in case of computing bilinear forms of the solution of a system of linear algebraic equations) or $\theta^{(k)}$ (in case of computing bilinear forms of matrix powers),
- 1 operation for obtaining the next power of q, q^i (this applies for the *Resolvent* Monte Carlo only; for the *Power* Monte Carlo it's equal to zero),
- 3 operations for obtaining the next term of the summation.

So, the mathematical expectation of the operations needed for each move of any Markov chain is

$$E\delta = \left[(\varkappa + \gamma_A)l_A + \frac{1}{2}dl_L\right], \tag{9.9}$$

where l_A and l_L are the numbers of suboperations of the arithmetic and logical operations, respectively.

In order to obtain the initial density vector $2n$ operations for computing $\{p_\alpha\}_{\alpha=1}^n$ are needed since

- $n-1$ summation and one multiplication is needed for obtaining the value of $\frac{1}{\sum_{i=1}^n |v_i|}$, and
- n multiplications are needed for obtaining the initial density vector, $\{p_\alpha\}_{\alpha=1}^n = \frac{|v_\alpha|}{\sum_{i=1}^n |v_i|}$.

To obtain the transition density matrix $\{p_{\alpha\beta}\}_{\alpha,\beta=1}^n$, the algorithm needs $2dn$ arithmetic operations, where d is the mean value of the number of non-zero elements per row because

- $d_i - 1$ summations and one multiplication for obtaining the value of $\frac{1}{\sum_{j=1}^n |a_{ij}|}$ for each row a_i, $\quad i = 1, \ldots, n$ of the matrix A are needed, and
- d_i multiplications for obtaining the i^{th} row of the transition density matrix are needed;
- we need to repeat the above mentioned procedure for every row of the matrix A, that is $n-1$ times.

Thus, the mathematical expectation of the total time becomes

$$ET_1(RMC) \approx 2\tau \left[(\varkappa + \gamma_A) l_A + \frac{1}{2} d l_L\right] kN + 2\tau n(1+d), \tag{9.10}$$

where k is the number of moves in every Markov chain, and N is the number of Markov chains and τ is the clock period.

It is worth noting that the main term of (9.10) does not depend on the size n of the matrix and the next term has $O(n)$ operations for sparse matrices and $O(n^2)$ operations for dense matrices. This result means that the time required for calculating the eigenvalue by the *Resolvent* MC practically does not depend on n. The parameter k depends on the spectrum of the matrix, but do not depend on the size n. The parameter N controls the stochastic error (for more details about choosing parameters k and N see Section 5.4). The above mentioned result was confirmed for a wide range of matrices during numerical experiments.

Let us consider now the pipeline model. In this case there are two levels of vectorization. The first (low) level consists in vectorization of every arithmetic operation using a sequence of suboperations (the number of suboperations for one arithmetic operation is usually equal to 4). The second (high) level consists in vectorization of *DO*-loops. For both type of vectorization one number of the result is produced at every clock period (with a periodicity τ). In comparison with the previous case, the scalar computer produces every number of the result at every τl_A moment. So, one can obtain

$$\begin{aligned} ET_{pipe}(RMC) &\approx \tau \left[\left(s + c + \varkappa + \gamma_A + l_A + \frac{1}{2} d l_L\right) + k + N\right] \\ &\quad + \tau[2 + s + n] + \tau[2 + s + d + n] \qquad (9.11) \\ &= \tau \left[4 + 3s + c + \varkappa + \gamma_A + l_A + 2n + \left(\frac{1}{2} l_L + 1\right) d + k + N\right], \end{aligned}$$

where τs is the startup time of the pipeline processor and the factor c (which is usually very small) describes the increase of the time for joint vectorization of the calculations of the r.v. $\overline{\theta}[u]_\nu[h, A]$ and $\overline{\theta}[d]_\nu[h, A]$ (see, (9.5) and (9.6)). In comparison with (4.8) in (9.11) there is summation instead of multiplication in the term expressing the number of arithmetic operations (low level vectorization) and, instead of multiplication by k and N, there is summation because k and N are parameters of *DO*-loops (high level vectorization). The second term of (9.11) expresses the low level of vectorization ($2 + n$ instead of $2 \times n$). The third term of (4.8) expresses the

low level ($2 + d$ instead of $2 \times d$) as well as high level ($+n$ instead of $\times n$) of vectorization.

Consider the third model - multiprocessor model consisting of p processors. In this case every Markov chain can be run independently and in parallel, so that we have

$$ET_p(RMC) \approx 2\tau \left[(\varkappa + \gamma_A) l_A + \frac{1}{2} d l_L \right] k \frac{N}{p} + 2\tau n \left[1 + d \frac{1}{p} \right], \qquad (9.12)$$

where p is the number of processors.

The expression (9.12) shows that the *Resolvent* MC can by very efficient for a large N even for a large number of processors p, which is quite interesting. Consider now the speed-up and the parallel efficiency for the second and the third model of computer architectures.

One can get for the pipeline model

$$S_{pipe}(RMC) \approx \frac{2\left[(\varkappa + \gamma_A) l_A + \frac{1}{2} d l_L \right] k + 2n(1+d)}{4 + 3s + c + \varkappa + \gamma_A + l_A + 2n + \left(\frac{1}{2} l_L + 1 \right) d + k + N}.$$

Let us assume that

$$n(1+d) << \left[(\varkappa + \gamma_A) l_A + \frac{1}{2} d l_L \right] kN;$$

$$k << N;$$

$$s + \varkappa + \gamma_A + l_A \leq (\varkappa + \gamma_A) l_A.$$

Then

$$S_{pipe}(RMC) \geq \frac{2k}{\frac{1}{N} + \frac{1}{B}},$$

where

$$B = (\varkappa + \gamma_A) l_A + \frac{1}{2} d l_L.$$

As for the parallel efficiency, we have

$$E_{pipe}(RMC) \geq \frac{2k}{\frac{1}{N} + \frac{1}{B}},$$

since there is just one processor.

For a very rough estimate (supposing $N \approx B$) one can use the following simple expression for the speed-up

$$S_{pipe}(RMC) \approx kN.$$

For the multiprocessor model with p processors one can get

$$S_p(RMC) \approx \frac{\left[(\varkappa+\gamma_A)l_A+\frac{1}{2}dl_L\right]kN+n(1+d)}{\left[(\varkappa+\gamma_A)l_A+\frac{1}{2}dl_L\right]l\frac{N}{p}+n\left[1+\frac{d}{p}\right]}.$$

Suppose that

$$\left[(\varkappa+\gamma_A)l_A+\frac{1}{2}dl_L\right]kN=\frac{1}{\varepsilon}n(p+d),$$

where ε is a small positive number. Then for every $p\geq 1$

$$S_p(RMC)\geq\frac{p}{1+\varepsilon}\geq 1.$$

For the parallel efficiency we have

$$\frac{1}{1+\varepsilon}\leq E_p(RMC)\leq 1.$$

The last inequality shows that the parallel efficiency of RMC algorithm can be really very close to 1.

9.3 Algorithms for Boundary Value Problems

Here the following classical problem will be considered:

$$\Delta u=-f(x),\quad x\in\Omega, \tag{9.13}$$

$$u(x)=\psi(x),\quad x\in\partial\Omega, \tag{9.14}$$

where $x=(x_{(1)},\ldots,x_{(d)})\in\overline{\Omega}\subset C^d=\{0\leq x\leq 1,\ i=1,2,\ldots,d\}$.

Our aim is to calculate the linear functional

$$J(u)=(g,u)=\int_\Omega g(x)u(x)dx. \tag{9.15}$$

We will next consider three Monte Carlo algorithms for the evaluation of this inner product.

Here we also estimate the *computational complexity* in terms of the definition given in Chapter 1. The product $t\sigma^2(\theta)$ is considered as the *computational complexity* of the algorithm.

9.3.1 *Algorithm $\mathcal{A}$ (Grid Algorithm)*

Here we consider the grid Monte Carlo algorithm described in Section 4.3.2. Using a regular discretization with step h, the problem (9.13) is approximated by the difference equation

$$\Delta_h^{(d)} u = -f_h, \tag{9.16}$$

or solved for the ith point $i = (i_1, \ldots, i_n)$:

$$u_i = L_h u + \frac{h^2}{2d} f_i, \tag{9.17}$$

where $\Delta_h^{(d)}$ is the Laplace difference operator, and L_h is an averaging operator.

The approximation error of (9.16), (9.17) is

$$|u_i - u(x_i)| = O(h^2). \tag{9.18}$$

We need $h = \sqrt{\varepsilon}$ to ensure consistency of the approximation error and the probability error.

Every transition in the Markov chain is done following the algorithm

Algorithm 9.3.

(i) generation of a random number (it is usually done in $\varkappa$ arithmetic operations, where $\varkappa = 2$ or 3);

(ii) determination of the next point which includes a random number of logical operations [3] with expectation equal to d, and maximum $2d$ (d is the space dimension) and one algebraic operation to calculate the coordinates of the point.

Let $m_i(h, d)$ be the mean number of steps required to reach the boundary for a process (or a chain) starting at the point i and $m(h, d)$ is the vector with coordinates m. Then $m(h, d)$ is the solution of the following finite difference problem

$$\begin{cases} m_i = L_h m + \varphi_i, \; i \in \Omega_h, & \varphi_i = \frac{2d}{h^2}, \\ m_i = 0, & i \in \partial\Omega_h. \end{cases} \tag{9.19}$$

If $\{m_i'(h, d)\}$ is the solution of problem (9.19) in the unit cube $\mathbf{E}^d$, then the following inequality holds:

$$m_i'(h, d) \geq m_i(h, d), \quad i = 1, 2, \ldots.$$

[3] Here logical operation means testing the inequality "$a < b$".

The difference problem for $m'_i(h,d)$ could be approximated by the partial differential problem

$$\begin{cases} \Delta m'(x) = -\frac{2d}{h^2}, & x \in \mathbf{E}^d, \\ m'(x) = \frac{1}{h^2}\sum_{l=1}^{d}(x_l - x_l^2), & x \in \partial\mathbf{E}^d, \end{cases}$$

where the error is of order $O(1)$. Note that to estimate the computational complexity we approximate the finite difference problem by a differential problem which has an exact solution.

Let M be the maximum of $m'(x)$, i.e.,

$$M = \max_{\mathbf{E}^d} \; m'(x) = \frac{1}{4h^2} = \frac{1}{4\varepsilon}. \tag{9.20}$$

This value is independent of the problem dimension and could be reached when $x_l = \frac{1}{2}, l = 1, 2, \ldots, d$.

The result in (9.20) is natural because the inverse operator of $\Delta_h^{(d)}$ is uniformly bounded over h, and so it follows that

$$m_i(h,d) = \frac{c(d)}{h^2} + O\left(\frac{1}{h^2}\right).$$

Thus, M is an upper bound for the expectation of the number of steps in a Markov chain, which is a realization of the r.v. θ. To achieve a probable error ε, it is necessary to average N realizations of the r.v. θ, where

$$N = c_{0.5}^2 \frac{\sigma^2(\theta)}{\varepsilon^2}.$$

The expectation of the number of all transitions R in the Markov chain will then be

$$R \le MN = \frac{1}{4} c_{0.5}^2 \frac{1}{\varepsilon^3} \sigma^2(\theta) = t\sigma^2(\theta). \tag{9.21}$$

Let l_A and l_L be the number of suboperations of the arithmetic and logical operations, respectively. Then the expectation of the time required to achieve a probable error on the serial model using the algorithm $\mathcal{A}$ is

$$ET_1(\mathcal{A}) = \tau\left[(\varkappa + 1 + \gamma_A)l_A + (d + 1 + \gamma_L)l_L\right]\left(\frac{1}{4}(c_{0.5}\sigma(\theta))\right)^2 \frac{1}{\varepsilon^3},$$

where γ_A and γ_L are the number of arithmetic and logical operations to calculate the distance from a given point to the boundary $\partial\Omega_h$. Here we assume that $f(x) = 0$ in (9.13), because the number of operations to calculate a value of $f(x)$ in an arbitrary point x varies for different functions f.

In the case of the third model we suppose that it is possible to choose the number of processors to be

$$p = \left(c_{0.5} \frac{\sigma(\theta)}{\varepsilon} \right)^2 .$$

Then,

$$ET_p(\mathcal{A}) = \tau \left[(\varkappa + 1 + \gamma_A) l_A + (d + 1 + \gamma_L) l_L \right] \frac{1}{4\varepsilon},$$

$$S_p(\mathcal{A}) = p$$

and

$$E_p(\mathcal{A}) = 1,$$

i.e., the parallel efficiency has its maximum value in terms of the modified criterion.

The speed-up can be improved using the fourth model. In this case all $2d$ logical operations at each step of the Markov chain are done simultaneously, and so

$$ET_{2dp}(\mathcal{A}) = \frac{\tau}{4\varepsilon} \left[(\varkappa + 1 + \gamma_A) l_A + 3 l_L \right],$$

$$S_{2dp}(\mathcal{A}) = p \left(1 + \frac{d + 1 + \gamma_L}{\varkappa + 1 + \gamma_A} \frac{l_L}{l_A} \right) \Bigg/ \left(1 + \frac{3}{\varkappa + 1 + \gamma_A} \frac{l_L}{l_A} \right), \qquad (9.22)$$

and $S_{2dp}(\mathcal{A}) > S_p(\mathcal{A})$ when $d + 1 + \gamma_L \sim 3$.

The parallel efficiency is

$$E_{2np}(\mathcal{A}) = \frac{1}{2d} \left(1 + \frac{d + 1 + \gamma_L}{\varkappa + 1 + \gamma_A} \frac{l_L}{l_A} \right) \Bigg/ \left(1 + \frac{3}{\varkappa + 1 + \gamma_A} \frac{l_L}{l_A} \right).$$

Assuming the pipeline model to have l_A segments, we obtain the following results:

$$ET_{pipe}(\mathcal{A}) = \tau [s + \varkappa + l_A + \gamma_A + (d + 1 + \gamma_L) l_L] \left(\frac{1}{2} (c_{0.5} \sigma(\theta)) \right)^2 \frac{1}{\varepsilon^3},$$

$$S_{pipe}(\mathcal{A}) = \left(1 + \frac{\varkappa + 1 + \gamma_A}{d + 1 + \gamma_L} \frac{l_A}{l_L} \right) \Bigg/ \left(1 + \frac{s + \varkappa + l_A + \gamma_A}{d + 1 + \gamma_L} \frac{1}{l_L} \right).$$

From (9.22) follows that $S_{pipe}(\mathcal{A}) > 1$ when s, l_A and γ_A meet the inequality

$$s < (l_A - 1)(\varkappa + \gamma_A), \qquad (9.23)$$

which is not a real restriction, but it becomes obvious that algorithms like $\mathcal{A}$ are not well suited for the pipeline model. This follows from the formula for $S_{pipe}(\mathcal{A})$ which does not have a factor $1/\varepsilon$, whereas $S_{pipe}(\mathcal{A}) = O(\varepsilon^{-2})$.

Special Monte Carlo algorithms can be considered for the pipeline computers which are known as vector algorithms. Such an algorithm will be considered later in this chapter.

9.3.2 *Algorithm $\mathcal{B}$ (Random Jumps on Mesh Points Algorithm)*

Let us consider again the problem of calculating the inner product (g, u) (see (9.15)) with a preset probable error ε.

The discretization formula for the Laplace equation

$$U_{i,j,k} = \frac{1}{6}\Lambda_{i,j,k},$$

stated in Appendix A, leads to the r.v.

$$\theta = \psi(\xi),$$

where ξ is a random point on the boundary $\partial\Omega_h$. The expectation of this r.v. coincides with the required value.

The Monte Carlo algorithm for estimating the expectation of θ consist in calculating repeated realizations of θ which are Markov chains with initial density p_0 permissible to g (see, Definition 4.1 in Section 4.2), and transition probabilities

$$p_{\alpha\beta} = \begin{cases} \frac{1}{6}, & \text{when } \alpha \in \Omega_h, \\ \delta_{\alpha\beta}, & \text{when } \alpha \in \partial\Omega_h, \end{cases}$$

where

$$\delta_{\alpha\beta} = \begin{cases} 1, & \alpha = \beta, \\ 0, & \alpha \neq \beta, \end{cases}$$

is the Kronecker notation. Next, we consider the problem of estimating the number of operations R required to reach the probable error ε.

From (9.21) it follows that the estimators for the number of realizations N and the number of transitions M in one Markov chain have to be obtained.

The number of realizations N is independent of the algorithm used, and is given by

$$N = \left(c_{0.5} \frac{\sigma(\theta)}{\varepsilon} \right)^2. \tag{9.24}$$

Let us find an upper bound for the number of steps M in a Markov chain. To do that we consider the following problem. Let D be a plane in $\mathbb{R}^3$ and Ω be the half-space bounded by D. The domain Ω is replaced by a set of points ω_h using a regular discretization. The Markov chains in the algorithm $\mathcal{B}$ are random walks in ω_h that begin at a point chosen

according to the initial density p_0; the next point is selected according to the following stipulations.

Algorithm 9.4.

- **(i) Determine** the maximum approximation molecule (scheme)[4] $\mu(\alpha)$ for the inner point α.
- **(ii) Select** the point β uniformly at random on $\mu(\alpha)$ with probability $p_{\alpha\beta} = \frac{1}{6}$.
- **(iii)** If the point β is boundary mesh-point, **terminate** the Markov chain and **start** the process from the point chosen correspondingly to p_0. If the point β is an inner point of ω_h, then **do** step (ii) using β in place of α.

Our aim is to estimate the number of inner points passed by until a boundary point is reached.

Let T be a r.v. that is the number of steps in a Markov chain, and T_ν be the number of steps in the νth Markov chain. Then

$$P\{T_\nu = k\} = \left(\frac{5}{6}\right)^{k-1} \frac{1}{6},$$

and the expectation of the r.v. T is given by

$$ET = \sum_{k=1}^{\infty} k \left(\frac{5}{6}\right)^{k-1} \frac{1}{6} = 6.$$

This result gives an upper bound for the number of steps in a Markov chain, i.e.,

$$M \leq 6;$$

so in this case

$$R < 6 \left(c_{0.5} \frac{\sigma(\theta)}{\varepsilon}\right)^2.$$

For one transition in the Markov chain we have to do the following operations:

- γ_A arithmetic and 5 logical operations to find (calculate) the distance from the point to the boundary $\partial\Omega$ and 1 logical operation to verify if this distance is nonzero, i.e., the point is not on the boundary;

[4]The approximation molecule (scheme) $\mu(\alpha)$ for the differential operator is the set of mesh points used for approximation of the derivatives in the operator at the point α.

- $\varkappa$ arithmetic operations for the generation of random number;
- 3 logical and 1 arithmetic operations for calculating the coordinates of the next point.

So, for a serial model it is obtained that

$$ET_1(\mathcal{B}) = 6\tau\left[(\varkappa + 1 + \gamma_A)l_A + 9l_L\right]\left(c_{0.5}\frac{\sigma(\theta)}{\varepsilon}\right)^2,$$

and for the third model with $p = (c_{0.4}\sigma(\theta)/\varepsilon)^2$ processors the expectation is

$$ET_p(\mathcal{B}) = 6\tau\left[(\varkappa + 1 + \gamma_A)l_A + 9l_L\right].$$

Then the coefficients for the speed-up and the performance are

$$S_p(\mathcal{B}) = p,$$

and

$$E_p(\mathcal{B}) = \frac{S_p(\mathcal{B})}{p} = 1, \tag{9.25}$$

which means that one could achieve the optimum performance.

The calculation of repeated realizations of a r.v. does not need any communication between the processors, so the mathematical expectation of the idle time of the processors is zero.

The value of $S_p(\mathcal{B})$ could be improved with the fourth model. In this case all logical operations for determining the next point in the Markov chain and those for finding the distance to the boundary could be fulfilled at two clock periods, and so

$$ET_{6p}(\mathcal{B}) = 6\tau\left((\varkappa + 1 + \gamma_A)l_A + 5l_L\right) \tag{9.26}$$

and

$$S_{6p}(\mathcal{B}) = \frac{(\varkappa + 1 + \gamma_A)l_A + 9l_L}{(\varkappa + 1 + \gamma_A)l_A + 5l_L}\left(c_{0.5}\frac{\sigma(\theta)}{\varepsilon}\right)^2$$

$$= p\left(1 + \frac{9}{\varkappa + 1 + \gamma_A}\frac{l_L}{l_A}\right)\Big/\left(1 + \frac{5}{\varkappa + 1 + \gamma_A}\frac{l_L}{l_A}\right) > S_p(\mathcal{B}). \tag{9.27}$$

For the performance coefficient we obtain

$$E_{6p}(\mathcal{B}) = \frac{1}{6}\left(1 + \frac{9}{\varkappa + 1 + \gamma_A}\frac{l_L}{l_A}\right)\Big/\left(1 + \frac{5}{\varkappa + 1 + \gamma_A}\frac{l_L}{l_A}\right) < 1. \tag{9.28}$$

It is obvious that increasing the number of processing elements will involve increasing the idle time, i.e., efficiency will decrease.

Let us now consider the pipeline model with l_A segments for performing the arithmetic operations. In this case the coefficients of interest become

$$ET_{pipe}(\mathcal{B}) = 6\tau(s + \varkappa + l_A + \gamma_A + 9l_L)\left(c_{0.5}\frac{\sigma(\theta)}{\varepsilon}\right)^2, \tag{9.29}$$

$$S_{pipe}(\mathcal{B}) = \frac{\left(1 + \frac{1}{9}(\varkappa + 1 + \gamma_A)\frac{l_A}{l_L}\right)}{\left(1 + \frac{1}{9}(s + \varkappa + l_A + \gamma_A)\frac{1}{l_L}\right)}, \tag{9.30}$$

and the sufficient condition for $S_{pipe}(\mathcal{B}) > 1$ is

$$s < (l_A - 1)(\varkappa + \gamma_A), \tag{9.31}$$

which holds for all real computers.

As in the case of the algorithm $\mathcal{A}$, one can see that this algorithm is not the best one for pipeline computers.

9.3.3 *Algorithm $\mathcal{C}$ (Grid-Free Algorithm)*

There exists a different approach to the solution of problem (9.13), (9.14) which uses the following integral representation [Muller (1956); Ermakov and Mikhailov (1982)].

$$u(x) = \frac{1}{4\pi}\int\left[\frac{1}{|x-y|}\frac{\partial u(y)}{\partial n_y} - u(y)\frac{\partial}{\partial n_y}\frac{1}{|x-y|}\right]ds_y, \tag{9.32}$$

for $f(x) \equiv 0$ and $n = 3$.

Representation (9.32) allows us to consider a Monte Carlo algorithm, called *spherical process* for computing the inner product $(g(x), u(x))$ (see, (9.15)). The algorithm is as follows.

Algorithm 9.5.

- **(i) Choose** an ε-strip $\partial\Omega_\varepsilon$ of the boundary $\partial\Omega$ and suppose that the solution of problem (9.13) ,(9.14) is known in $\partial\Omega_\varepsilon$.
- **(ii)** The starting point x_0 is to be **chosen** with respect to initial density $p(x)$ which is permissible for the function $g(x)$.
- **(iii)** The next point is **chosen** to be equally distributed on the maximum sphere in Ω with the center in x_0. The formula for computing x_k from x_{k-1} is

$$x_k = x_{k-1} + \omega_n d(x_{k-1}), \quad k = 1, 2, \ldots, \tag{9.33}$$

where ω is a unit vector uniformly distributed on the unit sphere. Then, **if** $x_k \in \Omega \setminus \partial\Omega_\varepsilon$, the process **continues** by determining the next point from (9.33), **if** $x_k \in \partial\Omega_\varepsilon$, the process is **terminated** with setting up $\theta_k = u(x_k)$.

The value θ is the k^{th} realization of the r.v. θ. Then, using N such realizations, we consider the approximation of the solution

$$(g, u) \approx \frac{1}{N} \sum_{k=1}^{N} \theta_k. \tag{9.34}$$

The number of steps M in a realization of the r.v. θ is

$$M = c|\log \varepsilon|, \tag{9.35}$$

where c depends on the boundary $\partial\Omega$.

To ensure a probable error ε, it is necessary to perform N realizations of the r.v. where N is chosen from (9.24). Then for the expectation of the number of steps we obtain

$$R = c(c_{0.5}\sigma(\theta))^2 \frac{|\log \varepsilon|}{\varepsilon^2}, \tag{9.36}$$

and each step consists of γ_A arithmetic and γ_L logical operations for finding $d(x), 2\varkappa$ arithmetic operations for generating two pseudo-random numbers (when $d = 3$), q_A arithmetic operations for calculation of the coordinates of the next point and 1 logical operation to determine whether $x \in \partial\Omega_\varepsilon$.

It then follows that

$$ET_1(\mathcal{C}) = \tau \left[(2\varkappa + \gamma_A + q_A)l_A + (\gamma_L + 1)l_A\right] c(c_{0.5}\sigma(\theta))^2 \frac{|\log \varepsilon|}{\varepsilon^2}, \tag{9.37}$$

whereas for the third model with $p = (c_{0.5}\sigma(\theta)/\varepsilon)^2$ processor it follows that

$$ET_p(\mathcal{C}) = \tau \left((2\varkappa + \gamma_A + q_A)l_A + (\gamma_L + 1)l_L\right) c|\log \varepsilon|, \tag{9.38}$$

and so for the speed-up and parallel efficiency we obtain

$$S_p(\mathcal{C}) = p = O(\varepsilon^{-2}) \tag{9.39}$$

and

$$E_p(\mathcal{C}) = 1, \tag{9.40}$$

If the fourth model is considered, a greater speed-up can be achieved:

$$S_{6p}(\mathcal{C}) = p \frac{\left(1 + \frac{2\varkappa+\gamma_A+q_A}{\gamma_L+1} \frac{l_A}{l_L}\right)}{\left(1 + \frac{\varkappa+\gamma_A+q_A}{\gamma_L+1} \frac{l_A}{l_L}\right)} > S_p(\mathcal{C}), \tag{9.41}$$

but the parallel efficiency decreases:

$$E_{6p}(\mathcal{A}) = \frac{1}{6}\frac{\left(1+\frac{2\varkappa+\gamma_L+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right)}{\left(1+\frac{\varkappa+\gamma_L+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right)} < 1. \tag{9.42}$$

In case of the pipeline model with l_A segments, the respective results are

$$ET_{pipe}(\mathcal{C}) = \tau\left[s+2\varkappa+\gamma_A+q_A+l_A-1+(\gamma_L+1)l_L\right] \times c(c_{0.5}\sigma(\theta))^2\frac{|\log\varepsilon|}{\varepsilon^2}, \tag{9.43}$$

and

$$S_{pipe}(\mathcal{C}) = \frac{\left(1+\frac{2\varkappa+\gamma_A+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right)}{\left(1+\frac{s+2\varkappa+\gamma_A+q_A+l_A-1}{\gamma_L+1}\frac{1}{l_L}\right)}, \tag{9.44}$$

and $S_{pipe}(\mathcal{C}) > 1$ when the inequality

$$s < (2\varkappa+\gamma_A+q_A-1)(l_A-1) \tag{9.45}$$

holds. But it is met for the real computers where a speed-up greater than one could be obtained. This effect is known as *superlinear speed-up*.

All results concerning algorithm $\mathcal{A}, \mathcal{B}$ and $\mathcal{C}$ in the case $d=3, f(x)\equiv 0$ and $p=(c_{0.5}\sigma(\theta)/\varepsilon)^2$ are enumerated in the appendix.

9.3.4 *Discussion*

One can see that the rates of increase of the time required to achieve a present probable error ε using algorithm $\mathcal{A}$, $\mathcal{B}$ and $\mathcal{C}$ are $O(1/\varepsilon)$, $O(1)$ and $O(|\log\varepsilon|)$, respectively. Thus, algorithm $\mathcal{B}$ is faster than algorithms $\mathcal{A}$ and $\mathcal{C}$.

On the other hand, it is obvious that the speed-up and the parallel efficiency coefficients are greater for MIMD architectures (models (3) and (4)) that those for the pipeline architecture (model (2)). That is so, because the increase of the number of processing elements in models (3) and (4) involves an increase of the speed-up with the same factor, while the parallel efficiency remains constant. The formulae in Appendix B show that third model is the best one for such algorithms, because their parallel efficiency is unity, i.e., the idle time of its processing elements is zero.

The Monte Carlo algorithms described include a wide range of problems with probable error of the superconvergent type $r_N = cN^{-1/2-\psi(d)}$, where $\psi(d) > 0$ is a nonnegative function of dimensionality of the problem. Such

type of Monte Carlo algorithms are considered in Sections 2.4 and 2.5, as well as in Chapter 3 for calculation of integrals of smooth functions from $\mathbf{W}^k(\|f\|;\Omega)$ with partially continuous and limited derivatives. Methods of this type are called *superconvergent* and can be applied for Fredholm integral equations of second kind (see [Dimov and Tonev (1989)]). The main idea of the *superconvergent* Monte Carlo methods is that the random points are generated in different subdomains Ω_j. This allows to obtain a greater rate of convergence with $\psi(d) > 0$ (for $\mathbf{W}^1(\|f\|;\Omega)$, $\psi(d) = 1/d$; see Section 2.4).

The algorithm results from the special way of separation of the domain Ω. In this case every Markov chain is defined in one of the subdomains Ω_j. This means that the only difference between the algorithm described above and this algorithms is the probability density function which does not change the speed-up and efficiency estimators.

Here,

- an optimum (with a minimum standard deviation) algorithm for BVP problem, is considered. This algorithm has a minimum probable error.
- It is obtained using the idea of the importance sampling technique for defining the transition probabilities in the Markov chain.
- As the only difference between this algorithm and the classical one is the probability density function, it is obvious that the estimates for the speed-up and the efficiency are the same.
- However, the Monte Carlo algorithms described above are not well suited for pipeline architectures. Thus, if one wants to achieve greater speed-up on pipeline computers, it is necessary to construct special vector Monte Carlo algorithms.

9.3.5 *Vector Monte Carlo Algorithms*

There are Monte Carlo algorithms called *vector algorithms* that are more efficient for pipeline computers.

Let us now consider an approach to solve the problem

$$R_m(\Delta)u(x) = (-1)^m f(x), \quad x \in \Omega \subset I\!R^2, \tag{9.46}$$

$$\Delta^k u(x) \to f_{km}(x), \quad x \to x_0, \quad x_0 \in \partial\Omega, \tag{9.47}$$

where

$$R_m(\lambda) = \lambda^m + c_1\lambda^{m-1} + \cdots + c_{m-1}\lambda + c_m \tag{9.48}$$

is a polynomial with real zeros and Δ is Laplace operator. This is a common problem in the investigation of the beam penetration in multilayer targets.

Suppose that all conditions ensuring the existence and the uniqueness of the solution of problem (9.46), (9.47) are met and all zeros of the polynomial $R_m(\lambda)$ are real numbers. Let $\lambda_1, \ldots, \lambda_m$ be the zeros of $R_m(\lambda)$.

Then the problem (9.46), (9.47), (9.48) can be given the following representation:

$$\begin{cases} (\Delta - \lambda_1)u_1 = (-1)^m f(x), \\ (\Delta - \lambda_2)u_2 = u_1, \\ \qquad\qquad \vdots \\ (\Delta - \lambda_m)u_m = u_{m-1}, \\ \qquad\quad u_m = u, \end{cases}$$

with the boundary conditions

$$u_s(x) \to f_s(x) + b_1^s f_{s-1}(x) + \cdots + b_{s-1}^s f_1 = W_s(x), \quad s = 1, \ldots, m,$$

where

$$\begin{cases} b_j^s = b_j^{s+1} + \lambda_s b_{j-1}^s, \ j = 1, \ldots, s-1, \\ b_0^s = 1, \quad b_k^{m+1} = c_k, \ k = 1, \ldots, m-1. \end{cases}$$

Then the discrete problem becomes

$$u_{k,0} = \frac{1}{4}\sum_{i=1}^{4} u_{k,i} - \frac{1}{4}h^2\lambda_k u_{k,0} - \frac{1}{4}h^2 u_{k-1,0}, \quad k = 1, \ldots, m,$$

$$u_{0,0} = f_0 = (-1)^m f(x), \quad x \in \Omega_h.$$

Using the notation $k_i = (1 - \frac{1}{4}h^2\lambda_i)^{-1}, \quad i = 1, \ldots, m$, one can obtain the following system of linear algebraic equations:

$$\begin{cases} u_{m,0} = k_m \left(\frac{1}{4}\sum_{k=0}^{4} u_{m,i} - \frac{1}{4}h^2 u_{m-1,0} \right), \\ u_{m-1,0} = k_{m-1} \left(\frac{1}{4}\sum_{i=0}^{4} u_{m-1,i} - \frac{1}{4}h^2 u_{m-2,0} \right), \\ \qquad \vdots \\ u_{1,0} = k_1 \left(\frac{1}{4}\sum_{i=0}^{4} u_{1,i} - \frac{1}{4}h^2 f_0 \right), i = 1, \ldots, m, \end{cases}$$

or represented in a matrix form:

$$\mathbf{U} + A\mathbf{U} + \mathbf{F}, \tag{9.49}$$

where $\mathbf{U} = (u_{m,0}, u_{m-1,0}, \ldots, u_{1,0})^T$ and each $u_{k,0}$ is a vector with m_1, m_2 components (here m_1 and m_2 are the number of points along x_1 and x_2, respectively),

$$A = \begin{pmatrix} k_m L_h & -\frac{1}{4}h^2 k_m k_{m-1} L_h & \cdots & \left(\frac{1}{4}h^2\right)^{m-1} \prod_{i=1}^{m} k_i L_h \\ 0 & k_{m-1} L_h & \cdots & \left(-\frac{1}{4}h^2\right)^{m-2} \prod_{i=1}^{m-1} k_i L_h \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & k_1 L_h \end{pmatrix},$$

$$\mathbf{F} = \left(\prod_{i=1}^{m} k_i \left(-\frac{1}{4}h^2\right)^m, \ldots, \left(-\frac{1}{4}h^2\right)^2 k_2 k_1, \left(-\frac{1}{4}h^2\right)^2 k_1, \right)^T (-1)^m f(x).$$

In what follows we will denote by $\mathbf{W} = (w_n(x), \ldots, w_1(x))^T$ the boundary conditions vector.

This interpretation allows to formulate a Monte Carlo vector algorithm. We introduce matrix weights in the following manner:

$$Q_0 = \{\delta_{ij}\}_{i.j=1}^{m}, \quad Q_j = Q_{j-1} \frac{k(x_{j-1}, x_j)}{p(x_{j-1}, x_j)},$$

where

$$k(x_{n-1}, x_n) = \prod_{l=i}^{j} k_{m+1-l} \left(-\frac{1}{4}h^2\right)^{l-i} L_h, \quad i, j = 1, \ldots, m.$$

Then the r.v. ξ_x with $E\xi_x = J(\mathbf{U})$ is

$$\xi_x = \sum_{j=0}^{i^*} Q_j f(x_j),$$

where i^* is a r.v. that is the length of the Markov chain, and $J(\mathbf{U})$ is a linear functional of the solution $\mathbf{U}$. Each Markov chain is a realization of the r.v. ξ_x. From the law of large numbers it follows that we need N realizations to estimate the mathematical expectation of ξ_x, i.e., the linear functional of the solution $\mathbf{U}$.

Each realization is done according to the following algorithm.

Algorithm 9.6. Vector Monte Carlo Algorithm

(i) The process begins at the point x_0, chosen according to the initial density, and accumulate in ξ_0 the values of $\mathbf{F}_0$.

(ii) From the point x_k the process goes to the next point x_l depending on the transition densities p_{kl} and

- if $x \in \Omega_h$, then $\xi_0 = \xi_0 + Q_1\mathbf{F}_l$ and the process continues with a transition to the next point (according to step (ii));
- if $x \in \partial\Omega_h$, then $\xi_0 = \xi_0 + Q_1\mathbf{W}_l$ and the process is terminated;
- if the boundary $\partial\Omega_h$ is reached at the point x_{i^*} and on the s^{th} step, then

$$\xi_0 = \xi_0 + Q_s\mathbf{W}_{i^*}.$$

This algorithm is used to solve the following practical computational problem:

$$\Delta\Delta\Delta u(x) = (-1)^3 f(x), \quad x \in \Omega, \quad x = (x_1, x_2),$$

$$\Delta^k u(x) \to f_{k+1}(x), \quad \text{when} \quad x \to x_0, \quad x_0 \in \partial\Omega, \ k = 0, 1, 2,$$

where $\Omega = \{(x_{(1)}, x_{(2)}) : 0 \le x_{(1)} \le 1, \ 0 \le x_{(2)} \le 1\}$ is the unit square, and

$$\begin{cases} f(x) = 2 \cdot 3^6 \sin 3x_{(1)} + 2^6 \cos 2x_{(2)}, \\ f_1(x) = 2 \sin 3x_{(1)} + \cos 2x_{(2)}, \\ f_2(x) = -18 \sin 3x_{(1)} - 4 \cos 2x_{(2)}, \\ f_3(x) = 2 \cdot 3^4 \sin 3x_{(1)} + 2^4 \cos 2x_{(2)}. \end{cases}$$

The corresponding system of differential equations is

$$\begin{cases} \Delta u_1 = (-1)^3 f(x), \\ \Delta u_2 = u_1, \\ \Delta u_3 = u_2, \\ u_3 = u, \end{cases}$$

and using a grid of mesh size $h = 0.1$, we obtain the finite difference system

$$\mathbf{u} = A\mathbf{u} + \mathbf{F},$$

where

$$\mathbf{u} = (u_{3,0}, u_{2,0}, u_{1,0})^T,$$

$$A = \begin{pmatrix} L_h & -\frac{1}{4}h^2 L_h & (-\frac{1}{4}h^2)^2 L_h \\ 0 & L_h & -\frac{1}{4}h^2 L_h \\ 0 & 0 & L_h \end{pmatrix},$$

Table 9.1 Solution of the problem. The number of Markov chains is $N = 3000$

Coordinates of points	Exact solution	Monte Carlo solution	Relative error
$(0.6, 0.8)$	2.6444	2.6790	−0.013
$(0.2, 0.5)$	1.6696	1.6895	−0.012

$$\mathbf{F} = \left((-\frac{1}{4}h^2)^3, (-\frac{1}{4}h^2)^2, (-\frac{1}{4}h^2)\right)^T (-1)^3 f(x),$$

and the boundary conditions vector is

$$\mathbf{F} = (f_3(x), f_2(x), f_1(x))^T .$$

The random vector ξ_k is

$$\xi_k = \xi_k + Q_s \Phi_s,$$

where

$$Q_s \Phi_l = \begin{cases} Q_s \mathbf{F}_l, & x_l \in \Omega_h, \\ Q_s \mathbf{W}_l, & x_l \in \partial\Omega_h, \end{cases}$$

$$Q_s \mathbf{F}_l = \begin{pmatrix} R(s+1)\left(-\frac{1}{4}h^2\right)^3 \\ (s+1)\left(-\frac{1}{4}h^2\right)^2 \\ -\frac{1}{4}h^2 \end{pmatrix} (-1)^3 f(x),$$

$$Q_s \mathbf{W}_l = \begin{pmatrix} f_3 + s\left(-\frac{1}{4}h^2\right) f_2 + R(s)\left(-\frac{1}{4}h^2\right)^2 f_1 \\ f_2 + s\left(-\frac{1}{4}h^2\right) f_1 \\ f_1 \end{pmatrix},$$

where $R(1) = 1, R(s) = s + R(s-1)$.

In Table 9.1 the results from the solution of the problem in two points over $N = 3000$ realizations of the r.v. are shown. One can see that the relative error of MC result obtained using the vector algorithm is around 1% at the points chosen with coordinates $(0.6, 0.8)$ and $(0.2, 0.5)$.

Chapter 10

Applications for Transport Modeling in Semiconductors and Nanowires

There are many important applications of Monte Carlo methods in statistical physics, finances, environmental sciences, biology, medicine and many other fields. In this chapter algorithms for modeling of transport phenomena in semiconductors and nanowires are presented. This choice is made because in this field a very significant improvement was made by deploying the numerical Monte Carlo approach during the last decade. The basic algorithms for simulation of the semiclassical Boltzmann equation (BE) are discussed from the point of view of the numerical Monte Carlo method. Such an approach is a base for developing models for solving equations generalizing the BE towards quantum transport. It is further used for solving purely quantum problems, e.g. the Wigner equation. These cases are considered below. Convergency proof for each corresponding algorithm is presented. A number of numerical experiments are performed. A Grid application to modeling of carrier transport in nanowires is presented. Numerical results based on this grid application are discussed.

10.1 The Boltzmann Transport

The history of the MC application in the semiconductor physics probably begins in 1966 [Kurosawa (1966)]. Since then, a large variety of physical systems and phenomena have been investigated, accounting for all types of interactions with the lattice imperfections (phonons, impurities) and other carriers, full band structure, the effect of the exclusion principle or to analyze space and/or time dependent phenomena [Jacoboni and Reggiani (1983); Jacoboni and Lugli (1989)]. Different MC algorithms have been created to meet the challenge of the concrete physical tasks. The problem of their convergency, despite demonstrated so widely in practice (the suc-

cessive completion of any MC code), was not discussed in the literature. The variety of the semiclassical MC algorithms can be summarized in three main groups. The algorithms of the first one simulate the natural chain of events happening during the physical process of the particle transport. The algorithms belonging to the other two generate the particle history back in time or modify the weight of the elementary events, thus achieving variance reduction in the desired regions of the phase space.

The general space-time dependent BE and the algorithms from the three main groups - Ensemble MC [Phillips and Price (1977)], MC Backwards [Jacoboni *et al.* (1988)] and Weighted Ensemble MC [Rota *et al.* (1989)] will be introduced. It will be shown that they are generalized by the numerical Monte Carlo algorithm applied to appropriate integral form of the BE.

The convergency of the particular technique follows from the convergency of its Neumann series.

The understanding of the MC algorithms for solving the BE is essentially assisted by consideration of the concrete process of the semiconductor carrier transport.

Within the semiclassical transport concepts the point-like particles move in the six dimensional phase space $\mathbf{R} \times \mathbf{K}$. The distribution function $f(\mathbf{r}, \mathbf{k}, \mathbf{t})$ gives the particle number density in the phase space points of position and wave vector (the set $\mathbf{K}$ of all elements $\mathbf{k}$ is an Euclidean space).

The particles move along Newton trajectories, defined by:

$$\dot{\mathbf{k}} = e\mathbf{E}/\hbar \qquad \dot{\mathbf{r}} = \mathbf{v} \qquad \mathbf{v} = \frac{\mathbf{1}}{\hbar}\nabla_{\mathbf{k}}\varepsilon(\mathbf{k}), \tag{10.1}$$

which is called drift process. Here $\mathbf{E}(\mathbf{r})$ is the electric field, $\mathbf{v}$ is the electron velocity and $\varepsilon(\mathbf{k})$ is the electron energy. The drift is interrupted by scattering events due to the lattice imperfections and are considered local in position and instantaneous in time. They are accounted for by a function $S(\mathbf{r}, \mathbf{k}, \mathbf{k}')d\mathbf{k}'$ giving scattering frequency from $\mathbf{k}$ to $d\mathbf{k}'$ around $\mathbf{k}'$. S is calculated within a quantum mechanical description of the considered electron-imperfection interaction.

The transport process is presented by the following chain of events: The electron is accelerated by the field along the Newton trajectory. When scattering occurs it disappears in state $\mathbf{r}, \mathbf{k}$ and in the same moment appears in the state $\mathbf{r}, \mathbf{k}'$, which is an initial point of a new trajectory. This trajectory is in turn interrupted by a scattering event, etc.

The common integro-differential form of the BE is:

$$\left(\frac{\partial}{\partial t} + e\mathbf{E}/\hbar.\nabla_k + \mathbf{v}(\mathbf{k}).\nabla_r\right) f(\mathbf{r}, \mathbf{k}, t)$$

$$= \int d\mathbf{k}' S(\mathbf{r}, \mathbf{k}', \mathbf{k}) f(\mathbf{r}, \mathbf{k}', t) - \lambda(\mathbf{r}, \mathbf{k}) f(\mathbf{r}, \mathbf{k}, t), \tag{10.2}$$

where $\lambda(\mathbf{r}, \mathbf{k}) = \int d\mathbf{k}' S(\mathbf{r}, \mathbf{k}, \mathbf{k}')$. It is accomplished by an initial condition, giving the particle distribution at time zero. As a rule we are interested in the solution in a confined space domain and boundary conditions are also present. We will use the fact that the space domain is confined, but in order to keep the attention to the algorithmic aspect of the problem, boundary conditions will not be considered here. We also mention that, because of the periodicity of the carrier properties with respect to the wave vector, the $\mathbf{K}$ space is confined in a domain, called first Brillouin zone.

There are two main approaches for understanding the relations between the BE and the MC: understanding the MC algorithm in terms of the transport events (leading to the BE) and understanding the transport phenomena in terms of numerical MC, applied to the BE. The first one introduces the imitation technique EMC.

The EMC is based on the simulation of the natural processes of drift and scattering events, existing in the *real world* of the semiconductor electrons. A given simulation electron is followed through random generation of the probabilities for the drift duration and the after scattering state. They are easily obtained from the λ and S which participate also as coefficients in the Boltzmann equation. For example the probability $w(\mathbf{r}(t), \mathbf{k}(t))$ for drift without scattering during the time interval t is obtained as follows. If n particles are located in $\mathbf{r}, \mathbf{k}$ at time 0, after time interval t their number in point $\mathbf{r}(t), \mathbf{k}(t)$ will decrease to $n.w(\mathbf{r}(t)\mathbf{k}(t))$. The following relation holds:

$$n.w(\mathbf{r}(t+dt), \mathbf{k}(t+dt)) = n.w(\mathbf{r}(t), \mathbf{k}(t)) - n.w(\mathbf{r}(t), \mathbf{k}(t))\lambda(\mathbf{r}(t), \mathbf{k}(t))dt.$$

So for w the expression $w(\mathbf{r}(t), \mathbf{k}(t)) = e^{-\int_0^t \lambda(\mathbf{r}(\tau), \mathbf{k}(\tau))d\tau}$ is obtained. Another probability which will be often used in this chapter is given by the product $\lambda(\mathbf{r}(t), \mathbf{k}(t))w(\mathbf{r}(t), \mathbf{k}(t))dt$ - it is the probability for scattering in time interval dt after a successive drift for time $0 - t$.

The motions of a number of particles with a given initial distribution are simulated in the phase space. The distribution function in a given phase space point $\mathbf{r}, \mathbf{k}$ at given moment t is proportional to the relative number of simulated electrons located in a unit volume about the point at that moment. During such imitation of the elementary transport processes the particle number balance, leading to the BE, is performed automatically. In this sense the imitation MC is another way of stating the BE.

The second approach introduces the techniques MCB, WEMC and the IA. MCB and WEMC are still within the concept of drift and scattering events, but in a more sophisticated way. They are oriented towards the

mathematical aspects of the BE rather than to the real transport processes. MCB inverses the order of the real evolution of the carrier system - of both the drift and the scattering processes. The simulated trajectories begin in a given point $\mathbf{r}, \mathbf{k}$ and time t, evolve back in time and finish in time 0 when the initial distribution is accounted. The gain is that we can obtain $f(\mathbf{r}, \mathbf{k}, t)$ in domains, practically unattainable by the EMC. The WEMC, as the EMC, follows the real evolution, but modifies the corresponding probabilities for drift and scattering. Thus rare transport events are generated more frequently. After each simulated elementary event, a weight given by the ratio between the real and modified probabilities is calculated. The product of the weights is then used to recover the real carrier distribution.

10.1.1 *Numerical Monte Carlo Approach*

Equation (10.2) can be transformed into an integral form [Vitanov *et al.* (1994)]:

$$\begin{aligned} f(\mathbf{r}, \mathbf{k}, t) = & \int_0^t dt' \int d\mathbf{k}' f(\mathbf{r}(t'), \mathbf{k}', t') S(\mathbf{r}(t'), \mathbf{k}', \mathbf{k}(t')) \\ & \times \exp\left\{ - \int_{t'}^t \lambda(\mathbf{r}(y), \mathbf{k}(y)) dy \right\} \\ & + \exp\left\{ - \int_0^t \lambda(\mathbf{r}(y), \mathbf{k}(y)) dy \right\} f\left(\mathbf{r}(0), \mathbf{k}(0), 0\right), \end{aligned} \tag{10.3}$$

where now $\mathbf{r}(\tau)$, $\mathbf{k}(\tau)$ are the Newton coordinates at time τ which lead to $\mathbf{r}$, $\mathbf{k}$ at time t. We introduce the short notations: $x = (\mathbf{r}, \mathbf{k}, t)$ and $x' = (\mathbf{r}', \mathbf{k}', t')$. Formally by introducing $\delta(\mathbf{r}' - \mathbf{r}(t'))$ together with an integration over $\mathbf{r}'$, the equation can be written as: $f(x) = \int \mathcal{K}(x, x') f(x') dx' + f_\circ(x)$.

We choose the following two MC estimators ν_i^f and ν_i^b :

$$\nu_i^f = \frac{f_\circ(x_0)}{p(x_0)} \frac{\mathcal{K}(x_1, x_0)}{p(x_0, x_1)} \cdots \frac{\mathcal{K}(x_i, x_{i-1})}{p(x_{i-1}, x_i)} \delta(x - x_i), \tag{10.4}$$

$$\nu_i^b = \delta(x - x_0) \frac{\mathcal{K}(x_0, x_1)}{p(x_0, x_1)} \cdots \frac{\mathcal{K}(x_{i-1}, x_i)}{p(x_{i-1}, x_i)} f_\circ(x_i), \tag{10.5}$$

which reflect generation of trajectories forward – from the initial to the final distribution – and backward – from fixed point x back to the initial distribution. Depending on the choice of $p(x)$ and $p(x, x')$ – the initial and the transition probability densities there are numerous ways for obtaining the distribution function.

Special choices of the probability densities should provide the EMC, WEMC and MCB algorithms. They are recovered if for $p(x, x')$ we combine the heuristic functions for trajectory construction with the normalization requirement: $\int p(x', x'')dx'' = 1, \quad \forall x'$.

For obtaining the p_{EMC} we consider the expression [Vitanov and Nedjalkov (1991)]:

$$\frac{S(\mathbf{r}', \mathbf{k}', \mathbf{k}'')}{\lambda(\mathbf{r}', \mathbf{k}')}\lambda(\mathbf{r}'(t''), \mathbf{k}''(t''))e^{-\int_{t'}^{t''}\lambda(\mathbf{r}'(y),\mathbf{k}''(y))dy}. \tag{10.6}$$

It is the product of the probability $P_1(A)$ for event $A =$ scattering from a given point $\mathbf{r}', \mathbf{k}', t'$ to $\mathbf{r}', \mathbf{k}''(t'), t'$ and the probability $P_2(B/A)$ for event $B/A =$ drift until the moment t'' (in which the next scattering will occur) in the condition that event A has taken place. Hence (10.6) provides the real transport transition from $x' = (\mathbf{r}', \mathbf{k}', t')$ to $x'' = (\mathbf{r}'(t''), \mathbf{k}''(t''), t'')$. Comparison with the kernel in (10.3) shows that the expression (10.6) is equal to $\frac{\lambda(x'')\mathcal{K}(x'',x')}{\lambda(x')}$. This expression should be used for times t'' lower that the time of interest t. The inequality $t'' > t$ means that the time boundary is reached so that a new trajectory should be initiated. The total contribution of such events is $\frac{\mathcal{K}(x'',x')}{\lambda(x')}$. Finally for the EMC transition density the following expression is obtained:

$$p_{EMC}(x', x'') = \lambda(x'')\frac{\mathcal{K}(x'', x')}{\lambda(x')} + \frac{\mathcal{K}(x'', x')}{\lambda(x')}\delta(t'' - t).$$

Direct calculations show that the normalization condition is satisfied for any x'. Accordingly

$$P_{EMC}(x_0) = \lambda(x_0)f_\circ(x_0) + f_\circ(x_0)\delta(t_0 - t).$$

When P_{EMC} and p_{EMC} are replaced in (10.4), exact cancelations occur so that always $\nu^f = \delta(x - x_i)$, independently on i. Thus $f(x)dx$ is given by the relative number of simulation trajectories which start from the initial distribution and stop in dx around x.

It is seen that the particularly chosen probability densities reproduce the EMC algorithm. A generalization to WEMC is straightforward: weights due to the ratio between p_{WEMC} and p_{EMC} will appear in the estimator ν^f; thus providing the proper $f(x)$ value despite the deviation of the simulated trajectories from the real ones.

The MCB algorithm is reproduced by the following transition density:

$$p_{MCB}(x', x'') = \lambda(\mathbf{r}'(t''), \mathbf{k}'(t''))e^{-\int_{t''}^{t'}\lambda(\mathbf{r}'(y),\mathbf{k}'(y))dy}\frac{S(\mathbf{r}'(t''),\mathbf{k}'',\mathbf{k}'(t''))}{\lambda^*(\mathbf{r}'(t''),\mathbf{k}'(t''))}$$
$$+ \qquad e^{-\int_0^{t'}\lambda(\mathbf{r}'(y),\mathbf{k}'(y))dy}\delta(t'')\delta(k'' - k'(t'')).$$

In terms of conditional probabilities, A corresponds to a drift from $x' = (\mathbf{r}', \mathbf{k}', t')$ to $x^+ = (\mathbf{r}'(t''), \mathbf{k}'(t''), t'')$ back in the time. B/A is a scattering from x^+ to $x'' = (\mathbf{r}'(t''), \mathbf{k}'', t'')$ in direction, opposite to the real process. The denominator $\lambda^*(\mathbf{r}, \mathbf{k}') = \int S(\mathbf{r}, \mathbf{k}, \mathbf{k}')d\mathbf{k}$ ensures the proper normalization of the *scattering term*. This form of the backwards transition density is chosen from purely physical considerations: if p_{MCB} reflects the natural sequence of events in the opposite direction, the terms giving the maximum contribution to f will be simulated most often.

Direct calculations show that the x'' integral of p_{MCB} is unity for any x'. Then the backward estimator ν^b appears to be a product of $\frac{\lambda^*}{\lambda}$ in the intermediate points x_j^+ obtained in the process of simulation. In this way we established the generalization of the phenomenological MC algorithms by the numerical MC algorithms for solving integral equations. The successful application of the latter ones requires convergence of the Neumann series of the BE. Its proof ensures the convergency of the EMC, WEMC and MCB algorithms.

10.1.2 *Convergency Proof*

To investigate the Neumann series convergency of equation (10.3) we can exploit the fact that the integral form of the BE is of Volterra type with respect to the time variable. Following [Nedjalkov *et al.* (1996)], a similar to the Wigner equation estimate is expected: the i^{th} term of the Neumann series is less than the i^{th} term of the Ae^{Mt} series; the constants A and M depend on the concrete kernel. From the other side, the heuristic MC algorithms prompt that there should be a more precise estimation, for example since the contribution of the forward estimator of any iteration order is always less than unity. Our proof is based on the forward model. The following considerations are also necessary: In accordance with the semiconductor physics, λ is a nonnegative, continuous function. Owing to the limited phase space domain of interest, λ reaches its maximum, M, inside the domain. The phase space subdomains where $\lambda = 0$ can be enclosed by surfaces, inside which the solution is known. It is provided either by the homogeneous form of equation (10.2), or by the accounting of the generation caused by the scattering processes. In the first case the solution is analytical while in the second the BE must be coupled with the BE's in the neighborhood domains responsible for the generation term. The effect of such subdomains is to provide additional boundary conditions on the surfaces. This ensures $\lambda > 0$ in the main domain of interest.

Now let us consider an equation, conjugate to (10.3):

$$\phi(\mathbf{r},\mathbf{k},t') = \int_{t'}^{t} dt'' \int d\mathbf{k}' \int d\mathbf{r} S(\mathbf{r},\mathbf{k},\mathbf{k}') e^{-\int_{t'}^{t''} \lambda(\mathbf{r}(y),\mathbf{k}'(y))dy} \phi(\mathbf{r}(t''),\mathbf{k}'(t''),t'') + \delta(\mathbf{r}-\mathbf{r}_1)\delta(\mathbf{k}-\mathbf{k}_1)\delta(t'-t). \tag{10.7}$$

It can be obtained that f is given by the expression:

$$f(\mathbf{r}_1,\mathbf{k}_1,t) = \int_0^t dt' \int d\mathbf{k} \int d\mathbf{r} f(\mathbf{r},\mathbf{k},0) \times \exp\{-\int_0^{t'} \lambda(\mathbf{r}(y),\mathbf{k}(y))dy\} \phi(\mathbf{r}(t'),\mathbf{k}(t'),t'). \tag{10.8}$$

Another way to see this is to replace the iterations of ϕ in equation (10.8). Such a procedure recovers the numerator in the forward estimator (10.4). An equivalent to equation (10.7) equation is obtained by dividing it by λ:

$$\Phi(\mathbf{r},\mathbf{k},t') = \int_{t'}^{t} dt'' \int d\mathbf{k}' \int d\mathbf{r} \frac{S(\mathbf{r},\mathbf{k},\mathbf{k}')}{\lambda(\mathbf{r},\mathbf{k})} \times \lambda(\mathbf{r}(t''),\mathbf{k}'(t''))e^{-\int_{t'}^{t''} \lambda(\mathbf{r}(y),\mathbf{k}'(y))dy} \Phi(\mathbf{r}(t''),\mathbf{k}'(t''),t'') + \frac{\delta(\mathbf{r}-\mathbf{r}_1)\delta(\mathbf{k}-\mathbf{k}_1)\delta(t-t')}{\lambda(\mathbf{r},\mathbf{k})}. \tag{10.9}$$

Here $\Phi = \phi/\lambda$. It is interesting to note that by replacing its iterations in (10.8) we recover the denominator in the forward estimator.

The convergency proof is based on the form of the kernel in (10.9).

The t'' integral of the $\lambda e^{\cdots}$ term gives

$$\int_{t'}^{t} dt'' \lambda(\mathbf{r}(t''),\mathbf{k}'(t''))e^{-\int_{t'}^{t''} \lambda(\mathbf{r}(y),\mathbf{k}'(y))dy} = 1 - e^{-\int_{t'}^{t} \lambda(\mathbf{r}(y),\mathbf{k}'(y))dy}.$$

The value of this expression is less than the value of $B = 1 - e^{-Mt}$ hence the integral of the kernel with respect to t'' and $\mathbf{k}'$ is smaller than the constant B. This result allows to conclude that the Neumann series converges as a geometric progression.

10.1.3 *Error Analysis and Algorithmic Complexity*

Let us first consider an error analysis. Using N realizations of the r.v., we construct estimates of the following form:

$$f_i(x) \simeq \frac{1}{N} \sum_s \nu_{i,s}.$$

For the estimated functional $Y = f_i(x)$ and for the r.v. $f_{i,N}$ it follows that:

$$Ef_{i,N} = Y, \qquad Df_{i,N} = \frac{D\nu}{N} = \delta_N^2.$$

Since finite number of moves in every Markov chain is used, there is a truncation error of order ε, that is:

$$|E\nu_\varepsilon - Y|^2 \leq c\varepsilon^2 = \delta_s^2,$$

where c is a constant and ν_ε is the biased estimate of ν.

Now consider a family of estimates ν_ε. For balancing of both systematic and stochastic errors, the following equality has to be fulfilled:

$$\frac{D\nu_\varepsilon}{N} = c\varepsilon^2.$$

The systematic error δ depends on the norm of the integral operator:

$$\|K\| = vrai \sup_x \int |K(x, x')|dx'$$

and on the length, i, of the Markov chains defined by (10.4) or (10.5). Let

$$\delta = \delta_N + \delta_s$$

be the given error, fixed in advance. This means that the systematic error is also fixed, because we would like to have a good balancing of errors, i.e.

$$\delta_s(\|K\|, i) = \frac{1}{2}\delta.$$

Since the norm $\|K\|$ can be calculated, the average length of the Markov chain could be estimated. Let $t(i)$ be the average complexity of the realization of the r.v. ν. Obviously $t(i) = O(i)$. Suppose $t(i) = c_1 i$ (where c_1 is a constant). Now the computational complexity C_N (see Definition 1.9 in Introduction) can be estimated as:

$$C_N = t(i).N.$$

Using the equality

$$\frac{D\nu}{N} + c\varepsilon^2(\|K\|, i) = \delta^2$$

and the condition $c\varepsilon^2(\|K\|, i) < \delta^2$, one can obtain:

$$C_N = \frac{c_1 i D\nu}{\delta^2 - c\varepsilon^2(\|K\|, i)}.$$

For any given integral operator K (respectively $\|K\|$) and for any fixed error δ, there is an optimal average length i of the Markov chain, which can be found from the following equality:

$$c_1 D\nu(\delta^2 - c\varepsilon^2) = -cc_1 i D\nu \frac{\partial}{\partial i}[\varepsilon^2(\|K\|, i)].$$

The last condition could be used to choose the optimal value of i.

10.2 The Quantum Kinetic Equation

The development of the device physics involves in the last years so small space and time scales that the applicability of the semiclassical transport fails and a quantum description is needed. As a rule the equations describing quantum processes do not allow direct probabilistic interpretation. They depend on functions and objects which are related to the physical observable only after a second stage of mathematical processing. This constitutes a limit for the direct application of the semiclassical algorithms to quantum transport processes. For some cases these algorithms can be successively modified [Rossi *et al.* (1994a)]. But the lack of transparency in semiclassical terms of the quantum processes involved makes their application difficult.

Here we consider a Monte Carlo algorithm for space homogeneous, ultrashort time quantum transport. The physical model corresponds to an initially photoexcited electron distribution relaxing due to interaction with phonons in one-band semiconductor. The kinetic equations set for the electron dynamics is obtained in [Schilp *et al.* (1994)].

Under the assumption for equilibrium phonons, from this set an integral equation for the electron distribution function is derived.

A comparison with the Boltzmann equation is done, which provides some general conclusions about the physical aspects of the obtained equation. The main quantum effects originate from the energy-time uncertainty in the electron-phonon interaction and the finite lifetime of the interacting electron eigenstates. They result in a memory character of the obtained equation and collisional broadening of the corresponding semiclassical scattering process. Some of the properties of the solution related to its physical acceptability are discussed too.

It should be noted, that finding a correct initial condition is a quantum mechanical problem by itself. The reason is the finite time period of building up the initial electron distribution by the laser pulse. During this period electron-phonon interaction process takes part, while in the one-band model this period is neglected. The more realistic approach requires to solve the semiconductor Bloch set equations [Haug and Koch (1994)]. Although a simplified version, from physical point of view the one-band model contains important aspects of the quantum kinetic.

The Hamiltonian of a system of electrons interacting with phonons, $H = H_0 + H_{e-p}$ consists of a noninteracting part and a part accounting for

the electron-phonon coupling:

$$H_0 = \sum_k \varepsilon_k c_k^+ c_k + \sum_q \hbar\omega_q b_q^+ b_q, \tag{10.10}$$

$$H_{e-p} = \sum_{k,q} [g_q c_{k+q}^+ b_q c_k + g_q^* c_k^+ b_q^+ c_{k+q}], \tag{10.11}$$

where c_k^+ (c_k) and b_q^+ (b_q) are the electron and phonon creation (annihilation) operators, $\varepsilon_k = \hbar^2 k^2/2m$ is the electron energy, ω_q is the phonon frequency and the coupling g_q is for the case of Froehlich interaction, $\omega_q = \omega$:

$$g_q = -i \left[\frac{2\pi e^2 \hbar\omega}{V q^2} \left(\frac{1}{\varepsilon_\infty} - \frac{1}{\varepsilon_s} \right) \right]^{\frac{1}{2}}.$$

The distribution functions of electrons and phonons are given by the statistical average of the density operators:

$$f_k = \langle c_k^+ c_k \rangle, \qquad n_q = \langle b_q^+ b_q \rangle.$$

Their equations of motion initiate an hierarchy of equations (in some sources called quantum BBGKY hierarchy), where the time derivative of the averaged product of n one-particle operators contains also averaged products of order higher than n. To obtain a closed set of equations the hierarchy is truncated with the help of suitable approximations. Here we adopt the results obtained in [Schilp *et al.* (1994)], leading to the set:

$$\frac{d}{dt} f_k = 2 \sum_q Re[s_{k+q,k} - s_{k,k-q}], \tag{10.12}$$

$$\frac{d}{dt} n_q = 2 \sum_k Re[s_{k+q,k}], \tag{10.13}$$

$$\begin{aligned} \frac{d}{dt} s_{k+q,k} &= (i\Omega_{k+q,k} - \Gamma_{k+q} - \Gamma_k) s_{k+q,k} \\ &\quad + \frac{1}{\hbar^2} \|g_q\|^2 [f_{k+q}(1-f_k)(n_q+1) - f_k(1-f_{k+q}) n_q], \end{aligned} \tag{10.14}$$

where $\Gamma_k = (\Gamma_k^i + \Gamma_k^o)/2$, $\Omega_{k+q,k} = (\varepsilon_{k+q} - \varepsilon_k - \hbar\omega)/\hbar$ and Γ^i, Γ^o are the Boltzmann *in* and *out scattering rates*:

$$\Gamma_k^i = \frac{2\pi}{\hbar} \sum_{q,\pm} \|g_q\|^2 \delta(\varepsilon_{k+q} - \varepsilon_k \pm \hbar\omega) f_{k+q} (n_q + 1/2 \mp 1/2),$$

$$\Gamma_k^o = \frac{2\pi}{\hbar} \sum_{q,\pm} \|g_q\|^2 \delta(\varepsilon_{k+q} - \varepsilon_k \pm \hbar\omega) (1 - f_{k+q}) (n_q + 1/2 \pm 1/2).$$

Equations (10.12)-(10.14) are accomplished with initial conditions $f_k(0) = \phi_k$, $s_{k+q,k}(0) = 0$ (the phonon interaction is switched on at time 0). Within a phonon bath close to equilibrium equation (10.13) is removed and n_q is replaced by the Bose distribution

$$n = \frac{1}{e^{\frac{\hbar\omega}{KT}} - 1}.$$

Further, equation (10.14) has the form

$$\frac{ds(t)}{dt} = A(t)s(t) + B(t)$$

with an analytical solution

$$s(t) = e^{\int_0^t dt' A(t')} s(0) + \int_0^t dt' e^{\int_{t'}^t dt'' A(t'')} B(t').$$

Taking into account the zero initial condition, equation (10.14) obtains the following form:

$$s_{k+q,k}(t) = \frac{1}{\hbar^2}\|g_q\|^2 \int_0^t dt' e^{\int_{t'}^t dt''(i\Omega_{k+q,k} - \Gamma_{k+q} - \Gamma_k)(t'')} \times [f_{k+q}(1 - f_k)(n+1) - f_k(1 - f_{k+q})n]. \tag{10.15}$$

Now s can be replaced in the equation for the electron density, which, after changing q with $-q$ in the second term of the r.h.s. and a pass to integration over q, becomes:

$$\frac{d}{dt} f_k = \int_0^t dt' \int d^3q \{s'(k+q,k,t,t')[f_{k+q}(1 - f_k)(n+1) - f_k(1 - f_{k+q})n] - s'(k,k+q,t,t')[f_k(1 - f_{k+q})(n+1) - f_{k+q}(1 - f_k)n]\}, \tag{10.16}$$

where

$$s'(k+q,k,t,t') \tag{10.17}$$

$$= \frac{V}{(2\pi)^3\hbar^2}\|g_q\|^2 e^{-\int_{t'}^t dt''(\Gamma_{k+q} + \Gamma_k)(t'')} cos\left(\frac{\varepsilon_{k+q} - \varepsilon_k - \hbar\omega}{\hbar}(t - t')\right).$$

The nonlinear integro-differential equation (10.16) together with its initial condition is the starting point for developing the Monte Carlo approach to quantum-kinetic carrier dynamics. We consider the simpler nondegenerate case, when the distribution function is neglected with respect to the

unity. This is done straightforward in equation (10.16) and for Γ_k^o, while for the damping rate one can obtain

$$\begin{aligned}\Gamma_k &= (\Gamma_k^i + \Gamma_k^o)/2 \\ &= \int d^3q \frac{V}{2^3\pi^2\hbar} \sum_{\pm} \|g_q\|^2 \delta(\varepsilon_{k+q} - \varepsilon_k \pm \hbar\omega) \\ &\times (f_{k+q}(n + 1/2 \mp 1/2) + (n + 1/2 \pm 1/2)) \\ &= \int d^3q \frac{V}{2^3\pi^2\hbar} \sum_{\pm} \|g_q\|^2 \delta(\varepsilon_{k+q} - \varepsilon_k \pm \hbar\omega)(n + 1/2 \pm 1/2).\end{aligned} \tag{10.18}$$

Although this two step elimination of f (first in Γ^o and then in the whole Γ) effectively neglects the Γ^i contribution, it simplifies a lot the damping rate, making it time independent and equivalent to the Boltzmann one (see below). For equation (10.16) one can get:

$$\begin{aligned}\frac{d}{dt} f_k &= \int_0^t dt' \int d^3q \{s'(k+q, k, t, t')[f_{k+q}(n+1) - f_k n]\} \\ &- \int_0^t dt' \int d^3q \{-s'(k, k+q, t, t')[f_k(n+1) - f_{k+q} n]\} \\ &= \int_0^t dt' \int d^3q \{S(k+q, k, t-t') f_{k+q} - S(k, k+q, t-t') f_k\},\end{aligned} \tag{10.19}$$

where

$$\begin{aligned}S(k+q, k, t-t') &= \frac{V}{(2\pi)^3\hbar^2} \|g_q\|^2 e^{-(\Gamma_{k+q} + \Gamma_k)(t-t')} \\ &\times \left\{ cos\left(\frac{\varepsilon_{k+q} - \varepsilon_k - \hbar\omega}{\hbar}(t-t')\right)(n+1) \right. \\ &+ \left. cos\left(\frac{\varepsilon_{k+q} - \varepsilon_k + \hbar\omega}{\hbar}(t-t')\right) n \right\}.\end{aligned} \tag{10.20}$$

The integral form of the carrier density equation is then:

$$\begin{aligned}f(k,t) &= \int_0^t dt' \int_0^{t'} dt'' \int d^3q \{S(k+q, k, t'-t'') f(k+q, t'')\} \\ &- \int_0^t dt' \int_0^{t'} dt'' \int d^3q \{S(k, k+q, t'-t'') f(k, t'')\} + \phi(k).\end{aligned} \tag{10.21}$$

10.2.1 *Physical Aspects*

Before entering the numerical part of the problem, we can draw some general conclusions about the physics incorporated in the obtained equation.

For this it is convenient to make a comparison with the two equivalent integral forms of the BE (with zero electric field) [Jacoboni *et al.* (1988); Nedjalkov and Vitanov (1989)]:

$$f(k,t) = \int_0^t dt' \int d^3q\{S^B(k+q,k)f(k+q,t')\}$$

$$- \int_0^t dt' \int d^3q\{S^B(k,k+q)f(k,t')\} + \phi(k) \tag{10.22}$$

$$= e^{-\int_0^t dt' \Gamma^B(k)}\phi(k)$$

$$+ \int_0^t dt' \int d^3q e^{-\int_{t'}^t \Gamma^B(k)d\tau} S^B(k+q,k)f(k+q,t'), \tag{10.23}$$

where $\Gamma^B(k) = \int d^3q S^B(k,k+q)$ equals to Γ_k, so that S^B is given by the under the integral part of equation (10.19).

Both equations, the Boltzmann and the quantum-kinetic (10.21), provide the electron distribution function. Thus their solutions have to be positive with a time independent $\mathbf{L}_1$ norm (with positive initial conditions ϕ). Such requirements reflect the conservation of the career number during time evolution.

In the Boltzmann case $S^B \geq 0$ an integration with respect to k of equation (10.22) cancels the term in the curly brackets, showing that the solution keeps its k-integral constant in time. From equation (10.23) it follows that the solution is positive, so a physically acceptable solution is provided in the semiclassical case.

A comparison between equations (10.21) and (10.22) shows that due to the equivalent structure with respect to the k, q dependence, in the quantum case the solution also keeps its k-integral equal to the integral of the initial condition. But the positiveness of the quantum solution is not obvious as in the semiclassical case.

Important quantum aspects are encountered in (10.21). The extra time integration leads a non Markovian form of the equation: to obtain $f(k, t + dt)$ a memory for the distribution function in the whole time interval $(0, t)$ is required, while in the Boltzmann case a knowledge of $f(k, t)$ is sufficient. The equation (10.21) reflects the memory character of the quantum evolution of the electron distribution. Kernel (10.20) violates the classical energy conservation of the interaction. This effect is well pronounced for short evolution times. The delta function in (10.19) allows for contributions of the initial condition only to energy conservation related states. For short times contribution of (10.20) is equal for all states in the domain of integration.

Another quantum effect comes from the presence of a finite damping rate Γ. It prevents the long evolution time build up of the semiclassical energy conserving delta function thus leading to the effect of collisional broadening.

The following considerations clarify the above discussion. By using the identity:

$$\int_0^t dt' \int_0^{t'} dt'' = \int_0^t dt'' \int_{t''}^t dt'$$

one can write (10.21) as:

$$f(k,t) = \int_0^t dt'' \int d^3q \left[\left\{ \int_{t''}^t dt' S(k+q,k,t'-t'') \right\} f(k+q,t'') - \left\{ \int_{t''}^t dt' S(k,k+q,t'-t'') \right\} f(k,t'') \right] + \phi(k). \tag{10.24}$$

Formally (10.24) is written in the same way as the Boltzmann equation (10.21).

The physical interpretation is as if the electron system evolves under the time dependent *scattering term* $\{\int_{t''}^t dt' S(k,k+q,t'-t'')\}$ (if a probabilistic interpretation is possible). In contrast, in the semiclassical case the *scattering term*, being provided by a long time limit (called *golden rule*), does not depend explicitly on the time.

An analytical evaluation of the *scattering term* of (10.24) is possible due to the following relation:

$$\int_0^{t-t''} d\tau e^{-(\Gamma_{k+q}+\Gamma_k)\tau} \cos(\Omega_{k+q,k}\tau) = \frac{(\Gamma_{k+q}+\Gamma_k)}{\Omega_{k+q,k}^2 + (\Gamma_{k+q}+\Gamma_k)^2} + \frac{-(\Gamma_{k+q}+\Gamma_k)\cos(\Omega_{k+q,k}(t-t'')) + \Omega_{k+q,k} sin(\Omega_{k+q,k}(t-t''))}{\Omega_{k+q,k}^2 + (\Gamma_{k+q}+\Gamma_k)^2} \times e^{-(\Gamma_{k+q}+\Gamma_k)(t-t'')}.$$

It is seen that for values of t much larger than t'' the *scattering kernel* is positive and has a small time dependent part. Then the conclusions about the sign and norm of the semiclassical solution, deduced from equation (10.22) and (10.23) are expected to hold also for f in (10.24). But to assert this in the general case we have to rely on the numerical experiments.

The long time limit $t \to \infty$ cancels one of the time integrals and, thus, also the memory character of the equation. But this is still not the Boltzmann equation, because of the Lorentzian form of the *scattering kernel*. To cancel also this effect of collisional broadening we have to let the damping rate tend to zero, which recovers the energy delta function in (10.19).

10.2.2 *The Monte Carlo Algorithm*

The following estimator is chosen for this case:

$$\nu_i = \frac{\mathcal{K}(x_0, x_1)}{P(x_0, x_1)} \cdots \frac{\mathcal{K}(x_{i-1}, x_i)}{P(x_{i-1}, x_i)} \phi(x_i).$$

The equation (10.21) is written in the common form of an integral equation as follows:

$$f(k,t) = \int_0^t dt' \int_0^{t'} dt'' \int d^3q \{S(k+q, k, t'-t'')\}$$

$$- \int_0^t dt' \int_0^{t'} dt'' \int d^3q \{\lambda(k, t'-t'')\delta(q)\} f(k+q, t'') + \phi(k), \qquad (10.25)$$

where $\lambda(k, t'-t'') = \int d^3q S(k, k+q, t'-t'')$. Here we specify that the wave vectors k, q belong to a finite domain Ω (around the point $q = 0$) with a volume V_Ω.

The kernel has two components, $\mathcal{K}_A$ and $\mathcal{K}_B$, so that the i^{th} Neumann series term is represented by all possible combinations of the form:

$$\prod_{l_0} \mathcal{K}_A \prod_{l_1} \mathcal{K}_B \prod_{l_2} \mathcal{K}_A \cdots \prod_{l_i} \mathcal{K}_B \phi,$$

where the indices l_j are positive and $\sum_{j=0}^{i} l_j = i$.

We require that the q-integrals of the absolute value of any of the kernel components are limited by a constant $M \geq 0$. Then our statement is that the following estimate of the i^{th} Neumann series term holds:

$$|(\mathcal{K}_i \phi)(k,t)| \leq (\max \phi(k)) \frac{(2M)^i (t)^{2i}}{(2i)!}.$$

For $i = 1$ this estimate is true. Assuming that it is true for $i = p$, we have:

$$|(\mathcal{K}_{p+1}\phi)(k,t)|$$

$$= \left| \int_0^t dt' \int_0^{t'} dt'' \int d^3q \{\mathcal{K}(k+q, k, t'-t'') - \lambda(k, t'-t'')\delta(q)\}(\mathcal{K}_p(k+q, t'')) \right|$$

$$\leq \int_0^t dt' \int_0^{t'} dt''(M+M)(\max \phi(k)) \frac{(2M)^p (t'')^{2p}}{(2p)!} = (\max \phi(k)) \frac{(2M)^{p+1} (t)^{2p+2}}{(2p+2)!}.$$

Hence the convergence of the Neumann series of the carrier-density equation is better than the series of the exponent: $(\max \phi(k)) exp(2Mt^2)$.

The common approach to such problems is to look for a small parameter in the kernel of the equation under consideration. Our result shows that despite the fact that the kernel can be arbitrarily large, the series converges absolutely.

The existence of the constant M follows from the concrete form of S (see, equation (10.20)).

10.2.3 *Monte Carlo Solution*

In this section we discuss a concrete realization of the backward Monte Carlo algorithm. The choice of an isotropic initial condition ϕ decreases the number of variables in equation (10.21). Although the Monte Carlo advantages are expected in just opposite situations, a simpler form of (10.21) assists in the choice of the concrete algorithm.

We replace $k+q$ with k' and use spherical coordinates, with k'_z axes oriented along k. Let θ be the angle between k and k' so that

$$d^3k' = k'^2 dk' sin(\theta) d\theta d\xi, \quad \theta \in (0,\pi), \;\; \xi \in (0,2\pi).$$

From here on we denote by k and k' the absolute values of the corresponding vectors. All terms of S, equation (10.11), but $\|g_q\|^2$ depend only on k and k'. The norm $\|g_q\|^2$ is proportional to

$$\frac{1}{k'^2 + k^2 - 2kk'\theta}.$$

After integration on θ it results in $\frac{1}{kk'} log\left(\frac{k+k'}{|k-k'|}\right)$ contribution to the kernel of (10.12). The ξ integral contributes with 2π, so that only the k' integration remains in the equation, with a term $\dfrac{k'}{k} ln\left(\dfrac{k+k'}{|k-k'|}\right)$ in front of S. By introducing $F(k,t) = kf(k,t)$ and $\Phi(k,t) = k\phi(k,t)$ one obtains:

$$F(k,t) = \int_0^t dt' \int_0^{t'} dt'' \int_0^K dk' c.ln\left(\frac{k+k'}{|k-k'|}\right) \{S(k',k,t'-t'')F(k',t'') - \frac{k'}{k} S(k,k',t'-t'')F(k,t'')\} + \Phi(k).$$

The domain Ω is now the interval $(0,K)$. For completeness we give also $\Gamma(k)$ and $\phi(k)$:

$$\Gamma(k) = \frac{g(n+1)}{k} log\left(\frac{\sqrt{\omega_1}}{k-\sqrt{k^2-\omega_1}}\right) + \frac{gn}{k} log\left(\frac{\sqrt{\omega_1}}{-k+\sqrt{k^2+\omega_1}}\right),$$

where $\omega_1 = 2m\omega/\hbar$, and the first term is accounted if $k^2 \geq \omega_1$

$$\phi(k) = e^{-(b_1 k^2 - b_2)^2}.$$

Here c, g, b_1, b_2 stand for the contribution of the physical constants, taken to be the same as in [Schilp *et al.* (1994)]. The algorithm we used for finding a solution in a fixed point $x = (k,t)$ for one random walk is as follows:

Algorithm 10.1.

1 **Set** initial values $\nu = \phi(k), \quad W = 1, \quad \varepsilon$.

2 **Define** a value of x' with density $P(x, x') = p(t', t'')p(k, k')$ where $p(t', t'') = p(t')p(t''/t')$. Thus the value of t' is chosen with density $p(t') = \frac{2t'}{t^2}$ between 0 and t, while t'' is uniformly distributed within 0 and t. The value of k' is chosen uniformly distributed with density $1/K$.

3 **Calculate** the kernels $\mathcal{K}_A$ and $\mathcal{K}_B$.

4 **Choose** a value α of a uniformly distributed in $[0, 1]$ r.v.

5 **If**

$$\frac{|\mathcal{K}_A|}{|\mathcal{K}_A| + |\mathcal{K}_B|} \leq \alpha,$$

then

$$W = W \frac{sign(\mathcal{K}_A)(|\mathcal{K}_A| + |\mathcal{K}_B|)}{P(x, x')}$$

and

$$\nu = \nu + W.\phi(k').$$

5' **If**

$$\frac{|\mathcal{K}_A|}{|\mathcal{K}_A| + |\mathcal{K}_B|} > \alpha,$$

then

$$W = W \frac{-sign(\mathcal{K}_B)(|\mathcal{K}_A| + |\mathcal{K}_B|)}{P(x, x')}$$

and

$$\nu = \nu + W\phi(k).$$

6 **Repeat** steps 2 to 5 **or** 5′ **until** $|W| \leq \varepsilon$.

Simulation results of performing the above algorithm for zero lattice temperature are presented. At such temperature the semiclassical electron behavior is to emit phonons and to loose energy equal to multiple phonon energy $\hbar\omega$. For the electron distribution evolution, this results in the appearance of shifted down replicas of the initial distribution. Above the initial condition the distribution function remains zero, since the electrons cannot gain energy from the lattice.

In Figures 10.1 and 10.2 the $F(k^2)$ distribution for evolution times of 10, 20, 30 and 40 femtoseconds are presented. The simulation domain is

between 0 and $K = 100 \times 10^7 m^{-1}$. F is calculated using $N = 10^6$ random trajectories, in 50 points located in the region above the initial condition, where no classical electrons can appear. At such short times, the quantum approach results in finite probability to find electrons there. It decreases with the increase of the evolution time, associated with the tendency towards a classical transport behavior for larger times. It is attended with an increase of the distribution function oscillations, which is a typical quantum effect. The curves are similar to the ones given in [Schilp *et al.* (1994)].

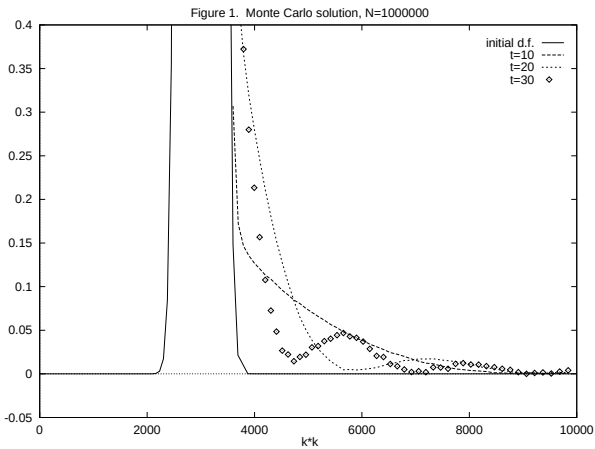


Fig. 10.1 $F(k^2)$ distribution for evolution times of 10, 20, and 30 femtoseconds.

Figures 10.3 and 10.4 show the $F(k^2)$ distribution for evolution times of 50, 100, 150 femtoseconds. The simulation domain is between 0 and $K = 66$ $10^7 m^{-1}$. F is calculated in 88 k points with a varying density. The curves demonstrate the appearance of the first replica. Due to the energy-time uncertainty its width is time dependent. The narrowing begins around the classical phonon emission time, 150 fs.

The numerical experience shows that with the increase of the evolution time the simulation efforts increase drastically. While for 50 fs evolution time $N = 10^5$ random walks ensure reliable statistics, for 200 fs a value of $N = 10^8$ is insufficient. This is associated with the increase of the time domain of simulation.

Figures 10.3 and 10.4 show in semilogarithmic scale the empirically calculated mean statistical error versus t^2, for $N = 10^5$ and $N = 10^6$ random walks. For example, for time 200 fs., the corresponding mean region where the r.v. values appear has a width of 5000 a.u. They compensate

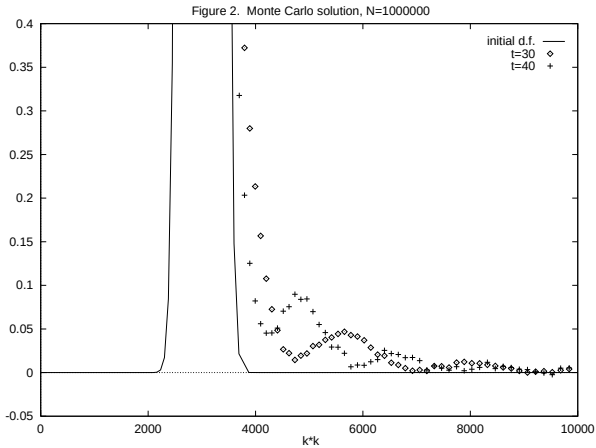


Fig. 10.2 $F(k^2)$ distribution for evolution times of 30, and 40 femtoseconds.

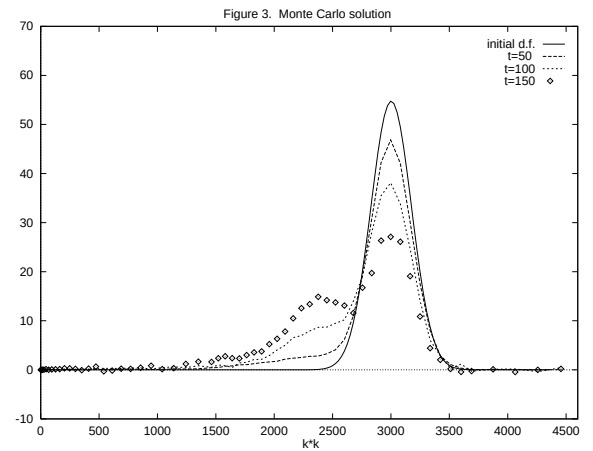


Fig. 10.3 $F(k^2)$ distribution for evolution times of 50, 100 and 150 femtoseconds.

each other to produce a mean value bellow 40 a.u. - the maximum of F for this evolution time.

10.3 The Wigner Quantum-Transport Equation

When describing some physical phenomena, the equation for the Wigner function is as general as the Liouville-von Neumann equation for the density

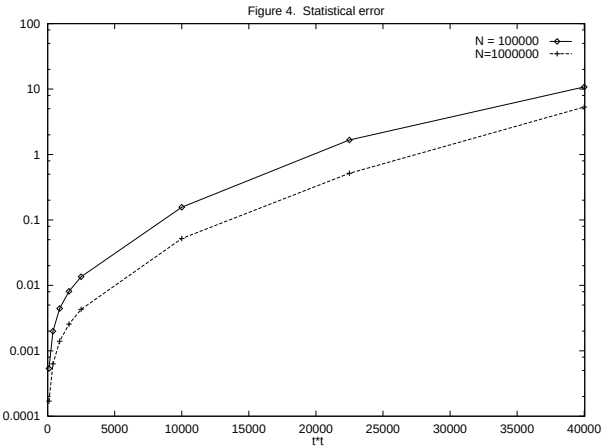


Fig. 10.4 Mean value of the statistical error versus t^2 for $N = 10^5$ and $N = 10^6$ random walks.

matrix. But the common definition of the Wigner equation meets difficulties with the convergence of the transfer-function integral for potentials having two different values when the coordinate approaches $+\infty$ and $-\infty$ respectively. It seems that this restricts the scope of physically actual potentials, which are accounted for in the Wigner picture. From a physical point of view it is clear that the potential value at very high distances from the region of interest has negligible contribution to the processes inside the region. That is why in the device modeling, the integral for the transfer function is taken over a finite interval.

A correct definition of the Wigner equation for such type of potentials is presented below. It introduces a damping factor in the Monte Carlo simulation, which ensures the convergence of the algorithm. For the sake of simplicity the one-dimensional case is considered.

Let $\rho(x, x')$ be the density matrix, where x and x' are eigenvalues of the position operator. Using the substitution $q = \frac{1}{2}(x + x')$, $\eta = x' - x$, the Wigner function is defined through the Weil-Wigner transform of the density matrix:

$$f^w(p, q) = \frac{1}{2\pi\hbar} \int_{-\infty}^{\infty} d\eta e^{i\eta p/\hbar} \rho(q - \eta/2, q + \eta/2).$$

The Fourier transformation concerns only the variable η. Since q and $\frac{d}{d\eta}$ commute, a description in terms of p (eigenvalues of $i\hbar\frac{d}{d\eta}$) and q does not

violate the uncertainty principle. f^w is a solution of the Wigner equation,

$$\{\pm t+(p/m)\pm q\}f^w(p,q,t) = \frac{1}{\hbar}\frac{1}{2\pi\hbar}\int_{-\infty}^{\infty} dp' V^w(p-p',q)f^w(p',q,t) \quad (10.26)$$

obtained from the Liouville-von Neumann equation for the density matrix. V^w is the transfer function:

$$V^w(p,q) = i\int_{-\infty}^{\infty} d\eta e^{i\eta p/\hbar}(V(q+\eta/2) - V(q-\eta/2)). \quad (10.27)$$

For solutions compatible with quantum mechanics in this section we will refer to the following: $f^w \in L_1$ i.e. such that the integral $\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}|f^w(p,q)|dpdq$ is finite, and $|f^w(p,q)| \leq (\pi\hbar)^{-1}$ [Tatarskii (1983)].

Let us consider a potential typical for the device modeling: a biased double barrier structure with length l, electric field, present in the structure F, barrier height and thickness H and a respectively. The total electric plus structure potential $V = V_1 + V_2$, is decomposed in the following way

$$V_1 = \begin{cases} Fl, & q<0 \\ F(l-q), & 0\leq q\leq l \\ 0, & q>l \end{cases} = Fl\theta(-q) + \begin{cases} 0, & q\leq 0 \\ F(l-q), & 0\leq q\leq l \\ 0, & q>l \end{cases} \quad (10.28)$$

$$V_2 = \begin{cases} 0, & q<0 \\ H, & 0\leq q\leq a \\ 0, & a<q<l-a \\ H, & l-a\leq q\leq l \\ 0, & q>l, \end{cases} \quad (10.29)$$

where $\theta(q)$ is the step-like function: $\theta(q) = 1$ for $q \geq 0$ and $\theta(q) = 0$ for $q < 0$.

In the framework of the finite-difference method, a self consistent scheme of discretization of *finite* domains of space and momentum variables is used [Frensley (1990)], satisfying the Fourier completeness relation. While the momentum integral in (10.26) can be approximated over a finite domain, because $f^w \to 0$ if $p \to \pm\infty$, the space integral in (10.27) appears to be undefined for step-like potentials, so that cutting off the $\eta \to \pm\infty$ regions should be regarded as a new definition of the transfer function rather than as an approximation. Another way to define (10.27) for such potentials is to include a damping $e^{\pm\alpha\eta}$ in the integrand function in (10.27) [Rossi *et al.* (1994b)]. With this the integral for the transfer function $V^w(p,q,\alpha)$

becomes well defined. We can reduce the effect of this artificial factor to the physical picture by decreasing the value of α. But in what sense should we understand the limit $\lim_{\alpha\to 0} V^w(p,q,\alpha)$, which recovers our potentials, will be clarified below.

Let us recall that the integral (10.27) that defines the transfer function is used in the Wigner equation (10.26), inside a second integral in dp' involving also the Wigner function, which is, by itself, a Fourier transformation of the density matrix. This suggests to justify the Wigner equation for such potentials with the help of the properties of the density matrix. Let ρ be any continuous function from $\mathbf{L}_1$, so that $\int_{-\infty}^{\infty} |\rho(\eta)| d\eta$ is finite, then its Fourier transformation: $f(p) = \int_{-\infty}^{\infty} e^{i\eta p}\rho(\eta)d\eta$ together with the integrals $\int_{-\infty}^{\infty} e^{i\eta p}\theta(\pm\eta)\rho(\eta)d\eta$ are well defined. We can then obtain the existence of the following limit:

$$\begin{aligned}
&\int_{-\infty}^{\infty} e^{i\eta p}\theta(\pm\eta)\rho(\eta)d\eta = \int_{-\infty}^{\infty} \lim_{\alpha\to 0} e^{\mp\alpha\eta}e^{i\eta p}\theta(\pm\eta)\rho(\eta)d\eta \\
&= \lim_{\alpha\to 0}\int_{-\infty}^{\infty} e^{\mp\alpha\eta}e^{i\eta p}\theta(\pm\eta)\rho(\eta)d\eta \\
&= \frac{1}{2\pi}\lim_{\alpha\to 0}\int_{-\infty}^{\infty} e^{\mp\alpha\eta}e^{i\eta p}\theta(\pm\eta)\int_{-\infty}^{\infty} e^{-i\eta p'}f(p')dp'd\eta \qquad (10.30)\\
&= \frac{1}{2\pi}\lim_{\alpha\to 0}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{\mp\alpha\eta}e^{i\eta(p-p')}\theta(\pm\eta)f(p')d\eta dp'.
\end{aligned}$$

If $\rho(\eta)$ is the density matrix $\rho(q-\eta/2, q+\eta/2)$ then $f(p)$ will correspond to the Wigner function $f^w(p)$ (q and t are fixed). In order to apply this procedure to the integrals appearing in the Wigner equation, it is needed to prove that ρ belongs to $\mathbf{L_1}$:

If the density matrix is expressed trough a set of normalized states of the system

$$\rho(q-\eta/2, q+\eta/2) = \sum_i \gamma_i \Phi_i^*(q+\eta/2)\Phi_i(q-\eta/2);\quad \sum_i \gamma_i = 1;\ \gamma_i \geq 0, \forall i$$

then

$$\begin{aligned}
&\int_{-\infty}^{\infty}\left|\sum_i \gamma_i \Phi_i^*(q+\eta/2)\Phi_i(q-\eta/2)\right| d\eta \\
&\leq \sum_i \gamma_i \int_{-\infty}^{\infty} |\Phi_i^*(q+\eta/2)\Phi_i(q-\eta/2)|\, d\eta.
\end{aligned}$$

The Cauchy inequality provides the following relation:

$$\left(\int_{-\infty}^{\infty}|\Phi_i^*(q+\eta/2)\Phi_i(q-\eta/2)|d\eta\right)^2$$
$$\leq \int_{-\infty}^{\infty}|\Phi_i^*(q+\eta/2)|^2 d\eta \int_{-\infty}^{\infty}|\Phi_i(q-\eta/2)|^2 d\eta = 4,$$

where it is taken into account that the states Φ are normalized to unity. We obtain:

$$\int_{-\infty}^{\infty}|\rho(q-\eta/2, q+\eta/2)|d\eta \leq 2\sum_i \gamma_i = 2.$$

The above result permits the use of the limit $\alpha \to 0$ in the Wigner equation:

$$\begin{aligned}&\{\pm t + (p/m) \pm q\} f^w(p,q,t)\\ &= C\lim_{\alpha\to 0}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} e^{i\eta(p-p')/\hbar} e^{-\alpha|\eta|}\\ &\times(\theta(-q-\eta/2)-\theta(-q+\eta/2)) f^w(p',q,t)d\eta dp' + \ldots,\end{aligned} \tag{10.31}$$

where the dots refer to the other terms in the potential profile, which do not contain θ term and C is a constant. Since $\alpha = 0$ recovers our original formulation, this equation can be regarded as a definition of the WE for potentials which approach two different constants in $\pm\infty$.

The above limit can be accounted for in the framework of the distribution theory.

$$\begin{aligned}&\lim_{\alpha\to 0}\int_{-\infty}^{\infty}\int_{-\infty}^{\infty}\theta(\pm\eta)e^{\mp\alpha\eta}e^{i\eta(p)}\phi(p)d\eta dp\\ &= \pm i\lim_{\alpha\to 0}\int_{-\infty}^{\infty}\frac{\phi(p)}{p\pm i\alpha}dp = \pm i\left(\mp i\pi\phi(0) + Vp\int_{-\infty}^{\infty}\frac{\phi(p)}{p}dp\right),\end{aligned}$$

where it is required that $\phi(\pm\infty) = 0$ and $Vp\int_{-\infty}^{\infty}$ means $\lim_{\alpha\to 0}\left(\int_{-\infty}^{-\alpha} + \int_{\alpha}^{\infty}\right)$. Then we can write formally:

$$\int_{-\infty}^{\infty}\theta(\pm\eta)e^{i\eta(p)}d\eta = \pi\delta(p) + -\mathcal{P}\frac{1}{p}.$$

Hence, the transfer function $V^w(p,q) = \alpha_{\to 0}V^w(p,q,\alpha)$ is understood in terms of generalized functions.

In this section we will use a finite value for α under the following assumptions: f_α^w close to f_0^w means that the RHS of the corresponding Wigner equations are almost equal, i.e.,

$$\int_{-\infty}^{\infty}e^{\mp\alpha\eta}e^{i\eta p}\theta(\pm\eta)\rho(\eta)d\eta \simeq \int_{-\infty}^{\infty}e^{i\eta p}\theta(\pm\eta)\rho(\eta)d\eta. \tag{10.32}$$

in any value of t in the time evolution of f^w, i.e. α must be such that $e^{\pm\alpha\eta} \simeq 1$ for η values in which $\rho(\eta) \neq 0$, so the initial guess of α depends not only upon the initial "spread" of ρ. We must be sure *a priori* that the density matrix corresponding to our Wigner function will not spread out too much to violate condition (10.32).

10.3.1 *The Integral Form of the Wigner Equation*

When the damping $e^{-\alpha|\eta|}$ is included in (10.27), the transfer function is well defined in the integro-differential form of the Wigner equation (10.26). To determine the solution, this equation must be considered together with a function $f^w(p,q,t_0) = f_0^w(p,q)$ providing the initial condition at time t_0. This form of the Wigner equation is not convenient for the Monte Carlo approach. Equation (10.26) can be transformed into an integral form, by expressing the solution in terms of its Green function [Vitanov *et al.* (1994); Rossi *et al.* (1992)]. The left hand side of (10.26) is the Liouville operator with a force term 0, so that the Green function is known and introduces Newton phase trajectories, which for zero force have a simple form: a particle which is in the phase point p,q at time t was located in $p, q-(t-t')p/m$ at time $t' < t$. The following equation is obtained:

$$\begin{aligned} f^w(p,q,t) &= f^w(p, q-(t-t_0)p/m, t_0) \\ &+ \int_{t_0}^{t} dt' \int_{-\infty}^{\infty} dp' V^w(p-p', q-(t-t')p/m) f^w(p', q-(t-t')p/m, t'). \end{aligned} \tag{10.33}$$

It is seen, that the initial condition already participates explicitly in the equation. If we begin to substitute (10.33) into itself, a series expansion is obtained:

$$\begin{aligned} f^w(p,q,t) &= f^w(p, q-(t-t_0)p/m, t_0) \\ &+ \int_{t_0}^{t} dt_1 \int_{-\infty}^{\infty} dp_1 V^w(p-p_1, q-(t-t_1)p/m) \\ &\times f^w(p_1, q-(t-t_1)p/m-(t_1-t_0)p_1/m, t_0) \\ &+ \int_{t_0}^{t} dt_1 \int_{-\infty}^{\infty} dp_1 \int_{t_0}^{t_1} dt_2 \int_{-\infty}^{\infty} dp_2 V^w(p-p_1, q-(t-t_1)p/m) \\ &\times V^w(p_1-p_2, q-(t-t_1)p/m-(t_1-t_2)p_1/m) \\ &\quad f^w(p_2, q-(t-t_1)p/m-(t_1-t_2)p_1/m-(t_2-t_0)p_2/m, t_0) + \ldots \end{aligned} \tag{10.34}$$

10.3.2 *The Monte Carlo Algorithm*

For practical implementation of this general algorithm the variation of p and q has to be constrained over finite domains. Another peculiarity of the problem is that we can specify boundary conditions. For constant potential f^w is the Fermi-Dirac or, in the high energy limit, the Maxwell-Boltzmann function. Hence far, in the constant potential regions on both sides of the structure we can choose two points q_1 and q_2 where $f^w(p, q_i, t)$ is known for any time $t > t_0$. The solution of (10.33) is sought within the region Ω between q_1 and q_2. We have to make some compromise between the constant potential condition (i.e. q_i to be chosen far from the device, where the potential varies) and Ω not larger than a region where we can comfortably simulate and have negligible effect of the spatial damping.

By specifying the $f_0^w(p, q)$ values within Ω for the initial time t_0 we will have the information required for determining the solution.

Let the momentum variable be chosen to vary in $(-A, A)$, $A > 0$, thus constraining the p integrals in (10.33) and (10.34) to finite regions. The difference $p-p'$ is allowed to have values outside $(-A, A)$ because $V^w(p, ., .)$ is known analytically.

Hence, the simulated equation has the form:

$$f^w(p, q, t) = f^w(p, q - (t - t_b)p/m, t_b) \tag{10.35}$$
$$+ \int_{t_b}^{t} dt' \int_{-A}^{A} dp' V^w(p - p', q - (t - t')p/m) f^w(p', q - (t - t')p/m, t'),$$

where t_b is the $max(t_0, t_q)$ and t_q is the solution of the equation $q - (t - t_q)p/m = q_i$; here q_i is the boundary reached at a time less then t. So, we suppose that also in (10.33) t_0 is replaced by t_b.

We choose the estimator ν_i as in the quantum-kinetic case to be:

$$\nu_i = \frac{V^w(Q_0, Q_1)}{P(Q_0, Q_1)} \cdots \frac{V^w(Q_{i-1}, Q_i)}{P(Q_{i-1}, Q_i)} f_0^w(Q_i) \tag{10.36}$$

Provided that the corresponding Neumann series converges, the following particular algorithm can be used:

Algorithm 10.2.

1 The initial point $Q_0 = (p, t)$ together with the parametric value q are fixed for all iterative orders. This provides the phase space - time coordinates in which the Wigner function is calculated.
2 Introduce two estimators, ν and μ setting their initial values to 0 and 1 respectively, and set the label $i = 1$.

3 The value $p_{(i)}$ is randomly selected in the interval $(-A, A)$ with an arbitrary probability density P_p.
4 The value $t_{(i)}$ is randomly selected in the interval $(t_{(i-1)}, t_0)$, where $t_{(0)} = t$, with an arbitrary probability density P_t. Since generation of the initial time t_0 is required in the simulation process, $P_t(t_0)$ must be chosen in a way to ensure this.
5 The new position on the Newton trajectory $q(t_{(i)}) = q(t_{(i-1)}) + (t_{(i)} - t_0)p_{(i-1)}/m - (t_{(i)} - t_0)p_{(i)}/m$ is calculated.

If $q(t_{(i)})$ is outside Ω, the time t_b of crossing the boundary is calculated. Then μ is multiplied by

$$\frac{V^w(p_{(i-1)} - p_{(i)}, q(t_b))f^w(p_{(i)}, q_{1,2}, t_b)}{P_p(p_{(i)}) \int_{t_0}^{t_b} P_t(\tau)d\tau}.$$

If $q(t_{(i)})$ is inside Ω and $t_{(i)} = t_0$ μ is multiplied by

$$\frac{V^w(p_{(i-1)} - p_{(i)}, q(t_{(i-1)}))f^w(p_{(i)}, q(t_{(i)}), t_i)}{P_p(p_{(i)})P_t(t_{(i)})}.$$

With this the contribution of the present trajectory is accounted for, μ is added to ν and a generation of a new trajectory begins.
If $q(t_{(i)})$ is inside Ω and $t_{(i)} > t_0$, μ is multiplied by

$$\frac{V^w(p_{(i-1)} - p_{(i)}, q(t_{(i-1)})}{P_p(p_{(i)})P_t(t_{(i)})}$$

and the simulation process of the present trajectory starts again from step 3 with i set to $i + 1$.

It is seen that when the trajectory encounters the boundary, the knowledge of $f^w(p, q_{1,2}, t_b)$ results in sparing the simulation between times t_b and t_0. Another way for optimization of the algorithm is to choose P_pP_t proportional to $|V^w|$. Since V^w is an oscillating function this will ensure generation in p, t points, where V^w has significant values and will avoid points where it is almost zero.

With the above algorithm the solution of the Wigner equation is obtained as a sum of contributions coming from different *trajectories* in a sort of Feynman path integral. The Monte Carlo nature of the approach is realized through the random selection of trajectories. Each of these paths carries the correct phase information so that the overall phase coherence is preserved in this approach.

10.3.3 *The Neumann Series Convergency*

We will prove the convergence of the Neumann series obtained for (10.35). The p_1 integral in the first iteration can be estimated as follows:

$$\begin{aligned}
&\left| \int_{t_b}^{t} dt_1 \int_{-A}^{A} V^w(p-p_1, q-(t-t_1)p/m) \right. \\
&\times \left. f^w(p_1, q-(t-t_1)p/m-(t_1-t_0)p_1/m, t_0)dp' \right| \\
&\leq \int_{t_0}^{t} dt_1 \int_{-A}^{A} |V^w(p-p_1, q-(t-t_1)p/m)| \\
&\times |f^w(p_1, q-(t-t_1)p/m-(t_1-t_0)p_1/m, t_0)|\, dp' \\
&\leq 2A.M.\|f_0^w\|.(t-t_0),
\end{aligned}$$

where $M = \max_{p,q} |V^w(p,q)|$. The existence of this maximum is provided by the definition of transfer function:

$$|V^w(p,q)| \leq 2\int_{-\infty}^{\infty} d\eta |V(\eta/2)|,$$

so that if $V(\eta)$ is absolutely convergent, as is the potential with finite α considered in this section, the maximum exists. $\|f_0^w\|$ means:

$$\|f_0^w\| = max(\max_{p,q} |f^w(p,q,t_0)| \quad , \quad \max_{p,t} |f^w(p,q_{1,2},t)|.$$

One of the properties of the Wigner function is to be bounded by $(\pi\hbar)^{-1}$ at any time, so that the existence of $\|f_0^w\|$ does not restrict the scope of the solutions having physical meaning.

Let suppose that the estimate

$$|V_{(n-1)}^w f^w| \leq \|f_0^w\| \frac{(2A.M(t-t_0))^{(n-1)}}{(n-1)!}$$

holds. Then

$$\begin{aligned}
&|V_n^w f^w| \\
&= \left| \int_{t_b}^{t} dt' \int_{-A}^{A} dp' V^w(p-p', q-(t-t')p/m)(V_{(n-1)}^w f)(p', p, q, t') \right| \\
&\leq 2A.M\|f_0^w\| \int_{t_0}^{t} dt' \frac{(2A.M(t-t_0))^{(n-1)}}{(n-1)!} = \|f_0^w\| \frac{(2A.M(t-t_0))^{(n)}}{(n)!}.
\end{aligned}$$

Hence $\Sigma_n |V_n^w f^w|$ is limited by $\|f_0^w\| exp(2A.M(t-t_0))$ so that the Neumann series corresponding to (10.35) is absolutely convergent and has a unique continuous solution for any continuous initial condition. Thus (10.35) can be solved by Monte Carlo algorithm.

Equation (10.35) is obtained by cutting off the infinite integration boundaries in (10.33). We have to investigate what is the effect of this cutting off on the solution i.e. we should decide if the solution of (10.35) gives a correct approximation to the solution of (10.33).

Let f^w be a physical solution of (10.33), obtained from the Liouville-von Neumann equation for the density matrix, so that it has all the properties of the Wigner function, and let f_0^w be its initial and boundary conditions. If we use the same initial and boundary conditions in the corresponding equation (10.35) the solution, as just proved, is unique and is denoted by f_A^w. If we subtract (10.35) from (10.33), the following equation is obtained:

$$\delta f^w(p,q,t) = \varepsilon(p,q,t) + \int_{t_b}^{t} dt' \int_{-A}^{A} dp' V^w(p-p', q-(t-t')p/m)\delta f^w(p', q-(t-t')p/m, t'),$$

where $\delta f^w = f^w - f_A^w$ and

$$\varepsilon(p,q,t) = \int_{t_b}^{t} dt' \left(\int_{-\infty}^{-A} + \int_{A}^{\infty} \right) dp V^w(p-p', q-(t-t')p/m) \times f^w(p', q-(t-t')p/m, t').$$

From the existence of the infinite integral in (10.33) it follows that $\varepsilon(A,p,q,\tau) \to 0$ when $A \to \infty$ for any given p,q,τ values, $-A \le p \le A, \quad q \in \Omega, \quad t_0 \le \tau \le t$. From (10.33) we also can conclude that $\varepsilon(A,p,q,t)$ is continuous on p,q,t, as it is expressed through a sum of continuous functions of p,q,t.

Now we will prove that ε tends to zero uniformly as $A \to \infty$, i.e. that for any $\delta > 0$ there exists A_0 such that $\forall A \ge A_0$ and $\forall p,q,\tau$ in $-A \le p \le A, \quad q \in \Omega, \quad t_0 \le \tau \le t$ we have $|\varepsilon(A,p,q,\tau)| \le \delta$. To prove this, we suppose the contrary: let, for a given δ, a series $(A_n) \to \infty$ and points $p_n, q_n \tau_n$ exist so that $|\varepsilon(A_n,p_n,q_n,\tau_n)| \ge \delta$ when $n \to \infty$. Because the variables q and τ are bounded, points q_0, τ_0 exist, so that $|\varepsilon(A_n,p_n,q_0,\tau_0)| \ge \delta$ when $n \to \infty$. We make use of the fact that $f^w(p,q,t) \to 0$ when $p \to \pm\infty$ so that $|\varepsilon(A_n,p_n,q_0,\tau_0)| \to 0$ when $p_n \to \infty$ to conclude that $\{p_n\}$ is bounded. Hence a point p_0 exist in which $|\varepsilon(A_n,p_0,q_0,\tau_0)| \ge \delta$ when $n \to \infty$, so that there is violation of the condition $|\varepsilon(A,p,q,\tau)| \to 0$ when $A \to \infty$ Hence $\varepsilon(A,p,q,\tau) \to 0$ when $A \to \infty$ uniformly with respect to p,q,τ for $-A \le p \le A, \quad q \in \Omega, \quad t_0 \le \tau \le t$, i.e. $\|\varepsilon(A)\| \to 0$, when $A \to \infty$.

Now in the same manner as for (10.35), we can obtain that

$$|\delta f^w| \le \|\varepsilon(A)\| exp(2A.M(t-t_0)),$$

and hence, the solution of the approximate equation tends to the Wigner function uniformly with respect to the arguments in the domain of simulation.

To conclude this section we note that the Levinson and Barker-Ferry equations, which describe the femtosecond evolution of local electron packets in a quantum wires, were derived. An effect of ultrafast spatial transport is observed. Our numerical experiments show that the solutions of the two models begin to differ after around 200 femtoseconds of evolution. The next few hundreds of femtoseconds is the most interesting time domain for analysis of the features of the two models. Unfortunately the numerical burden increases rapidly with the evolution time. Novel numerical technologies, including Grid technologies, aiming to explore this time domain along with the effect of the electric field are needed to tackle such a complex task. In the next section we present an approach that demonstrates the power of these new computational technologies.

10.4 A Grid Computing Application to Modeling of Carrier Transport in Nanowires

Grid computing is an emerging computing model that provides the ability to perform higher throughput computing by taking advantage of many networked computers to model a virtual computer architecture that is able to distribute process execution across a parallel infrastructure. Grids use the resources of many separate computers connected by a network to solve large or very large computational problems. Grids provide the ability to perform computations on large data sets, by dividing them down into many smaller tasks [Foster and Kesselman (1998); Davies (2004); Buyya (2007)]. Dealing with grid computing one should accept that some of subtasks will not be performed, so that the result of grid computation should not be very sensitive to possible lost information from some of the subtasks. This is one of important differences between grid and distributed computing. We should also mention that many ideas for the new wave of grid computing were borrowed from high-performance computing.

10.4.1 *Physical Model*

An initial non-equilibrium electron distribution is created locally in a nanowire by e.g. an optical excitation. The electrons begin to propagate

along the wire, where they are characterized by two variables: the position z and the component of the wave vector is k_z. We note that these are Wigner coordinates so that a classical interpretation as *position and momentum of the electron* is not relevant any more. A general, time-dependent electric field $E(t)$ can be applied along the wire. The field changes the wave vector with the time: $k_z(t') = k_z - \int_{t'}^{t} eE(\tau)/\hbar d\tau$ where k_z is the wave vector value at the initialization time t. Another source of modifications of the wave vector are dissipative processes of electron-phonon interaction, responsible for the energy relaxation of the electrons.

The following considerations simplify the problem thus allowing to focus on the numerical aspects: *(i)* We regard the Levinson type of equation which bears the basic numerical properties of the physical model:

$$\begin{aligned}\left(\frac{\partial}{\partial t} + \frac{\hbar k_z}{m}\nabla_z\right) f_w(z,k_z,t) = \int d\mathbf{k}' \int_0^t dt' \\ \{S(k_z,k_z',\mathbf{q}_\perp',t,t') f_w(z(t'),k_z{}'(t'),t') \\ - S(k_z',k_z,\mathbf{q}_\perp',t,t') f_w(z(t'),k_z(t'),t')\},\end{aligned} \tag{10.37}$$

where both $\int_G d^3\mathbf{k}' = \int d\mathbf{q}_\perp' \int_{-Q_2}^{Q_2} dk_z'$ and the domain G is specified in the next section. The spatial part $z(t')$ of the trajectory is initialized by the value z at time t: $z(t') = z - \frac{\hbar}{m}\int_{t'}^{t}\left(k_z(\tau) - \frac{q_z'}{2}\right)d\tau; q_z' = k_z - k_z'$. The *scattering function* S is

$$\begin{aligned}S(k_z,k_z',\mathbf{q}_\perp',t,t') = \frac{2V}{(2\pi)^3}|\Gamma(\mathbf{q}_\perp')\mathcal{F}(\mathbf{q}_\perp',k_z-k_z')|^2 \\ \times\left[n(\mathbf{q})cos\left(\int_{t'}^{t}\frac{(\epsilon(k_z(\tau)) - \epsilon(k_z'(\tau)) - \hbar\omega_{\mathbf{q}'})\,d\tau}{\hbar}\right)\right. \\ \left. + (n(\mathbf{q})+1)cos\left(\int_{t'}^{t}\frac{(\epsilon(k_z(\tau)) - \epsilon(k_z'(\tau)) + \hbar\omega_{\mathbf{q}'})\,d\tau}{\hbar}\right)\right].\end{aligned} \tag{10.38}$$

The electron-phonon coupling $\mathcal{F}$ is chosen for the case of Fröhlich polar optical interaction:

$$\begin{aligned}\mathcal{F}(\mathbf{q}_\perp',k_z-k_z') = -\left[\frac{2\pi e^2\omega_{\mathbf{q}'}}{\hbar V}\left(\frac{1}{\varepsilon_\infty} - \frac{1}{\varepsilon_s}\right)\frac{1}{(\mathbf{q}')^2}\right]^{\frac{1}{2}}; \\ \Gamma(\mathbf{q}_\perp') = \int \mathbf{r}_\perp |\Psi(\mathbf{r}_\perp)|^2 e^{i\mathbf{q}_\perp'\cdot\mathbf{r}_\perp},\end{aligned}$$

where (ε_∞) and (ε_s) are the optical and static dielectric constants. $\Gamma(\mathbf{q}_\perp')$ is the Fourier transform of the square of the ground state wave function. Thus

we come to the second simplifying assumption: *(ii)* Very low temperatures are assumed, so that the electrons reside in the ground state $\Psi(\mathbf{r}_\perp)$ in the plane normal to the wire. The phonon distribution is described by the Bose function, $n_{\mathbf{q}'} = (e^{\frac{\hbar\omega_{\mathbf{q}'}}{KT}} - 1)^{-1}$ with K the Boltzmann constant and T is the temperature of the crystal. $\hbar\omega_{\mathbf{q}'}$ is the phonon energy which generally depends on $\mathbf{q}' = (\mathbf{q}'_\perp, q'_z)$, and $\varepsilon(k_z) = (\hbar^2 k_z^2)/2m$ is the electron energy. *(iii)* Finally we consider the case of no electric field applied along the wire: $E = 0$.

Next we need to transform the transport equation into the corresponding integral form:

$$f_w(z, k_z, t) = f_{w0}(z - \frac{\hbar k_z}{m}t, k_z) \tag{10.39}$$
$$+ \int_0^t dt'' \int_{t''}^t dt' \int_G d^3\mathbf{k}' \{K_1(k_z, k'_z, \mathbf{q}'_\perp, t', t'') f_w\left(z + z_0(k_z, q'_z, t, t', t''), k'_z, t''\right)\}$$
$$+ \int_0^t dt'' \int_{t''}^t dt' \int_G d^3\mathbf{k}' \{K_2(k_z, k'_z, \mathbf{q}'_\perp, t', t'') f_w\left(z + z_0(k_z, q'_z, t, t', t''), k_z, t''\right)\},$$

where

$$z_0(k_z, q'_z, t, t', t'') = -\frac{\hbar k_z}{m}(t - t'') + \frac{\hbar q'_z}{2m}(t' - t''),$$

$$K_1(k_z, k'_z, \mathbf{q}'_\perp, t', t'') = S(k'_z, k_z, \mathbf{q}'_\perp, t', t'') = -K_2(k'_z.k_z, \mathbf{q}'_\perp, t', t'').$$

Here we note that the evolution problem becomes inhomogeneous due to the spatial dependence of the initial condition f_{w0}. The shape of the wire modifies the electron-phonon coupling via the ground state in the normal plane. If the wire cross section is chosen to be a square with side a, the corresponding factor Γ becomes: $|\Gamma(\mathbf{q}'_\perp)|^2 = |\Gamma(q'_x)\Gamma(q'_y)|^2 = \left(\frac{4\pi^2}{q'_x a((q'_x a)^2 - 4\pi^2)}\right)^2 4sin^2(aq'_x/2) \left(\frac{4\pi^2}{q'_y a\left((q'_y a)^2 - 4\pi^2\right)}\right)^2 4sin^2(aq'_y/2)$. We note that the Neumann series of such type of integral equations as (10.39) converges [Gurov and Whitlock (2002)]. Thus, the functional (10.40) can be evaluated by a MC estimator [Mikhailov (1995)].

10.4.2 *The Monte Carlo Method*

The values of the physical quantities are expressed by the following general functional of the solution of (10.39):

$$J_g(f) \equiv (g, f) = \int_0^{\mathcal{T}} \int_D g(z, k_z, t) f_w(z, k_z, t) dz dk_z dt \tag{10.40}$$

by a MC method. Here we specify that the phase space point (z, k_z) belongs to a rectangular domain $D = (-Q_1, Q_1) \times (-Q_2, Q_2)$, and $t \in (0, \mathcal{T})$. The

function $g(z, k_z, t)$ depends on the quantity of interest. In particular the Wigner function, the wave vector and density distributions, and the energy density are given by:

$$
\begin{aligned}
&(i)\ \ g_W(z, k_z, t) = \delta(z - z_0)\delta(k_z - k_{z,0})\delta(t - t_0),\\
&(ii)\ \ g_n(z, k_z, t) = \delta(k_z - k_{z,0})\delta(t - t_0),\\
&(iii)\ \ g_k(z, k_z, t) = \delta(z - z_0)\delta(t - t_0),\\
&(iv)\ \ g(z, k_z, t) = \epsilon(k_z)\delta(z - z_0)\delta(t - t_0)/g_n(z, k_z, t).
\end{aligned}
$$

We consider a biased MC estimator for evaluating the functional (10.40) using backward time evolution of the numerical trajectories in the following way:

$$
\xi_s[J_g(f)] = \frac{g(z, k_z, t)}{p_{in}(z, k_z, t)} W_0 f_{w,0}(., k_z, 0) + \frac{g(z, k_z, t)}{p_{in}(z, k_z, t)} \sum_{j=1}^{s} W_j^{\alpha} f_{w,0}\left(., k_{z,j}^{\alpha}, t_j\right),
$$

where

$$
f_{w,0}\left(., k_{z,j}^{\alpha}, t_j\right) =
$$

$$
\begin{cases}
f_{w,0}\left(z + z_0(k_{z,j-1}, k_{z,j-1} - k_{z,j}, t_{j-1}, t'_j, t_j), k_{z,j}, t_j\right), & \text{if } \alpha = 1,\\
f_{w,0}\left(z + z_0(k_{z,j-1}, k_{z,j} - k_{z,j-1}, t_{j-1}, t'_j, t_j), k_{z,j-1}, t_j\right), & \text{if } \alpha = 2,
\end{cases}
$$

$$
W_j^{\alpha} = W_{j-1}^{\alpha} \frac{K_{\alpha}(k_{zj-1}, \mathbf{k}_j, t'_j, t_j)}{p_{\alpha} p_{tr}(\mathbf{k}_{j-1}, \mathbf{k}_j, t'_j, t_j)},\ \ W_0^{\alpha} = W_0 = 1,\ \ \alpha = 1, 2,\ \ j = 1, \ldots, s\,.
$$

The probabilities $p_{\alpha}, (\alpha = 1, 2)$ are chosen to be proportional to the absolute value of the kernels in (10.39). The initial density $p_{in}(z, k_z, t)$ and the transition density $p_{tr}(\mathbf{k}, \mathbf{k}', t', t'')$ are chosen to be permissible[1] to the function $g(z, k_z, t)$ and the kernels, respectively. The first point (z, k_{z0}, t_0) in the Markov chain is chosen using the initial density, where k_{z0} is the third coordinate of the wave vector $\mathbf{k}_0$. Next points $(k_{zj}, t'_j, t_j) \in (-Q_2, Q_2) \times (t_j, t_{j-1}) \times (0, t_{j-1})$ of the Markov chain:

$$
(k_{z0}, t_0) \to (k_{z1}, t'_1, t_1) \to \ldots \to (k_{zj}, t'_j, t_j) \to \ldots \to (k_{zs}, t'_s, t_s)
$$

$(j = 1, 2, \ldots, s)$ do not depend on the position z of the electrons. They are sampled using the transition density $p_{tr}(\mathbf{k}, \mathbf{k}', t', t'')$. The z coordinate of the generated wave vectors is taken for the Markov chain, while the normal coordinates give values for $\mathbf{q}_{\perp}$. As the integral on $\mathbf{q}_{\perp}$ can be assigned to the kernel these values do not form a Markov chain but are independent on

[1] $r(x)$ is permissible of $g(x)$ if $r(x) > 0$ when $g(x) \neq 0$ and $r(x) \geq 0$ when $g(x) = 0$ (see Definition 4.1 in Section 4.2).

the consecutive steps. The time t'_j conditionally depends on the selected time t_j. The Markov chain terminates in time $t_s < \varepsilon_1$, where ε_1 is related to the truncation error introduced in first section.

In order to evaluate the functional (10.40) by N independent samples of the obtained estimator, we define a Monte Carlo method

$$\frac{1}{N}\sum_{i=1}^{N}(\xi_s[J_g(f)])_i \xrightarrow{P} J_g(f_s) \approx J_g(f), \tag{10.41}$$

where $\xrightarrow{P}$ means stochastic convergence as $N \to \infty$; f_s is the iterative solution obtained by the Neumann series of (10.39), and s is the number of iterations. The relation (10.41) still does not determine the computational algorithm. To define a MC algorithm we have to specify the initial and transition densities, as well the modeling function (or sampling rule). The modeling function describes the rule needed to calculate the states of the Markov chain by using uniformly distributed random numbers in the interval $(0, 1)$. The transition density is chosen: $p_{tr}(\mathbf{k}, \mathbf{k}', t', t'') = p(\mathbf{k}'/\mathbf{k})p(t, t', t'')$, where $p(t, t', t'') = p(t, t'')p(t'/t'') = \frac{1}{t}\frac{1}{(t-t'')}$ $p(\mathbf{k}'/\mathbf{k}) = c_1/(\mathbf{k}' - \mathbf{k})^2$],and c_1 is the normalized constant. Thus, if we know t, the next times t'' and t' are computed by using the inverse-transformation rule. The wave vector $\mathbf{k}'$ are sampled in the same way as described in [Gurov and Dimov (2004)]. The difference here is that we have to compute all three coordinates of the wave vector although we need only the third coordinate. The choice of $p_{in}(z, k_z, t)$ is according to the particular function $g(z, k_z, t)$.

10.4.3 *Grid Implementation and Numerical Results*

The computational complexity of an MC algorithm can be measured by the quantity $C_N = N \times \tau \times M(s_{\varepsilon_1})$. The number of the random walks, N, and the average number of transitions in the Markov chain, $M(s_{\varepsilon_1})$, are related to the stochastic and systematic errors [Gurov and Whitlock (2002)]. The mean time for modeling one transition, τ, depends on the complexity of the transition density functions and on the sampling rule, as well as on the choice of the random number generator (rng).

It is proved [Gurov and Whitlock (2002); Gurov *et al.* (2002)] that the stochastic error has order $O(\exp(c_2 t)/N^{1/2})$, where t is the evolution time and c_2 is a constant depending on the kernels of the quantum kinetic equation under consideration. This estimate shows that when t is fixed and $N \to \infty$ the error decreases, but for large t the factor $\exp(c_2 t)$ looks ominous. Therefore, the algorithm solves an NP-hard problem concerning

the evolution time. To solve this problem for long evolution times with small stochastic error we have to combine both MC variance reduction techniques and distributed or parallel computations.

As it was shown in Chapter 9, MC algorithms are very convenient for parallel implementations on parallel computer systems, because every value of the MC estimator can be done independently and simultaneously (see also [Dimov and Tonev (1993a)]). Although MC algorithms are well suited to parallel computation, there are a number of potential problems. The available computers can run at different speeds; they can have different user loads on them; one or more of them can be down; the random number generators that they use can run at different speeds; etc. On the other hand, these random number generators must produce independent and non-overlapping random sequences. Thus, the parallel realization of the MC algorithms is not a trivial task on different parallel computer systems.

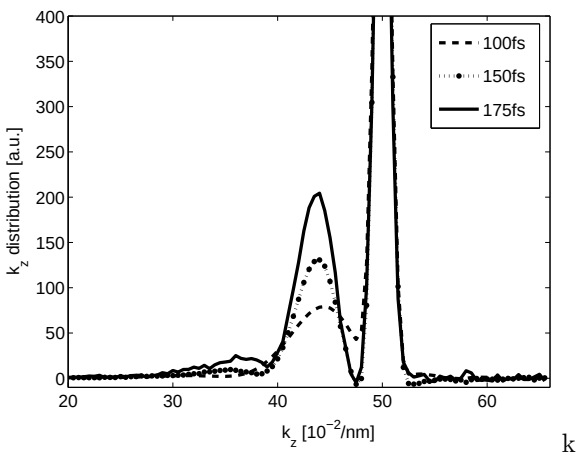


Fig. 10.5 Energy relaxation of the highly non-equilibrium initial condition. At $T = 0$ classical electrons form exact replicas of the initial condition towards low energies. The quantum solution shows broadening of the replicas. Electrons appear in the classically forbidden region above the initial condition. Here the variance of the solution for 175 fs is still high with respect to the solutions for shorter times, obtained with the same computational efforts.

By using the Grid infrastructure provided by the EGEE project middleware[2] [EGEE (2003)] we were able to reduce the computing time of the

[2]The Enabling Grids for E-sciencE (EGEE) project is funded by the European Commission and aims to build on recent advances in Grid technology and develop a service Grid infrastructure. For more information see http://www.eu-egee.org/.

MC algorithm under consideration. The simulations are parallelized on the existing Grid infrastructure by splitting the underlying random number sequences. The numerical results presented in Figure 10.5 are obtained for zero temperature and GaAs material parameters: the electron effective mass is 0.063, the optimal phonon energy is $36meV$, the static and optical dielectric constants are $\varepsilon_s = 10.92$ and $\varepsilon_\infty = 12.9$. The initial condition is a product of two Gaussian distributions of the energy and space. The k_z distribution corresponds to a generating laser pulse with an excess energy of about $150meV$. This distribution was estimated for 130 points in the interval $(0, 130)$, where $Q_2 = 66 \times 10^7 m^{-1}$. The z distribution is centered around zero (see Figures 10.6 and 10.7) and it is estimated for 400 points in the interval $(-Q_1, Q_1)$, where $Q_1 = 400 \times 10^9 m^{-1}$. The side a of the wire is chosen to be 10 nanometers.

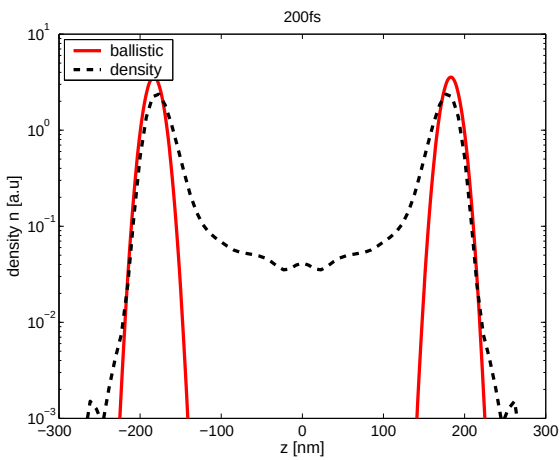


Fig. 10.6 Electron density along the wire after 200 fs. The ballistic curve outlines the largest distance which can be reached by classical electrons. The quantum solution reaches larger distances due to the electrons scattered in the classically forbidden energy region.

In our research, the MC algorithm has been implemented in C^{++}. The SPRNG library has been used to produce independent and non-overlapping random sequences [SPRNG (2007)]. Successful tests of the algorithm were performed at the Bulgarian SEE-GRID[3]. The MPI implementation was MPICH 1.2.6, and the execution is controlled from the Computing Element via the Torque batch system.

[3]South Eastern European Grid-enabled eInfrastructure Development (SEE-GRID) project is funded by the European Commission (see http://www.see-Grid.eu/.)

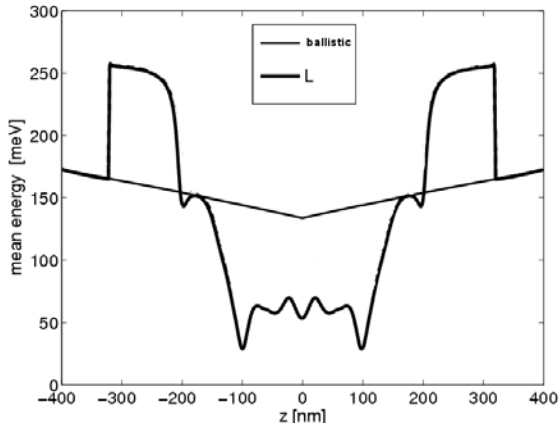


Fig. 10.7 Energy density after 175 fs evolution time. A comparison with the ballistic curve shows that the mean kinetic energy per particle is lower in the central region. On contrary hot electrons reside in the regions apart from the center. These electrons are faster than the ballistic ones and thus cover larger distances during the evolution.

Table 10.1 The CPU time (measured in seconds) for all points (in which the physical quantities are estimated), the speed-up, and the parallel efficiency. The number of Markov chains is $N = 10^5$. The evolution time is $100\,fs$.

Number of CPUs	CPU Time (s)	Speed-up	Parallel Efficiency
2	9790	-	-
4	4896	1.9996	0.9998
6	3265	2.9985	0.9995

The timing results for evolution time t=100 femtoseconds are shown in Table 10.1. The parallel efficiency is close to 100%. The results presented in Table 10.1 show a very high scalability of the algorithm. The scalability issue is very important for large-scale computational tasks running on Grids. Another important aspect of this particular Grid application is that the output results are not be very sensitive to possible lost information from some of the subtasks.

10.5 Conclusion

The Boltzmann transport is analysed from the point of view of numerical Monte Carlo methods. Convergency proof is given. Our approach is a base

for developing models for solving equations generalizing the BE towards quantum transport.

The generalized Wigner function provides a convenient instrumentality for derivation of quantum-kinetic models of the electron-phonon interaction. The corresponding hierarchy of Wigner function elements can be truncated at different levels giving rise to closed equations for the electron system. Derived are the inhomogeneous counterparts of the Levinson and Barker-Ferry equations, which describe the femtosecond evolution of local electron packets in a quantum wires. Basic are the hypotheses for an initially decoupled system, equilibrium phonons and the Markov approximation.

The physical aspects of the set of the incorporated assumptions is discussed. In particular it is argued that the relevance of both models is bounded at the long time limit. The solutions of the equations are rich of quantum effects already at the case of zero electric field. Along with the collisional broadening and retardation, an effect of ultrafast spatial transport is observed.

A quantum-kinetic model for the evolution of an initial electron distribution in a quantum wire has been introduced in terms of the electron Wigner function. The physical quantities, expressed as functionals of the Wigner function are evaluated within a stochastic approach. The developed MC method is characterized by the typical for quantum algorithms computational demands. The stochastic variance grows exponentially with the evolution time. Grid technologies are implemented, which allow to obtain simulation results for times up to 175 femtoseconds.

This page intentionally left blank

Appendix A

Jumps on Mesh Octahedra Monte Carlo

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 $x_i = (i_1 h, I_2 h, i_3 h)$ is

$$\Lambda_1(u_i) - 6u_i = 0, \tag{A.3}$$

where

$$\begin{aligned}\Lambda_1(u_i) = {} & u((i_1+1)h, i_2 h, i_3 h) + u((i_1-1)h, i_2 h, i_3 h) + u(i_1 h, (i_2+1)h, i_3 h) \\ & + u(i_1 h, (i_2-1)h, i_3 h) + u(i_1 h, i_2 h, (i_3+1)h) + u(i_1 h, i_2 h, (i_3-1)h),\end{aligned} \tag{A.4}$$

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 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 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 A.2. *For an arbitrary integer k it follows that*

$$\tilde{\Lambda}_k(u_i) = \begin{cases} \displaystyle\sum_{l=0}^{(k-3)/2} (-1)^l \left(12\Lambda_{k-2l-1}(u_i) - \overline{\Lambda}_{k-2l-1}(u_i)\right) + (-1)^{[k/2]}\tilde{\Lambda}_1(u_i), \\ \qquad\qquad\qquad\qquad\qquad\qquad k \quad \textit{odd}, \\ \displaystyle\sum_{l=0}^{(k-2)/2} (-1)^l \left(12\Lambda_{k-2l-1}(u_i) - \overline{\Lambda}_{k-2l-1}(u_i)\right) + (-1)^{[k/2]}\tilde{\Lambda}_0(u_i), \\ \qquad\qquad\qquad\qquad\qquad\qquad k \quad \textit{even}, \end{cases} \tag{A.11}$$

where [t] means the integer part of t, and

$$\begin{aligned}\overline{\Lambda}_k(u_i) = {} & u(i_1+k,i_2+1,i_3-1)+u(i_1+k,i_2+1,i_3+1)+u(i_1+k,i_2-1,i_3+1)\\ & +u(i_1+k,i_2-1,i_3-1)+u(i_1-k,i_2+1,i_3-1)+u(i_1-k,i_2+1,i_3+1)\\ & +u(i_1-k,i_2-1,i_3+1)+u(i_1-k,i_2-1,i_3-1)+u(i_1+1,i_2+k,i_3-1)\\ & +u(i_1-1,i_2+k,i_3-1)+u(i_1-1,i_2+k,i_3+1)+u(i_1+1,i_2+k,i_3+1)\\ & +u(i_1+1,i_2-k,i_3-1)+u(i_1-1,i_2-k,i_3+1)+u(i_1-1,i_2-k,i_3-1)\\ & +u(i_1+1,i_2-k,i_3+1)+u(i_1+1,i_2-1,i_3+k)+u(i_1+1,i_2+1,i_3+k)\\ & +u(i_1-1,i_2+1,i_3+k)+u(i_1-1,i_2-1,i_3+k)+u(i_1+1,i_2-1,i_3-k)\\ & +u(i_1+1,i_2+1,i_3-k)+u(i_1-1,i_2+1,i_3-k)+u(i_1-1,i_2-1,i_3-k).\end{aligned}$$

Proof. 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). □

Lemma A.3. *For an arbitrary integer k the following formula holds:*

$$\overline{\Lambda}_k(u_i) = \begin{cases} 8\displaystyle\sum_{l=0}^{(k-3)/2} (-1)^l \Lambda_{k-2l-1}(u_i) + (-1)^{[k-2]}\overline{\Lambda}_1(u_i),\ k \quad \textit{odd}, \\ 8\displaystyle\sum_{l=0}^{(k-2)/2} (-1)^l \Lambda_{k-2l-1}(u_i) + (-1)^{[k-2]}\overline{\Lambda}_0(u_i),\ k \quad \textit{even}, \end{cases} \tag{A.13}$$

Proof. 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). □

Proof. **of Theorem A.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

$$u_i = \frac{1}{36}\left[\Lambda_{n+1}(u_i) + 30u_i\right] \quad or \quad u_i = \frac{1}{6}\Lambda_{n+1}(u_i). \tag{A.17}$$

The case when $k = 2m$ is similar. □

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 **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).

This page intentionally left blank

Appendix B

Performance Analysis for Different Monte Carlo Algorithms

Here all the results for the values of interest are summarized.

B.1. Algorithm $\mathcal{A}$ $\quad (f(x) \equiv 0, \quad p = (c_{0.5}\sigma(\theta)/\epsilon)^2)$

$$ET_1(\mathcal{A}) = \tau\left((k+1+\gamma_A)l_A) + (n+1+\gamma_L)l_L\right)\frac{1}{4\epsilon}\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$ET_{pipe}(\mathcal{A}) = \tau(s+k+l_A+\gamma_A+(n+1+\gamma_L)l_L)\frac{1}{4\epsilon}\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$ET_p(\mathcal{A}) = \tau\left((k+1+\gamma_A)l_A + (n+1+\gamma_L)l_L\right)\frac{1}{4\epsilon},$$

$$ET_{2np}(\mathcal{A}) = \tau\left((k+1+\gamma_A)l_A + 3l_L\right)\frac{1}{4\epsilon},$$

$$S_{pipe}(\mathcal{A}) = \left(1+\frac{k+1+\gamma_A}{n+1+\gamma_L}\frac{l_A}{l_L}\right)\Big/\left(1+\frac{s+k+l_A+\gamma_A}{n+1+\gamma_L}\frac{1}{l_L}\right),$$

$$S_p(\mathcal{A}) = p,$$

$$S_{2np}(\mathcal{A}) = p\left(1+\frac{n+1+\gamma_L}{k+1+\gamma_A}\frac{l_L}{l_A}\right)\Big/\left(1+\frac{3}{k+1+\gamma_A}\frac{l_L}{l_A}\right),$$

$$E_p(\mathcal{A}) = 1,$$

$$E_{2np}(\mathcal{A}) = \frac{1}{2n}\left(1+\frac{n+1+\gamma_L}{k+1+\gamma_A}\frac{l_L}{l_A}\right)\Big/\left(1+\frac{3}{k+1+\gamma_A}\frac{l_L}{l_A}\right).$$

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 $\mathcal{C}$ $\quad (f(x)\equiv 0, \quad n=3, \quad p=(c_{0.5}\sigma(\theta)/\epsilon)^2)$

$$ET_1(\mathcal{C}) = \tau\left((2k+\gamma_A+q_A)l_A\right)+(\gamma_L+1)l_L)\,c\ln\epsilon\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$ET_{pipe}(\mathcal{C}) = \tau\left(s+2k+\gamma_A+q_A+l_A-1+(\gamma_L+1)l_L\right)c\ln\epsilon\left(c_{0.5}\frac{\sigma(\theta)}{\epsilon}\right)^2,$$

$$ET_p(\mathcal{C}) = \tau\left((k+\gamma_A+q_A)l_A+(\gamma_L+1)l_L\right)c\ln\epsilon,$$

$$ET_{6p}(\mathcal{C}) = \tau\left((2k+\gamma_A+q_A)l_A+(\gamma_L+1)l_L\right)c\ln\epsilon,$$

$$S_{pipe}(\mathcal{C}) = \left(1+\frac{2k+\gamma_A+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right)\Big/\left(1+\frac{s+2k+\gamma_A+q_A+l_A-1}{\gamma_L+1}\frac{1}{l_L}\right),$$

$$S_p(\mathcal{C}) = p,$$

$$S_{6p}(\mathcal{C}) = p\left(1+\frac{2k+\gamma_L+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right)\Big/\left(1+\frac{k+\gamma_A+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right),$$

$$E_p(\mathcal{C}) = 1,$$

$$E_{6p}(\mathcal{C}) = \frac{1}{6}\left(1+\frac{2k+\gamma_L+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right)\Big/\left(1+\frac{k+\gamma_A+q_A}{\gamma_L+1}\frac{l_A}{l_L}\right).$$

Appendix C

Sample Answers of Exercises

(1) **Random Variables**

(a)

$$E\gamma^3 = \int_0^1 x^3 p(x)dx \quad \text{Since } \int_0^1 p(x)dx = 1 \quad \rightarrow p(x) \equiv 1.$$

$$E\gamma^3 = \int_0^1 x^3 dx = \frac{x^4}{4}\Big|_0^1 = \frac{1}{4}$$

$$D\gamma^3 = E\gamma^6 - (E\gamma^3)^2$$

$$E\gamma^6 = \int_0^1 x^6 dx = \frac{x^7}{7}\Big|_0^1 = \frac{1}{7}$$

$$D\gamma^3 = \frac{1}{7} - \left(\frac{1}{4}\right)^2 = \frac{1}{7} - \frac{1}{16} = \frac{9}{112} \approx 0.08036$$

(b)

$$\gamma \in [0, 1]$$

Divide the interval $[0, 1]$ into subintervals of length $\Delta_i = p_i = \frac{1}{n}$. Since $\sum_{i=1}^n p_i = 1$, $\sum_{i=1}^n \Delta_i = 1$

$$P\{\xi = x_i\} = P\{\gamma \in \Delta_i\} = \Delta_i = \frac{1}{n} = p_i.$$

Thus, $P\{\xi = x_i\} = \frac{1}{n}$.

(2) **Plain Monte Carlo Algorithm**

$$
\begin{aligned}
I = & \int_0^1 x^{1/5} dx; \quad p(x) \equiv 1 \quad \text{because} \quad \int_0^1 p(x) dx = 1. \\
I = & \int_0^1 x^{1/5} dx = \frac{5}{6} x^{\frac{6}{5}} \Big|_0^1 = \frac{5}{6}; \\
\theta = & \ \gamma^{1/5}, \quad \text{where } \gamma \text{ is a c.u.d.r.v. in } [0,1]
\end{aligned}
$$

$$
\begin{aligned}
D\theta = & \ E\theta^2 - (E\theta)^2 \\
E\theta^2 = & \int_0^1 x^{2/5} dx = \frac{5}{7} x^{\frac{7}{5}} \Big|_0^1 = \frac{5}{7} \approx 0.7143 \\
D\theta = & \ \frac{5}{7} - \left(\frac{5}{6}\right)^2 = \frac{5}{7} - \frac{25}{36} = \frac{5}{252} \approx 0.01984 << I^2
\end{aligned}
$$

(3) **Importance Sampling Algorithm**

$$
\begin{aligned}
\hat{p} = & \ c|f(x)| \quad \text{Since} \int_0^1 \hat{p}(x) dx = 1 \\
c = & \left[\int_0^1 |e^x| dx \right]^{-1} = \frac{1}{e-1}. \\
\hat{p} = & \ \frac{1}{e-1} e^x
\end{aligned}
$$

According to the definition of the *Importance Sampling Algorithm*

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

Thus,

$$
\begin{aligned}
\theta_0(x) = & \ \frac{e^x}{e^x}(e-1) = e-1, \quad \text{for } x \in [0,1]. \\
D\theta_0(x) = & \int_0^1 (e-1)^2 dx - (e-1)^2 = 0
\end{aligned}
$$

(4) **Symmetrization of the Integrand**

Since

$$\int_0^1 p(x)dx = 1 \quad p(x) \equiv 1 \text{ and } f(x) = e^x.$$

$$I = \int_0^1 e^x dx = e^x|_0^1 = e - 1, \quad f_1(x) = \frac{1}{2}(e^x + e^{1-x})$$

The symmetrized random variable is

$$\theta' = \frac{1}{2}\left(e^{\gamma} + e^{1-\gamma}\right), \text{ where } \gamma \text{ is a c.u.d.r.v. in } [0,1].$$
$$E\theta' = \frac{1}{2}\int_0^1 \left(e^x + e^{1-x}\right) dx = \frac{1}{2}\left[e^x|_0^1 - e^{1-x}|_0^1\right] = \frac{1}{2}(e + e - 2) = e - 1$$
$$D\theta' = E(\theta')^2 - (E\theta')^2$$
$$D\theta' = \frac{1}{4}[2e - (e-1)(3e-5)] \approx 0.00392 << (e-1)^2 = I^2$$

(5) **Separation of the principle part**

a)

$$I = \int_0^1 e^x dx = e^x|_0^1 = e - 1,$$
$$I' = \int_0^1 h(x)p(x)dx = \int_0^1 x dx = \left.\frac{x^2}{2}\right|_0^1 = \frac{1}{2}$$
$$\theta' = f(x) - h(x) + I'; \quad \theta' = e^x - x + \frac{1}{2}$$

b)

$$E\theta' = \int_0^1 \left(e^x - x + \frac{1}{2}\right) dx = e^x|_0^1 - \left.\frac{x^2}{2}\right|_0^1 + \frac{1}{2} = e - 1 - \frac{1}{2} + \frac{1}{2} = e - 1$$

c)

$$
\begin{aligned}
D\theta' &= E(\theta')^2 - (E\theta')^2 \\
E(\theta')^2 &= \int_0^1 \left(e^x - x + \frac{1}{2}\right)^2 dx \\
&= \int_0^1 \left[(e^x - x)^2 + e^x - x + \frac{1}{4}\right] dx \\
&= \int_0^1 \left(e^{2x} - 2e^x x + x^2\right) dx + \int_0^1 (e^x - x)dx + \frac{1}{4} \\
&= \frac{1}{2}\, e^{2x}\big|_0^1 - 2J + \frac{x^3}{3}\bigg|_0^1 + e^x\big|_0^1 - \frac{x^2}{2}\bigg|_0^1 + \frac{1}{4} \\
&= \frac{1}{2}(e^2 - 1) + \frac{1}{3} + (e - 1) - \frac{1}{2} + \frac{1}{4} - 2J \\
&= \frac{1}{2}(e^2 - 1) + (e - 1) - 2J + \frac{4 - 6 + 3}{12} \\
&= \frac{1}{2}(e^2 - 1) + (e - 1) - 2J + \frac{1}{12} \\
J &= \int_0^1 xe^x dx = \int_0^1 x d(e^x) = xe^x\big|_0^1 - \int_0^1 e^x dx = e - (e - 1) = 1 \\
E(\theta')^2 &= \frac{1}{2}(e^2 - 1) + (e - 1) - 2 + \frac{1}{12} \\
&= \frac{1}{2}(e - 1)(e + 1 + 2) - \frac{23}{12} \\
&= \frac{1}{2}(e - 1)(e + 3) - \frac{23}{12} \\
D\theta' &= \frac{1}{2}(e - 1)(e + 3) - \frac{23}{12} - (e - 1)^2 \\
&= \frac{1}{2}(e - 1)\left[(e + 3) - 2(e - 1)\right] - \frac{23}{12} \\
&= \frac{1}{2}(e - 1)(5 - e) - \frac{23}{12}
\end{aligned}
$$

Thus,

$$
D\theta' = (e - 1)(5 - e)/2 - \frac{23}{12} \approx 0.0437 << (e - 1)^2 = I^2.
$$

(6) **Integration on Subdomain Monte Carlo Algorithm**

$$
\begin{aligned}
I = & \int\int_{\Omega}(2-y)dxdy = \int_0^1 dx \int_0^{1+\frac{x}{4}}(2-y)dy = -\int_0^1 \frac{(2-y)^2}{2}\Big|_0^{1+\frac{x}{4}} dx \\
= & \frac{1}{2}\int_0^1 \left[4-\left(1-\frac{x}{4}\right)^2\right]dx = 2-\frac{1}{2}\int_0^1\left(1-\frac{x}{2}+\frac{x^2}{16}\right)dx \\
= & \ 2-\frac{1}{2}+\frac{1}{8}-\frac{1}{2\times 48} = \frac{3}{2}+\frac{1}{8}-\frac{1}{96} = \frac{13}{8}-\frac{1}{96} = \frac{155}{96} \approx 1.61458
\end{aligned}
$$

To find $p(x,y)$ we have to take into account that

$$\int_0^1\int_0^{1+1/4} p(x,y)dxdy = 1$$

Since $S_\Omega = 1\frac{1}{8} = \frac{9}{8}$, we have

$$p(x,y) = \frac{8}{9}, \quad (x,y)\in\Omega.$$

Then

$$I = \int\int_{\Omega} \underbrace{\frac{9}{8}(2-y)}_{f(x,y)} . \underbrace{\frac{8}{9}}_{p(x,y)} dxdy.$$

$$
\begin{aligned}
I' = & \int_0^1\int_0^1 (2-y)dxdy = \frac{3}{2}; \\
c = & \int_0^1\int_0^1 p(x,y)dxdy = \int_0^1\int_0^1 \frac{8}{9}dxdy = \frac{8}{9}.
\end{aligned}
$$

Now one can define the r.v. θ' for *Integration on Subdomain* Monte Carlo:

$$\theta' = \ I' + (1-c)f(\xi'),$$

where the random point $\xi' \in \Omega_1 = \Omega \setminus \Omega'$ has a density function

$$p_1(x,y) = \frac{p(x,y)}{1-c} = \frac{8/9}{1-8/9} = \frac{8.9}{9.1} = 8.$$

$$
\begin{aligned}
D\theta' &= E(\theta')^2 - (E\theta')^2 \quad \left(D\theta' = (1-c)^2 Df(\xi')\right) \\
Df(\xi') &= E\left(f(\xi')\right)^2 - \left(Ef(\xi')\right)^2 \\
Ef(\xi') &= \int_0^1 \int_1^{1+x/4} \frac{9}{8}(2-y).8dxdy = 9\int_0^1 dx \int_1^{1+x/4} (2-y)dy \\
&= -\frac{9}{2}\int_0^1 (2-y)^2\big|_1^{1+x/4} dx = \frac{9}{2}\int_0^1 \left[1-\left(1-\frac{x}{4}\right)^2\right] dx \\
&= \frac{9}{2}\left[1-\int_0^1 \left(1-\frac{x}{2}+\frac{x^2}{16}\right) dx\right] \\
&= \frac{9}{2}\left[1-1+\frac{1}{4}-\frac{1}{48}\right] = \frac{3}{2}.\frac{11}{16} = \frac{33}{32} = 1\frac{1}{33} \approx 1.03125
\end{aligned}
$$

$$
\begin{aligned}
E(f(\xi'))^2 &= \int_0^1 \int_1^{1+x/4} \frac{81}{64}(2-y)^2.8dxdy = \frac{81}{8}\int_0^1 dx \int_1^{1+x/4} (2-y)^2 dy \\
&= -\frac{81}{24}\int_0^1 (2-y)^3\big|_1^{1+x/4} dx = \frac{27}{8}\int_0^1 \left[1-\left(1-\frac{x}{4}\right)^3\right] dx \\
&= \frac{27}{8}\left[1-\int_0^1 \left(1-\frac{x}{4}\right)^3 dx\right] \\
&= \frac{27}{8}\left[1+\left(1-\frac{x}{4}\right)^4\Big|_0^1\right] = \frac{27}{8}\left[1+\left(\frac{3}{4}\right)^4 - 1\right] \\
&= \frac{27}{8}\times\frac{81}{256} = \frac{3^7}{2.4^5} \approx 1.0678
\end{aligned}
$$

$$
\begin{aligned}
Df(\xi') &= \frac{2187}{2048} - \frac{1089}{1024} = \frac{9}{2048} \approx 0.0043945 \\
D\theta' &= (1-c)^2 Df(\xi') = \frac{1}{9^2}\times\frac{9}{2048} = \frac{1}{18432} \approx 5.4253.10^{-5} << I^2
\end{aligned}
$$

For the *Plain Monte Carlo Algorithm* we have:

$$
\begin{aligned}
\theta &= f(\xi) = \frac{9}{8}(2-\xi) \\
E\theta &= \iint_\Omega \underbrace{\frac{9}{8}(2-y)}_{f(x,y)}.\underbrace{\frac{8}{9}}_{p(x,y)} dxdy = \frac{155}{96} \approx 1.61458
\end{aligned}
$$

$$\begin{aligned}
E\theta^2 &= \int\int_{\Omega} \frac{81}{64}(2-y)^2.\frac{8}{9}dxdy \\
&= \frac{9}{8}\int_0^1 dx \int_0^{1+x/4} (2-y)^2 dy = -\frac{9}{24}\int_0^1 (2-y)^3\Big|_0^{1+x/4} dx \\
&= \frac{3}{8}\int_0^1 \left[8-\left(1-\frac{x}{4}\right)^3\right] dx = 3-\frac{3}{8}\int_0^1 \left(1-\frac{x}{4}\right)^3 dx \\
&= 3+\frac{3}{8}\left(1-\frac{x}{4}\right)^4\Big|_0^1 = 3+\frac{3}{8}\left[-1+\left(\frac{3}{4}\right)^4\right] \\
&= 3-\frac{3}{8}+\frac{3^5}{8.4^4} = \frac{21}{8}+\frac{3^5}{8.4^4} = \frac{21.4^4+3^5}{8.4^4} \\
&= \frac{5376+243}{8.4^4} = \frac{5619}{2048} \approx 2.74365
\end{aligned}$$

Thus

$$D\theta = \frac{5619}{8.4^4} - \frac{155^2}{(4^2.3.2)^2} = \frac{2521}{18432} \approx 0.136773.$$

Now, it's easy to check that

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

Indeed,

$$5.4253.10^{-5} \leq \frac{1}{9} \times 0.136773.$$

This page intentionally left blank

Appendix D

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 point x and 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 r^{th} derivatives, such that

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

where

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

is the r^{th} derivative, $r = (r_1, r_2, \ldots, r_d), |r| = r_1 + r_2 + \cdots + r_d$, and $D_i = \frac{\partial}{\partial x_{(i)}}$
- 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 i^{th} row and j^{th} 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 - k^{th} 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
- $T_i = (\xi_0 \to \xi_1 \to \cdots \to \xi_i)$ - random trajectory
- $\bar{\xi}_n = \frac{1}{n}\sum_{i=1}^n \xi_i$ - mean value of n realizations of the r.v. ξ
- r_n - probable error defined as a value r_n for which

$$Pr\left\{|\bar{\xi}_n - J| \le r_n\right\} = \frac{1}{2} = Pr\left\{|\bar{\xi}_n - J| \ge r_n\right\}$$

Bibliography

Alexandrov, V., Atanassov, E. and Dimov, I. (2004). Parallel Quasi-Monte Carlo methods for linear algebra problems, *Monte Carlo Methods and Applications* **10**, 3-4, pp. 213–219.

Andrieu, C., de Freitas, N., Doucet, A. and Jordan, M. (2003). An introduction to MCMC for machine learning, *Machine Learning* **50**, 1-2, pp. 5–43, URL `citeseer.ist.psu.edu/andrieu03introduction.html`.

Arioli, M., Ptak, V. and Strakos, Z. (1998). Krylov sequences of maximal lenght and convergence of GMRES, *BIT* **38**, pp. 636–643.

Arnoldi, W. E. (1951). The principle of minimized iterations in the solution of the matrix eigenvalue problem, *Quart. Appl. Math.* **9**, pp. 17–29.

Atanassov, E. and Dimov, I. (1999). A new Monte Carlo method for calculating integrals of smooth functions, *Monte Carlo Methods and Applications* **2**, 3-4, pp. 149–167.

Bai, Z., Demmel, J., Dongarra, J., Ruhe, A. and van der Vorst, H. (2000). *Templates for the Solution of Algebraic Eigenvalue Problems: A Practical Guide* (SIAM, Philadelphia).

Bakhvalov, N. S. (1959). On the approximate computation of multiple integrals, *Vestnik Moscow State University* **4**, 3-4, pp. 3–18.

Bakhvalov, N. S. (1961). Average estimation of the remainder therm of quadrature formulas, *USSR Comput. Math. and Math. Phys.* **1(1)**, 3-4, pp. 64–77.

Bakhvalov, N. S. (1964). On the optimal estimations of convergence of the quadrature processes and integration methods, in *Numerical methods for solving differential and integral equations* (Nauka, Moscow), pp. 5–63.

Beattie, C., Embree, M. and Rossi, J. (2004). Convergence of restarted Krylov subspaces to invariant subspaces, *SIAM J. Matrix Anal. Appl.* **25**, 4, pp. 1074–1109.

Berg, B. A. (2004). *Markov Chain Monte Carlo Simulations and Their Statistical Analysis* (World Scientific, Singapore).

Binder, K. and Heermann, D. W. (1998). *Monte Carlo Simulation in Statistical Physics: An Introduction*, Springer Series in Solid-State Sciences, 80 (Springer–Verlag).

Bitzadze, A. V. (1982). *Equations of the Mathematical Physics* (Nauka, Moscow).

Bradley, P. and Fox, B. (1980). Implementing Sobol's Quasi Random Sequence Generator, *ACM Trans. Math. Software* **14(1)**, pp. 88–100.

Bremaud, P. (1999). *Markov chains: Gibbs Fields, Monte Carlo Simulation, and Queues*, Texts in Applied Mathematics, 31 (Springer–Verlag).

Buyya, R. (2007). Grid computing info centre (grid infoware), `http://www.gridcomputing.com/`.

Chen, M. H. and Shao, Q. M. (2000). *Monte Carlo Methods in Bayesian Computation*, Springer Series in Statistics (Springer–Verlag).

Chib, S. and Greenberg, E. (1995). Understanding the Metropolis-Hastings algorithm, *American Statistician* **49**, 4, pp. 327–335.

Curtiss, J. H. (1954). Monte Carlo methods for the iteration of linear operators, *J. Math. Phys.* **32**, pp. 209–232.

Curtiss, J. H. (1956). 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, in *Proc. Symposium on Monte Carlo Methods* (John Wiley and Sons), pp. 191–233.

Davies, A. (2004). Computational intermediation and the evolution of computation as a commodity, *Applied Economics* **36**, pp. 1131–11142.

Davis, P. and Rabinowitz, P. (1984). *Methods of numerical integration. Comp. Sci. and Appl. Math.*, 2nd edn. (Academic Press).

de Boor, C. (2001). *Practical guide for splines*, Applied Mathematical Sciences , Vol. 27 (Springer).

de Buffon, G. C. (1777). Essai d'arithmetique morale, *Supplement a l'Histoire Naturalle* **4**.

Dimov, I. (1989). Efficiency estimator for Monte Carlo algorithms, in *Numerical Methods and Applications, II Intern. Conf. on Numerical Methods and Appl.* (Publ. House of the Bulg. Acad. Sci., Sofia), pp. 111–115.

Dimov, I. (1991). Minimization of the probable error for some Monte Carlo methods, in *Proc. of the Summer School on* ***Mathematical Modelling and Scientific Computations*** (Publ. House of the Bulg. Acad. Sci., Sofia), pp. 159–170.

Dimov, I. (1994). Efficient and overconvergent Monte Carlo methods, in I. Dimov and O. Tonev (eds.), *Advances in Parallel Algorithms* (IOS Press, Amsterdam), pp. 100–111.

Dimov, I. (2000). Monte Carlo algorithms for linear problems, *Pliska* **13**, pp. 57–77.

Dimov, I. and Alexandrov, V. (1998). A new highly convergent Monte Carlo method for matrix computations, *Mathematics and Computers in Simulation* **47**, pp. 165–181.

Dimov, I., Alexandrov, V. and Karaivanova, A. (2001). Parallel resolvent Monte Carlo algorithms for linear algebra problems, *Mathematics and Computers in Simulation* **55**, pp. 25–35.

Dimov, I. and Atanassov, E. (2007). Exact error estimates and optimal randomized algorithms for integration, in T. Boyanov (ed.), *Proceedings of* ***NMA 2006***, Vol. 4310 (Springer LNCS), pp. 131–139.

Dimov, I., Dimov, T. and Gurov, T. (1997). A new iterative Monte Carlo ap-

proach for inverse matrix problem, *Journal of Computational and Applied Mathematics* **92**, pp. 15–35.

Dimov, I. and Gurov, T. (1993). Parallel Monte Carlo algorithms for calculation integrals, in K. Boyanov (ed.), *Proceedings of* ***WP&DP***, pp. 426–434.

Dimov, I. and Gurov, T. (2000). Monte Carlo algorithm for solving integral equations with polynomial non-linearity. parallel implementation, *Pliska* **13**, pp. 117–132.

Dimov, I. and Karaivanova, A. (1994). Monte Carlo parallel algorithms, in *Proc. III Intern. Conf. on* ***Applications of Supercomputers in Engineering - ASE/93,*** *Comp. Mech. Publ.* (Elsevier, Appl. Sci., London, New York), pp. 479–495.

Dimov, I. and Karaivanova, A. (1996). Iterative Monte Carlo algorithms for linear algebra problems, in *Numerical Analysis and its Applications. Proc. First Workshop on Numerical Analysis and Applications, Rousse, Bulgaria, June 24-27, 1996* (Springer LNCS # 1196), pp. 150–160.

Dimov, I. and Karaivanova, A. (1998). Parallel computations of eigenvalues based on a Monte Carlo approach, *Monte Carlo Method and Applications* **4**, 1, pp. 33–52.

Dimov, I. and Karaivanova, A. (1999). A power method with Monte Carlo iterations, in O. Iliev, M. Kaschiev, B. Sendov and P. Vassilevski (eds.), *Recent Advances in Numerical Methods and Applications* (World Scientific), pp. 239–247.

Dimov, I., Karaivanova, A., Kuchen, H. and Stoltze, H. (1996). Monte Carlo algorithms for elliptic differential equations. data parallel functional approach, *Journal of Parallel Algorithms and Applications* **9**, pp. 39–65.

Dimov, I. and Tonev, O. (1989). Monte Carlo numerical methods with overconvergent probable error, in *Numerical Methods and Applications, Proc. 2nd Intern. Conf. on Numerical Methods and Appl.* (Publ. House of the Bulg. Acad. Sci., Sofia), pp. 116–120.

Dimov, I. and Tonev, O. (1992). Criteria estimators for speed-up and efficiency of some parallel Monte Carlo algorithms for different computer architectures, in B. Sendov and I. Dimov (eds.), *Proc.* ***WP & DP'91*** *(Sofia, April 16-19)* (North-Holland Publ. Co., Amsterdam), pp. 243–262.

Dimov, I. and Tonev, O. (1993a). Monte Carlo algorithms: performance analysis for some computer architectures, *J. of Computational and Applied Mathematics* **48**, pp. 253–277.

Dimov, I. and Tonev, O. (1993b). Random walk on distant mesh points Monte Carlo methods, *J. of Statistical Physics* **70**, 5/6, pp. 1333–1342.

Doroshkevich, A. and Sobol, I. (1987). Monte Carlo evaluation of integrals encountered in nonlinear theory of gravitational instability, *USSR Comput. Math. and Math. Phys.* **27(10)**, pp. 1577–1580.

Dubi, A. (2000). *Monte Carlo Applications in Systems Engineering* (Weley).

Dupach, V. (1956). Stochasticke pocetni metody, *Cas. Pro. Pest. Mat.* **81**, 1, pp. 55–68.

EGEE (2003). EGEE GRID middleware, `http://lcg.web.cern.ch/LCG/Sites/releases.html`.

Ermakov, S. M. (1967). On the admissibleness of Monte Carlo procedures, *Dokl. Acad. Nauk SSSR* **72**, 2, pp. 262– 264.

Ermakov, S. M. (1975). *Monte Carlo Methods and Mixed Problems* (Nauka, Moscow).

Ermakov, S. M. and Mikhailov, G. A. (1982). *Statistical Modelling* (Nauka, Moscow).

Ermakov, S. M., Nekrutkin, V. V. and Sipin, A. S. (1984). *Random processes for solving classical equations of mathematical physics* (Nauka, Moscow).

Ermakov, S. M. and Zolotukhin, V. G. (1960). Polynomial approximations in the Monte Carlo method, *Probability Theory and Appl.* **4**, 4, pp. 473–476.

Evans, M. J. and Swartz, T. (2000). *Approximating Integrals Via Monte Carlo and Deterministic Methods*, Oxford Statistical Science Series, 20 (Oxford University Press).

Fishman, G. S. (1995). *Monte Carlo: Concepts, Algorithms, and Applications* (Springer-Verlag, New York).

Foster, I. and Kesselman, C. (1998). Computational grids, in I. Foster and C. Kesselman (eds.), *The Grid: Blueprint for a New Computing Infrastructure* (Morgan Kaufmann Publishers, San Francisco, California), pp. 15–52.

Frensley, W. (1990). Boundary conditions for open quantum systems driven far from equilibrium, *Rev. Mod. Phys.* **62**, 3, pp. 745–791.

Gelfand, A. E. and Smith, A. F. M. (1990). Sampling-based approaches to calculating marginal densities, *J. American Statistical Association* **85**, pp. 398–409.

Gilks, W. R., Richardson, S. and Spiegelhalter, D. J. (1995). *Markov Chain Monte Carlo in Practice* (Chapman & Hall).

Godunov, S. K. (1991). Spectral portraits of matrices and criteria of spectrum dichotomy, in J. Herzberger and L. Atanasova (eds.), *International symposium on computer arithmetic and scientific computation* (Oldenburg, Germany, North-Holland).

Golub, G. H. and Van-Loan, C. F. (1983). *Matrix computations* (Johns Hopkins Univ. Press, Baltimore).

Golub, G. H. and Van-Loan, C. F. (1996). *Matrix computations*, 3rd edn. (Johns Hopkins Univ. Press, Baltimore).

Gould, H. and Tobochnik, J. (1989). *An Introduction to Computer Simulation Methods, Part 2, Applications to Physical Systems* (Addison Wesley).

Green, P. J. and Mira, A. (2001). Delayed rejection in reversible jump Metropolis–Hastings, *Biometrika* **88**, pp. 1035–1053.

Grimmett, G. R. and Stirzaker, D. R. (1992). *Probability and Random Processes* (Clarendon Press, Oxford).

Gurov, T. (1994). Monte Carlo methods for nonlinear equations, in I. Dimov (ed.), *Advances in Numerical Methods and Applications* (World Scientific, Singapore), pp. 127–135.

Gurov, T. and Dimov, I. (2004). A parallel Monte Carlo method for electron quantum kinetic equation, in *Lect. Notes in Comp. Sci.*, *LNCS*, Vol. 2907 (Springer-Verlang), pp. 151–161.

Gurov, T. and Whitlock, P. A. (2002). An efficient backward Monte Carlo estimator for solving of a quantum kinetic equation with memory kernel, *Mathematics and Computers in Simulation* **60**, pp. 85–105.

Gurov, T. V., Nedjalkov, M., Whitlock, P. A., Kosina, H. and Selberherr, S. (2002). Femtosecond relaxation of hot electrons by phonon emission in presence of electric field, *Physica B* **314**, pp. 301–304.

Haario, H., Saksman, E. and Tamminen, J. (2001). An adaptive Metropolis algorithm, *Bernoulli* **7**, pp. 223–242.

Haber, S. (1966). A modified Monte Carlo quadrature, *Math. of Comput.* **20**, 95, pp. 361–368.

Haber, S. (1967). A modified Monte Carlo quadrature, *Math. of Comput.* **21**, 99, pp. 388–397.

Hammersley, J. M. and Handscomb, D. C. (1964). *Monte Carlo methods* (John Wiley & Sons, inc., New York, London, Sydney, Methuen).

Harris, T. (1963). *The Theory of Branching Processes* (Springer–Verlag, Berlin).

Hastings, W. K. (1970). Monte Carlo sampling methods using Markov chains and their applications, *Biometrika* **57**, 1, pp. 97–109.

Haug, H. and Koch, S. W. (1994). *Quantum Theory of the Optical and Electronic Properties of Semiconductors*, 3rd edn. (World Scientific, Singapore).

Hellekalek, P. and Larcher, G. (1998). *Random and Quasi-Random Point Sets*, Lecture Notes in Statistics, 138 (Springer–Verlag).

Hoare, C. A. R. (1961). Partition: Algorithm 63; quicksort: Algorithm 64; and find: Algorithm 65, *Comm. ACM* **4**, 7, pp. 321–322.

Isaacsons, E. and Keller, H. B. (1994). *Analysis of Numerical Methods* (Courier Dover Publications).

Jacoboni, C. and Lugli, P. (1989). *The Monte Carlo method for semiconductor device simulation* (Springer-Verlag).

Jacoboni, C., Poli, P. and Rota, L. (1988). A new Monte Carlo technique for the solution of the Boltzmann transport equation, *Solid State Electronics* **31**, 3/4, pp. 523–526.

Jacoboni, C. and Reggiani, L. (1983). The Monte Carlo method for the solution of charge transport in semiconductors with applications to covalent materials, *Rev. Mod. Phys.* **55**, 3, pp. 645–705.

Jost, J. (2002). *Partial differential equations* (Springer-Verlag, New York).

Kahn, H. (1950a). Random sampling (Monte Carlo) techniques in neutron attenuation problems, *Nucleonics* **6**, 5, pp. 27–33.

Kahn, H. (1950b). Random sampling (Monte Carlo) techniques in neutron attenuation problems, *Nucleonics* **6**, 6, pp. 60–65.

Kalos, M. H. and Whitlock, P. A. (1986). *Monte Carlo Methods, Volume I: Basics* (Wiley, New York).

Kantorovich, L. W. and Akilov, G. P. (1982). *Functional analysis* (Pergamon Press, Oxford).

Kantorovich, L. W. and Krylov, V. I. (1964). *Approximate Methods of Higher Analysis* (Interscience, New York).

Karaivanova, A. and Dimov, I. (1998). Error analysis of an adaptive Monte Carlo method for numerical integration, *Mathematics and Computers in Simula-*

tion **47**, pp. 201–213.

Kimmel, M. and Axelrod, D. E. (2002). *Branching Processes in Biology* (Springer, New York).

Kloek, T. and van Dijk, H. (1978). Bayesian estimates of equation system parameter. an application of integration by Monte Carlo, *Econometrica* **46**, pp. 1–19.

Knuth, D. (1997). *The Art of Computer Programming, Volume 3: Sorting and Searching* (Addison-Wesley), pages 113-122 of section 5.2.2: Sorting by Exchanging.

Krommer, A. R. and Ueberhuber, C. W. (1998). *Computational integration* (SIAM).

Kurosawa, T. (1966). Monte Carlo calculation of hot electron problems, *J. Phys. Soc. Japan, Supl. (Proceedings of the International Conference on the Physics of Semiconductors)* **21**, pp. 424–426.

Landau, D. P. and Binder, K. (2000). *A Guide to Monte Carlo Simulations in Statistical Physics* (Cambridge University Press).

Lepage, G. P. (1978). A new algorithm for adaptive multidimensional integration, *Journal of Computational Physics* **27**, pp. 192–203.

Levitan, Y., Markovich, N., Rozin, S. and Sobol, I. (1988). On quasi-random sequences for numerical computations, *USSR Comput. math. and Math. Phys.* **28**, 5, pp. 755–759.

Liu, J. S. (2001). *Monte Carlo Strategies in Scientific Computing*, Springer Series in Statistics (Springer–Verlag).

MacKeown, P. (1997). *Stochastic Simulation in Physics* (National University of Singapore,Singapore).

Mahotkin, O. A. and Pirimkulov, M. I. (1984). Application of splines for some problems of statistical modelling, in G. A. Mikhailov (ed.), *Theory and Practice of Statistical Modelling* (Computing Center, Novosibirsk), pp. 43–53.

Maire, S. (2003a). An iterative computation of approximatons on Korobov-like spaces, *Journal of Computational and Applied Mathematics* **157**, pp. 261–281.

Maire, S. (2003b). Reducing variance using iterated control variates, *The Journal of Statistical Computation and Simulation* **71**, 1, pp. 1–29.

Maire, S. and DeLuigi, C. (2006). Quasi-Monte Carlo quadratures for multivariate smooth functions, *Applied Numerical Mathematics archive* **56**, Issue 2 (February 2006), pp. 146–162.

Manly, B. F. J. (1997). *Randomization, Bootstrap and Monte Carlo Methods in Biology* (Chapman & Hall).

Manno, I. (1999). *Introduction to the Monte Carlo Method* (Budapest: Akademiai Kiado).

Medvedev, I. N. and Mikhailov, G. A. (2007). A new criterion for finiteness of weight estimator variance in statistical simulation, Communication; the paper is prepared for publication.

Megson, G., Alexandrov, V. and Dimov, I. (1994). Systolic matrix inversion using a Monte Carlo method, *J. of Parallel Algorithms and Appl.* **3**, 3/4, pp. 311–

330.

Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. and Teller, E. (1953). Equations of state calculations by fast computing machines, *Journal of Chemical Physics* **21**, 6, pp. 1087–1092.

Metropolis, N. and Ulam, S. (1949). The Monte Carlo method, *J. of Amer. Statistical Assoc.* **44**, 247, pp. 335–341.

Mikhailov, G. A. (1987). *Optimization of Wieght Monte Carlo methods* (Nauka, Moscow).

Mikhailov, G. A. (1995). *New Monte Carlo Methods with Estimating Derivatives* (Utrecht, The Netherlands).

Mikhailov, G. A. and Sabelfeld, K. K. (1992). *Optimization of Weighted Monte Carlo Methods*, Springer Series in Computational Physics (Springer–Verlag).

Miranda, C. (1955). *Equasioni alle dirivate parziali di tipo ellittico* (Springer-Verlag, Berlin).

Monahan, J. F. (2001). *Numerical Methods of Statistics* (Cambridge University Press).

Morton, K. W. (1957). A generalization of the antithetic variate technique for evaluating integrals, *J. Math. Phys.* **36**, 3, pp. 289–293.

Muller, M. (1956). Some continuous Monte Carlo methods for the dirichlet problem, *Ann. Math. Statistics* **27**, 3, pp. 569–589.

Nedjalkov, M., Dimov, I., Rossi, F. and Jacoboni, C. (1996). Convergence of the Monte Carlo algorithm for the solution of the Wigner quantum-transport equation, *Journal of Mathematical and Computer Modeling* **23**, 8/9, pp. 159–166.

Nedjalkov, M. and Vitanov, P. (1989). Iteration approach for solving the Boltzmann equation with the Monte Carlo method, *Solid State Electronics* **32**, 10, pp. 893–896.

Newman, M. E. J. and Barkema, G. (1999). *Monte Carlo Methods in Statistical Physics* (Oxford University Press, Oxford).

Niederreiter, H. (1987). Point sets and sequences with small discrepancy, *Monatsh. Math.* **104**, pp. 273–337.

Niederreiter, H. (1992). *Random number generation and Quasi-Monte Carlo Methods. Number **63** in **CBMS-NSF Series in Applied Mathematics*** (SIAM, Philadelphia).

Nikolskiy, S. M. (1988). *Quadrature formulas* (Nauka, Moscow).

Novak, E. and Ritter, K. (1996). High dimensional integration of smooth functions over cubes, *Numerishche Mathematik* , pp. 1–19.

Phillips, A. and Price, P. J. (1977). Monte Carlo calculations on hot electron tails, *Applied Physics Letters* **30**, pp. 528–530.

Pitman, J. (1993). *Probability* (Springer-Verlag, New York).

Plateau, B. (1994). Apache: Algorithmique parallele et partagede charge, Tech. rep., Institut IMAG, Grenoble, France.

Press, W. P. and Farrar, G. R. (1990). Recursice stratified sampling for multidimensional Monte Carlo integration, *Computer in Physis* **4**, pp. 190–195.

Robert, C. P. and Casella, G. (2004). *Monte Carlo Statistical Methods*, 2nd edn.

(Springer-Verlag, New York).
Roberts, G. and Rosenthal, J. (2001). Optimal scaling for various Metropolis–Hastings algorithms, *Statistical Science* **16**, pp. 351–367.
Rossi, F., Haas, S. and Kuhn, T. (1994a). Ultrafast relaxation of photoexcited carriers: The role of coherence in the generation process, *Phys. Rev. Lett.* **72**, 1, pp. 152–155.
Rossi, F., Jacoboni, C. and Nedjalkov, M. (1994b). A Monte Carlo solution of the Wigner transport equation, *Semicond. Sci. Technol.* **9**, p. 934.
Rossi, F., Poli, P. and Jacoboni, C. (1992). The weighted Monte Carlo approach to electron transport in semiconductors, *Semicond. Sci. Technol.* **7**, pp. 1017–1035.
Rota, L., Jacoboni, C. and Poli, P. (1989). Weighted ensemble Monte Carlo, *Solid State Electronics* **32**, pp. 1417–1421.
Rubinstein, R. Y. (1981). *Simulation and the Monte Carlo Method* (Wiley & Sons, New York).
Rubinstein, R. Y. (1992). *Monte Carlo Optimization, Simulation and Sensitivity of Queuing Networks* (Krieger Publishing).
Sabelfeld, K. K. (1989). *Algorithms of the Method Monte Carlo for Solving Boundary Value Problems* (Nauka, Moscow).
Sabelfeld, K. K. (1991). *Monte Carlo Methods in Boundary Value Problems*, Springer Series in Computational Physics (Springer–Verlag).
Schilp, J., Kuhn, T. and Mahler, G. (1994). Electron-phonon quantum kinetics in pulse-excited semiconductors: Memory and renormalization effects, *Physical Review B* **50**, 8, pp. 5435–5447.
Schurer, R. (2003). A comparison between (Quasi-)Monte Carlo and cubature rule based methods for solving high-dimensional integration problems, *Mathematics and Computers in Simulation* **62**, pp. 509–517.
Sendov, B., Andreev, A. and Kjurkchiev, N. (1994). *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, New York).
Shaw, J. (1988). Aspects of numerical integration and summarization, *Bayesian Statistics* **3**, pp. 411–428.
Shreider, Y. A. (1964). *Method of Statistical Testing. Monte Carlo method.* (Elsevierm Publishing Co., 335 Jan Van Galenstraat, P.O. Box 211, Amsterdam (Netherlands)).
Shreider, Y. A. (1966). *The Monte Carlo method. Method of Statistical Testing*, 2nd edn. (Pergamon Press Ltd., Ozford, London, Edinburgh, New York, Paris, Frankfurt).
Sloan, I. H. and Joe, S. (1994). *Lattice Methods for Multiple Integration* (Oxford University Press, Oxford).
Sobol, I. M. (1973). *Monte Carlo numerical methods* (Nauka, Moscow).
Sobol, I. M. (1979). On the systematic search in a hypercube, *SIAM J. Numerical Analysis* **16**, pp. 790–793.
Sobol, I. M. (1989). On quadratic formulas for functions of several variables satisfying a general lipschitz condition, *USSR Comput. Math. and Math. Phys.*

29, 6, pp. 936–941.

Sobol, I. M. (1991). Quasi - Monte Carlo methods, in B. Sendov and I. Dimov (eds.), *International Youth Workshop on* ***Monte Carlo Methods and Parallel Algorithms*** *- Primorsko* (World Scientific, Singapore), pp. 75–81.

Sobol, I. M. (1994). *A Primer for the Monte Carlo Method* (CRC Press, Boca Raton, Florida), translated from the 4th Russian edition (1985).

Spanier, J. and Gelbard, E. (1969). *Monte Carlo Principles and Neurton Transport Problems* (Addison-Wesley, Reading, Massachusetts).

SPRNG (2007). Scalable Parallel Random Number Generators Library for Parallel Monte Carlo computations. SPRNG 1.0 and SPRNG 2.0, `http://sprng.cs.fsu.edu`.

Stewart, L. (1983). Bayesian analysis using Monte Carlo integration - a powerful methodology for handling some difficult problems, *Statistician* **32**, pp. 195–200.

Stewart, L. (1985). Multiparameter bayesian inference using Monte carlo integration - some techniques for bivariate analysis, in D. L. J.M.Bernardo, M.H. de Groot and A. Smith (eds.), *Bayesian Statistics*, Vol. 2 (North-Holland, Amsterdam), pp. 157–166.

Stewart, L. (1987). Hieralhical bayesian analysis using Monte carlo integration computing posterior distributions when there are many possible models, *Statistician* **36**, pp. 211–219.

Stewart, L. and Davis, W. (1986). Bayesian posterior distribution over sets of possible models with inferences computed by Monte Carlo integration, *Statistician* **35**, pp. 175–182.

Tanaka, H. and Nagata, H. (1974). Quasi-random number method for the numerical integration, *Supplement of the Progress of Theoretical Physics* **56**, pp. 121–131.

Tatarskii, W. (1983). The Wigner representation of quantum mechanics, *Sov. Phys. Usp.* **26**, 4, pp. 311–327.

Tijms, H. (2004). *Understanding Probability: Chance Rules in Everyday Life* (Cambridge University Press, Cambridge).

Tikchonov, A. N. and Samarskii, A. A. (1977). *Equations of the Mathematical Physics* (Nauka, Moscow).

Ton, K. C. and Trefethen, L. N. (1996). Calculation of pseudospectra by the Arnoldi iteration, *SIAM, J. Sci. Comput.* **17**, pp. 1–15.

Torn, A. (1966). Crude Monte Carlo quadrature in infinite variance case and the central limit theorem, *BIT Numerical Mathematics* **6**, 4, pp. 339–346.

Trefethen, L. N. (1991). Pseudospectra of matrices, in D. Griffiths and G. Watson (eds.), *14th Dundee Biennal Conference on Numerical Analysis*, pp. 234–266.

Trefethen, L. N. (1999). Computation of pseudospectra, in *Acta Numerica*, Vol. 8 (Cambridge University Press, Cambridge, UK), pp. 247–295.

van Dijk, H., Hop, J. and Louter, A. (1987). An algorithm for the computation of posterior moments and densities using simple importance sampling, *Statistician* **37**, pp. 83–90.

van Dijk, H. and Kloek, T. (1983). Monte Carlo analysis of skew posterior distri-

butions: An illustrative econometric example, *Statistician* **32**, pp. 216–223.
van Dijk, H. and Kloek, T. (1985). Experiments with some alternatives for simple importance sampling in Monte Carlo integration, in D. L. J. Bernardo, M.H. de Groot and A. Smith (eds.), *Bayesian Statistics*, Vol. 2 (North-Holland, Amsterdam), pp. 511–530.
Vitanov, P. and Nedjalkov, M. (1991). Iteration approach for solving the inhomogeneous Boltzmann equation, *COMPEL* **10**, 4, pp. 531–538.
Vitanov, P., Nedjalkov, M., Jacoboni, C., Rossi, F. and Abramo, A. (1994). Unified Monte Carlo approach to the Boltzmann and Wigner equations, in I. Dimov and O. Tonev (eds.), *Advances in Parallel Algorithms* (IOS Press, Amsterdam), pp. 117–128.
von Neumann, J. (1951). Various techniques used in connection with random digits. Monte Carlo methods, *Nat. Bureau Standards* **12**, pp. 36–38.
Weyl, H. (1916). Ueber die gleichverteilung von zahlen mod eins, *Math. Ann.* **77**, 3, pp. 313–352.
Zletev, Z. and Dimov, I. (2006). *Computational and Numerical Challengies in Environmental Modelling* (Elsevier, Amsterdam).

Subject Index

Author Index