Available online at www.sciencedirect.com

ScienceDirect

Applied Mathematical Modelling 32 (2008) 1511–1529

APPLIED MATHEMATICAL MODELLING

www.elsevier.com/locate/apm

Robustness and applicability of Markov chain Monte Carlo algorithms for eigenvalue problems ☆

I.T. Dimov [a,b,*], B. Philippe [c], A. Karaivanova [b], C. Weihrauch [a]

[a] *Centre for Advanced Computing and Emerging Technologies, The University of Reading, Whiteknights, P.O. Box 217, Reading RG6 6AH, UK*

[b] *IPP, Department of Parallel Algorithms, Bulgarian Academy of Sciences, Acad. G. Bonchev St., 25 A, 1113 Sofia, Bulgaria*

[c] *IRISA/INRIA-Rennes, Campus de Beaulieu, 35042 Rennes cedex, France*

Received 1 December 2006; received in revised form 1 April 2007; accepted 25 April 2007
Available online 30 June 2007

Abstract

In this paper we analyse applicability and robustness of Markov chain Monte Carlo algorithms for eigenvalue problems. We restrict our consideration to real symmetric matrices.

Almost Optimal Monte Carlo (MAO) algorithms for solving eigenvalue problems are formulated. Results for the structure of both – systematic and probability error 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 is presented. It shows that the 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 constructed.

A number of numerical tests for large symmetric dense matrices are performed in order to study experimentally the dependence of the systematic error from the structure of matrix spectrum. We also study how the probability error depends on the balancing of the matrix.

© 2007 Elsevier Inc. All rights reserved.

Keywords: Monte Carlo algorithms; Markov chain; Eigenvalue problem; Robust algorithms

☆ This work was partially supported by BIS-21^{++} project and by the Ministry of Science and Education of Bulgaria under grant I-1405/2004.

* Corresponding author. Address: Centre for Advanced Computing and Emerging Technologies, The University of Reading, Whiteknights, P.O. Box 217, Reading RG6 6AH, UK.

E-mail addresses: I.T.Dimov@reading.ac.uk (I.T. Dimov), Bernard.Philippe@irisa.fr (B. Philippe), anet@parellel.bas.bg (A. Karaivanova), c.weihrauch@reading.ac.uk (C. Weihrauch).

URLs: http://www.personal.reading.ac.uk/~sis04itd/ (I.T. Dimov), http://www.irisa.fr/sage/bernard/ (B. Philippe), http://www.bas.bg (A. Karaivanova), http://acet.rdg.ac.uk/~cw/ (C. Weihrauch).

0307-904X/$ - see front matter © 2007 Elsevier Inc. All rights reserved.
doi:10.1016/j.apm.2007.04.012

1. Introduction

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. Monte Carlo algorithms give statistical estimates of the required solution, by performing random sampling of a random variable, whose mathematical expectation is the desired solution [1–4]. Let J be the exact solution of the problem under consideration. Suppose it is proved that there exists a random variable ξ, such that $E\{\xi\} = J$. If one can produce N values of ξ, i.e. $\xi_1, \dots, \xi_i, \dots, \xi_N$, then

$$\bar{\xi}_N = \sum_{i=1}^{N} \xi_i,$$

can be considered as a MC approximation of J. The *probability error* of the MC algorithm is defined as follows.

Definition 1.1. If J is the exact solution of the problem, then the probability error is the least possible real number R_N, for which:

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

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

Several authors have presented work on the estimation of computational complexity of linear algebra problems [5–7,17–25]. In this paper we consider bilinear forms of matrix powers, which is 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 polynomials.

By $h = (h_1, \dots, h_n)$ and $v = (v_1, \dots, v_n)$, we usually denote given vectors of dimension n, i.e., $h, v \in \mathbb{R}^n$. By $x = (x_1, \dots, x_n)$, $x \in \mathbb{R}^n$ we denote the unknown vector. By $e^{(j)} \equiv (0, \dots, 0, \underbrace{1}_{j}, 0, \dots, 0)$ we denote an unit vector all elements, of which are zeros except the jth element $e_j^{(j)}$, which is equal to 1. Sometimes we will need to use a vector containing squares of elements of the original vector. We will use the notation $\hat{h} = \{h_i^2\}_{i=1}^n$. The notation $\bar{h}$ will be used to denote vector containing absolute values of elements of vector h, i.e., $\bar{h} = \{|h_i|\}_{i=1}^n$.

By A and B we denote matrices of size $n \times n$, i.e., $A, B \in \mathbb{R}^{n \times n}$. We use the following presentation of matrices:

$$A = \{a_{ij}\}_{i,j=1}^n = (a_1, \dots, a_i, \dots, a_n)^{\mathrm{T}}, \quad \text{where } a_i = (a_{i1}, \dots, a_{in}),\ i = 1, \dots, n$$

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 $\overline{A}$ we denote the matrix containing the absolute values of elements of a given matrix A:

$$\overline{A} = \{|a_{ij}|\}_{i,j=1}^n.$$

By

$$p_k(A) = \sum_{i=0}^{k} c_i A^i, \quad c_i \in \mathbb{R}$$

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:

$$(v, p_k(A)h).$$

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).

2. Formulation of the problems

Basically, we are interested in evaluation of forms

$$(v, p_k(A)h). \tag{2}$$

2.1. Bilinear form of matrix powers

In a special case of $p_k(A) = A^k$ the form (2) becomes

$$(v, A^k h), \quad k \geqslant 1.$$

2.2. Eigenvalues of matrices

Suppose that a real symmetric matrix A is diagonalisable, i.e.,

$$x^{-1}Ax = \mathrm{diag}(\lambda_1, \ldots, \lambda_n),$$

where $x = (x_1, \ldots, x_n)$ and $|\lambda_1| > |\lambda_2| \geqslant \cdots \geqslant |\lambda_{n-1}| > |\lambda_n|$. Values λ, for which the equality

$$Ax = \lambda x$$

is fulfilled are called eigenvalues. If A is a symmetric matrix, then the values λ are real numbers, i.e., $\lambda \in \mathbb{R}$.

The well-known Power method [8] 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 $v, h \in \mathbb{R}^n$ are arbitrary vectors. The Rayleigh quotient is used to obtain an approximation to λ_1:

$$\lambda_1 \approx \frac{(v, A^k h)}{(v, A^{k-1} h)}, \tag{3}$$

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 expression:

$$p_i(A) = \sum_{k=0}^{i} q^k C^k_{m+k-1} A^k, \tag{4}$$

where C^k_{m+k-1} are binomial coefficients, and the characteristic parameter q is used as acceleration parameter of the algorithm [6,9,10]. This approach is a discrete analogue of the resolvent analytical continuation method used in functional analysis [11].

If $|q|\|A\| < 1$ and $i \to \infty$, then the expression (4) becomes the resolvent matrix [9,12]:

$$p_\infty(A) = p(A) = \sum_{k=0}^{\infty} q^k C^k_{m+k-1} A^k = [I - qA]^{-m} = R^m_q,$$

where $R_q = [I - qA]^{-1}$ is the resolvent matrix of the equation

$$x = qAx + h. \tag{5}$$

Values $q_1, q_2, \ldots (|q_1| \leqslant |q_2| \leqslant \cdots)$ for which Eq. (5) is fulfilled are called characteristic values of the Eq. (5). The resolvent operator

$$R_q = [I - qA]^{-1} = I + A + qA^2 + \cdots \tag{6}$$

exists if the sequence (6) converges. The systematic error of the presentation (6) when m terms are used is

$$R_s = \mathrm{O}[(|q|/|q_1|)^{m+1} m^{\rho-1}], \tag{7}$$

where ρ is multiplicity of the roots q_1. Estimation (7) shows that the MC algorithm converges if $|q| < |q_1|$. When $|q| \geqslant |q_1|$ the algorithm does not converge for $q = q_* = 1$, but the solution of (5) exists (and moreover, the solution is unique). In this case one may apply a mapping of the spectral parameter q described in [12]. The mapping procedure consists in choosing a domain D lying inside the definition domain of

$$R_q h = \sum_{k=0}^{\infty} c_k q^k, \quad c_k = A^{k+1} h \tag{8}$$

as a function of q such that all characteristic values are outside of D, $q_* = 1 \in D$, $0 \in D$. Consider a variable δ in the unit disk $\Delta(|\delta| < 1)$ of the complex plane and a function $q = \Psi(\delta)$, which maps the domain Δ onto D. We show in [12] that the matrix resolvent (8) can be replaced by

$$R_{\Psi(\delta)} h = \sum_{j=0}^{\infty} b_j \delta^j, \quad b_j = \sum_{k=1}^{j} d^{(j)}_k c_k, \quad b_0 = 1, \tag{9}$$

where $d^{(j)}_k = \frac{1}{j!}\left[\frac{\partial^j}{\partial\delta^j}[\Psi(\delta)]^k\right]_{\delta=0}$. So, we assume that the matrix resolvent exists. In MC calculations we keep $m+1$ terms of the sequence (9) (see [12]):

$$R_{q_*} h \approx \sum_{k=0}^{m} g^{(m)}_k c_k, \quad g^{(m)}_k = \sum_{j=k}^{m} d^{(j)}_k \delta^j_*, \tag{10}$$

where coefficients $d^{(j)}_k$ depend on the mapping function $q = \Psi(\delta)$. Normally, the coefficients $d^{(j)}_k$ are calculated at advance and then are used in all MC calculations.

Let us consider the ratio:

$$\lambda = \frac{(v, Ap(A)h)}{(v, p(A)h)} = \frac{(v, AR^m_q h)}{(v, R^m_q h)}.$$

If $q < 0$, then

$$\frac{(v, AR^m_q h)}{(v, R^m_q h)} \approx \frac{1}{q}\left(1 - \frac{1}{\mu^{(k)}}\right) \approx \lambda_n, \tag{11}$$

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

$$\frac{(v, AR_q^m h)}{(v, R_q^m h)} \approx \lambda_1, \tag{12}$$

where $\lambda_1 = \lambda_{\max}$ is the dominant eigenvalue.

The approximate Eqs. (3), (11), (12) can be used to formulate efficient Monte Carlo algorithms for evaluating both the dominant and the minimal by modulo eigenvalue of real symmetric matrices.

The two problems formulated in this section rely on the bilinear form $(v, p_k(A)h)$. The latter fact allow us to concentrate our study on MC algorithms for computing

$$(v, A^k h), \quad k \geqslant 1. \tag{13}$$

In the next section we are going to consider Almost Optimal Markov Chain Monte Carlo algorithms for computing both bilinear forms of matrix powers $(v, A^k h)$ (Section 3.2) and extremal eigenvalues of real symmetric matrices (Section 3.3). Since expressions (4) and (11) exist the technique can also be used to compute the eigenvalue of minimum modulus λ_n.

It should be mentioned that the use of acceleration parameter based on the resolvent presentation is one way to decrease the computational complexity. Another way is to apply a variance reduction technique [13] in order to get the required approximation of the solution with a smaller number of operations. The variance reduction technique for particle transport eigenvalue calculations proposed in [13] uses Monte Carlo estimates of the forward and adjoint fluxes.

3. Almost optimal Markov chain Monte Carlo

We shall use the so-called MAO algorithm studied in [5,12,14,15]. Here we give a brief presentation of MAO.

Suppose we have a Markov chain

$$T = \alpha_0 \to \alpha_1 \to \alpha_2 \to \cdots \to \alpha_k \to \cdots$$

with n states. The random trajectory (chain) T_k of length k starting in the state α_0 is defined as follows:

$$T_k = \alpha_0 \to \alpha_1 \to \cdots \to \alpha_j \to \cdots \to \alpha_k, \tag{14}$$

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

Assume that

$$P(\alpha_0 = \alpha) = p_\alpha, \quad P(\alpha_j = \beta | \alpha_{j-1} = \alpha) = p_{\alpha\beta}, \tag{15}$$

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. We require that

$$\sum_{\alpha=1}^{n} p_\alpha = 1 \quad \text{and} \quad \sum_{\beta=1}^{n} p_{\alpha\beta} = 1, \quad \text{for any } \alpha = 1, 2, \ldots, n. \tag{16}$$

3.1. MAO density distributions

Suppose the distributions created from the density probabilities p_α and $p_{\alpha\beta}$ are *tolerant*, according to the following definition:

Definition 3.1. The distribution $(p_{\alpha_1}, \ldots, p_{\alpha_n})$ is tolerant to vector v, if

$$\begin{cases} p_{\alpha_s} > 0 & \text{when } v_{\alpha_s} \neq 0, \\ p_{\alpha_s} \geqslant 0 & \text{when } v_{\alpha_s} = 0. \end{cases} \tag{17}$$

Similarly, the distribution $p_{\alpha_{s-1},\alpha_s}$ is tolerant to matrix A, if

$$\begin{cases} p_{\alpha_{s-1},\alpha_s} > 0 & \text{when } a_{\alpha_{s-1},\alpha_s} \neq 0, \\ p_{\alpha_{s-1},\alpha_s} \geqslant 0 & \text{when } a_{\alpha_{s-1},\alpha_s} = 0. \end{cases} \tag{18}$$

We will consider a special choice of tolerant densities that are called *permissible*[1] p_i and $p_{i,j}$ defined as follows:

$$p_i = \frac{|v_i|}{\|v\|}, \quad p_{ij} = \frac{|a_{ij}|}{\|a_i\|}. \tag{19}$$

Such density distributions lead to almost optimal algorithms in sense that for a class of matrices A and vectors h such a choice coincides with optimal weighted algorithms defined in [1] and studied in [16] (for more details see [15]). The reason to use MAO instead of Uniform Monte Carlo is that MAO normally gives much smaller variances. From 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. The procedure makes the optimal algorithms very expensive.

3.2. MC algorithm for computing bilinear forms of matrix powers $(v, A^k h)$

The pair of density distributions (19) defines a finite chain of vector and matrix entries:

$$v_{\alpha_0} \longrightarrow a_{\alpha_0\alpha_1} \longrightarrow \cdots \longrightarrow a_{\alpha_{k-1}\alpha_k}. \tag{20}$$

The latter chain induces (defines) 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 (20) defined by densities (19), are included. For such a choice of densities p_i and p_{ij} we can prove the following lemma.

Lemma 3.1

$$E\{h_{\alpha_k}\} = \frac{\operatorname{sign}\{A_v^k\}}{\|A_v^k\|}(v, A^k h).$$

Proof. Consider the value $\theta^{(k)} = \operatorname{sign}\{A_v^k\}\|A_v^k\|h_{\alpha_k}$ for $k \geqslant 1$. We have

$$\begin{aligned} \theta^{(k)} &= \operatorname{sign}\{A_v^k\}\|A_v^k\|h_{\alpha_k} = \operatorname{sign}\left\{v_{\alpha_0}\prod_{s=1}^{k} a_{\alpha_{s-1}\alpha_s}\right\}\|v\|\prod_{s=1}^{k}\|a_{\alpha_{s-1}}\|h_{\alpha_k} \\ &= \operatorname{sign}\left\{v_{\alpha_0}\prod_{s=1}^{k} a_{\alpha_{s-1}\alpha_s}\right\}\|v\|\|a_{\alpha_0}\|\ldots\|a_{\alpha_{k-1}}\|h_{\alpha_k} = \frac{v_{\alpha_0}}{|v_{\alpha_0}|}\frac{a_{\alpha_0\alpha_1}\ldots a_{\alpha_{k-1}\alpha_k}}{|a_{\alpha_0\alpha_1}|\ldots|a_{\alpha_{k-1}\alpha_k}|}\|v\|\|a_{\alpha_0}\|\ldots\|a_{\alpha_{k-1}}\|h_{\alpha_k}. \end{aligned} \tag{21}$$

Let as 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 (19). The rules (19) ensure that the Markov chain *visits* non-zero elements only. From (21) and taking into account (19) one can get:

[1] Note that we distinguish *tolerant* from *permissible* densities. *Permissible* densities are equal to zero whenever $v_{\alpha_s}, a_{\alpha_{s-1},\alpha_s} = 0$.

$$\begin{aligned}
E\{\theta^{(k)}\} &= E\left\{\frac{v_{\alpha_0}}{|v_{\alpha_0}|}\frac{a_{\alpha_0\alpha_1}\dots a_{\alpha_{k-1}\alpha_k}}{|a_{\alpha_0\alpha_1}|\dots|a_{\alpha_{k-1}\alpha_k}|}\|v\|\|a_{\alpha_0}\|\dots\|a_{\alpha_{k-1}}\|h_{\alpha_k}\right\}\\
&= E\left\{\frac{v_{\alpha_0}}{p_{\alpha_0}}\frac{a_{\alpha_0\alpha_1}\dots a_{\alpha_{k-1}\alpha_k}}{p_{\alpha_0\alpha_1}\dots p_{\alpha_{k-1}\alpha_k}}h_{\alpha_k}\right\}\\
&= \sum_{\alpha_0,\dots\alpha_k=1}^{n}\frac{v_{\alpha_0}}{p_{\alpha_0}}\frac{a_{\alpha_0\alpha_1}\dots a_{\alpha_{k-1}\alpha_k}}{p_{\alpha_0\alpha_1}\dots p_{\alpha_{k-1}\alpha_k}}h_{\alpha_k}p_{\alpha_0}p_{\alpha_0\alpha_1}\dots p_{\alpha_{k-1}\alpha_k}\\
&= \sum_{\alpha_0=1}^{n}v_{\alpha_0}\sum_{\alpha_1=1}^{n}a_{\alpha_0\alpha_1}\dots\sum_{\alpha_{k-1}=1}^{n}a_{\alpha_{k-2}\alpha_{k-1}}\sum_{\alpha_k=1}^{n}a_{\alpha_{k-1}\alpha_k}h_{\alpha_k} = \sum_{\alpha_0=1}^{n}v_{\alpha_0}\sum_{\alpha_1=1}^{n}a_{\alpha_0\alpha_1}\dots\sum_{\alpha_{k-1}=1}^{n}a_{\alpha_{k-2}\alpha_{k-1}}(Ah)_{\alpha_{k-1}}\\
&= \sum_{\alpha_0=1}^{n}v_{\alpha_0}(A^kh)_{\alpha_0} = (v, A^kh). \qquad \square
\end{aligned}$$

Obviously, the standard deviation $\sigma\{h_{\alpha_k}\}$ is finite. Since we proved that the random variable $\theta^{(k)} = \text{sign}\{A_v^k\}\|A_v^k\|h_{\alpha_k}$ is a unbiased estimate of the form (v, A^kh), Lemma 3.1 can be used to construct a MC algorithm.

Let us consider N realizations of the Markov chain T_k (14) defined by the pair of density distributions (19). Denote by $\theta_i^{(k)}$ the ith realization of the random variable $\theta^{(k)}$. Then the value

$$\bar{\theta}^{(k)} = \frac{1}{N}\sum_{i=1}^{N}\theta_i^{(k)} = \frac{1}{N}\sum_{i=1}^{N}\{\text{sign}(A_v^k)\|A_v^k\|h_{\alpha_k}\}_i, \quad k \geqslant 1 \tag{22}$$

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:

$$R_N^{(k)} = |(v, A^kh) - \bar{\theta}^{(k)}| = c_p\sigma\{\theta^{(k)}\}N^{-\frac{1}{2}}, \tag{23}$$

where the constant c_p only depends on the probability P used in Definition 1.1 and does not depend on N and on $\theta^{(k)}$. Note that sometimes the constant c_p is taken for convenience to be 2 or 3 (for $c_p = 2$ the approximate probability in (1) is $P \approx 0.95$). Sometimes this error is refereed to as *statistical* error. Because of the finiteness of the standard deviation the probability error is always finite.

In fact, (22) together with the rules (19) defines a MC algorithm. The expression (22) gives a MC approximation of the form (v, A^kh) 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.*

3.3. MC algorithm for computing extremal eigenvalues

Now consider again the pair of density distributions (19) defining a finite chain of vector and matrix entries (20). For such a choice of densities p_i and p_{ij} we can prove the following theorem.

Theorem 3.1. *Consider a real symmetric matrix A and the chain of vector and matrix entries (20) defined by MAO density distributions (19).*

Then

$$\lambda_1 = \lim_{k\to\infty}\text{sign}\{a_{\alpha_{k-1}\alpha_k}\}\|a_{\alpha_{k-1}}\|\frac{E\{h_{\alpha_k}\}}{E\{h_{\alpha_{k-1}}\}}.$$

Proof. First consider the density of the Markov chain T_k of length k $\alpha_0 \to \alpha_1 \to \dots \to \alpha_k$ as a point in $n(k+1)$-dimensional Euclidian space $T_{k+1} = \underbrace{\mathbb{R}^n \times \dots \times \mathbb{R}^n}_{k+1}$:

$$P\{\alpha_0 = t_0, \alpha_1 = t_1, \dots, \alpha_k = t_k\} = p_{t_0}p_{t_0t_1}p_{t_1t_2}\dots p_{t_{k-1}t_k}.$$

To prove the theorem we have to show that $\text{sign}\{a_{\alpha_{k-1}\alpha_k}\}\|a_{\alpha_{k-1}}\|\frac{E\{h_{\alpha_k}\}}{E\{h_{\alpha_{k-1}}\}} = \mu_k = \frac{(v,A^kh)}{(v,A^{k-1}h)}$.

From the definition of sign function and MAO distributions (19) one can write:

$$\begin{aligned}\mathrm{sign}\{a_{\alpha_{k-1}\alpha_k}\}\|a_{\alpha_{k-1}}\|\frac{E\{h_{\alpha_k}\}}{E\{h_{\alpha_{k-1}}\}} &= \frac{a_{\alpha_{k-1}\alpha_k}}{|a_{\alpha_{k-1}\alpha_k}|}\|a_{\alpha_{k-1}}\|\frac{E\{h_{\alpha_k}\}}{E\{h_{\alpha_{k-1}}\}}\\ &= \frac{\frac{v_{\alpha_0}}{|v_{\alpha_0}|}\frac{a_{\alpha_0\alpha_1}\ldots a_{\alpha_{k-2}\alpha_{k-1}}a_{\alpha_{k-1}\alpha_k}}{|a_{\alpha_0\alpha_1}|\ldots|a_{\alpha_{k-2}\alpha_{k-1}}||a_{\alpha_{k-1}\alpha_k}|}\|v\|\|a_{\alpha_0}\|\ldots\|a_{\alpha_{k-2}}\|\|a_{\alpha_{k-1}}\|}{\frac{v_{\alpha_0}}{|v_{\alpha_0}|}\frac{a_{\alpha_0\alpha_1}\ldots a_{\alpha_{k-2}\alpha_{k-1}}}{|a_{\alpha_0\alpha_1}|\ldots|a_{\alpha_{k-2}\alpha_{k-1}}|}\|v\|\|a_{\alpha_0}\|\ldots\|a_{\alpha_{k-2}}\|}\frac{E\{h_{\alpha_k}\}}{E\{h_{\alpha_{k-1}}\}}\\ &= \frac{\mathrm{sign}\{A_v^k\}\|A_v^k\|E\{h_{\alpha_k}\}}{\mathrm{sign}\{A_v^{k-1}\}\|A_v^{k-1}\|E\{h_{\alpha_{k-1}}\}}.\end{aligned}$$

According to Lemma 3.1 we have:

$$\frac{\mathrm{sign}\{A_v^k\}\|A_v^k\|E\{h_{\alpha_k}\}}{\mathrm{sign}\{A_v^{k-1}\}\|A_v^{k-1}\|E\{h_{\alpha_{k-1}}\}} = \frac{(v,A^kh)}{(v,A^{k-1}h)} = \mu_k. \qquad \square$$

Obviously, since the standard deviation $\sigma\{h_{\alpha_k}\}$ is finite Theorem 3.1 allows to define a biased estimate of the extremal eigenvalue λ_1. Since, according to Theorem 3.1 for large enough values of k

$$\lambda_1 \approx \mu_k = \frac{E\{\mathrm{sign}(a_{\alpha_{k-1}\alpha_k})\|a_{\alpha_{k-1}}\|h_{\alpha_k}\}}{E\{h_{\alpha_{k-1}}\}}$$

and the computational formula of the algorithm can be presented in the following form:

$$\lambda_1 \approx \{\mu_k\}_N := \frac{1}{\sum_{i=1}^N h^{(i)}_{\alpha_{k-1}}}\sum_{i=1}^N \mathrm{sign}(a^{(i)}_{\alpha_{k-1}\alpha_k})\|a^{(i)}_{\alpha_{k-1}}\|h^{(i)}_{\alpha_k},$$

where the upper subscript (i) denotes the (i)th realization of the Markov chain, so that $h^{(i)}_{\alpha_{k-1}}$ is the value of the corresponding element of vector h after the $k-1$-st jump in the ith Markov chain, $h^{(i)}_{\alpha_k}$ is the value of the corresponding element of vector h after the kth jump in the ith Markov chain, $\|a^{(i)}_{\alpha_{k-1}}\|$ is the corresponding vector norm of the row which element is last visited by the Markov chain number i after the kth 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 (4) (see, also (11) instead of just bilinear forms of matrix powers).

3.4. Robust MC algorithms

Definition 3.2. MC algorithm for which the standard deviation does not increase with increasing matrix powers k is called robust MC algorithm.

We can prove the following lemma.

Lemma 3.2. *If MC algorithm is robust, then there exist a constant M such that*

$$\lim_{k\to\infty}\sigma\{\theta^{(k)}\} \leqslant M,$$

where $\theta^{(k)}$ is defined in the proof of Lemma 3.1.

Proof. Let us choose the constant M as:

$$M = \|v\| \times \|a_{\alpha_0}\| \times \sigma\{h_{\alpha_1}\}.$$

Consider the equality (23):

$$R_N^{(k)} = c_p\sigma\{\theta^{(k)}\}N^{-\frac{1}{2}}.$$

If a MC algorithm is robust, then for any fixed pair (N, P) (number of realizations N and probability P) we have:

$$\sigma\{\theta^{(k)}\} \leqslant \sigma\{\theta^{(k-1)}\}.$$

Since the smallest possible value of k is 1 (according to our requirement (13))

$$\begin{aligned}\sigma\{\theta^{(k)}\} \leqslant \sigma\{\theta^{(k-1)}\} \leqslant \ldots \leqslant \sigma\{\theta^{(1)}\} &= \sigma\{\mathrm{sign}\{v_{\alpha_0} a_{\alpha_0\alpha_1}\} \times \|v\| \times \|a_{\alpha_0}\| \times h_{\alpha_1}\} \\ &= \|v\| \times \|a_{\alpha_0}\| \times \sigma\{h_{\alpha_1}\} = M. \qquad \square\end{aligned} \tag{24}$$

It is interesting to answer the question:

- *How small could be the probability error? and*
- *Is it possible to construct MC algorithms with zero probability error?*

To answer the first question one has to analyse the structure of the variance. Then it will be possible to answer the second question concerning the existence of algorithms with zero probability error.

3.5. Interpolation MC algorithms

Definition 3.3. MC algorithm for which the probability error is zero is called interpolation MC algorithm.

The next theorem gives the structure of the variance for MAO algorithm.

Theorem 3.2. *Let*

$$\hat{h} = \{h_i^2\}_{i=1}^n, \quad \bar{v} = \{|v_i|\}_{i=1}^n, \quad \overline{A} = \{|a_{ij}|\}_{i,j=1}^n.$$

Then

$$D\{\theta^{(k)}\} = \|A_v^k\|(\bar{v}, \overline{A}^k \hat{h}) - (v, A^k h)^2.$$

Proof. Taking into account MAO density distributions (19) the random variable $\theta^{(k)}$ can be presented in the following form:

$$\theta^{(k)} = \mathrm{sign}\{A_v^k\}\|A_v^k\| h_{\alpha_k} = \frac{v_{\alpha_0}}{|v_{\alpha_0}|} \frac{a_{\alpha_0\alpha_1} \ldots a_{\alpha_{k-1}\alpha_k}}{|a_{\alpha_0\alpha_1}| \ldots |a_{\alpha_{k-1}\alpha_k}|} \|v\| \|a_{\alpha_0}\| \ldots \|a_{\alpha_{k-1}}\| h_{\alpha_k} = \frac{v_{\alpha_0}}{p_{\alpha_0}} \frac{a_{\alpha_0\alpha_1} \ldots a_{\alpha_{k-1}\alpha_k}}{p_{\alpha_0\alpha_1} \cdots p_{\alpha_{k-1}\alpha_k}} h_{\alpha_k}.$$

We deal with the variance

$$D\{\theta^{(k)}\} = E\{(\theta^{(k)})^2\} - (E\{\theta^{(k)}\})^2. \tag{25}$$

Consider the first term of (25).

$$\begin{aligned}E\{(\theta^{(k)})^2\} &= E\left\{\frac{v_{\alpha_0}^2}{p_{\alpha_0}^2} \frac{a_{\alpha_0\alpha_1}^2 \ldots a_{\alpha_{k-1}\alpha_k}^2}{p_{\alpha_0\alpha_1}^2 \cdots p_{\alpha_{k-1}\alpha_k}^2} h_{\alpha_k}^2\right\} = \sum_{\alpha_0,\ldots\alpha_k=1}^n \frac{v_{\alpha_0}^2}{p_{\alpha_0}^2} \frac{a_{\alpha_0\alpha_1}^2 \ldots a_{\alpha_{k-1}\alpha_k}^2}{p_{\alpha_0\alpha_1}^2 \cdots p_{\alpha_{k-1}\alpha_k}^2} h_{\alpha_k}^2 p_{\alpha_0} p_{\alpha_0\alpha_1} \cdots p_{\alpha_{k-1}\alpha_k} \\ &= \sum_{\alpha_0,\ldots\alpha_k=1}^n \frac{v_{\alpha_0}^2}{p_{\alpha_0}} \frac{a_{\alpha_0\alpha_1}^2 \ldots a_{\alpha_{k-1}\alpha_k}^2}{p_{\alpha_0\alpha_1} \cdots p_{\alpha_{k-1}\alpha_k}} h_{\alpha_k}^2 = \sum_{\alpha_0,\ldots\alpha_k=1}^n \frac{v_{\alpha_0}^2}{|v_{\alpha_0}|} \|v\| \frac{a_{\alpha_0\alpha_1}^2 \ldots a_{\alpha_{k-1}\alpha_k}^2}{|a_{\alpha_0\alpha_1}| \ldots |a_{\alpha_{k-1}\alpha_k}|} \|a_{\alpha_0}\| \ldots \|a_{\alpha_{k-1}}\| h_{\alpha_k}^2 \\ &= \|A_v^k\| \sum_{\alpha_0,\ldots\alpha_k=1}^n |v_{\alpha_0}||a_{\alpha_0\alpha_1}| \ldots |a_{\alpha_{k-1}\alpha_k}| h_{\alpha_k}^2 = \|A_v^k\| \sum_{\alpha_0=1}^n |v_{\alpha_0}| \sum_{\alpha_1=1}^n |a_{\alpha_0\alpha_1}| \ldots \sum_{\alpha_{k-1}=1}^n |a_{\alpha_{k-2}\alpha_{k-1}}| \sum_{\alpha_k=1}^n |a_{\alpha_{k-1}\alpha_k}| h_{\alpha_k}^2 \\ &= \|A_v^k\| \sum_{\alpha_0=1}^n |v_{\alpha_0}| \sum_{\alpha_1=1}^n |a_{\alpha_0\alpha_1}| \ldots \sum_{\alpha_{k-1}=1}^n |a_{\alpha_{k-2}\alpha_{k-1}}| (\overline{A}\hat{h})_{\alpha_{k-1}} = \|A_v^k\| \sum_{\alpha_0=1}^n |v_{\alpha_0}| (\overline{A}^k \hat{h})_{\alpha_0} = \|A_v^k\| (\bar{v}, \overline{A}^k \hat{h}).\end{aligned}$$

According to Lemma 3.1 the second term of (25) is equal to $(v, A^k h)^2$.

Thus, $D\{\theta^{(k)}\} = \|A_v^k\|(\bar{v}, \overline{A}^k \hat{h}) - (v, A^k h)^2 t$. $\square$

Now we can formulate an important corollary that gives a sufficient condition for constructing an interpolation MC algorithm.

Corollary 3.1. *Consider vectors* $h = (1, \ldots, 1)^{\mathrm{T}}$, $v = (\frac{1}{n}, \ldots, \frac{1}{n})$ *and the stochastic matrix* $A = hv = \begin{pmatrix} \frac{1}{n} & \ldots & \frac{1}{n} \\ \vdots & & \\ \frac{1}{n} & \ldots & \frac{1}{n} \end{pmatrix}$, $h \in \mathbb{R}^{n\times 1}$, $v \in \mathbb{R}^{1\times n}$, $A \in \mathbb{R}^{n\times n}$. *Then MC algorithm defined by density distributions* (19) *is an interpolation MC algorithm.*

Proof. To prove the corollary it is sufficient to show that the variance $D\{\theta^{(k)}\}$ is zero. Obviously, $\|v\| = 1$ and $\|a_i\| = 1$ for any $i = 1, \ldots, n$. Thus,

$$\|A_v^k\| = \|v\|\|a_{\alpha_0}\| \ldots \|a_{\alpha_{k-1}}\| = 1. \tag{26}$$

The following equalities are true:

$$Ah = \begin{pmatrix} \frac{1}{n} & \ldots & \frac{1}{n} \\ \vdots & & \\ \frac{1}{n} & \ldots & \frac{1}{n} \end{pmatrix} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}.$$

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

$$\overline{A} = \begin{pmatrix} \left|\frac{1}{n}\right| & \ldots & \left|\frac{1}{n}\right| \\ \vdots & & \\ \left|\frac{1}{n}\right| & \ldots & \left|\frac{1}{n}\right| \end{pmatrix} = A$$

and $\hat{h} = \{|h_i^2|\}_{i=1}^n = h$, and taking into account (26) we have:

$$\|A_v^k\|(\bar{v}, \overline{A}^k \hat{h}) = 1.$$

Thus, we proved that $D\{\theta^{(k)}\} = 0$. $\square$

4. 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, $\mathscr{A}$, 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:

$$\mathscr{A} = \{A : Pr(R_{k,N} \leqslant \varepsilon) \geqslant c\}. \tag{27}$$

There exists a question: which algorithm in the set $\mathscr{A}$ has the smallest computational cost? In this paper we are not going to analyse 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 [5,6,10,15]).

We assume that the probability error $R_{k,N}$ is fixed by the value of ε and the probability $c < 1$ in (27) 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 4.1. Computational cost of 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 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 of the number of steps in each random trajectory (number of iterations) k.

4.1. 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:

$$\left| (v, A^k h) - \frac{1}{N} \sum_{i=1}^{N} \theta_i^{(k)} \right| \leqslant \varepsilon.$$

We consider the case of balanced errors, i.e.,

$$R_{k,N} = R_{k,s} = \frac{\varepsilon}{2}.$$

When a mapping procedure (9) is applied one may assume that there exists a positive constant $\alpha < 1$ such that

$$\alpha \geqslant |g_i^{(k)}| \times \|A\| \quad \text{for any } i \text{ and } k. \tag{28}$$

Then

$$\frac{\varepsilon}{2} \leqslant \frac{(|g_i^{(k)}| \|A\|)^{k+1} \|h\|}{1 - |g_i^{(k)}| \|A\|} \leqslant \frac{\alpha^{k+1} \|h\|}{1 - \alpha}$$

and for k should be chosen the smallest natural number for which

$$k \geqslant \frac{|\log \delta|}{|\log \alpha|} - 1, \quad \delta = \frac{\varepsilon(1-\alpha)}{2\|h\|}. \tag{29}$$

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 $\alpha \geqslant \|A\|$ exists. Then the number of iterations k should be chosen according to (29).

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 Eq. (5) is $\rho = 1$. Assume also that there exists a positive constant q_*, such that

$$q_* \leqslant |q_1|.$$

Then we have

$$R_s = \frac{\varepsilon}{2} = c_s \left| \frac{q}{q_*} \right|^{k+1},$$

$$k \geqslant \frac{\left| \log \left(\frac{\varepsilon}{2c_s} \right) \right|}{\left| \log \left| \frac{q}{q_*} \right| \right|} - 1.$$

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.

4.2. Method for choosing the number of chains N

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

$$\sigma \geqslant \sigma(\theta^{(k)}). \tag{30}$$

Then we have

$$\varepsilon = 2R_N^{(k)} = 2c_p\sigma(\theta^{(k)})N^{-\frac{1}{2}} \geqslant 2c_p\sigma N^{-\frac{1}{2}}$$

and

$$N \geqslant \left\{ \frac{2c_p\sigma}{\varepsilon} \right\}^2. \tag{31}$$

Taking into account relations (29) and (31) one can get estimates of computational cost of biased MC algorithms. Let us stress that to obtain relations (29) and (31) we needed some *a priori* information in form of (28) and (30). 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 (30) 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. 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.

5.1. Power Monte Carlo algorithm for computing the dominant eigenvalue

The corresponding matrix polynomial is $p_k(A) = A^k$, so that

$$(v, A^k h) = E\{\theta^{(k)}\} \approx \bar{\theta}^{(k)} = \frac{1}{N}\sum_{i=1}^{N}\{\text{sign}(A_v^k)\|A_v^k\|h_{\alpha_k}\}_i \tag{32}$$

and the computational formula is

$$\lambda_{\max} \approx \frac{\bar{\theta}^{(k)}}{\bar{\theta}^{(k-1)}} = \frac{1}{\sum_{i=1}^{N} h^{(i)}_{\alpha_{k-1}}} \sum_{i=1}^{N} \mathrm{sign}(a^{(i)}_{\alpha_{k-1}\alpha_k}) \| a^{(i)}_{\alpha_{k-1}} \| h^{(i)}_{\alpha_k}.$$

5.2. Resolvent MC algorithm for eigenvalue problems

For the Inverse shifted (Resolvent) MC the matrix polynomial is $p_k(A) = \sum_{k=0}^{\infty} q^k C^k_{m+k-1} A^k$. If $|qA| < 1$, then $p_k(A) = \sum_{k=0}^{\infty} q^k C^k_{m+k-1} A^k = [I - qA]^{-m} = R^m_q$ (R is the resolvent matrix) and

$$\lambda = \frac{(v, Ap(A)h)}{(v, p(A)h)} = \frac{(v, AR^m_q h)}{(v, R^m_q h)}.$$

If $q < 0$, then $\frac{(v,AR^m_q h)}{(v,R^m_q h)} \approx \frac{1}{q}(1 - \frac{1}{\mu^{(m)}}) \approx \lambda_{\min}$ ($\mu^{(m)}$ is the approximation to the dominant eigenvalue of the resolvent matrix R).

For a positive q ($q > 0$): $\frac{(h,AR^m_q f)}{(h,R^m_q f)} \approx \lambda_{\max}$.

Thus the computational formula for the smallest by magnitude eigenvalue is

$$\lambda \approx \frac{E\sum_{k=0}^{l} q^k C^k_{k+m-1} \theta^{(k+1)}}{E\sum_{k=0}^{l} q^k C^k_{k+m-1} \theta^{(k)}}, \tag{33}$$

where $\theta^{(0)} = \frac{v_{k_0}}{p_{k_0}}$ and the r.v. $\theta^{(k)}$ are defined according to (22). The value v_{k_0} is the entrance k_0 of the arbitrary vector v chosen according to the initial distribution p_{k_0}.

If $q > 0$ the algorithm described by (33) evaluates $\lambda_{\max}$, if $q < 0$, the algorithm evaluates $\lambda_{\min}$.

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 $|\lambda_1| > |\lambda_2| \geqslant \cdots \geqslant |\lambda_{n-1}| > |\lambda_n|$ as well as $|\mu_1| > |\mu_2| \geqslant \cdots \geqslant |\mu_{n-1}| > |\mu_n|$.

The systematic error that appears from the Power method is:

$$\mathrm{O}\left(\left| \frac{\mu_2}{\mu_1} \right|^k \right),$$

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 $\mathrm{O}(\sigma(\theta^{(k)})N^{-1/2})$.

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:

$$J(q, A) = \frac{1 + |q|\lambda_1}{1 + |q|\lambda_2}, \tag{34}$$

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

$$q = -\frac{1}{2\|A\|}. \tag{35}$$

The *systematic error* in this case is

$$\mathrm{O}\left[\left(\frac{2\lambda_1+\lambda_n}{2\lambda_1+\lambda_{n-1}}\right)^m\right], \tag{36}$$

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 $\mathrm{O}[(1/3)^m]$. Such a convergence can be 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 $\mathrm{O}[(2/3)^m]$ for $\alpha=\frac{1}{2}$. In case of $\alpha=\frac{9}{10}$ the convergence can not be better than $\mathrm{O}[(1/19)^m]$.

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 1). The last column of Table 1 contains results of the convergence (the ratio μ_2/μ_1 characterizes the rate of convergence). Obviously, the smaller is the ratio μ_2/μ_1 the faster is the convergence.

The results presented in Table 1 show that for some classes of matrices the convergence is relatively slow, but for some other classes it is very fast. 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 $\mathrm{O}(0.0830)^m$ (see the last row of Table 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 $q=-\frac{1}{2\|A\|}$ 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 $q=-\frac{9}{10\|A\|}$ 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 non-balancing leads to increasing stochastic errors with increasing 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 on Fig. 1. The first impression is that the results are good.

But more precise consideration shows that the error increases with increasing matrix powers (see Fig. 2). Since we consider values of matrix powers $\frac{v^TA^kv}{\|v\|}$ we eliminate the systematic error (the Monte Carlo estimate is a unbiased estimate) in this special case. So that considering the results from Fig. 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's iteration 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 number of iterations the standard deviation (respectively the probability error) also increases (see Section 3.4). 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 perturbation the matrices are not any more stochastic. Bigger the perturbation farther the matrix from the 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 $\frac{(v,A^kh)}{(v,h)}$ since in this case there is no system-

Table 1
Illustration of the convergence of the resolvent MC algorithm

Matrix	α	$\lambda_1(A)$	$\lambda_{n-1}(A)$	$\lambda_n(A)$	$\mu_1(R_q)$	$\mu_2(R_q)$	$\mu_n(R_q)$	$\frac{\mu_1}{\mu_2}$
A_1	$\frac{1}{2}$	0.5	0.22	0.05	0.9524	0.8197	0.6667	0.8600
A_2	$\frac{1}{2}$	1.0	0.95	−0.94	1.8868	0.6780	0.6667	0.3590
A_2	$\frac{4}{5}$	1.0	0.95	−0.94	4.0323	0.5682	0.5556	0.1409
A_2	$\frac{9}{10}$	1.0	0.95	−0.94	6.4935	0.5391	0.5263	0.0830

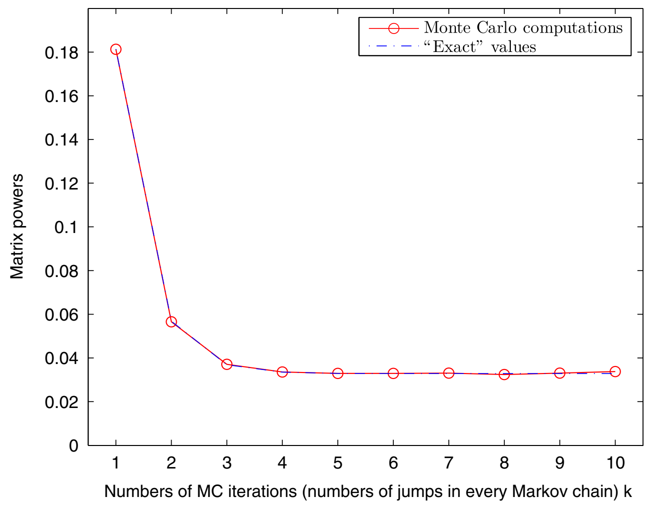

Fig. 1. Monte Carlo and "Exact" values $\frac{v^{\mathrm{T}}A^k v}{\|v\|}$ for a random non-balanced matrix A_3 of size 128×128. In all experiments the number N of Markov chains is 10^6.

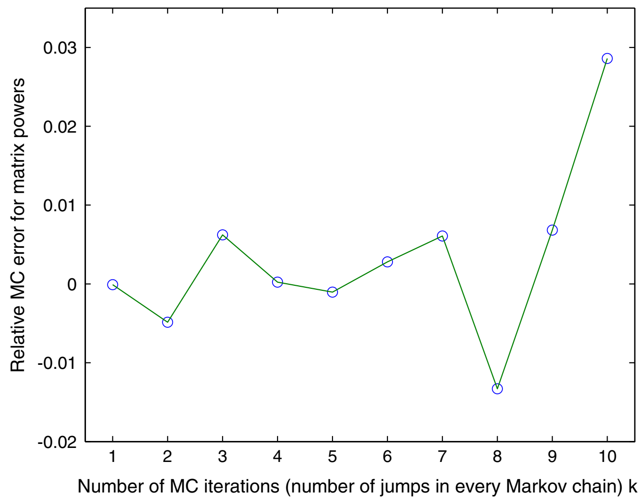

Fig. 2. Relative MC error for values $\frac{v^{\mathrm{T}}A^k v}{\|v\|}$ for a random non-balanced matrix of size 128×128. In all experiments the number N of Markov chains is 10^6.

atic 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 $(v, h) = 1$.

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.

In Fig. 3 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:

$$R_N^{(k)} = \left| \frac{1}{N} \sum_{i=1}^{N} \theta_i^{(k)} - \frac{(v, A^k h)}{(v, h)} \right|, \quad Rel_N^{(k)} = \frac{(v, h)}{(v, A^k h)} R_N^{(k)}.$$

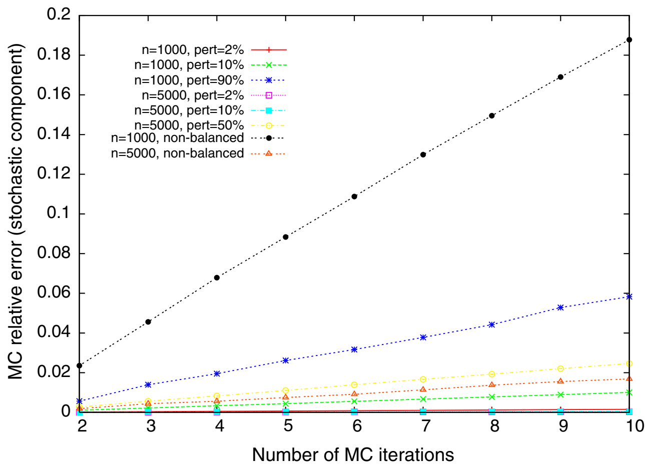

Fig. 3. The dependence of MC relative error (stochastic component) on power of the matrix A.

From Fig. 3 we can see that the error increases linearly if k is increasing. 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 Fig. 4. 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 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 k (see results shown on Fig. 3). 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 Figs. 5 and 6.

In Fig. 5 the Monte Carlo errors for matrix size of $n = 1000$ 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.

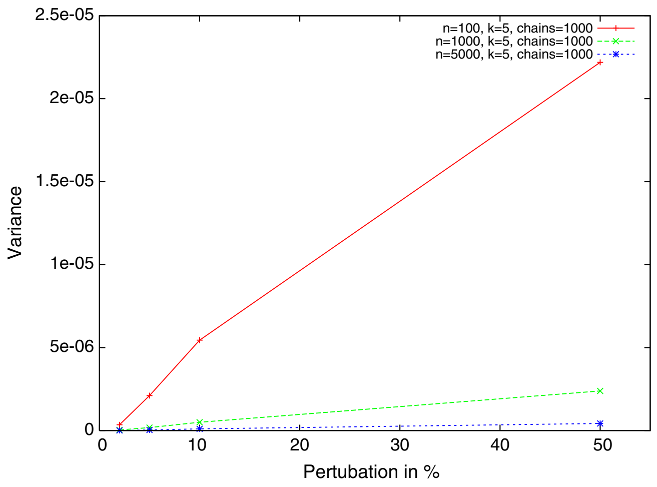

Fig. 4. The dependence of variance of the r.v. on perturbation of the matrix entries.

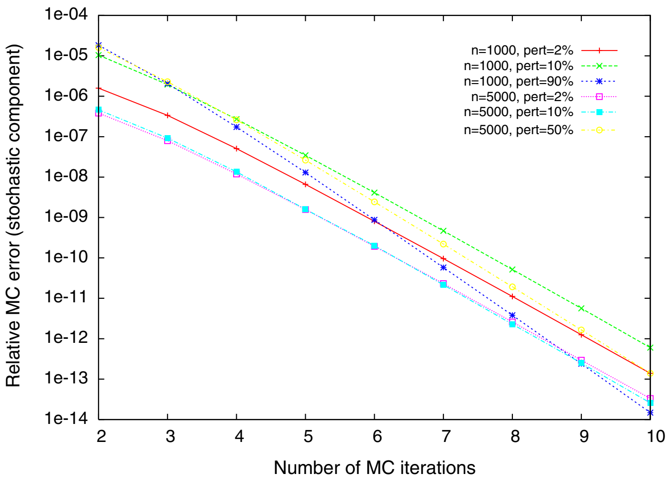

Fig. 5. The dependence of MC error on power of matrices k with "small" spectral norms ($\|A\| \approx 0.1$).

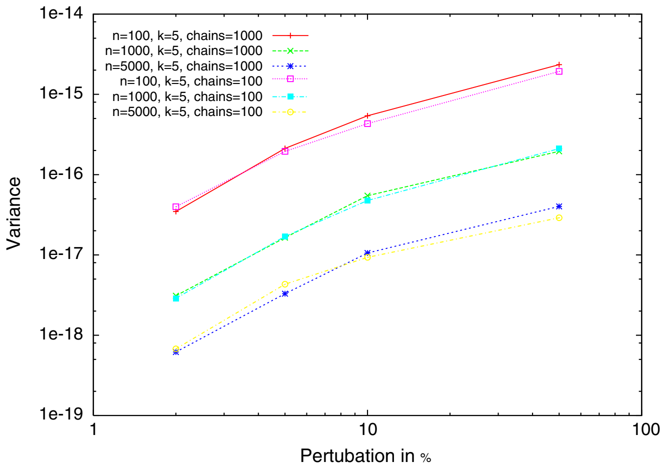

Fig. 6. The dependence of variance of the r.v. on perturbation of matrices with "small" spectral norms ($\|A\| \approx 0.1$).

The results shown on Fig. 6 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 Fig. 6 is 10^{10} smaller than the variance shown in Fig. 4. It increases with increasing of perturbations because matrices are getting more and more unbalanced. The last results 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 [5,6,9]. Nevertheless, it is not trivial 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

Table 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	1000	39	1.0000	-1.9000
A_5	2000	56	1.0000	64.0000

Table 3
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

parallelization seems to be one of the most efficient from the point of view of good load balancing [9]. Experiments are performed using Message Passing Interface (MPI). The matrices used in our experiments are presented in Table 2.

The results of runs on a parallel system are given in Table 3. The computational cost τ of the MAO algorithm is measured in milliseconds.

The main observations from the computational experiments are the following.

- The systematic error depends very much on the spectrum of the iterating 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 on the mathematical expectation of the number of non-zero elements per row.
- There is a linear dependence of the computational cost on the number of Markov chains N.
- When MAO algorithm is run on parallel systems the speedup is almost linear when the computational cost τ for every processor is not too small.

All observations are expected; they confirm the theoretical analysis of MAO algorithm.

6. Conclusion

In this paper 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 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.

References

[1] S.M. Ermakov, G.A. Mikhailov, Statistical Modelling, Nauka, Moscow, 1982.
[2] I.M. Sobol, Monte Carlo Numerical Methods, Nauka, Moscow, 1973.
[3] J. Spanier, E. Gelbard, Monte Carlo Principles and Neutron Transport Problem, Addison-Wesley, 1969.
[4] J.R. Westlake, A Handbook of Numerical Matrix Inversion and Solution of Linear Equations, John Wiley & Sons, Inc., New York, London, Sydney, 1968.
[5] I. Dimov, Monte Carlo algorithms for linear problems, Pliska (Studia Mathematica Bulgarica) 13 (2000) 57–77.
[6] I. Dimov, V. Alexandrov, A. Karaivanova, Parallel resolvent Monte Carlo algorithms for linear algebra problems, J. Math. Comput. Simul. 55 (2001) 25–35.
[7] I. Dimov, O. Tonev, Random walk on distant mesh points Monte Carlo methods, J. Statist. Phys. 70 (5/6) (1993) 1333–1342.
[8] G.V. Golub, C.F. Van Loon, Matrix Computations, third ed., Johns Hopkins Univ. Press, Baltimore, 1996.
[9] I. Dimov, A. Karaivanova, Parallel computations of eigenvalues based on a Monte Carlo approach, J. Monte Carlo Method Appl. 4 (1) (1998) 33–52.
[10] I. Dimov, A. Karaivanova, in: O. Iliev, M. Kaschiev, Bl. Sendov, P. Vassilevski (Eds.), A Power Method with Monte Carlo Iterations, Recent Advances in Numerical Methods and Applications, World Scientific, Singapore, 1999, pp. 239–247.
[11] L.W. Kantorovich, V.I. Krylov, Approximate Methods of Higher Analysis, Interscience, New York, 1964.
[12] I.T. Dimov, V. Alexandrov, A new highly convergent Monte Carlo method for matrix computations, Math. Comput. Simul. 47 (1998) 165–181.
[13] J.D. Densmore, E.W. Larsen, Variational variance reduction for particle transport eigenvalue calculations using Monte Carlo adjoint simulation, J. Comput. Phys. 192 (2) (2003) 387–405.
[14] V. Alexandrov, E. Atanassov, I. Dimov, Parallel quasi-Monte Carlo methods for linear algebra problems, Monte Carlo Methods Appl. 10 (3–4) (2004) 213–219.
[15] I. Dimov, Minimization of the probable error for some Monte Carlo methods, in: Proc. Int. Conf. on Mathematical Modeling and Scientific Computation, Albena, Bulgaria, Sofia, Publ. House of the Bulgarian Academy of Sciences, 1991, pp. 159–170.
[16] G.A. Mikhailov, Optimization of Monte Carlo Weighted Methods, Springer-Verlag, 1992.
[17] J.H. Curtiss, A theoretical comparison of the efficiencies of two classical methods and a Monte Carlo method for computing one component of the solution of a set of linear algebraic equations, Proc. Symposium on Monte Carlo Methods, John Wiley and Sons, 1956, pp. 191–233.
[18] J.H. Curtiss, Monte Carlo methods for the iteration of linear operators, J. Math. Phys. 32 (4) (1954) 209–232.
[19] I.T. Dimov, A.N. Karaivanova, Iterative Monte Carlo algorithms for linear algebra problems, in: First Workshop on Numerical Analysis and Applications, Rousse, Bulgaria, June 24–27, 1996, in Numerical Analysis and Its Applications, Springer Lecture Notes in Computer Science, ser. 1196, pp. 150–160.
[20] I. Dimov, O. Tonev, Performance analysis of Monte Carlo algorithms for some models of computer architectures, in: Bl. Sendov, I. Dimov (Eds.), International Youth Workshop on Monte Carlo Methods and Parallel Algorithms – Primorsko, World Scientific, Singapore, 1990, pp. 91–95.
[21] I. Dimov, O. Tonev, Monte Carlo algorithms: performance analysis for some computer architectures, J. Comput. Appl. Math. 48 (1993) 253–277.
[22] G.E. Forsythe, R.A. Leibler, Matrix inversion by a Monte Carlo method, MTAC 4 (1950) 127–129.
[23] J.M. Hammersley, D.C. Handscomb, Monte Carlo Methods, John Wiley & Sons Inc., New York, London, Sydney, Methuen, 1964.
[24] M. Mascagni, A. Karaivanova, A parallel quasi-Monte Carlo method for computing extremal eigenvalues, Monte Carlo and Quasi-Monte Carlo Methods, Springer, 2000.
[25] G.M. Megson, V.N. Aleksandrov, I.T. Dimov, Systolic matrix inversion using a Monte Carlo method, J. Parallel Algorith. Appl. 3 (3/4) (1993) 311–330.