Provided for non-commercial research and educational use.
Not for reproduction, distribution or commercial use.

PLISKA
STUDIA MATHEMATICA
BULGARICA

ПЛИСКА
БЪЛГАРСКИ
МАТЕМАТИЧЕСКИ
СТУДИИ

The attached copy is furnished for non-commercial research and education use only.
Authors are permitted to post this version of the article to their personal websites or
institutional repositories and to share with other researchers in the form of electronic reprints.
Other uses, including reproduction and distribution, or selling or
licensing copies, or posting to third party websites are prohibited.

For further information on
Pliska Studia Mathematica Bulgarica
visit the website of the journal http://www.math.bas.bg/~pliska/
or contact: Editorial Office
Pliska Studia Mathematica Bulgarica
Institute of Mathematics and Informatics
Bulgarian Academy of Sciences
Telephone: (+359-2)9792818, FAX:(+359-2)971-36-49
e-mail: pliska@math.bas.bg

Pliska Stud. Math. Bulgar. **13** (2000), 133-149

PLISKA
STUDIA MATHEMATICA
BULGARICA

STATISTICAL NUMERICAL METHODS FOR EIGENVALUE PROBLEM. PARALLEL IMPLEMENTATION

Ivan Dimov, Aneta Karaivanova

The problem of evaluating the smallest eigenvalue of real symmetric matrices using statistical numerical methods is considered.

Two new almost optimal Monte Carlo algorithms are presented:

- Resolvent Monte Carlo algorithm (RMC). The algorithm uses Monte Carlo iterations by the resolvent matrix and includes parameter controlling the rate of convergence;
- Monte Carlo algorithm with inverse iterations (MCII).

Numerical tests are performed for a number of large sparse symmetric test matrices. The tests are performed on supercomputers Intel-PARAGON (which is a distributed memory multicomputer) and CRAY Y-MP C92A (a two-processor vector machine).

Some information about the parallel efficiency of the algorithms is obtained, showing that the algorithms under consideration are *well-parallized* and *well vectorizable.*

Keywords: Monte Carlo algorithms, Eigenvalue, Efficiency estimator, Markov chain, Parallel algorithm

MSC subject classification: 65C05, 65U05.

1 Introduction

In this work we present Monte Carlo algorithms for evaluating eigenvalues of a symmetric matrix A, i.e. the values of λ for which

$$Au = \lambda u \tag{1}$$

holds.

Here we consider algorithms for both problems: evaluating the dominant eigenvalue and the smallest one. It is known that the problem of calculating the smallest eigenvalue of a matrix A is more difficult from numerical point of view than the problem of evaluating the dominant eigenvalue. Nevertheless, for many important applications in physics and engineering it is necessary to estimate the value of the smallest eigenvalue, because it usually defines the most stable state of the system which is described by the considered matrix.

There are several basic advantage of the Monte Carlo algorithms. It is well known that Monte Carlo algorithms are parallel algorithms. They have high parallel efficiency when parallel computers are used [5]. Monte Carlo algorithms are also very efficient when the problem under consideration is too large or too intricate for other treatments.

There are, also, many problems in which it is important to have an efficient algorithm which is parallel and/or vectorizable. And for matrices with a large size which often appear in practice it is not easy to find efficient algorithms for evaluating eigenvalues when modern high-speed parallel computers are used. For example, the problem of plotting the spectral portraits of matrices is one of the important problems where high-efficient parallel algorithms are needed. 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.

Monte Carlo methods give statistical estimates for the functional of the solution by performing random sampling of a certain chance variable whose mathematical expectation is the desired functional.

Let I be any functional that we estimate by Monte Carlo method; θ_n be the estimator, where n is the number of trials. The probable error for the usual Monte Carlo method (which does not use any additional a priori information about the regularity of the solution) [12] is defined as:

$$P\{|I-\theta_n| \geq r_n\} = 1/2 = P\{|I-\theta_n| \leq r_n\}. \tag{2}$$

If the standard deviation $\sigma(\theta_n) < \infty$, the normal convergence in the Central Limit Theorem holds.

$$P\{|I-\theta_n| \leq x\sigma(\theta_n)n^{-1/2}\} \approx \Phi(x).$$

Obviously, from (2) and $\Phi(0.6745) \approx \frac{1}{2}$ we have

$$r_n \approx 0.6745\sigma(\theta_n)n^{-1/2}. \tag{3}$$

The algorithms under consideration are almost optimal Monte Carlo algorithms (i.e. MAO algorithms). The value of the variance is reduced by means of a special kind of transition-density matrices.

2 Description of the Monte Carlo Algorithms

Here we present a *stationary linear iterative Monte Carlo algorithm* for evaluating eigenvalues of matrices.

2.1 Almost Optimal Markov Chains

The presented algorithms contain iterations with the original matrix, as well as Monte Carlo iterations with a resolvent matrix (used as iterative operator) of a given matrix.

Consider a matrix A:

$$A = \{a_{ij}\}_{i,j=1}^{n}, \quad A \in \mathbb{R}^{n\times n} \tag{4}$$

and a vector

$$f = (f_1, \dots, f_n)^t \in \mathbb{R}^n \tag{5}$$

The matrix A can be considered as a linear operator $A[\mathbb{R}^n \to \mathbb{R}^n]$, so that the linear transformation

$$Af \in \mathbb{R}^n \tag{6}$$

defines a new vector in $\mathbb{R}^n$.

Since iterative Monte Carlo algorithms using the transformation (6) will be considered, the linear transformation (6) will be called *iteration*. The algebraic transformation (6) plays a fundamental role in iterative Monte Carlo algorithms.

Now consider the following problem **P** for the matrix A:

Problem P. Evaluating of eigenvalues $\lambda(A)$:

$$Au = \lambda(A)u. \tag{7}$$

It is assumed that

$$(i) \quad \begin{cases} 1. & A \text{ is a symmetric matrix, i.e. } a_{ij} = a_{ji} \text{ for all } i,j = 1,\dots,n; \\ 2. & \lambda_{\min} = \lambda_n < \lambda_{n-1} \le \lambda_{n-2} \le \dots \le \lambda_2 < \lambda_1 = \lambda_{\max}. \end{cases}$$

For the **problem P** under conditions (i) an iterative process of the type (6) can be used for calculating the dominant eigenvalue:

$$\lambda_{\max}(A) = \lim_{i\to\infty} \frac{(h, A^i f)}{(h, A^{i-1} f)}, \tag{8}$$

since for symmetric matrices $\lambda_{\max}(A)$ is a real number.

We will be interested in evaluating both: the smallest eigenvalue $\lambda_{\min}(A)$ and the dominant one $\lambda_{\max}(A)$ using an iterative process of the type (6). It will be done by introducing a new matrix for realizing Monte Carlo iterations.

Let $A = \{a_{ij}\}_{i,j=1}^{n}$ be a given matrix and $f = \{f_i\}_{i=1}^{n}$ and $h = \{h_i\}_{i=1}^{n}$ are vectors. For the problems P we create a stochastic process using the matrix A and vectors f and h. Consider an initial density vector $p = \{p_i\}_{i=1}^{n} \in \mathbb{R}^n$, such that $p_i \ge 0, i = 1,\dots,n$ and $\sum_{i=1}^{n} p_i = 1$. Consider also a transition density matrix $P = \{p_{ij}\}_{i,j=1}^{n} \in \mathbb{R}^{n\times n}$, such that $p_{ij} \ge 0,\ i,j = 1,\dots,n$ and $\sum_{j=1}^{n} p_{ij} = 1$, for any $i = 1,\dots,n$.

Define sets of permissible densities P_h and P_A. The initial density vector $p = \{p_i\}_{i=1}^{n}$ is called *permissible* to the vector $h = \{h_i\}_{i=1}^{n} \in \mathbb{R}^n$, i.e. $p \in P_h$, if

$$p_i > 0, \ \text{when } h_i \ne 0 \text{ and } p_i = 0, \quad \text{when } h_i = 0 \text{ for } \ i = 1,\dots,n. \tag{9}$$

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

(10) $$p_{ij} > 0, \text{when } a_{ij} \neq 0 \text{ and } p_{ij} = 0, \text{ when } a_{ij} = 0 \text{ for } i, j = 1, \ldots, n.$$

Consider the following Markov chain:

(11) $$k_0 \to k_1 \to \ldots \to k_i.$$

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

The rules for constructing the chain (11) are:

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

Assume that

(13) $$p = \{p_\alpha\}_{\alpha=1}^{n} \in P_h, \quad P = \{p_{\alpha\beta}\}_{\alpha,\beta=1}^{n} \in P_A.$$

Now define the random variables W_j using the following recursion formula:

(14) $$W_0 = \frac{h_{k_0}}{p_{k_0}}, \quad W_j = W_{j-1}\frac{a_{k_{j-1}k_j}}{p_{k_{j-1}k_j}}, \quad j = 1, \ldots, i.$$

The random variables W_j, $j = 1, \ldots, i$ can also be considered as weights on the Markov chain (12).

From all possible permissible densities we choose the following

(15) $$p = \{p_\alpha\}_{\alpha=1}^{n} \in P_h, \; p_\alpha = \frac{|h_\alpha|}{\sum_{\alpha=1}^{n} |h_\alpha|};$$

(16) $$P = \{p_{\alpha\beta}\}_{\alpha,\beta=1}^{n} \in P_A, p_{\alpha\beta} = \frac{|a_{\alpha\beta}|}{\sum_{\beta=1}^{n} |a_{\alpha\beta}|}, \alpha = 1, \ldots, n.$$

Such a choice of the initial density vector and the transition density matrix leads to 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.*

It is easy to show [12], that under the conditions (i), the following equalities are fulfilled:

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

(18) $$\frac{E\{W_i f_{k_i}\}}{E\{W_{i-1} f_{k_{i-1}}\}} \approx \lambda_{\max}(A), \quad \text{for sufficiently large "i"} \quad (P).$$

2.2 The Resolvent Monte Carlo (RMC) algorithm

Now consider an algorithm based on Monte Carlo iterations by the resolvent operator $R_q = [I - qA]^{-1} \in \mathbb{R}^{n \times n}$.

The following presentation

$$[I - qA]^{-m} = \sum_{i=0}^{\infty} q^i C^i_{m+i-1} A^i, \; |q|\lambda < 1 \tag{19}$$

is valid because of behaviors of binomial expansion and the spectral theory of linear operators (the matrix A is a linear operator) [8]. The eigenvalues of the matrices R_q and A are connected with the equality $\mu = \frac{1}{(1-q\lambda)}$, and the eigenfunctions coincide. According to (8), the following expression

$$\mu^{(m)} = \frac{([I - qA]^{-m} f, h)}{([I - qA]^{-(m-1)} f, h)} \to_{m \to \infty} \mu = \frac{1}{1 - q\lambda}, \quad f \in R^n, h \in R^n. \tag{20}$$

is valid. For a negative value of q, the dominant eigenvalue $\mu_{\max}$ of R_q corresponds to the smallest eigenvalue $\lambda_{\min}$ of the matrix A. For a positive value of q, the dominant eigenvalue $\mu_{\max}$ of R_q corresponds to the dominant eigenvalue $\lambda_{\max}$ of the matrix A. Now, for constructing the method it is sufficient to prove the following theorem.

Theorem 1. *Let $\lambda'_{\max}$ be the largest eigenvalue of the matrix $A' = \{|a_{ij}|\}^n_{i,j=1}$ If q is chosen such that $|\lambda'_{\max} q| < 1$, then*

$$([I - qA]^{-m} f, h) = E\{\sum_{i=0}^{\infty} q^i C^i_{m+i-1} W_i f(x_i)\}. \tag{21}$$

PROOF. Since the expansion (19) converges in uniform operator topology [8] it converges for any vector $f \in \mathbb{R}^n$:

$$([I - qA]^{-m} f, h) = \sum_{i=0}^{\infty} q^i C^i_{m+i-1} (A^i f, h). \tag{22}$$

For obtaining (22) from (21) one needs to apply (17) and to average every term of the presentation (22). Such averaging will be correct if A, f, h and q in (21) are replaced by their absolute values. If it is done the sum (21) will be finite since the condition $|\lambda'_{\max} q| < 1$ is fulfilled. Thus, for a finite sum (21) there is a finite majorant summed over all terms and the expansion can be average over all terms. The theorem is proved. □

After some calculations one can obtain

$$\lambda \approx \frac{1}{q}(1 - \frac{1}{\mu^{(m)}}) = \frac{(A[I - qA]^{-m} f, h)}{([I - qA]^{-m} f, h)} =$$

$$\frac{E\sum_{i=1}^{\infty} q^{i-1} C^{i-1}_{i+m-2} W_i f(x_i)}{E\sum_{i=0}^{\infty} q^i C^i_{i+m-1} W_i f(x_i)}. \tag{23}$$

The coefficients C^n_{n+m} are calculated using the presentation

$$C^i_{i+m} = C^i_{i+m-1} + C^{i-1}_{i+m-1}.$$

From the representation

$$\mu^{(m)} = \frac{1}{1-|q|\lambda^{(m)}} \approx \frac{(h,[I-qA]^{-m}f)}{(h,[I-qA]^{-(m-1)}f)}, \tag{24}$$

we obtain the following Resolvent Monte Carlo (RMC) algorithm for evaluating the smallest (largest) eigenvalue:

$$\lambda \approx \frac{1}{q}\left(1-\frac{1}{\mu^{(m)}}\right) \approx \frac{E\sum_{i=0}^{l} q^i C^i_{i+m-1} W_{i+1} f(x_i)}{E\sum_{n=0}^{l} q^i C^i_{i+m-1} W_i f(x_i)}, \tag{25}$$

where $W_0 = \frac{h_{k_0}}{p_{k_0}}$ and W_i are defined by (14).

When we are interested in evaluating the smallest eigenvalue the parameter $q < 0$ has to be chosen so that to minimize the following expression

$$J(q,A) = \frac{1+|q|\lambda_1}{1+|q|\lambda_2}, \tag{26}$$

or if $\lambda_1 = \alpha\lambda_2, \quad (0 < \alpha < 1)$,

$$J(q,A) = 1 - \frac{|q|\lambda_2(1-\alpha)}{1+|q|\lambda_2}. \tag{27}$$

We choose

$$q = -\frac{1}{2\|A\|} \tag{28}$$

but sometimes a slightly different value of q might give better results when a number of realizations of the algorithm is considered.

2.3 Monte Carlo Algorithm with Inverse Iterations (MCII) for the smallest eigenvalue

Here an **Monte Carlo algorithm with inverse iterations** (MCII) is also considered.

This algorithm can be applied when A is a non-singular matrix. The algorithm has a high efficiency when the smallest by modules eigenvalue of A is much smaller then other eigenvalues. This algorithm can be realized as follow:

1. Calculate the inversion of the matrix A.

2. Starting from the initial vector $f_0 \in R^n$ calculate the sequence of Monte Carlo iterations:

$$f_1 = A^{-1}f_0,\ f_2 = A^{-1}f_1 \ldots,\ f_i = A^{-1}f_{i-1}, \ldots$$

For the Monte Carlo methods it is more efficient first to evaluate the inverse matrix using the algorithm proposed in and after that to apply the Monte Carlo iterations.

The vectors $f_i \in R^n$ converge to the eigenvector which corresponds to the smallest by modules eigenvalue of A. In fact, we calculate the functionals

$$\frac{(Af_i, h_i)}{(f_i, h_i)} = \frac{(f_{i-1}, h_i)}{(f_i, h_i)}.$$

It is not necessary to calculate A^{-1} because the vectors f_k can be evaluated solving the following system of equations:

$$Af_1 = f_0$$
$$Af_2 = f_1$$
$$\dots$$
$$Af_i = f_{i-1}.$$

3 Parallel Implementation

In this section we consider some estimators of the quality of the parallel algorithm – *Speed-up* and *Parallel efficiency.* Here we also give a brief description of the supercomputers Intel-PARAGON and CRAY Y-MP C92A used for implementation of the algorithm. In the end of the section we present some numerical results of parallel implementation of the algorithm for some test matrices.

3.1 Estimators of the quality of the algorithm

In this section the estimations for the mathematical expectation of the time, speed-up and parallel efficiency will be presented. All three parameters define the quality of the parallel algorithms.

To get theoretical estimates of the time, speed-up and parallel efficiency a model of multiprocessor configuration consisting of p processors is considered. Every processor of the multiprocessor system performs its own instructions on the data in its own memory.

The inherent parallelism of the Monte Carlo methods lies in the possibility of calculating each realization of the random variable θ on a different processor (or computer). There is no need for communication between the processors during the time of calculating 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 algorithm, we use the criterion grounded in [5].

We consider the following estimator for the *speed-up* of the Monte Carlo algorithm [5]

(29) $$S_p(X) = \frac{ET_1(X)}{ET_p(X)},$$

where $ET_p(X)$ is the mathematical expectation of the computational complexity (or the time of the algorithm) needed for realizing the algorithm X on p processors.

We shall call the algorithm B *p-the-best* if

$$ET_p(X) \geq ET_p(B). \tag{30}$$

(Obviously, if an algorithm D is a deterministic algorithm, then $ET_p(D) = T_p(D)$, for any $p = 1, 2, \ldots.$)

The *parallel efficiency* is defined as

$$E_p(X) = S_p(X)/p. \tag{31}$$

For many 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 different. The parallel efficiency does not depend on p and may be very close to 1 for a large number of processors p.

Here we shall consider the Monte Carlo algorithms for calculating the smallest eigenvalue of symmetric matrices and will get estimations for speed-up and parallel efficiency.

Every move in a Markov chain is done according to the following algorithm:

(i) generation of a random number (it is usually done in k arithmetic operations where $k = 2$ or 3);

(ii) determination of the next element of the matrix : this step includes a random number of logical operations [*];

(iii) calculating the corresponding random variable.

Since Monte Carlo Almost Optimal (MAO) algorithm 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. Obviously, the number of logical operations γ_L in every move of the Markov chain can be estimated using the following expression

$$E\gamma_L \approx \frac{1}{2}\frac{1}{n}\sum_{i=1}^{n} d_i = \frac{1}{2}d. \tag{32}$$

Since γ_L is a random variable we need an estimation of the probable error of (32). It depends on the balance of the matrix. For matrices which are not very Dis balanced and of not vary small-size ($n = 2, 3$), the probable error of (32) is negligible small in comparison with γ_L.

The number of arithmetic operations, excluding the number of arithmetic operations k for generating the random number is γ_A.

The mathematical expectation of the operations needed for each move of any Markov chain is

[*] Here the logical operation deals with testing the inequality "$a < b$".

$$E\delta = \tau \left[(k + \gamma_A) l_A + \frac{1}{2} d l_L \right], \tag{33}$$

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 and the transition density matrix, the algorithm needs d_i multiplications for obtaining the i-th row of the transition density matrix and $2dn$ arithmetic operations for constructing $\{p_{\alpha\beta}\}_{\alpha,\beta=1}^{n}$, where d is determined by (32).

Thus, the mathematical expectation of the total number of operations becomes

$$ET_1(RMC) \approx 2\tau \left[(k + \gamma_A) l_A + \frac{1}{2} d l_L \right] lN + 2\tau n(1 + d), \tag{34}$$

where l is the numbers of moves in every realization of the Markov chain, and N is the number of realizations.

It is worth noting that the main term of (34) does not depend on the size n of the matrix and the next term (which corresponds to creating the transition density matrix and can considered as a preprocessing) 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 RMC practically does not depend n. The parameters l and N depend on the spectrum of the matrix, but not depend on the size n. The above mentioned result was confirmed for a wide range of matrices during the realized numerical experiments.

Let as also note that the main term of the estimate (34) can be written in the following form

$$ET_1'(RMC) = 2\tau \left[(k + \gamma_A) l_A + \frac{1}{2} d l_L \right] lN = (k_1 + k_2 d) lN, \tag{35}$$

where k_1 and k_2 are constants which do not depend on the matrix, and parameters l and N.

The numerical results performed on Intel-PARAGON machine show that the following values can be used as an approximation to the constants k_1 and k_2

$$k_1 \approx 10^{-3};$$
$$k_2 \approx 4.5 \cdot 10^{-4}.$$

3.2 Numerical Tests

Here we present some numerical results of implementation of the algorithm under consideration. The code is written in FORTRAN 77 and is performed on supercomputers Intel PARAGON and CRAY Y-MP C92A.

Intel PARAGON is a particular form of a parallel machine which consists of a set of independent processors, each with its own memory, capable of operating on its own data. The PARAGON machine on which our experiments are performed consists of a mesh-grid of $16 \times 4 = 64$ nodes.

Each processor executes the same program for N/p number of random trajectories, that is N/p independent realizations of the random variable (25). At the end of this part of computations - the host processor collects the results of all realizations and computes the average value which corresponds to the considered eigenvalue.

Table 1: Test matrices

Name	Size	Non — zero el. per row	$\lambda_{\min}$	$\lambda_{\max}$
$m128.52$	128	52	1.0000	64.0000
$m512.178$	512	178	1.0000	64.0000
$m1000.39$	1000	39	-1.9000	1.0000
$m1024.56$	1024	56	1.0000	64.0000
$m1024.322$	1024	322	1.0000	64.0000
$m2000.56$	2000	56	1.0000	64.0000

Numerical tests are performed for a number of test matrices - general symmetric sparse matrices and band sparse symmetric matrices. The test matrices are produced using a specially created generator of symmetric matrices called *MATGEN*. This generator allows to generate matrices with a given size, given sparsity and fixed largest and smallest eigenvalue (fixed condition number). All other eigenvalues are chosen to be randomly distributed. Using MATGEN-program it is also possible to put a gap of a given size into the spectrum of the matrix. For producing such matrices in MATGEN Jacobi rotations are used such that the angle of rotation and the place of appearing the non-zero entrances are randomly chosen. The test matrices used in our numerical experiments are of size 128, 512, 1000, 1024, and 2000 and have different number of non-zero elements. Some of the most important parameters of the matrices are shown in Table 1. The name of matrices contains the size of the matrix and also a parameter which indicates the sparsity. So, we a able to control the parallel behaviors of the algorithm for different levels of sparsity and to study the dependence between the computational time and the size of the matrices.

Some information about the computations complexity, speed-up and parallel efficiency of the algorithm is presented in Tables 2- 6.

Table 2 presents results for a matrix of size 512×512 with a given sparsity when a small number of Monte Carlo iterations is needed to receive a good accuracy. The subtable "a" show the dependence between calculated values and the number of random trajectories. The subtable "b" contain an information about the dependence of the computational complexity, speed-up and parallel efficiency from the used number of processors p of Intel PARAGON.

Table 3 presents results for a matrix of size 2000×2000 with a given sparsity when

Table 2: **Resolvent Monte Carlo Method (RMC) for tr.min ($\lambda_{\min} = 1.0$).**

a) **The solution when number of trajectories increases.**

Number of traject.	10^3	10^4	10^5	10^6
Calculated $\lambda_{\min}$	1.0278	0.9958	0.999984	1.000010

b) **Implementation on PARAGON (Num. of tr. $N = 10^6$). Calculated $\lambda_{\min} = 1.000010$.**

Numb. nodes	1	2	3	4	5	6	7	8	9	10
Time (s)	70.75	35.97	24.9	18.3	14.78	12.96	11.49	9.63	8.67	7.68
Speed-up S	1	1.97	2.84	3.86	4.78	5.46	6.16	7.34	8.16	9.21
Efficiecy E	1	0.983	0.947	0.966	0.957	0.909	0.879	0.918	0.906	0.921

a **large** number of iterations are needed ($m = 91$). Here an information about the computational error, time, speed-up and parallel efficiency is given. Our results show that the parallel efficiency increases when the number of random trajectories increases.

Table 4 shows that:

- the computational time is almost independent of the size of the matrix;
- a linear dependence between the computational time and the number of the random trajectories is observed;
- a linear dependence between the computational time and the mean value of the number of non-zero entries of each row of the matrix is realized.

Some results for the speed-up when different numbers of processors of Intel PARAGON machine are used are also received. When the number of the random trajectories is large the speed-up is almost linear and it is closed to the best value of the speed-up, i.e. the value of p. When the number of processors is small (with respect to the computational complexity) the speed-up is linear. If the number of processors p is large and the computational time is small the speed-up is not so good. It may happen for tasks with a very small computational complexity when such large number of processors is not needed.

Table 3: **Resolvent Monte Carlo Method (RMC) for tr2000.2min ($\lambda_{\min} = 1.0$). Parameters of the problem: M=91, L=5, f()- unit vector**

a) The solution when number of trajectories increases.

Number of traject.	10^3	10^4	10^5	10^6
Calculated $\lambda_{\min}$	0.88238	0.9074	0.9248	0.9262

b) Implementation on PARAGON (Num. of tr. $N = 10^6$). Calculated $\lambda_{\min} = 1.000010$.

Nnodes	$\lambda_{\min}$	Time (s)	Speed - up	*Efficiency*
1	0.9262	107.488	1	1
2	0.9266	54.752	1.963	0.981
3	0.9265	37.088	2.822	0.940
4	0.9256	27.424	2.822	0.705
5	0.9268	22.432	3.919	0.783
6	0.9253	19.456	4.791	0.783
7	0.9256	17,312	5.524	0.798
8	0.9279	14.496	6.208	0.789
9	0.9260	12.992	7.415	0.776
10	0.9282	12.288	8.273	0.823

We also consider some numerical results obtained on CRAY Y-MP C92A. CRAY Y-MP C92A is a typical pipeline machine, which contain two vector processors.

Some numerical results are presented on Tables 5 and 6. One can see that MCII algorithm gives good results even in case of small values of parameters m and N. It is also shown that the size of the HWM-memory is relatively small, which is important for such kind of machine like CRAY Y-MP C92A. The obtained information about the efficiency of vectorization show that MCII algorithm is well vectorizable.

Remarks:

1. The values of CP-time and HWM-memory are for CRAY Y-MP C92A.

2. "$\star$" - no estimated CP-time; the values of CP-time are between 5.296 s and 5.514 s.

3. In comparison with case b), CP-time decreases very slowly for more then 10-times decreasing of the number of moves m.

4. The corresponding NAG-routine for solving the same problem needs CP-time =

Table 4: **Computing time and matrix size**

a) **Results for matrix 128 × 128. Number of nonzero elements = 6714. Exact $\lambda_{\max} = 64.0$. parametersd: M=47, L=7.**

Number of traject.	Calculated $\lambda_{\max}$	Time
1000	64.1350	0.256
10000	63.3300	2.112
100000	63.1843	21.600
1000000	63.189	208.256

b) **Results for matrix 1000 × 1000. Number of nonzero elements = 38748. Exact $\lambda_{\max} = 1.0$. parametersd: M=47, L=7.**

Number of traject.	Calculated $\lambda_{\max}$	Time
1000	0.9981	0.128
10000	0.9997	1.344
100000	1.000051	13.184
1000000	1.000033	132.288

c) **Results for matrix 1024 × 1024. Number of nonzero elements = 57538. Exact $\lambda_{\max} = 64.0$. parametersd: M=47, L=7.**

Number of traject.	Calculated $\lambda_{\max}$	Time
1000	64.3462	0.256
10000	64.1883	1.856
100000	64.1724	18.176
1000000	64.1699	181.504

d) **Results for matrix 2000 × 2000. Number of nonzero elements = 112594. Exact $\lambda_{\max} = 64.0$. parametersd: M=47, L=7.**

Number of traject.	Calculated $\lambda_{\max}$	Time
1000	63.9943	0.192
10000	64.0050	1.408
100000	64.0204	13.312
1000000	64.0265	133.248

Table 5: **Monte Carlo algorithm with inverse iterations (IMCM) for MS512.2 ($\lambda_{\min} = 0.2736$). (A general symmetric matrix of size 512.)**
a) **The number of Markov chains is fixed $N = 80$.**

m	*Calculated* $\lambda_{\min}$	*Error,*
2	0.2736	0.0000
3	0.2733	0.0011
4	0.2739	0.0011
5	0.2740	0.0015
10	0.2732	0.0015
50	0.2738	0.0007
100	0.2757	0.0076

b) **The number of iterations (number of moves in every Markov chain) m is fixed - $m = 50$.**

N	*Calculated* $\lambda_{\min}$	*Error,*	$CP - time,$ s	$HWM-$ *memory*
20	0.2729	0.0026	5.356	1137378
40	0.2742	0.0022	5.396	1137378
60	0.2748	0.0044	5.468	1137378
80	0.2739	0.0011	5.524	1137378
100	0.2736	0.0000	5.573	1137378
500	0.2737	0.0004	6.666	1137378
1000	0.2739	0.0011	8.032	1137378

Remark: The values for CP-time and HWM-memory are for CRAY Y-MP C92A.

Table 6: **The number of iterations (number of moves in every Markov chain) for MS512.2 ($\lambda_{\min} = 0.2736$) m is small and fixed - $m = 4$.**

N	*Calculated* $\lambda_{\min}$	*Error,*	$CP - time,$ s	$HWM-$ *memory*
20	0.2737	0.0004	5.296	1137378
40	0.2749	0.0058	⋆	1137378
60	0.2754	0.0066	⋆	1137378
80	0.2739	0.0011	⋆	1137378
100	0.2736	0.0000	⋆	1137378
500	0.2737	0.0004	⋆	1137378
1000	0.2738	0.0007	5.514	1137378

5.452 s and HWM-mem = 1 220 676.

4 Concluding Remarks

In this paper:

- A parallel Resolvent Monte Carlo Algorithm and a Monte Carlo algorithm with inverse iterations for evaluating eigenvalues of real symmetric matrices have been presented and implemented.
- Estimations for the computational complexity, speed-up and parallel efficiency are obtained.
- The studied algorithms are ***almost optimal*** from statistical point of view, i.e. the variance of the random variable which is equal to $\lambda_{\min}$ or $\lambda_{\max}$ is ***almost minimal*** in the meaning of definition given in [2].
- The studied algorithms are implemented on supercomputers Intel PARAGON and CRAY Y-MP C92A.
- The convergence of the algorithm depends on spectrum of the matrix. The systematic error is

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

  where m is the power (the number of iterations). When $\lambda_{\max} \approx -2\lambda_{\min}$ the convergence is very good. It is clear from (36) that the positive or negative defined matrices the convergence decreases, so that the best convergence which can be reached is $O[(2/3)^m]$.
- The presented algorithms have strong requirements about matrices for which it can be applied: the error from the Power method applied on the resolvent matrix determines the value of the parameter m; the error which comes from the representation the resolvent matrix as a series determines the value of the parameter l, and also the values of m and l are not independent since they determine the binomial coefficients C^l_{m+l-1} which grow exponentially with l.
- The numerical results obtained by Resolvent Monte Carlo algorithm for sparse matrices show that:
  - The computational time is almost independent of the size of the matrix.
  - There is a linear dependence between the computational time and the number of the random trajectories.

- There is a linear dependence between the computational time and the mean value of the number of non-zero entries of each row of the considered matrix.
- The speed-up of the algorithm is almost linear when the computational effort for every processor is not too small.

• The presented Monte Carlo algorithms can be efficiently used for solving other important linear algebra problems, where one is interested in computing powers of matrices. Such a problem is evaluating polynomials of large sparse matrices $p(A)$, which are used for obtaining some information about the spectrum of the matrices and also for studying the stability of large systems.

Acknowledgments.
This work was partially supported by the Ministry of Science, Education and Technology of Bulgaria under grant I 501/95.
The authors thank Professor Bernard Philippe – coordinator of the INCO-COPERNICUS project: 960237 for the helpful discussions and for possibility to use the Intel PARAGON machine for our computations.

REFERENCES

[1] S. K. Godunov. Spectral portraits of matrices and criteria of spectrum dichotomy. In: International symposium on computer arithmetic and scientific computation (eds. J. Herzberger, L. Atanasova). Oldenburg, Germany, North-Holland (1991).

[2] I. Dimov. Minimization of the Probable Error for Some Monte Carlo methods. Proc. Int. Conf. on Mathematical Modeling and Scientific Computation, Varna, 1991.

[3] I. Dimov, O. Tonev. Monte Carlo methods with overconvergent probable error. In: Numerical Methods and Applications, Proc. of Intern. Conf on Numerical Methods and Appl.,Sofia, House of Bulg. Acad. Sci., Sofia, 1989, 116-120.

[4] I. Dimov, O. Tonev. Random walk on distant mesh points Monte Carlo methods. *Journal of Statistical Physics*, **70**(5/6), 1993, 1333-1342.

[5] I. Dimov, O. Tonev. Monte Carlo algorithms: performance analysis for some computer architectures. *Journal of Computational and Applied Mathematics* **48** (1993), 253-277.

[6] V. Dupach. Stochasticke pocetni metody. *Cas. pro pest. mat.* **81**(1) (1956), 55-68.

[7] H. Kahn. Random sampling (Monte Carlo) techniques in neutron attenuation problems. *Nucleonics* **6** (5) (1950), 27-33; **6**(6) (1950) 60-65.

[8] L. V. Kantorovich, G. P. Akilov. Functional analysis. Nauka, Moscow, 1977.

[9] G. Megson, V. Aleksandrov, I. Dimov. Systolic Matrix Inversion Using a Monte Carlo Method, *Journal of Parallel Algorithms and Applications*, **3**(3/4) (1994), 311-330.

[10] G. A. Mikhailov. A new Monte Carlo algorithm for estimating the maximum eigenvalue of an integral operator. *Docl. Acad. Nauk SSSR*, **191**(5) (1970), 993-996.

[11] G. A. Mikhailov. Optimization of the "weight" Monte Carlo methods. Nauka, Moscow, 1987.

[12] I. M. SOBOL. Monte Carlo numerical methods. Nauka, Moscow, 1973.

[13] L. N. TREFETHEN. Pseudospectra of matrices. In: 14th Dundee Biennal Conference on Numerical Analysis (eds. D. F. Griffiths, G. A. Watson), 1991.

Central Laboratory for Parallel Computing
Bulgarian Academy of Sciences
Acad. G. Bonchev St.,bl. 25 A, Sofia, 1113, Bulgaria
e-mail: `anet@amigo.acad.bg`, `dimov@amigo.acad.bg`
Web site: http://www.acad.bg/BulRTD/math/dimov2.html