A New Walk on Equations Monte Carlo Method for Linear Algebraic Problems

Ivan Tomov Dimov, Sylvain Maire, Jean-Michel Sellier

A New *Walk on Equations* Monte Carlo Method for Linear Algebraic Problems ⋆

Ivan Dimov [a,*] Sylvain Maire [b] Jean Michel Sellier [c]

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

[b] *Laboratoire LSIS Equipe Signal et Image, Université du Sud Toulon-Var, AV. Georges Pompidou BP 56 83162 La Valette du Var Cedex*

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

Abstract

A new *Walk on Equations* (WE) Monte Carlo algorithm for Linear Algebra (LA) problem is proposed and studied. This algorithm relies on a non-discounted sum of an absorbed random walk. It can be applied for either real or complex matrices. Several techniques like simultaneous scoring or the sequential Monte Carlo method are applied to improve the basic algorithm. Numerical tests are performed on examples with matrices of different size and on systems coming from various applications. Comparisons with standard deterministic or Monte Carlo algorithms are also done.

Key words: Monte Carlo algorithms, Markov chain, eigenvalue problem, robust algorithms.
1991 MSC: 65C05, 65U05, 65F10, 65Y20

⋆ This work was partially supported by the European Commission under FP7 project AComIn (FP7 REGPOT-2012-2013-1, by the Bulgarian NSF Grant DCVP 02/1, as well as by the Université du Sud Toulon-Var.

* Corresponding author.
Email addresses: ivimov@bas.bg (Ivan Dimov), sylvain.maire@univ-tln.fr (Sylvain Maire), jeanmichel.sellier@parallel.bas.bg (Jean Michel Sellier).
URLs: http://parallel.bas.bg/dpa/BG/dimov/index.html (Ivan Dimov), http://maire.univ-tln.fr/ (Sylvain Maire), http://www.bas.bg (Jean Michel Sellier).

1 Introduction

In this paper we analyze applicability and efficiency of the proposed WE algorithm. The algorithm is based on Markov chain Monte Carlo (MC) approach. We focus our consideration to real symmetric matrices, but a wider class of matrices can also be treated.

Many scientific and engineering applications are based on the problems of solving systems of linear algebraic equations. For some applications it is also important to compute directly the inner product of a given vector and the solution vector of a linear algebraic system. For many LA problems it is important to be able to find a preconditioner with relatively small computational complexity. That is why, it is important to have relatively *cheap* algorithm for matrix inversion. 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 [2,3,7,23,29–31]. Let J be the exact solution of the problem under consideration. Suppose it is proved that there exists a random variable θ, such that $E\{\theta\} = J$. If one can produce N values of θ, i.e. $\theta_1, \ldots, \theta_N$, then the value $\bar{\theta}_N = \sum_{i=1}^{N} \theta_N$ can be considered as a Monte Carlo (MC) approximation of J. The probability error of the MC algorithm is defined as the least possible real number R_N, for which: $P = Pr\{|\sum_{i=1}^{N} \theta_N - J| \leq R_N\}$, where $0 < P < 1$.

Several authors have presented works on MC algorithms for LA problems and on the estimation of computational complexity [1,6,15,8,10,11]. There are MC algorithms for

- computing components of the solution vector;
- evaluating linear functionals of the solution;
- matrix inversion, and
- computing the extremal eigenvalues.

An overview of all these algorithms is given in [7]. The well-known Power method [17] gives an estimate for the dominant eigenvalue λ_1. This estimate uses the so-called *Rayleigh quotient*. In [9] the authors consider bilinear forms of matrix powers, which is used to formulate a solution for the eigenvalue problem. A 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 is considered. In [9] 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). It is shown how one has to choose the acceleration parameter

q in case of using Resolvent Power MC. The systematic error is analyzed and it is shown that the convergence can not be better than $O\left(\frac{1+|q|\lambda_n}{1+|q|\lambda_{n-1}}\right)^m$, where λ_n is the smallest by modulo eigenvalue, q is the characteristic parameter, and m is the number of MC iterations by the resolvent matrix. To construct an algorithm for evaluating the minimal by modulo eigenvalue λ_n, one has to consider the following matrix polynomial $p_k(A) = \sum_{k=0}^{\infty} q^k C_{m+k-1}^k A^k$, where C_{m+k-1}^k are binomial coefficients, and the characteristic parameter q is used as acceleration parameter of the algorithm [8,12,13]. This approach is a discrete analogues of the resolvent analytical continuation method used in the functional analysis [24]. There are cases when the polynomial becomes the resolvent matrix [11,12,7]. 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 [4] 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 [4] uses Monte Carlo estimates of the forward and adjoint fluxes. In [27] an unbiased estimation of the solution of the system of linear algebraic equations is presented. The proposed estimator can be used to find one component of the solution. Some results concerning the quality and the properties of this estimator are presented. Using this estimator the author gives error bounds and constructs confidence intervals for the components of the solution. In [16] a Monte Carlo algorithm for matrix inversion is proposed and studied. The algorithm is based on the solution of simultaneous linear equations. In our further consideration we will use some results from [16] and [7] to show how the proposed algorithm can be used for approximation of the inverse of a matrix.

2 Formulation of the Problem: Solving Linear Systems and Matrix Inversion

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

$$A = \{a_{ij}\}_{i,j=1}^n = (a_1, \ldots, a_i, \ldots, a_n)^t,$$

where $a_i = (a_{i1}, \ldots, a_{in}),\ i = 1, \ldots, n$ and the symbol t means *transposition*.

The following norms of vectors (l_1-norm):

$$\| b \| = \| b \|_1 = \sum_{i=1}^n |b_i|, \quad \| a_i \| = \| a_i \|_1 = \sum_{j=1}^n |a_{ij}|$$

and matrices

$$\| A \| = \| A \|_1 = \max_j \sum_{i=1}^{n} |a_{ij}|$$

are used.

Consider a matrix $A \in I\!R^{n \times n}$ and a vector $b = (b_1, \ldots, b_n)^t \in I\!R^{n \times 1}$. The matrix A can be considered as a linear operator $A[I\!R^n \to I\!R^n]$ [24,7], so that the linear transformation

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

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

Since iterative Monte Carlo algorithms using the transformation (1) will be considered, the linear transformation (1) is called *iteration*. The algebraic transformation (1) plays a fundamental role in the iterative Monte Carlo algorithms [7].

Now consider the following three LA problems **Pi (i=1,2,3)**:

Problem P1. Evaluating the inner product

$$J(u) = (v, x) = \sum_{i=1}^{n} v_i x_i$$

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

$$Bx = f,$$

where $B = \{b_{ij}\}_{i,j=1}^{n} \in I\!R^{n \times n}$ is a given matrix; $f = (f_1, \ldots, f_n)^t \in I\!R^{n \times 1}$ and $v = (v_1, \ldots, v_n)^t \in I\!R^{n \times 1}$ are given vectors.

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

Then

$$x = Ax + b. \tag{2}$$

It will be assumed that

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

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

$$Ax = \lambda(A)x$$

is satisfied. If the conditions (i) are fulfilled, then a *stationary linear iterative algorithm* [7,6] can be used:

$$x_k = Ax_{k-1} + b, \quad k = 1, 2, \ldots \tag{3}$$

and the solution x can be presented in a form of a von Neumann series

$$x = \sum_{k=0}^{\infty} A^k b = b + Ab + A^2 b + A^3 b + \ldots \tag{4}$$

The *stationary linear iterative Monte Carlo algorithm* [7,6] is based on the above presentation and will be given later on. As a result, the convergence of the Monte Carlo algorithm depends on the truncation error of the series (3) (see, [7]).

Problem P2. Evaluating all components $x_i,\ i = 1, \ldots n$ of the solution vector.

In this case we deal with the equation (2) assuming that $A = I - MB$, as well as, $b = Mf$.

Problem P3. Inverting of matrices, i.e. evaluating of matrix

$$G = B^{-1},$$

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

Assumed that the following conditions are fulfilled:

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

Obviously, if the condition (i) is fulfilled, the solution of **the problem P1** can be obtained using the iterations (3).

For **problem P3** the following *iterative* matrix:

$$A = I - B$$

can be constructed.

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

$$G = \sum_{i=0}^{\infty} A^i.$$

For problems $\mathbf{P_i}(\mathbf{i = 1, 2, 3})$ one can create a stochastic process using the matrix A and vectors b, and possibly v.

Consider an initial density vector $p = \{p_i\}_{i=1}^n \in I\!R^n$, such that $p_i \geq 0, i = 1, \ldots, n$ and $\sum_{i=1}^n p_i = 1$. Consider also a transition density matrix $P = \{p_{ij}\}_{i,j=1}^n \in I\!R^{n\times n}$, such that $p_{ij} \geq 0,\ i, j = 1, \ldots, n$ and $\sum_{j=1}^n p_{ij} = 1$, for any $i = 1, \ldots, n$. Define sets of *permissible* densities $\mathcal{P}_b$ and $\mathcal{P}_A$.

Definition 2.1 *The initial density vector $p = \{p_i\}_{i=1}^n$ is called permissible to the vector $b = \{b_i\}_{i=1}^n \in I\!R^n$, i.e. $p \in \mathcal{P}_b$, if*

$$\begin{cases} p_{\alpha_s} > 0 \quad when\ \ v_{\alpha_s} \neq 0 \\ p_{\alpha_s} = 0 \quad when\ \ v_{\alpha_s} = 0. \end{cases} \tag{5}$$

Similarly, 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. $P \in \mathcal{P}_A$, if

$$\begin{cases} p_{\alpha_{s-1},\alpha_s} > 0 \ \text{ when } \ a_{\alpha_{s-1},\alpha_s} \neq 0 \\ p_{\alpha_{s-1},\alpha_s} = 0 \ \text{ when } \ a_{\alpha_{s-1},\alpha_s} = 0. \end{cases} \tag{6}$$

In this work we will be dealing with with *permissible* densities. It is obvious that in such a way the random trajectories constructed to solve the problems under consideration never visit *z*ero elements of the matrix. Such an approach decreases the computational complexity of the algorithms. It is also very convenient when large sparse matrices are to be treated.

We shall use the so-called MAO algorithm studied in [7,6,5,9]. Here we give a brief presentation of MAO. Suppose we have a Markov chain:

$$T = \alpha_0 \to \alpha_1 \to \ldots \alpha_k \to \ldots, \tag{7}$$

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

$$T_k = \alpha_0 \to \alpha_1 \to \ldots \alpha_j \to \ldots \alpha_k, \tag{8}$$

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

Assume that

$$P(\alpha_0 = \alpha) = p_\alpha \ \text{ and } \ P(\alpha_j = \beta | \alpha_{j-1} = \alpha) = p_{\alpha\beta}, \tag{9}$$

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. For the standard Markov chain Monte Carlo we normally require that

$$\sum_{\alpha=1}^{n} p_\alpha = 1 \ \text{ and } \ \sum_{\beta=1}^{n} p_{\alpha\beta} = 1 \text{ for any } \alpha = 1, \ldots, n. \tag{10}$$

The above allows the construction of the following random variable, that is a unbiased estimator for the **P**roblem P1:

$$\theta[b] = \frac{b_{\alpha_0}}{p_0} \sum_{m=0}^{\infty} Q_m b_{\alpha_m} \ , \tag{11}$$

where

$$Q_0 = 1; \qquad Q_m = Q_{m-1} \frac{a_{\alpha_{m-1},\alpha_m}}{p_{\alpha_{m-1},\alpha_m}}, \qquad m = 1, 2, \ldots \tag{12}$$

and $\alpha_0, \alpha_1, \ldots$ is a Markov chain on elements of the matrix A constructed by using an initial probability p_0 and a transition probability $p_{\alpha_{m-1},\alpha_m}$ for choosing the element $a_{\alpha_{m-1},\alpha_m}$ of the matrix A.

Now define the random variables Q_m using the formula (12). One can see, that the random variables $Q_m, m = 1, \ldots, i$ can also be considered as weights on the Markov chain (7).

One possible possible permissible densities can be chosen in the following way:

$$p = \{p_\alpha\}_{\alpha=1}^{n} \in \mathcal{P}_b, \quad p_\alpha = \frac{|b_\alpha|}{\| \ b \ \|};$$

$$P = \{p_{\alpha\beta}\}_{\alpha,\beta=1}^{n} \in \mathcal{P}_A, \quad p_{\alpha\beta} = \frac{|a_{\alpha\beta}|}{\| \ a_\alpha \ \|}, \alpha = 1, \ldots, n. \tag{13}$$

Such a choice of the initial density vector and the transition density matrix leads to an *Almost Optimal Monte Carlo* (MAO) algorithm (see [5,7]). 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* [5].

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

$$\Delta_i = \delta_0 \times \delta_1 \times \ldots \times \delta_i.$$

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

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

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

$$E\{\sum_{i=0}^{N} \Delta_i Q_i b_{k_i}\} = (h, x), \quad (P1),$$

$$E\{\sum_{i|k_i=r'} \Delta_i Q_i\} = c_{rr'}, \quad (P3),$$

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

The new WE algorithm studied in this paper was proposed by the second author. One important difference from the previously known Markov chain Monte Carlo algorithms is the special choice of the absorbtion probabilities. In this case the Markov chain has $n+1$ states, $\{1, \ldots, n, n+1\}$. The transition density matrix (10) is such that $\sum_{\beta=1}^{n} p_{\alpha\beta} \leq 1$ and the probability for absorbtion at each stage accept the starting one is $p_{\alpha,n+1} = p_\alpha = 1 - \sum_{\beta=1}^{n} p_{\alpha\beta}, \alpha = 1, \ldots, n$.

3 Description of the probabilistic representation for the WE algorithm

We start the description of the probabilistic representation of the algorithm with two simple examples. The first example is for the case when all the coefficients of the matrix are non-negative. The second example deals with the case when some matrix coefficients may be negative real numbers. In this section we give a probabilistic representation for the WE algorithm for real-valued matrices, as well as for complex-valued ones.

3.1 A simple example: the positive case

We start with a simple example of positive case meaning that $a_{ij} \geq 0,\ i,j = 1,\ldots,n$. For simplicity, we also assume that $\sum_{j=1}^{n} a_{ij} \leq 1$.

We describe on a very simple example how we obtain our probabilistic representation. We consider the system of two equations:

$$x_1 = \frac{1}{2}x_1 + \frac{1}{4}x_2 + 1,\ x_2 = \frac{1}{3}x_1 + \frac{1}{3}x_2 + 2$$

with unknowns x_1 and x_2. We have obviously $x_1 = 1 + E(X)$ where

$$P(X = x_1) = \frac{1}{2}, P(X = x_2) = \frac{1}{4}, P(X = 0) = \frac{1}{4}$$

and $x_2 = 2 + E(Y)$ where

$$P(Y = x_1) = \frac{1}{3}, P(Y = x_2) = \frac{1}{3}, P(Y = 0) = \frac{1}{3}.$$

We will use only one sample to approximate the expectations and compute a score initialized by zero along a random walk on the two equations. For instance, to approximate x_1 we add 1 to the score of our walk and then either we stop with probability $\frac{1}{4}$ because we know the value of X which is zero or either we need again to approximate $x(1)$ or $u(2)$ with probability $\frac{1}{2}$ or $\frac{1}{4}$. If the walk continues and that we need to approximate $x(2)$, we add 2 to the score of our walk, we stop with probability $\frac{1}{3}$ or either we need again to approximate x_1 or $x(2)$ with probability $\frac{1}{3}$. The walk continues until one of the random variables X or Y takes the value zero and the score is incremented along the walk as previously. The score is an unbiased estimator of x_1 and we make the average of the scores of independent walks to obtain the Monte Carlo approximation of x_1.

This algorithm is very similar to the ones used to compute the Feynman-Kac representations of boundary value problems by means of discrete or continuous walks like walk on discrete grids or the walk on spheres method. At each step of the walk, the solution at the current position x is equal to the expectation of the solution on a discrete or continuous set plus eventually a source term. The key ingredient is the double randomization principle which says that we can replace the expectation of the solution on this set by the solution at one random point picked according to the appropriate distribution. For example, for the walk on spheres methods this distribution is the uniform law on the sphere centered at x and of radius the distance to the boundary.

3.2 Simple example: general real matrix

We slightly modify our system by considering now the system of two equations

$$x_1 = \frac{1}{2}x_1 - \frac{1}{4}x_2 + 1,\ x_2 = \frac{1}{3}x_1 + \frac{1}{3}x_2 + 2$$

with unknowns x_1 and x_2. We have $x_1 = 1 + E(W)$ where

$$P(W = x_1) = \frac{1}{2}, P(W = -x_2) = \frac{1}{4}, P(W = 0) = \frac{1}{4}$$

and $x_2 = 2 + E(Z)$ where

$$P(Z = x_1) = \frac{1}{3}, P(Z = x_2) = \frac{1}{3}, P(Z = 0) = \frac{1}{3}.$$

To approximate x_1 we add 1 to the score of our walk and then either we stop with probability $\frac{1}{4}$ because we know the value of X which is zero or either we need to approximate x_1 or $-x_2$ with probability $\frac{1}{2}$ or $\frac{1}{4}$. If the walk continues and that we need to approximate $-x_2$, we add -2 to the score of our walk, we stop with probability $\frac{1}{3}$ or either we need to approximate $-x_1$ or $-x_2$ with probability $\frac{1}{3}$. If the walk continues again and that we need to approximate $-x_1$, we add -1 to the score of our walk, we stop with probability $\frac{1}{4}$ or either we need to approximate $-x_1$ or x_2 with probability $\frac{1}{2}$ or $\frac{1}{4}$. The walk continues until one of the random variables W or Z takes the value zero and the score is incremented along the walk as previously. The same kind of reasoning holds for a complex-valued matrix in the representation given later on in Subsection 3.4.

3.3 Probabilistic representation for real matrices

Consider a real linear system of the form $u = Au + b$ where the matrix A of size n is such that $\varrho(A) < 1$, its coefficients $a_{i,j}$ are real numbers and

$$\sum_{j=1}^{n} |a_{i,j}| \leq 1,\ \forall 1 \leq i \leq n.$$

We now define a Markov chain T_k with $n+1$ states $\alpha_1, \ldots, \alpha_n, n+1$, such that

$$P(\alpha_{k+1} = j/\alpha_k = i) = |a_{i,j}|$$

if $i \neq n+1$ and

$$P(\alpha_{k+1} = n+1/\alpha_k = n+1) = 1.$$

We also define a vector c such that $c(i) = b(i)$ if $1 \leq i \leq n$ and $c(n+1) = 0$. Denote by $\tau = (\alpha_0, \alpha_1, \ldots, \alpha_k, n+1)$ a random trajectory that starts at the initial state $\alpha_0 < n+1$ and passes through $(\alpha_1, \ldots, \alpha_k)$ until the absorbing state $\alpha_{k+1} = n+1$. The probability to follow the trajectory τ is $P(\tau) = p_{\alpha_0} p_{\alpha_0\alpha_1}, \ldots p_{\alpha_{k-1,k}\alpha_k} p_{\alpha_k}$. We use the MAO algorithm defined by (13) for the initial density vector $p = \{p_\alpha\}_{\alpha=1}^n$ and for the transition density matrix $P = \{p_{\alpha\beta}\}_{\alpha,\beta=1}^n$, as well. The weights Q_α are defined in the same way as in (12), i.e.,

$$Q_m = Q_{m-1} \frac{a_{\alpha_{m-1},\alpha_m}}{p_{\alpha_{m-1},\alpha_m}}, \quad m = 1, \ldots, k, \quad Q_0 = \frac{c_{\alpha_0}}{p_{\alpha_0}}. \tag{14}$$

The estimator $\theta_\alpha(\tau)$ can be presented as $\theta_\alpha(\tau) = c_\alpha + Q_k \frac{a_{\alpha_k\alpha}}{p_{\alpha_k}}, \quad \alpha = 1, \ldots n$ taken with a probability $P(\tau) = p_{\alpha_0} p_{\alpha_0\alpha_1}, \ldots p_{\alpha_{k-1,k}\alpha_k} p_{\alpha_k}$.

Now, we can prove the following theorem.

Theorem 3.1 *The random variable $\theta_\alpha(\tau)$ is an unbiased estimator of x_α, i.e.*

$$E\{\theta_\alpha(\tau)\} = x_\alpha.$$

P r o o f.

$$\begin{aligned}
E\{\theta_\alpha(\tau)\} &= \sum_{\tau=(\alpha_0,\ldots,\alpha_k,n+1)} \theta_\alpha(\tau) P(\tau) \\
&= \sum_{\tau=(\alpha_0,\ldots,\alpha_k,n+1)} \left(c_\alpha + Q_k(\tau) \frac{a_{\alpha_k\alpha}}{p_{\alpha_k}} \right) P(\tau) \\
&= c_\alpha + \sum_{\tau=(\alpha_0,\ldots,\alpha_k,n+1)} Q_k(\tau) \frac{a_{\alpha_k\alpha}}{p_{\alpha_k}} P(\tau) \\
&= c_\alpha + \sum_{\tau=(\alpha_0,\ldots,\alpha_k,n+1)} \frac{c_{\alpha_0}}{p_{\alpha_0}} \frac{a_{\alpha_0\alpha_1} \ldots a_{\alpha_{k-1}\alpha_k}}{p_{\alpha_0\alpha_1} \ldots p_{\alpha_{k-1}\alpha_k}} \frac{a_{\alpha_k\alpha}}{p_{\alpha_k}} p_{\alpha_0} \\
&\times p_{\alpha_0\alpha_1} \ldots p_{\alpha_{k-1}\alpha_k} p_{\alpha_k} \\
&= c_\alpha + \sum_{k=0}^{\infty} \sum_{\alpha_0=1}^{n} \ldots \sum_{\alpha_k=1}^{n} a_{\alpha_k\alpha} a_{\alpha_{k-1}\alpha_k} \ldots a_{\alpha_0\alpha_1} c_{\alpha_0} \\
&= c_\alpha + (Ac)_\alpha + (A^2 c)_\alpha + \ldots
\end{aligned}$$

$$= \left(\sum_{i=0}^{\infty} \left(A^i c \right) \right)_{\alpha}. \tag{15}$$

Since $\varrho(A) < 1$, the last sum in (15) is finite and

$$\left(\sum_{i=0}^{\infty} \left(A^i c \right) \right)_{\alpha} = x_{\alpha}, \quad \alpha = 1, \ldots, n.$$

Thus,

$$E\{\theta_{\alpha}(\tau)\} = x_{\alpha}. \qquad \diamondsuit$$

Let as stress on the fact that among elements $c_{\alpha_0}, a_{\alpha_0\alpha_1} \ldots a_{\alpha_{k-1}\alpha_k}$ there are no elements equal to zero because of the special choice of *permissible* distributions p_{α} and $p_{\alpha\beta}$ defined by MAO densities (13). The rules (13) ensure that the Markov chain *visits* non-zero elements only.

One may also observe that the conjugate to $\theta_{\alpha}(\tau)$ random variable $\theta^*_{\alpha}(\tau) = \sum_{i=0}^{k} Q_i \frac{a_{\alpha_i\alpha}}{p_{\alpha_i}} c_{\alpha}$ has the same mean value, i.e., x_{α}. It can be immediately checked following the same technique as in the proof of Theorem 3.1.

Assume one can compute N values of the random variable θ_{α}, namely $\theta_{\alpha,i},\ i = 1, \ldots, N$. Let us consider the value $\bar{\theta}_{\alpha,N} = \frac{1}{N} \sum_{i=1}^{N} \theta_{\alpha,i}$ as a Monte Carlo approximation of the component x_{α} of the solution. For the random variable $\bar{\theta}_{\alpha,N}$ the following theorem is valid:

Theorem 3.2 *The random variable $\bar{\theta}_{\alpha,N}$ is an unbiased estimator of x_{α}, i.e.*

$$E\{\bar{\theta}_{\alpha,N}\} = x_{\alpha} \tag{16}$$

and the variance $D\{\bar{\theta}_{\alpha,N}\}$ vanishes when N goes to infinity, i.e

$$\lim_{N\to\infty} D\{\bar{\theta}_{\alpha,N}\} = 0. \tag{17}$$

P r o o f. Equality (16) can easily be proved using properties of the mean value. Using the variance properties one can also show that (17) is true. $\diamondsuit$

Moreover, using the Kolmogorov theorem and the fact that the sequences $(\theta_{\alpha,N})_{N\geq 1}$ are independent and also independently distributed with a finite mathematical expectation equal to x_{α} one can show that $\bar{\theta}_{\alpha,N}$ converges almost surely to x_{α}:

$$P(\lim_{N\to\infty} \bar{\theta}_{\alpha,N} = x_{\alpha}) = 1.$$

The theorems proved (Th.3.1 and Th.3.2) with the last fact allows us to use the Monte Carlo method with the random variable θ_α for solving algebraic systems and ensure that the estimate is unbias and converges almost surely to the solution. These facts do not give us a measure of the rate of convergence and do not answer the question *how one could control the quality of the Monte Carlo algorithm?* At the same time it is clear that the quality of the algorithm, meaning how fast is the convergence and how small is the probability error, should depend on the properties of the matrix A, and possibly of the right-hand side vector b. Our study shows that the *balancing* of the matrix together with the matrix spectral radius is an important issue for the quality of the Monte Carlo algorithm. And this is important for all three problems under consideration. To analyse this issue let us consider the variance of the random variable $\theta_\alpha(\tau)$ for solving **Problem P1**, i.e., evaluating the inner product

$$J(x) = (v, x) = \sum_{i=1}^{n} v_i x_i$$

of a given vector $v = (v_1, \ldots, v_n)^t \in I\!R^{n\times 1}$ and the solution x of the system (2):

$$x = Ax + b, \quad A \in I\!R^{n\times n}, \quad b = (b_1, \ldots, b_n)^t \in I\!R^{n\times 1}.$$

By chousing v to be an unit vector $e^{(j)} \equiv (0, \ldots, 0, \underbrace{1}_{j}, 0, \ldots, 0)$ all elements, of which are zeros except the j^{th} element $e_j^{(j)}$, which is equal to 1 the functional $J(x)$ becomes equal to x_j. Denote by$\bar{A}$ the matrix containing the absolute values of elements of a given matrix A: $\bar{A} = \{|a_{ij}|\}_{i,j=1}^{n}$ and by $\hat{c}$ the vector of squared entrances of the vector c, i.e., $\hat{c} = \{c_i^2\}_{i=1}^{n+1}$. The pair of MAO density distributions (13) define a finite chain of vector and matrix entrances:

$$c_{\alpha_0} \to a_{\alpha_0\alpha_1} \to \ldots \to a_{\alpha_{k-1}\alpha_k}. \tag{18}$$

The latter chain induces the following product of matrix/vector entrances and norms:

$$A_c^k = c_{\alpha_0} \prod_{s=1}^{k} a_{\alpha_{s-1}\alpha_s},$$

where the vector $c \in I\!R^{n\times 1}$ is defined as $c(i) = b(i)$ if $1 \le i \le n$ and $c(n+1) = 0$. Obviously, the variance of the random variable $\psi_\alpha^k(\tau)$ defined as

$$\psi_\alpha^k(\tau) = \frac{c_{\alpha_0}}{p_{\alpha_0}} \frac{a_{\alpha_0\alpha_1}}{p_{\alpha_0\alpha_1}} \frac{a_{\alpha_1\alpha_2}}{p_{\alpha_1\alpha_2}} \cdots \frac{a_{\alpha_{k-1}\alpha_k}}{p_{\alpha_{k-1}\alpha_k}} \frac{c_{\alpha_k}}{p_{\alpha_k}} = \frac{A_c^k c_{\alpha_k}}{P^k(\tau)}$$

plays an important role in defining the quality of the proposed WE Monte Carlo algorithm. Smaller is the variance of $D\{\psi_\alpha^k(\tau)\}$, better convergency of the algorithm will be achieved.

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

Theorem 3.3

$$D\{\psi_\alpha^k(\tau)\} = \frac{c_{\alpha_0}}{p_{\alpha_0}p_\alpha}(\bar{A}_c^k\hat{c})_\alpha - (A_c^k c)_\alpha^2.$$

P r o o f.

We deal with the variance

$$D\{\psi_\alpha^k(\tau)\} = E\{(\psi_\alpha^k(\tau))^2\} - (E\{\psi_\alpha^k(\tau))^2. \tag{19}$$

Consider the first term of (19).

$$E\{(\psi_\alpha^k(\tau))^2\} = E\left\{\frac{c_{\alpha_0}^2}{p_{\alpha_0}^2}\frac{a_{\alpha_0\alpha_1}^2\dots a_{\alpha_{k-1}\alpha_k}^2}{p_{\alpha_0\alpha_1}^2\dots p_{\alpha_{k-1}\alpha_k}^2}\frac{c_{\alpha_k}^2}{p_{\alpha_k}}\right\}$$

$$= \sum_{\alpha_0,\dots\alpha_k=1}^{n}\frac{c_{\alpha_0}^2}{p_{\alpha_0}^2}\frac{a_{\alpha_0\alpha_1}^2\dots a_{\alpha_{k-1}\alpha_k}^2}{p_{\alpha_0\alpha_1}^2\dots p_{\alpha_{k-1}\alpha_k}^2}\frac{c_{\alpha_k}^2}{p_{\alpha_k}^2}p_{\alpha_0}p_{\alpha_0\alpha_1}\dots p_{\alpha_{k-1}\alpha_k}p_{\alpha_k}$$

$$= \sum_{\alpha_0,\dots\alpha_k=1}^{n}\frac{c_{\alpha_0}^2}{p_{\alpha_0}}|a_{\alpha_0\alpha_1}|\dots|a_{\alpha_{k-1}\alpha_k}|\frac{c_{\alpha_k}^2}{p_{\alpha_k}}$$

$$= \sum_{\alpha_0,\dots\alpha_k=1}^{n}\frac{|c_{\alpha_0}|}{p_{\alpha_0}p_{\alpha_k}}|c_{\alpha_0}||a_{\alpha_0\alpha_1}|\dots|a_{\alpha_{k-1}\alpha_k}|c_{\alpha_k}^2$$

$$= \frac{|c_{\alpha_0}|}{p_{\alpha_0}p_\alpha}\times(\bar{A}_c^k\hat{c})_\alpha.$$

In the same way one can show that the second term of (19) is equal to $(A_c^k c)_\alpha^2$.

Thus, $D\{\psi_\alpha^k(\tau)\} = \frac{c_{\alpha_0}}{p_{\alpha_0}p_\alpha}(\bar{A}_c^k\hat{c})_\alpha - (A_c^k c)_\alpha^2\diamondsuit$

The problem of robustness of Monte Carlo algorithms is considered in [9] and [7]. We will not go into details here, but we should mention that according to the definition given in [7] the proposed WE algorithm is *robust*. In [7] the problem of existing of *interpolation* MC algorithms is also considered. Simply specking, MC algorithm for which the probability error is zero is called interpolation MC algorithm. Now we can formulate an important corollary that gives a sufficient condition for constructing an interpolation MC algorithm.

Corollary 3.1 *Consider a perfectly balanced stochastic matrix* $A = \{a_{ij}\}_{i,j=1}^n$, *where* $a_{ij} = 1/n$, *for* $i, j = 1,\dots,n$ *and the vector* $c = (1,\dots,1)^t$. *Then MC algorithm defined by density distribution* $P^k(\tau)$ *is an interpolation MC algorithm.*

P r o o f. To prove the corollary it is sufficient to show that the variance $D\{\psi_\alpha^k(\tau)\}$ is zero. Obviously,

$$Ac = \begin{pmatrix} \frac{1}{n} & \dots & \frac{1}{n} \\ \vdots & & \\ \frac{1}{n} & \dots & \frac{1}{n} \end{pmatrix} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix},$$

and also,

$$A^k c = \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix}.$$

Since

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

and $\hat{c} = \{c_i^2\}_{i=1}^n = c$, we have: $\frac{|c_{\alpha_0}|}{p_{\alpha_0} p_\alpha} \times (\bar{A}_c^k \hat{c})_\alpha = 1$. Thus, we proved that $D\{\psi_\alpha^k(\tau)\} = 0$ $\diamondsuit$

First of all, let us mention that solving systems with perfectly balanced stochastic matrices does not make sense. Nevertheless, it is possible to consider matrix-vector iterations $A^k c$ since they are the basics for the von Neumann series that approximate the solution of systems of linear algebraic equations under some conditions given in Section 2. The idea of this consideration was to demonstrate that, as closer the matrix A is to the perfectly balanced matrix, as smaller is the probability error of the algorithm.

3.4 Probabilistic representation for complex-valued matrices

Assume that one wants to solve a linear system of the form $u = Au + b$, where the square matrix $A \in \mathbf{C}^{n \times n}$ with complex-valued entrances is of size n. Assume also that $\varrho(A) < 1$, its coefficients $a_{i,j}$ verify $\sum_{j=1}^n |a_{i,j}| \leq 1$, $\forall 1 \leq i \leq n$. We also define a Markov T_k with $n+1$ states $\alpha_1, \dots, \alpha_n, n+1$, such that

$$P(\alpha_{k+1} = j / \alpha_k = i) = |a_{i,j}|$$

if $i \neq n+1$ and

$$P(\alpha_{k+1} = n+1/\alpha_k = n+1) = 1.$$

Like in the real case, we also define a vector c such that $c(i) = b(i)$ if $1 \leq i \leq n$ and $c(n+1) = 0$. Denote by $\tau = (\alpha_0, \alpha_1, \ldots, \alpha_k, n+1)$ a random trajectory that starts at the initial state $\alpha_0 < n+1$ and passes through $(\alpha_1, \ldots, \alpha_k)$ until the absorbing state $\alpha_{k+1} = n+1$. The probability to follow the trajectory τ is $P(\tau) = p_{\alpha_0} p_{\alpha_0 \alpha_1}, \ldots p_{\alpha_{k-1,k} \alpha_k} p_{\alpha_k}$.

Additionally to the Markov chain T_k and to the vector c we define a random variable S_{α_k} such that

$$S_{\alpha_0} = 1,\ S_{\alpha_k} = S_{\alpha_{k-1}} \exp\{i \arg(a_{\alpha_{k-1}\alpha_k})\}$$

which reduces to

$$S_{\alpha_0} = 1,\ S_{\alpha_k} = S_{\alpha_{k-1}} sgn(a_{\alpha_{k-1},\alpha_k})$$

in the real case. Then, we have the following probabilistic representation:

$$x_\alpha = E\left\{c_\alpha + S_{\alpha_k} \frac{a_{\alpha_k \alpha}}{p_{\alpha_k}}\right\},\ \alpha = 1, \ldots, n.$$

The proof of this probabilistic representation is similar to the proof in the real case given in Theorem 3.1.

4 Description of the walk on equations (WE) algorithms

We start this section with a remark of how the matrix inversion problem (see **Problem P3** presented in Section 2) can be treated by the similar algorithm for linear systems of equations. Consider the following system of linear equations:

$$Bu = f, \quad B \in I\!R^{n \times n};\ \ f, u \in I\!R^{n \times 1}. \tag{20}$$

The inverse matrix problem is equivalent to solving n-times the problem (20), i.e.

$$Bg_j = f_j, \quad j = 1, \ldots, n \tag{21}$$

where $f_j \equiv e_j \equiv (0, \ldots, 0, \underbrace{1}_{j}, 0, \ldots, 0)$ and $g_j \equiv (g_{j1}, g_{j2}, \ldots, g_{jn})^t$ is the j-th column of the inverse matrix $G = B^{-1}$.

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

$$A = I - DB, \tag{22}$$

where D is a diagonal matrix $D = diag(d_1, \ldots, d_n)$ and $d_i = \frac{\gamma}{b_{ii}},\ i = 1, \ldots, n$, and $\gamma \in (0, 1]$ is a parameter that can be used to accelerate the convergence. The system (20) can be presented in the following form:

$$u = Au + b, \tag{23}$$

where $B = Df$.

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

$$\sum_{j=1}^{n} |a_{ij}| \leq 1 \qquad i = 1, \ldots, n. \tag{24}$$

4.1 A new WE Monte Carlo algorithm for computing one component of the solution

Here we describe a new WE Monte Carlo algorithm for computing the component u_{i_0} of the solution.

Algorithm 4.1 *:*

Computing one component x_{i_0} of the solution $x_i,\ i = 1, \ldots n$
1. Initialization **Input** *initial data: the matrix B, the vector* $\mathbf{f}$*, the constant γ and the number of random trajectories N.*
2. Preliminary calculations (preprocessing):
2.1. **Compute** *the matrix A using the parameter $\gamma \in (0, 1]$:*

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

2.2. **Compute** *the vector asum:*

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

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

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

3. **Set** $S := 0$.
4. **for** $k = 1$ **to** N **do** *(MC loop)*
4.1 **set** $m := i_0$.
4.2 **set** $S := S + b(m)$.
4.3. $test = 0;\ sign = 1$
4.4 **while** $(test \neq 0$ **do***:*
4.4.1. generate an uniformly distributed random variable $r \in (0, 1)$
4.4.2. **if** $r \geq asum(m)$ **then** $test = 1$*;*
4.4.3. **else** *chose the index* j *according to the probability density matrix* p_{ij}*;*
4.4.4. **set** $sign = sign * sign\{a_{mj}\};\ m = j$*;*
4.4.5 **update** $S := S + sign * b_m$*;*
4.4.6. **endif** .
4.5 **endwhile**
5. **enddo**

4.2 A new WE Monte Carlo algorithm for computing all components the solution

Here we describe the WE algorithm for computing all the components of the solution. For simplicity we will consider the case when all the matrix entrances are nonnegative. So, we assume that $a_{ij} \geq 0$, for $i, j = 1, \ldots n$. We will skip the part of initialization and preprocessing, which is the same as in the previous subsection.

Algorithm 4.2 *:*

Computing all components $x_i,\ i = 1, \ldots n$ **of the solution**
1. Initialization.
2. Preprocessing.
3. **for** $i = 1$ **to** n **do**
3.1. $S(i) := 0$*;* $V(i) := 0$
3.2. **for** $k = 1$ **to** N **do** *(MC loop)*
3.1 **set** $m := rand(1 : n)$.

3.2 **set** $test := 0$; $m_1 := 0$;
3.3 **while** ($test \neq 1$ **do**:
3.3.1. $V(m) := V(m) + 1$; $m = m_1 + 1$; $l(m_1) = m$;
3.3.2. **for** $q = 1, m_1$ **do**:
$S(l(q)) := S(l(q)) + b(l(q))$;
3.3.3. **if** $r > asum(m)$, **then** $test = 1$;
else *chose* j *according to the probability density matrix* $p_{i,j}$;
3.3.4. **set** $m = j$.
3.3.5. **endif**
3.4. **endwhile**
3.5. **enddo**
3.6. **for** $j = 1, n$ **do**:
$V(j) := \max\{1, V(j)\}$;
$S(j) = \frac{S(j)}{V(j)}$.

4.3 Sequential Monte Carlo

The sequential Monte Carlo (SMC) method for linear systems has been introduced by John Halton in the sixties [19], and further developed in more recent works [20–22]. The main idea of the SMC is very simple. Assume that one needs to solve the problem:

$$x = Ax + b, \quad A \in I\!R^{n \times n}, \quad b = (b_1, \ldots, b_n)^t \in I\!R^{n \times 1}.$$

We put: $x = y + z$, where y is a current estimate of the solution x using a given Monte Carlo algorithm, say WE algorithm and z is the correction. Then the correction, z satisfies $z = Az + d$, where $d = Ay - y + b$. At each stage, one uses a given Monte Carlo to estimate z, and so, the new estimate y. If the sequential computation of d is itself approximated, numerically or stochastically, then the expected time for this process to reach a given accuracy is linear per number of steps, but the number of steps is dramatically reduced.

Normally, the SMC leads to dramatic reduction of the computational time and to high accuracy. Some numerical results demonstrating this fact will be shown in the next section. The algorithms of this kind can successfully compete with optimal deterministic algorithms based on Krilov subspace methods and the conjugate gradient methods (CGM) (many modifications of these methods are available). But one should be careful, if the matrix-vector multiplication Ay is performed in a deterministic way, it needs $O(n^2)$ operations. It means that for a high quality WE algorithm combined with SMC one needs to have very few sequential steps. In the next section some numerical results with high accuracy obtained in just several sequential steps will be presented.

5 Numerical tests and discussion

In this section we show numerical tests divided into several groups. In the first group we compare the results obtained by the proposed WE algorithm combined with SMC to the results obtained by standard deterministic Jacobi algorithm for some test matrices. In the second group of numerical experiments we study how the quality of the WE algorithm depends on the balancing of the matrices. In the first two groups of experiments we deal with matrices of different size n ranging from $n = 10$ to $n = 25000$. The matrices are either randomly generated, or created with some prescribed properties we want to have, in order to study the dependance of the convergency of the algorithm to different properties. In the third group of experiments we run our algorithm to some large arbitrary matrices describing real-life problems taken from some available collections of matrices. In this particular study we use matrices from the Harwell-Boeing Collection. It is a well-know fact that Monte Carlo methods are very efficient to deal with large and very large matrices. This is why we pay special attention to these matrices.

5.1 Comparison of WE with deterministic algorithms

In this subsection we compare the numerical results for solving systems of linear equations by WE Monte Carlo combined with SMC with the well known Jacobi method. By both methods we compute all the component of the solution. Here we deal with dense matrices of both types: randomly generated, as well as deterministic matrices. The randomly generated matrices are such that $\max_i \sum_{j=1}^{n} |a_{ij}| = \max \parallel a_i \parallel = a \leq 1$. In most of the graphs presented here the parameter a was chosen to be equal to 0.9. On the graphs we present the relative error of the solution of the first component x_1, since the results for all other components are very similar. So, on the graphs we present the value of

$$\varepsilon(x_i) = \left| \frac{x_{i,comp} - x_{i,exact}}{x_{i,exact}} \right|, \quad i = 1, \ldots, n,$$

where $x_{i,comp}$ is the computed value of the i-th component of the solution vector, and $x_{i,exact}$ is the i-th component of the exact solution-vector. On Figure 1 results for the relative error of the solution for the proposed algorithm and for the Jacobi method are presented.

One can observe that for the proposed WE Monte Carlo method the convergence is much faster and just after 30 iterations the solution is very accurate: the relative error is about 10^{-12}. At the same time, the convergence of Jacobi method is slower, and after 100 iterations the relative error is between 10^{-2} and 10^{-3}. Similar results are obtained for much large dense matrix of size

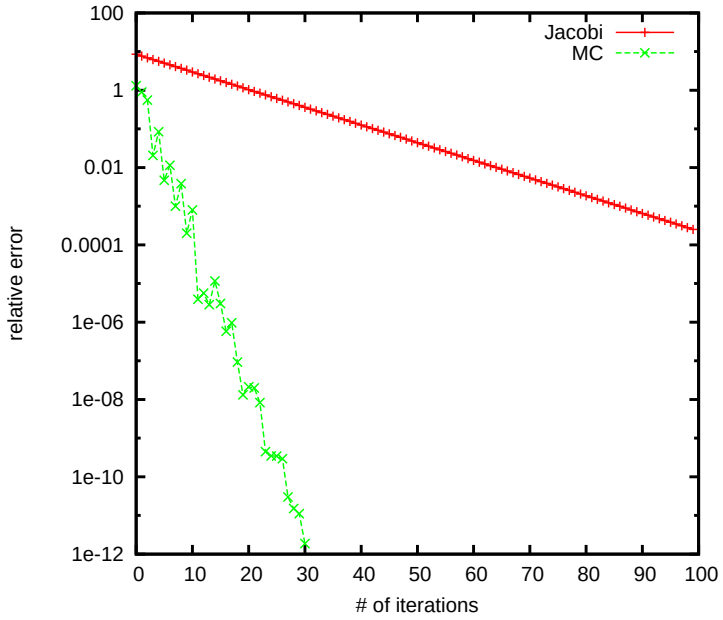

Fig. 1. Relative error for the WE Monte Carlo for all components of the solution and Jacobi method for a matrix of size 5000×5000. The number of random trajectories for MC is $5n$.

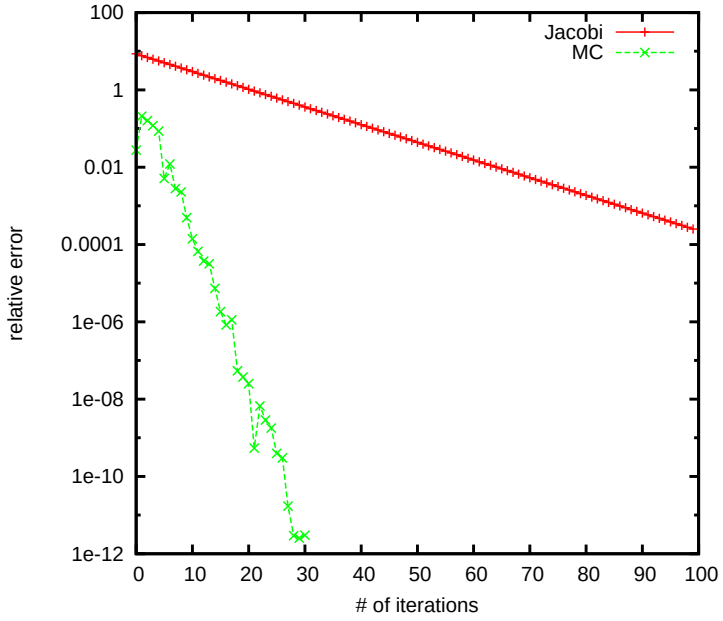

Fig. 2. Relative error for the WE Monte Carlo for all components of the solution and Jacobi method for a matrix of size 25000×25000. The number of random trajectories for MC is $5n$.

$n = 25000$. These results are shown on Figure 2. The results are similar.

In order to see if the residual can be considered as a good measure for the error we compare the relative error with the residual for a matrix of size $n = 5000$

(see. Figure 3). One can observe that the curves presenting the relative error and the residual are very close to each other.

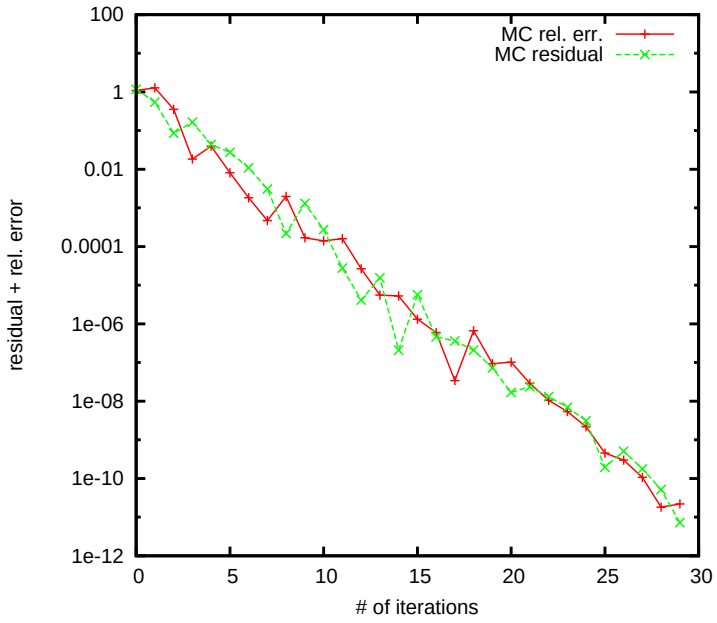

Fig. 3. Comparison the relative error with the residual for the WE Monte Carlo 100×100. The number of random trajectories for MC is $5n$.

5.2 Role of balancing for the quality of the algorithms

Let us mention that the balancing of matrices is a vary important issue for the deterministic algorithms. That is why for any arbitrary matrix a balancing procedure is needed as a preprocessing. Monte Carlo algorithms are not that sensitive to the balancing, but nevertheless, as it was shown in Subsection 3.3 the balancing affects the convergence of the WE Monte Carlo algorithm. The first example is shown on Figure 4 where we consider two matrices of small size $n = 10$. The balanced matrix is such that $a_{ij} = 0.1$, such that the parameter a defined above is equal to 0.9. For the unbalanced matrix the parameter a is equal to 0.88, but the unbalanced matrix is extremity unbalanced (the ration between the largest and the smallest by modulo elements is equal to 40). Let us stress on the fact that this ratio is considered large when it is equal to 5.

The unbalanced matrix we deal with is the following: $A = \begin{pmatrix} 0.4 & 0.4 & 0.01 & \dots & 0.01 \\ 0.01 & 0.4 & 0.4 & \dots & 0.01 \\ \vdots & & & & \\ 0.4 & 0.01 & 0.01 & \dots & 0.4 \end{pmatrix}$.

Obviously, for the unbalanced matrix there is no convergence at all. For the balanced matrix the convergence is fast, but the *red* curve is rough. The reason

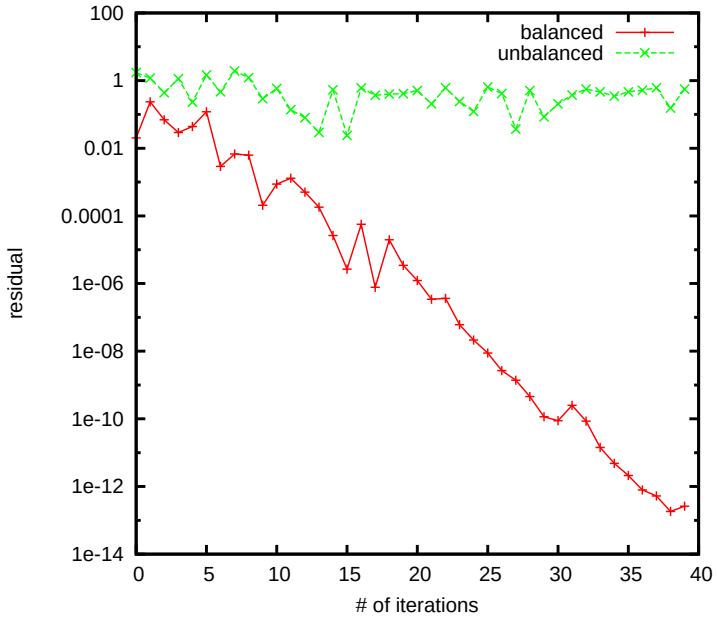

Fig. 4. Residual for the WE Monte Carlo for balanced and extremity unbalanced matrix of size $n = 10$. The number of random trajectories is $N = 5n$.

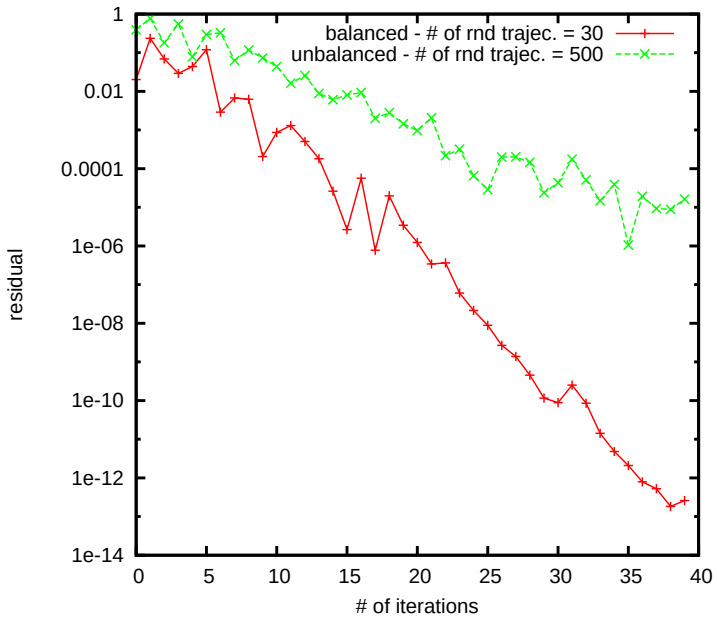

Fig. 5. Residual for the WE Monte Carlo for balanced and extremity unbalanced matrix of size $n = 10$. The number of random trajectories for the balanced matrix is $N = 5n$; for the unbalanced matrix there two cases: (i) $N = 5n$ – green curve; (ii) $N = 50n$ – blue curve.

for that is that we use a very small number of random trajectories ($N = 5n$). Now we can consider the same matrices, but let us increase the number of the random trajectories to $N = 500$. At the same time we may decrease the number of the random trajectories (and also the computational time) from

$N = 50$ to $N = 30$ for the balanced matrix. The results of this numerical experiments are presented on Figure 5.

One can clearly see, that the WE algorithm starts to converge and after 35 to 40 iterations we have 3 correct digits. For the balanced matrix the convergence is still very fast, but again the *red* curve is rough, because of the small number of random trajectories. For a randomly generated matrices of size $n = 5000$ similar results are presented on Figure 6. The unbalanced matrix is generated by perturbing randomly the elements of the balanced matrix. The *green* and the *blue* curves correspond to the same unbalanced matrix, but the numerical results presented by the *blue* curve are obtained by using 10 times more random trajectories. In such a way, one can see that increasing the number N of random trajectories one can improve the convergence, as it is clear from the theory. In our particular numerical experiments shown on Figure 6 the green

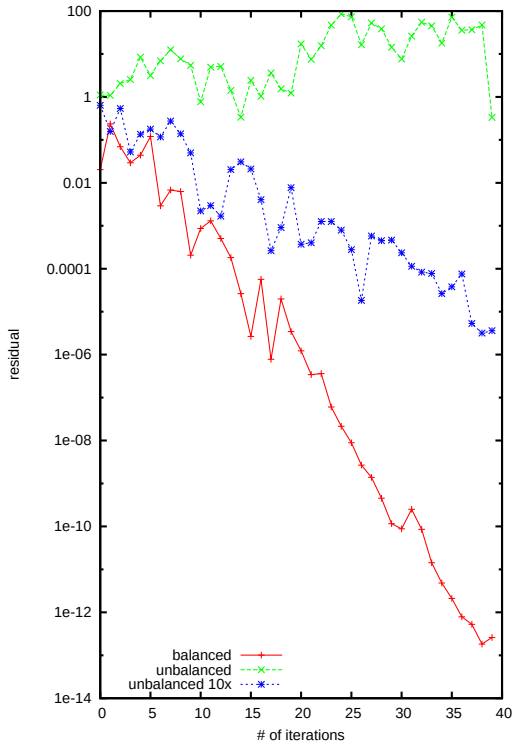

Fig. 6. Residual for the WE Monte Carlo; the matrix of size is 5000×5000. The number of random trajectories for MC is $5n$.

curve is obtained after a number of trajectories $N = 5n$, while the blue curve is based on $N = 50n$ number of trajectories. One can see that in the first case the algorithm does not converge, while in the second case it converges. These experiments show that the WE Monte Carlo has another degree of freedom (which does not exist in deterministic algorithms) to improve the convergence even for very bad balanced matrices. The price we pay to improve the convergence is the increased complexity (or, the computational time) since we increase the number of simulation trajectories.

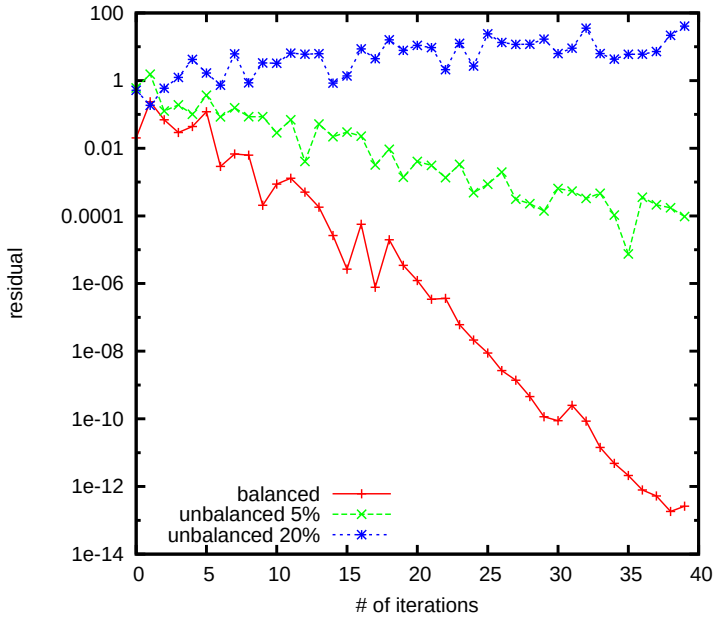

Fig. 7. Residual for the WE Monte Carlo; the matrix of size is 100×100. Matrix elements are additionally perturbed by 5% and by 20%.

It is interesting to see how the disbanding affects the convergence of the WE Monte Carlo. We have performed the following numerical experiment: taking randomly generated matrices of differen size we were perturbing the matrix elements randomly by some percentage making the matrix more and more imbalanced. On Figure 7 we present the results for the matrix of size $n = 100$. The randomly generated entrances of the matrix are additionally perturbed by 5% and by 20%.

5.3 Comparison of WE with the preconditioned conjugant gradient (PCG) method

We compare our results with the optimal deterministic preconditioned conjugant gradient (PCG) method [18,25,28]. Here is more precipe information about our implementation of the PCG method. We want to solve the linear system of equations $Bx = b$ by means of the PCG iterative method. The input arguments are the matrix B, which in our case is the square large matrix $NOS4$ taken from the well-known Harwell-Boeing Collection. B should be symmetric and positive definite. In our implementation we use the following parameters:

- b is the right-hand side vector.
- tol is the required relative tolerance for the residual error, namely $b - Bx$. The iteration stops if $\| b - Bx \| \leq tol * \| b \|$.
- $maxit$ is the maximum allowable number of iterations.

- $m = m1 * m2$ is the (left) preconditioning matrix, so that the iteration is (theoretically) equivalent to solving by PCG $Px = m\ b$, with $P = m\ B$.
- x_0 is the initial guess.

A comparison for convergency of the WE Monte Carlo and the PCG for the matrix $NOS4$ is presented in the Figure 8. This particular matrix is taken from an application connected to finite element approximation of a problem describing a beam structure in constructive mechanics [32].

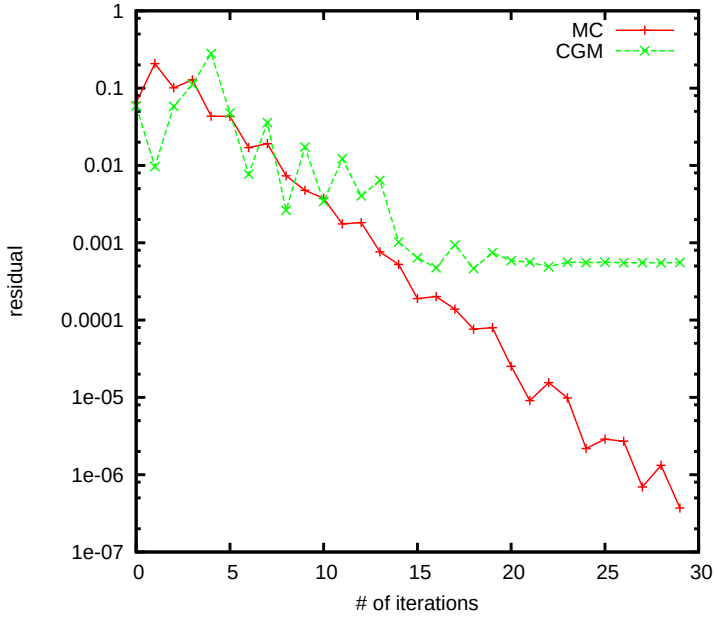

Fig. 8. Comparison of the WE Monte Carlo and the PCG method for a matrix $NOS4$ from the Harwell-Boeing Collection.

The results presented on Figure 8 show that the convergence for the WE Monte Carlo is better: the curve presenting the residual with the number of iterations is smoother and goes down to $10^{-6}-10^{-7}$, while the curve presenting the PCG achieves an accuracy of 10^{-3}. Let us stress on the fact that for the experiments presented the needed accuracy is set to $tol = 10^{-8}$. One can not guarantee that such an effect happens for every matrices, but there are cases in which the WE Monte Carlo performs better that the PCG.

6 Concluding remarks

A new Walk on Equations Monte Carlo algorithm for solving linear algebra problems is presented and studied. The algorithm can be used for evaluating all the components of the solution of systems. The algorithm can also be used to approximate the matrix inversion of square matrices. The WE Monte Carlo

is applied to real-valued, as well as to complex-valued matrices. It is proved that:

- the proposed random variable based on random walk on equations is an unbiased estimate of the solution vector;
- for the Monte Carlo estimator, the mean value of N realizations of the random variable, it is proven that the variance vanishes when N goes to infinity and converges almost surely to the solution;
- the structure of the relative stochastic error is analysed; it is shown that an interpolation Monte Carlo algorithm is possible for perfectly balanced matrices.

The analysis of the structure of the stochastic error shows that the rate of convergence of the proposed WE Monte Carlo algorithm depend on the balancing of the matrix. This dependance is not that strong as for deterministic algorithms, but still exists.

A number of numerical experiments are performed. The analysis of the results show that:

- the proposed WE Monte Carlo algorithm combined with the sequential Monte Carlo for computing all the components of the solution converges much faster than some well-known deterministic iterative methods, like Jacobi method; this is true for matrices of different size and the effect is much bigger for larger matrices of size up to $n = 25000$;
- the balancing of matrices is still important for the convergency; many numerical results demonstrate that one should pay special attention to the balancing;
- the comparison of numerical results obtained by the WE Monte Carlo and the best optimal deterministic method, the preconditioned conjugant gradient, show that for some matrices WE Monte Carlo gives better results.

Acknowledgment

The research reported in this paper is partly supported by the European Commission under FP7 project AComIn (FP7 REGPOT-2012-2013-1, by the Bulgarian NSF Grant DCVP 02/1, as well as by the Université du Sud Toulon-Var.

References

[1] V. Alexandrov, E. Atanassov, I. Dimov, *Parallel Quasi-Monte Carlo Methods for Linear Algebra Problems*, **Monte Carlo Methods and Applications, Vol. 10, No. 3-4 (2004)**, pp. 213–219.

[2] Curtiss J.H., *Monte Carlo methods for the iteration of linear operators*, **J. Math Phys., vol 32**, No 4 (1954), pp.209–232.

[3] Curtiss J.H. *A Theoretical Comparison of the Efficiencies of two classical methods and a Monte Carlo method for Computing one component of the solution of a set of Linear Algebraic Equations.* ,Proc. Symposium on Monte Carlo Methods , John Wiley and Sons, 1956, pp.191–233.

[4] J. D. Densmore and E. W. Larsen, *Variational variance reduction for particle transport eigenvalue calculations using Monte Carlo adjoint simulation*, **Journal of Computational Physics, Vol. 192**, No 2 (2003), pp. 387–405.

[5] Dimov I. *Minimization of the Probable Error for Some Monte Carlo methods*. Proc. Int. Conf. on Mathematical Modeling and Scientific Computation, Albena, Bulgaria, Sofia, **Publ. House of the Bulgarian Academy of Sciences**, 1991, pp. 159–170.

[6] I. Dimov, *Monte Carlo Algorithms for Linear Problems*, **Pliska (Studia Mathematica Bulgarica), Vol. 13 (2000)**, pp. 57–77.

[7] I.T. Dimov, *Monte Carlo Methods for Applied Scientists*, New Jersey, London, Singapore, **World Scientific** (2008), 291 p., ISBN-10 981-02-2329-3.

[8] I. Dimov, V. Alexandrov, A. Karaivanova, *Parallel resolvent Monte Carlo algorithms for linear algebra problems*, **J. Mathematics and Computers in Simulation, Vol. 55 (2001)**, pp. 25–35.

[9] I.T. Dimov, B. Philippe, A. Karaivanova, and C. Weihrauch, *Robustness and applicability of Markov chain Monte Carlo algorithms for eigenvalue problems*, **Applied Mathematical Modelling**, **Vol. 32** (2008) pp. 1511-1529.

[10] I. T. Dimov and A. N. Karaivanova, *Iterative Monte Carlo algorithms for linear algebra problems*, First Workshop on Numerical Analysis and Applications, Rousse, Bulgaria, June 24-27, 1996, in Numerical Analysis and Its Applications, **Springer Lecture Notes in Computer Science, ser. 1196**, pp. 150–160.

[11] I.T. Dimov, V. Alexandrov, *A New Highly Convergent Monte Carlo Method for Matrix Computations*, **Mathematics and Computers in Simulation, Vol. 47 (1998)**, pp. 165–181.

[12] I. Dimov, A. Karaivanova, *Parallel computations of eigenvalues based on a Monte Carlo approach*, **Journal of Monte Carlo Method and Applications, Vol. 4, Nu. 1, (1998)**, pp. 33–52.

[13] I. Dimov, A. Karaivanova, *A Power Method with Monte Carlo Iterations, Recent Advances in Numerical Methods and Applications*, (O. Iliev, M. Kaschiev, Bl. Sendov, P. Vassilevski, Eds., **World Scientific**, 1999, Singapore), pp. 239–247.

[14] I. Dimov, O. Tonev, *Performance Analysis of Monte Carlo Algorithms for Some Models of Computer Architectures*, International Youth Workshop on Monte Carlo Methods and Parallel Algorithms - Primorsko (Eds. Bl. Sendov, I. Dimov), **World Scientific**, Singapore, 1990, 91–95.

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

[16] Forsythe G.E. and Leibler R.A. *Matrix Inversion by a Monte Carlo Method*, **MTAC Vol.4** (1950), pp. 127–129.

[17] G. V. Golub, C. F. Van Loon, *Matrix computations (3rd ed.)*, **Johns Hopkins Univ. Press**, Baltimore, 1996.

[18] Gene H. Golub and Q. Ye, *Inexact Preconditioned Conjugate Gradient Method with Inner-Outer Iteration*, **SIAM Journal on Scientific Computing 21** (4): 1305. (1999). doi:10.1137/S1064827597323415.

[19] J. Halton, *Sequential Monte Carlo, Proceedings of the Cambridge Philosophical Society*, **Vol. 58** (1962) pp. 57-78.

[20] J. Halton, *Sequential Monte Carlo. University of Wisconsin*, Madison, Mathematics Research Center Technical Summary Report No. 816 (1967) 38 pp.

[21] J. Halton and E. A. Zeidman, *Monte Carlo integration with sequential stratification.* University of Wisconsin, Madison, Computer Sciences Department Technical Report No. 61 (1969) 31 pp.

[22] J. Halton, *Sequential Monte Carlo for linear systems - a practical summary*, **Monte Carlo Methods & Applications, 14** (2008) pp. 1–27.

[23] J.M. Hammersley, D.C. Handscomb, *Monte Carlo methods*, **John Wiley & Sons, inc.**, New York, London, Sydney,Methuen, 1964.

[24] L. W. Kantorovich and V. I. Krylov, *Approximate Methods of Higher Analysis*, **Interscience**, New York, 1964.

[25] A.V. Knyazev and I. Lashuk, *Steepest Descent and Conjugate Gradient Methods with Variable Preconditioning* **SIAM Journal on Matrix Analysis and Applications 29** (4): 1267, (2008). doi:10.1137/060675290

[26] S. Maire, *Reducing variance using iterated control variates*, **Journal of Statistical Computation and Simulation, Vol. 73(1)**, pp. 1-29, 2003.

[27] N. Rosca, *A new Monte Carlo estimator for systems of linear equations*, Studia Univ. Babes–Bolyai, Mathematica, Vol. LI, Number 2, June 2006, pp. 97–107.

[28] Y. Saad, *Iterative methods for sparse linear systems (2nd ed. – ed.)*. Philadelphia, Pa.: **Society for Industrial and Applied Mathematics** (2003) p. 195. ISBN 978-0-89871-534-7.

[29] I.M. Sobol *Monte Carlo numerical methods*, **Nauka**, Moscow, 1973.

[30] J. Spanier and E. Gelbard, *Monte Carlo Principles and Neutron Transport Problem*, **Addison–Wesley**, 1969.

[31] J.R. Westlake, *A Handbook of Numerical matrix Inversion and Solution of Linear Equations*, **John Wiley & Sons, inc.**, New York, London, Sydney, 1968.

[32] Website: Matrix market, *NOS4: Lanczos with partial reorthogonalization. Finite element approximation to a beam structure*, http://math.nist.gov/MatrixMarket/data/Harwell-Boeing/lanpro/nos4.html