Error Analysis of a Monte Carlo Algorithm for Computing Bilinear Forms of Matrix Powers*

Ivan Dimov, Vassil Alexandrov, Simon Branford, and Christian Weihrauch

Centre for Advanced Computing and Emerging Technologies
School of Systems Engineering, The University of Reading
Whiteknights, PO Box 225, Reading, RG6 6AY, UK
{v.n.alexandrov, s.j.branford, i.t.dimov, c.weihrauch}@reading.ac.uk
Institute for Parallel Processing, Bulgarian Academy of Sciences
Acad. G. Bonchev 25 A, 1113 Sofia, Bulgaria
ivdimov@bas.bg

Abstract. In this paper we present error analysis for a Monte Carlo algorithm for evaluating bilinear forms of matrix powers. An almost Optimal Monte Carlo (MAO) algorithm for solving this problem is formulated. Results for the structure of the probability error are presented and the construction of robust and interpolation Monte Carlo algorithms are discussed.

Results are presented comparing the performance of the Monte Carlo algorithm with that of a corresponding deterministic algorithm. The two algorithms are tested on a well balanced matrix and then the effects of perturbing this matrix, by small and large amounts, is studied.

Keywords: Monte Carlo algorithms, matrix computations, performance analysis, iterative process.

1 Introduction

The problems of inverting a real $n \times n$ matrix (MI), solving a system of linear algebraic equations (SLAE) or finding extreme eigenvalues are of unquestionable importance in many scientific and engineering applications: e.g real-time speech coding, digital signal processing, communications, stochastic modelling, and many physical problems involving partial differential equations. The direct methods of solution for SLAE require $O(n^3)$ sequential steps when using the usual elimination or annihilation schemes (e.g. Gaussian elimination, Gauss-Jordan methods) [7] and, similarly, the direct methods for MI or calculating extreme eigenvalues can be computationally expensive. Consequently the computation time for very large problems, or for real-time solution problems, can be prohibitive and this prevents the use of many established algorithms.

It is known that Monte Carlo methods give statistical estimates for elements of the inverse matrix, or for components of the solution vector of SLAE, by

* Partially supported by the Bulgarian Ministry of Education and Science, under grant I-1405/2004.

V.N. Alexandrov et al. (Eds.): ICCS 2006, Part III, LNCS 3993, pp. 632–639, 2006.

performing random sampling of a certain random variable, whose mathematical expectation is the desired solution [8, 9]. The problem of variance estimation, in the optimal case, has been considered for extremal eigenvalues [10]. In this paper we extend this by considering bilinear forms of matrix powers, which can be used to formulate solutions for all three problems, and study the effects, on the Monte Carlo algorithm, of perturbing a well balanced matrix.

The idea of Monte Carlo and an algorithm for the problem of bilinear forms of matrix powers is presented in Section 2; in Section 3 we detail the implementation we used for the algorithm; the results of the experimental runs of this implementation are also presented in this section; and we conclude the work in Section 4.

2 Formulation of the Monte Carlo Algorithm

2.1 Bilinear Forms of Matrix Powers

We are interested in the bilinear form of matrix powers:

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

For x, the solution of a SLAE $Bx = b$ then

$$(v, x) = \left(v, \sum_{i=0}^{k} A^i h \right), \tag{2}$$

where the Jacobi Over-relaxation Iterative Method has been used to transform the SLAE into the problem $x = Ax + h$. In cases where the Neumann series does not converge a resolvent method can be used [5, 6].

Matrix inversion is equivalent to solving (2) n times $Bc_j = e^{(j)}$, $j = 1, \ldots, n$ for the special case where $c_j \equiv (c_{1j}, \ldots, c_{nj})^T$ and $e^{(j)} \equiv (0, \ldots, 0, \underbrace{1}_{j}, 0, \ldots, 0)^T$.

For an arbitrary large natural number k the Rayleigh quotient can be used to obtain an approximation for λ_1, the dominant eigenvalue, of a matrix A:

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

Thus it is clear that having an efficient way of calculating (1) is important. This is especially important in cases where we are dealing with large and/or sparse matrices.

2.2 Almost Optimal Markov Chain Monte Carlo

We shall use the so-called Almost Optimal Monte Carlo (MAO) algorithm studied in [2, 3, 1, 5, 4]. Here we give a brief presentation of MAO.

Suppose we have a Markov chain

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

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

$$T_k = \alpha_0 \to \alpha_1 \to \ldots \to \alpha_j \to \ldots \to \alpha_k, \tag{3}$$

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

Assume that

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

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

$$\sum_{\alpha=1}^{n} p_\alpha = 1 \quad \text{and} \quad \sum_{\beta=1}^{n} p_{\alpha\beta} = 1, \quad \text{for any} \quad \alpha = 1, 2, ..., n. \tag{4}$$

We will consider a special choice of density distributions p_i and p_{ij} defined as follows:

$$p_i = \frac{|v_i|}{\| v \|}, \quad \| v \| = \sum_{i=1}^{n} |v_i| \quad \text{and} \quad p_{ij} = \frac{|a_{ij}|}{\| a_i \|}, \quad \| a_i \| = \sum_{j=1}^{n} |a_{ij}|. \tag{5}$$

2.3 Monte Carlo Algorithm for Computing Bilinear Forms of Matrix Powers $(v, A^k h)$

The pair of density distributions (5) defines a finite chain of vector and matrix entrances:

$$v_{\alpha_0} \to a_{\alpha_0\alpha_1} \to \ldots \to a_{\alpha_{k-1}\alpha_k}. \tag{6}$$

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

$$A_v^k = v_{\alpha_0} \prod_{s=1}^{k} a_{\alpha_{s-1}\alpha_s}; \quad \| A_v^k \| = \| v \| \times \prod_{s=1}^{k} \| a_{\alpha_{s-1}} \| .$$

Note, that the product of norms $\| A_v^k \|$ is not a norm of A_v^k. The rule for creating the value of $\| A_v^k \|$ is the following: the norm of the initial vector v, as well as norms of all row-vectors of matrix A visited by the chain (6), defined by densities (5), are included. For such a choice of densities p_i and p_{ij} we have

$$E\{h_{\alpha_k}\} = \frac{sign\{A_v^k\}}{\| A_v^k \|} \left(v, A^k h\right). \tag{7}$$

The standard deviation $\sigma\{h_{\alpha_k}\}$ is finite. Since random variable

$$\theta^{(k)} = sign\{A_v^k\} \times \| A_v^k \| \, h_{\alpha_k}$$

is an unbiased estimate of the form $(v, A^k h)$, (7) can be used to construct a MC algorithm.

Let us consider N realizations of the Markov chain T_k (3) defined by the pair of density distributions (5). Denote by $\theta_i^{(k)}$ the i^{th} realization of the random variable $\theta^{(k)}$. Then the value

$$\bar{\theta}^{(k)} = \sum_{i=1}^{N} \theta_i^{(k)} = sign\{A_v^k\} \parallel A_v^k \parallel \sum_{i=1}^{N} \{h_{\alpha_k}\}_i \tag{8}$$

can be considered as a MC approximation of the form $(v, A^k h)$. The probability error of this approximation can be presented in the following form:

$$R_N^{(k)} = \left| (v, A^k h) - \bar{\theta}^{(k)} \right| = c_p \sigma\{\theta^{(k)}\} N^{-\frac{1}{2}}, \tag{9}$$

where the constant c_p only depends on the probability $P = Pr\{|\bar{\theta} - J| \leq R_N\}$ (where J is the exact solution of the problem, R_N is the probability error and $0 < P < 1$) and does not depend on N and on $\theta^{(k)}$. Because of the finiteness of the standard deviation the probability error is always finite.

In fact, (8) together with the sampling rules using probabilities (5) defines a MC algorithm. The expression (8) gives a MC approximation of the form $(v, A^k h)$ with a probability error $R_N^{(k)}$. Obviously, the quality of the MC algorithm depends on the behaviour of the standard deviation $\sigma\{\theta^{(k)}\}$. So, there is a reason to consider a special class of *robust MC algorithms*.

2.4 Robust and Interpolation Monte Carlo Algorithms

Definition. A MC algorithm for which the standard deviation does not increase with the increasing of the matrix power k is called a *robust MC algorithm*.

Thus, if the MC algorithm is robust, then there exists a constant M such that

$$\lim_{k \to \infty} \sigma\{\theta^{(k)}\} \leq M.$$

Definition. A MC algorithm for which the probability error is zero is called an *interpolation MC algorithm*.

So, using the the following notations:

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

we can prove that

$$\sigma^2\{\theta^{(k)}\} = \parallel A_v^k \parallel \left(\bar{v}, \bar{A}^k \hat{h}\right) - (v, A^k h)^2,$$

where $\sigma^2\{\theta^{(k)}\}$ is the variance. A finite and small variance means that the associated Monte Carlo method should converge, and the probable error will be small.

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

Let $h = (1, \ldots, 1)$, $v = (\frac{1}{n}, \ldots, \frac{1}{n})$ and

$$A = \begin{pmatrix} \frac{1}{n} & \cdots & \frac{1}{n} \\ \vdots & & \\ \frac{1}{n} & \cdots & \frac{1}{n} \end{pmatrix}.$$

Then MC algorithm defined by density distributions (5) is an interpolation MC algorithm. Matrices A of the above form are *stochastic matrices*.

3 Experimental Results

For this paper two programs were implemented; one for the proposed Monte Carlo algorithm and one for a deterministic matrix-vector and vector-vector multiplication solving the same problem. The algorithms were written in Fortran 90 and compiled with the Intel Fortran 9.0 compiler. For all real numbers double precision was used to lower the influence of rounding errors. Random numbers were generated with the help of the Fortran 90 `RANDOM_NUMBER()` sub-routine which creates random numbers with a uniform distribution.

The results were generated on a SGI Prism equipped with 8 x 1.5 GHz Itanium II processors and 16 GByte of main memory. For the input data the vectors h and v and matrix A were generated for the sizes 100, 1000 and 5000. The vector h was filled with ones and v was filled with $\frac{1}{n}$. The matrix A was filled with elements of size $\frac{1}{n}$ and then perturbed by 2, 5, 10, 50 and 90%. The norm of such matrices is around 1. For comparison random non-balanced matrices were generated too. In Figure 1 we show the results for different perturbations for the Monte Carlo algorithm and for the deterministic code. We see that the results are very close for perturbations of up to 10% whereas the results for 50 and 90% differ up to 2% for matrices of size 1000 and 5000 and differ up to 14% for a matrix of size 100.

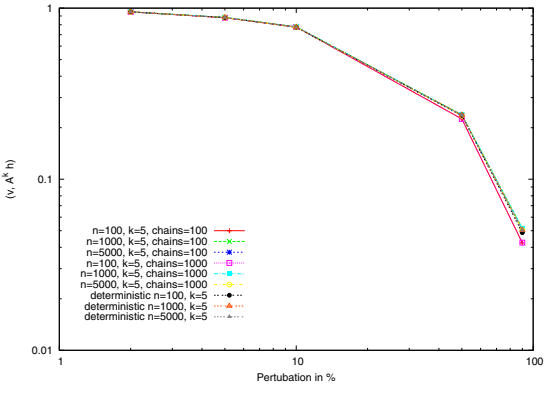

Fig. 1. The dependence of MC results on perturbation of A

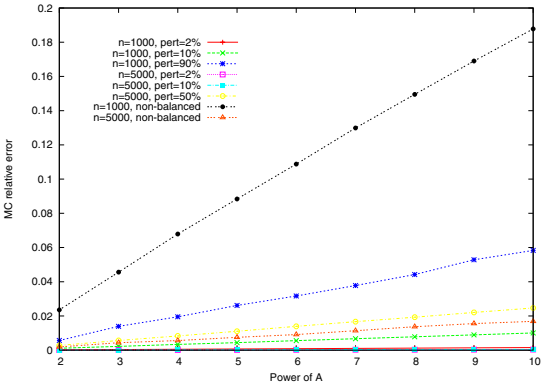

Fig. 2. The dependence of MC relative error on power of A

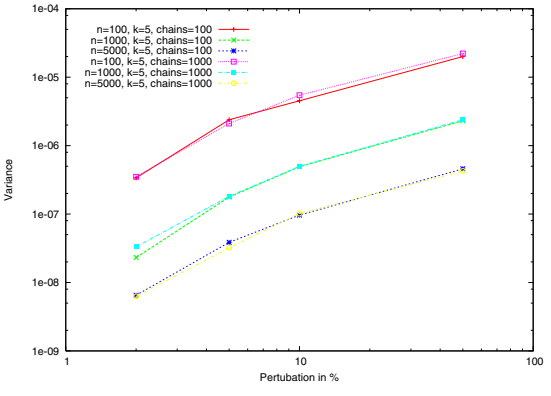

Fig. 3. The dependence of MC variance on perturbation of A

Since the deterministic computations were performed with a double precision we accept the results obtained as *exact results* and use them to analyse the accuracy of the results produced by our Monte Carlo code. In Figure 2 the relative error of the results for Monte Carlo algorithm is shown. The Monte Carlo relative error was computed by using the following formulas:

$$\text{MC error} = |\text{MC result - exact result}|,$$
$$\text{MC relative error} = \frac{\text{MC error}}{\text{exact result}}.$$

From Figure 2 we can see that the error increases linearly if k is increasing. The larger the matrix is, the smaller the influence of the perturbation. For comparison, the results for non-balanced matrices were included.

The variance of the results for the different perturbations are shown in Figure 3. In this figure we compare results for different sizes of the matrix and different chain lengths. Again it is obvious that the influence of the perturbation

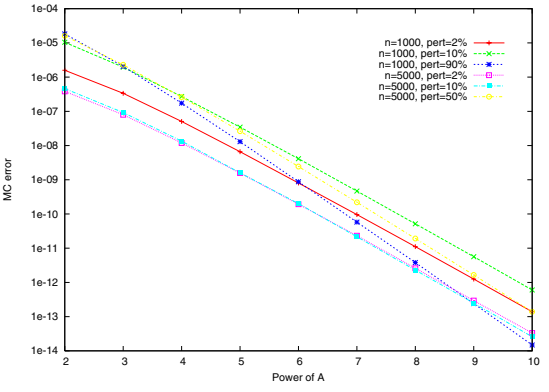

Fig. 4. The dependence of MC error on power of matrices with "small" spectral norms ($\| A \| \approx 0.1$)

is a lot bigger for smaller matrix of size 100. But over all a variance of 10e-06 is a good result and shows that the Monte Carlo algorithm works well with this kind of balanced matrices.

In order to test the robustness of the Monte Carlo algorithm, a re-run of the experiments was done with matrices of norm smaller than 1. Therefore the randomly generated matrices were re-used and their elements were divided by 10. The results for these experiments are shown in Figure 4 and Figure 5.

In Figure 4 the Monte Carlo errors for matrix size of $n = 1000$ and chain length of 1000 are shown. We can see that the Monte Carlo algorithm is very robust because with an increasing k the error is decreasing enormously.

The variance shown in Figure 5 is 10^{10} smaller than the variance shown in Figure 3.

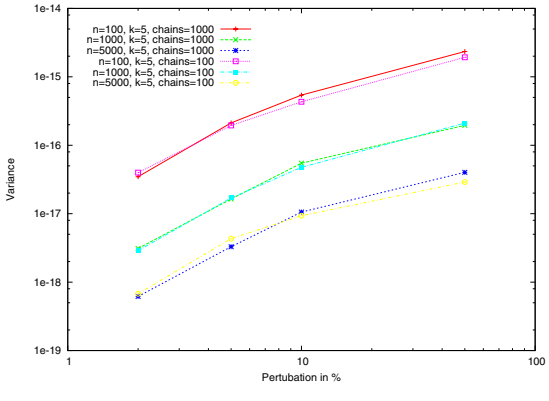

Fig. 5. The dependence of MC variance on perturbation of matrices with "small" spectral norms ($\| A \| \approx 0.1$)

4 Conclusion

In this paper we have analysed the error and robustness of the proposed MC algorithm for computing bilinear form of matrix powers $(v, A^k h)$. We have shown that with increasing the perturbations the error and the variance are increasing too. Especially small matrices have a high variance. For a rising power of A an increase of the relative error can be observed. The robustness of the Monte Carlo algorithm with balanced matrices with matrix norms much smaller than 1 has been demonstrated. In these cases the variance has improved a lot compared to cases were matrices have norms close to 1. We can conclude that the balancing of the input matrix is very important for MC computations. A balancing procedure should be performed as an initial (preprocessing) step in order to improve the quality of Monte Carlo algorithms. For matrices that are "close" in some sense to the stochastic matrices the accuracy of the MC algorithm is very high.

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. I. Dimov, *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.
3. I. Dimov, *Monte Carlo Algorithms for Linear Problems*, Pliska (Studia Mathematica Bulgarica), Vol. 13 (2000), pp. 57-77.
4. I. Dimov, T. Dimov, T. Gurov, *A New Iterative Monte Carlo Approach for Inverse Matrix Problem*, Journal of Computational and Applied Mathematics, Vol. 92 (1997), pp. 15-35.
5. 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.
6. 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.
7. G.V. Golub, C.F. Van Loon, *Matrix computations (3rd ed.)*, Johns Hopkins Univ. Press, Baltimore, 1996.
8. I.M. Sobol *Monte Carlo numerical methods*, Nauka, Moscow, 1973.
9. J.R. Westlake, *A Handbook of Numerical matrix Inversion and Solution of Linear Equations*, John Wiley & Sons, inc., New York, London, Sydney, 1968.
10. M. Mascagni, A. Karaivanova, *A Parallel Quasi-Monte Carlo Method for Computing Extremal Eigenvalues*, Monte Carlo and Quasi-Monte Carlo Methods (2000), Springer, pp. 369-380.