Comparison of the Computational Cost of a Monte Carlo and Deterministic Algorithm for Computing Bilinear Forms of Matrix Powers

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

Centre for Advanced Computing and Emerging Technologies
School of System Engineering, The University of Reading
Whiteknights, PO Box 225, Reading, RG6 6AY, UK
{c.weihrauch, i.t.dimov, s.j.branford, v.n.alexandrov}@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 consider bilinear forms of matrix polynomials and show that these polynomials can be used to construct solutions for the problems of solving systems of linear algebraic equations, matrix inversion and finding extremal eigenvalues. An almost Optimal Monte Carlo (MAO) algorithm for computing bilinear forms of matrix polynomials is presented.

Results for the computational costs of a balanced algorithm for computing the bilinear form of a matrix power is presented, i.e., an algorithm for which probability and systematic errors are of the same order, and this is compared with the computational cost for a corresponding deterministic method.

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

1 Introduction

Many scientific and engineering applications are based on the problems of finding extremal eigenvalues, solving a system of linear algebraic equations (SLAE), or inverting a real $n \times n$ matrix (MI). 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 methods give statistical estimates of the required solution, by performing random sampling of a random variable, whose mathematical expectation is the desired solution [10, 11].

Several authors have presented work on the estimation of computational complexity of linear algebra problems [4, 5, 6, 12, 13, 14]. In this paper we consider bilinear forms of matrix powers, which can be used to formulate solutions for all three problems. Considering the set, $\mathcal{A}$, of algorithms, A, for calculating bilinear forms of matrix powers

$$\mathcal{A} = \{A : Pr\{r_{n,N} \le \varepsilon\} \ge c\},$$

with a probability error less than a given constant ε, there is the practical question of which algorithm in the set has the smallest computational cost. In this paper we compare the computational cost of two such method - a Monte Carlo algorithm and a deterministic algorithm.

The formulation of MI, SLAE and finding extreme eigenvalues in terms of bilinear forms of matrix powers is presented in Section 2; in Section 3 we present a Monte Carlo algorithm for finding the bilinear form of a matrix power; the computational cost of the Monte Carlo algorithm and of the deterministic method are presented in Section 4; and we conclude the work in Section 5.

2 Formulation of the Problems

In this paper we are interested in the evaluation of forms

$$(v, p(A)h)\,, \tag{1}$$

where $p(A)$ is a matrix polynomial and $v, h \in I\!\!R^n$ are arbitrary vectors.

2.1 Bilinear Form of Matrix Powers

In a special case of $p(A) = A^k$ then (1) becomes

$$(v, A^k h).$$

2.2 Eigenvalues of Matrices

The well-known Power method [9] gives an estimate for the dominant eigenvalue λ_1, of a matrix A. This estimate is called *Rayleigh quotient*:

$$\lambda_1 = \lim_{k\to\infty} \frac{(v, A^k h)}{(v, A^{k-1} h)},$$

where $v, h \in I\!\!R^n$ are arbitrary vectors. The Rayleigh quotient is used to obtain, for an arbitrary large natural number k, an approximation:

$$\lambda_1 \approx \frac{(v, A^k h)}{(v, A^{k-1} h)}. \tag{2}$$

To construct an algorithm for evaluating the minimal by modulo eigenvalue, λ_n, one has to consider the following matrix polynomial:

$$p(A) = \sum_{k=0}^{\infty} q^k C^k_{m+k-1} A^k, \tag{3}$$

where C^k_{m+k-1} are binomial coefficients, and the characteristic parameter, q, is used as acceleration parameter of the algorithm [4, 7, 8].

If $|qA| < 1$, then the polynomial (3) becomes the resolvent matrix [6, 7]:

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

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

$$x = qAx + h. \tag{4}$$

Values $q_1, q_2, \dots \quad (|q_1| \le |q_2| \le \dots)$ for which equation (4) is fulfilled are called characteristic values of the equation. The resolvent operator $R_q = [I - qA]^{-1} = A + qA^2 + \dots$ exists if the sequence converges.

Let us consider the ratio:

$$\lambda = \frac{(v, Ap(A)h)}{(v, p(A)h)} = \frac{(v, AR_q^m h)}{(v, R_q^m h)}.$$

If $q < 0$, then

$$\frac{(v, AR_q^m h)}{(v, R_q^m h)} \approx \frac{1}{q}\left(1 - \frac{1}{\mu^{(k)}}\right) \approx \lambda_n, \tag{5}$$

where $\lambda_n = \lambda_{min}$ is the minimal by modulo eigenvalue, and $\mu^{(k)}$ is the approximation to the dominant eigenvalue of R_q.

If $|q| > 0$, then

$$\frac{(v, AR_q^m h)}{(v, R_q^m h)} \approx \lambda_1, \tag{6}$$

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

The approximate equations (2), (5) and (6) can be used to formulate efficient Monte Carlo algorithms for evaluating both the dominant and the minimal by modulo eigenvalue of real symmetric matrices.

2.3 Bilinear Forms of Solution of LAE Systems

Consider the bilinear form:

$$(v, x), \tag{7}$$

where x is the solution of the system:

$$Bx = b. \tag{8}$$

For a non-singular matrix B one can use the presentation of Jacobi Overrelaxation Iterative Method:

$$x = Ax + h. \tag{9}$$

Assume that matrix A satisfies:

$$\sum_{j=1}^{n} |a_{ij}| < 1, \quad i = 1, \dots, n. \tag{10}$$

Now, we can consider the *first-order stationary linear iterative process* for the system (9):

$$x^{(k)} = Ax^{(k-1)} + h, \quad k = 1, 2, \dots. \tag{11}$$

In fact, the presentation (11) defines a Neumann series

$$x^{(k)} = h + Ah + \ldots + A^{k-1}h + A^k x^{(0)}. \tag{12}$$

It is a well-known fact that the property (10) is a sufficient condition for convergence of the Neumann series, i.e.,

$$x = \lim_{k\to\infty} x^{(k)}.$$

It is clear, that every iterative algorithm (including those based on MC) uses a finite number of iterations k. If a MC algorithm is applied, then the k^{th} iteration can be computed with an additional statistical error. In practice the truncating parameter k is not a priori given parameter. Normally it is obtained from the condition that the difference between the stochastic approximation of two successive approximations is smaller than a given sufficiently small parameter ε. Thus, we approximate the bilinear form (7) by

$$(v, x) \approx \left(v, \sum_{i=0}^{k} A^i h\right). \tag{13}$$

One can see now, that for this problem, the matrix polynomial is of special type, i.e., $p(A) = \sum_{i=0}^{k} A^i$.

If the Neumann series (12) does not converge the technique of mapping can be applied. This gives us a resolvent method, with the bilinear form (7):

$$(v, x) \approx \left(v, \sum_{i=0}^{k} g_i^{(k)} A^i h\right). \tag{14}$$

This procedure leads to matrix polynomial of type: $p(A) = \sum_{i=1}^{k} g_i^{(k)} A^i$.

2.4 Matrix Inversion

Assume $B \in I\!R^{n\times n}$ is a non-singular matrix. The problem of finding the inverse matrix

$$C = B^{-1}$$

of B is equivalent to solve n times the problem (8) written in the following form:

$$Bc_j = e^{(j)}, \quad j = 1, \ldots, n,$$

where $e^{(j)} \equiv (0, \ldots, 0, \underbrace{1}_{j}, 0, \ldots, 0)^T$ and $c_j \equiv (c_{1j}, \ldots, c_{nj})^T$ is the j^{th} column of the inverse matrix $C = B^{-1}$. This leads to the following bilinear form:

$$c_{ij} = (e^{(i)}, c_j) \approx \left(e^{(i)}, \sum_{i=0}^{k} A^i d_j e^{(j)}\right). \tag{15}$$

It is easy to see, that the latter form (15) is the same as (13) for a special choice of vectors v and h: $v = e^{(i)}$ and $h = d_j e^{(j)}$.

3 Formulation of the MC Algorithm

We shall use the so-called *MAO* algorithm, studied in [1, 2, 3, 5, 6], where

$$p_i = \frac{|v_i|}{\parallel v \parallel}, \quad \parallel v \parallel = \sum_{i=1}^{n} |v_i| \quad \text{and} \quad p_{ij} = \frac{|a_{ij}|}{\parallel a_i \parallel}, \quad \parallel a_i \parallel = \sum_{j=1}^{n} |a_{ij}|. \tag{16}$$

3.1 MC Algorithm for Computing Bilinear Forms of Matrix Powers $(v, A^k h)$

From the pair of density distributions (16) we obtain a finite chain, which induces the matrix/vector powers $A_v^k = v_{\alpha_0} \prod_{s=1}^{k} a_{\alpha_{s-1}\alpha_s}$ and $\parallel A_v^k \parallel = \parallel v \parallel \times \prod_{s=1}^{k} \parallel a_{\alpha_{s-1}} \parallel$. With such densities we have that $E\{h_{\alpha_k}\} = \frac{sign\{A_v^k\}}{\|A_v^k\|} \left(v, A^k h\right)$.

If we consider N realizations of the Markov chain $T_k = \alpha_0 \rightarrow \alpha_1 \rightarrow \ldots \rightarrow \alpha_k$, then

$$\bar{\theta}^{(k)} = \sum_{i=1}^{N} \theta_i^{(k)} = sign\{A_v^k\} \parallel A_v^k \parallel \sum_{i=1}^{N} \{h_{\alpha_k}\}_i \tag{17}$$

is an MC approximation of the bilinear matrix power $(v, A^k h)$. The probability error of this approximation is

$$R_N^{(k)} = \left| (v, A^k h) - \bar{\theta}^{(k)} \right| = c_p \sigma\{\theta^{(k)}\} N^{-\frac{1}{2}}. \tag{18}$$

From (17), together with the sampling rules using (16), leads us to a MC algorithm for estimating $(v, A^k h)$ with a probability error $R_N^{(k)}$. The quality of the MC algorithm depends on the behaviour of the standard deviation $\sigma\{\theta^{(k)}\}$.

4 Performance Analysis

In this section we formulate the computational cost of a Monte Carlo algorithm and a deterministic algorithm for computing the bilinear forms of matrix powers $(v, A^k h)$. To do this we use the following notation: α is a cost of an addition or subtraction; β is a cost of a multiplication; and δ is a cost of a logical operation.

4.1 Computational Cost of MC Algorithm for Computing Bilinear Forms of Matrix Powers $(v, A^k h)$

To estimate the computational cost one needs to consider the following expression:

$$sign\{v_{\alpha_0}\} \sum_{\alpha} |v_\alpha| \left(\prod_{i=1}^{k} sign\{a_{\alpha_{i-1}\alpha_i}\} \sum_{\alpha_i} |a_{\alpha_{i-1}\alpha_i}| \right) h_{\alpha_k}. \tag{19}$$

The computational cost of (19) is thus:

$$\begin{aligned} &\delta + \beta + (n-1)\alpha + \beta + (k-1)\beta + k\delta + k(\beta + (n-1)\alpha) + \beta \\ &= (k+1)(n-1)\alpha + (2k+2)\beta + (k+1)\delta. \end{aligned} \tag{20}$$

For the generation of the required random numbers and for the selection of the elements of v and A the following operations are required:

$$m + \delta \log n + k\left(m + \delta \log n\right), \tag{21}$$

where m is the number of operations required to generate a random number. The $\delta \log n$ term is from the binary search method [15] used to select the next element in the Markov Chain, as used in [1].

To compute the MC approximation to $(v, A^k h)$ one needs to perform N realizations of the Markov chain, so that the computational cost of the algorithm is

$$\begin{aligned} &N\left[(k+1)(n-1)\alpha + 2(k+1)\beta\right. \\ &\left.+(k+1)\delta + m + \delta \log n + k\left(m + \delta \log n\right)\right]. \end{aligned} \tag{22}$$

4.2 Computational Cost of a Deterministic Method for Computing Bilinear Forms of Matrix Powers $(v, A^k h)$

For a matrix-vector multiplication: $\sum_{j=1}^{n} a_{ij} h_j, \quad \forall i \in 1, \ldots, n$, the computational cost is:

$$n\left((n-1)\alpha + n\beta\right) = n(n-1)\alpha + n^2\beta.$$

For a vector-vector multiplication: $\sum_{j=1}^{n} v_j h_j$ the computational cost is:

$$(n-1)\alpha + n\beta.$$

To compute the bilinear form of a matrix power, $(v, A^k h)$, one needs to perform k matrix-vector multiplications and one vector-vector multiplication. Thus, the computational cost of the deterministic method is:

$$\begin{aligned} &k[n(n-1)\alpha + n^2\beta] + (n-1)\alpha + n\beta \\ =\ &(kn^2 - kn + n - 1)\alpha + (kn^2 + n)\beta. \end{aligned} \tag{23}$$

4.3 Comparison of the Monte Carlo and Deterministic Method

For a rough estimation of the cost we may assume that $\alpha = \beta = \delta = 1$. Then, from (22), the computational cost for the Monte Carlo algorithm becomes:

$$N(kn + n + k\log n + \log n + km + m + 2k + 2) = Nkn + O(Nn) \tag{24}$$

and the computational cost for the deterministic method, from (23), is:

$$2kn^2 - kn + 2n - 1 = 2kn^2 + O(kn). \tag{25}$$

A comparison of costs (24) and (25) shows that, for a sufficiently large matrix, the Monte Carlo algorithm will be quicker than the deterministic method, provided that one can keep the number of Markov chains required to reach a suitably accurate solution low enough. As a rough criteria for choosing the proposed Monte Carlo algorithm we can use the following inequality:

$$\frac{1}{2}\frac{N}{n} \leq 1.$$

If the required accuracy to compute $(v, A^k h)$ is ε, then (according to (18)) the following inequality

$$2\varepsilon^2 n \geq c_p^2 \sigma^2(\Theta^{(k)})$$

should be fulfilled. Let us note that for many real-life applications the matrix size n is $10^7 - 10^8$ and the typical number of Markov chains N is $10^4 - 10^5$. In such cases the Monte Carlo algorithm is definitely preferable. However, it should be mentioned that if the matrix size n is close to N, then more accurate analysis of the computational cost taking into account weights α, β and δ of different operations has to be done.

5 Conclusion

In this paper we introduced the matrix polynomial form (1). Further (2), (5) and (6) show how this form can be used to find extreme eigenvalues; (13) and (14) show how this form can be used to solve SLAE; and (15) show how to extend the solution of SLAE to MI.

From there we concentrated on the bilinear form of matrix polynomials and constructed a Monte Carlo algorithm for solving this problem. This allowed us to estimate the computational cost of the Monte Carlo algorithm, as well as the cost of the deterministic approach. These results on computational cost show that the Monte Carlo algorithm will out perform the deterministic method on sufficiently large problems, which are common to many areas of mathematical modelling. In cases when the matrix size n is close to the required number of Markov chains N a careful error analysis of the MC algorithm is needed.

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, 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.

5. 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.
6. 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.
7. 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.
8. 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.
9. G.V. Golub, C.F. Van Loon, *Matrix computations (3rd ed.)*, Johns Hopkins Univ. Press, Baltimore, 1996.
10. I.M. Sobol *Monte Carlo numerical methods*, Nauka, Moscow, 1973.
11. J.R. Westlake, *A Handbook of Numerical matrix Inversion and Solution of Linear Equations*, John Wiley & Sons, inc., New York, London, Sydney, 1968.
12. 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.
13. I. Dimov, *Monte Carlo Algorithms for Linear Problems*, Pliska (Studia Mathematica Bulgarica), Vol. 13 (2000), Proceedings of the 9th International Summer School on Probability Theory and Mathematical Statistics, Sozopol, 1997, pp. 57-77.
14. I. Dimov, A. Karaivanova and P. Yordanova, *Monte Carlo Algorithms for calculating eigenvalues*, Second International Conference on Monte Carlo and Quasi-Monte Carlo methods in scientific computing, University of Salzburg, 8-12 July, 1996, (in: Proceedings of MC & QMC 96, Springer Notes in Statistics (H. Niederreiter, P. Hellekalek, G. Larcher and P. Zinterhof, Eds)), 1997, pp. 205-220.
15. D. Knuth, *The Art of Computer Programming, Volume 3: Sorting and Searching*, Third Edition. Addison-Wesley, 1997.