Improved Monte Carlo Linear Solvers Through Non-diagonal Splitting

Ashok Srinivasan[1] and Vikram Aggarwal[1]

Department of Computer Science, Florida State University,
Tallahassee FL 32306, USA
{asriniva,viaggarw}@cs.fsu.edu

Abstract. Fast, but approximate, solutions to linear algebra problems have many potential applications, such as in graph partitioning, preconditioning, information retrieval, etc. Monte Carlo techniques appear attractive for such needs. While Monte Carlo linear solvers have a long history, their application has been limited due to slow convergence. Despite the development of techniques to improve their accuracy, current methods suffer from the drawback that they are stochastic realizations of inherently poor iterative methods. The reason for such choices is the need for efficient Monte Carlo implementation, which has restricted the splittings that are considered. However, in this paper we demonstrate that such restrictions are not necessarily required, and that efficient Monte Carlo implementations are possible even with splittings that do not appear amenable to it.

1 Introduction

The use of Monte Carlo (MC) in linear algebra dates back to the work of von Neumann and Ulam (described by Forsythe and Leibler [1] in 1950). However, with the development of modern deterministic numerical techniques, MC started losing its appeal in numerical linear algebra. However, there are several applications [2] where approximate solutions would suffice, such as preconditioning, graph partitioning, information retrieval, and feature extraction. These applications are becoming increasingly important, and MC is an ideal technique for determining approximate solutions. Furthermore, parallel MC is very latency tolerant, and so should be effective in a Grid-like environments. MC can also yield specific components of the solution.

The MC technique for solving linear systems consists of generating a random walk, based on the values of the elements of the matrix. The solution is then estimated as a function of the states visited by the walk. Other linear algebra kernels, such as matrix-vector multiplication and eigenvalue determination, can be computed via similar procedures. MC techniques can be used, either as substitutes for conventional deterministic techniques, or to accelerate the convergence of deterministic iterative techniques, for example, through preconditioning.

A major problem with current MC techniques is that they are fundamentally based on stationary iterative methods with poor convergence properties. We shall summarize current schemes in Sect. 2 and point out the reason for poor convergence.

We will then explain the basis for our procedure in Sect. 3. Next, we will explain its implementation for dense matrices in Sect. 4, and give experimental results illustrating its convergence properties in Sect. 5. We shall then explain efficient implementation with sparse matrices, in Sect. 6, and then summarize our conclusions in Sect. 7.

2 Current Techniques

2.1 Matrix Vector Multiplication

The basis for MC linear algebra techniques is the ability to perform stochastic matrix-vector multiplication. Based on this ability, stochastic versions of other algorithms can be developed, such as stationary iterative techniques for linear solvers, power method for eigenvalues, etc. We first outline the idea behind stochastic matrix-vector multiplication.

Matrix-vector Products and Sums of Products Consider the matrix-vector product, $C \cdot h$, $C \in \Re^{n \times n}$ and $h \in \Re^n$. We construct "transition-probability" and "weight" matrices P and W satisfying the following constraint

$$C_{ij} = P_{ij} \times W_{ij}, 1 \leq i, j \leq n \ , \tag{1}$$

where

$$\sum_{i=1}^{n} P_{ij} = 1, 1 \leq j \leq n \ . \tag{2}$$

(The constraint on P, introduced by (2), can be relaxed by introducing absorbing states; however we shall not discuss that.) We similarly define "initial-probability" and "initial-weight" vectors p and w satisfying the following constraint

$$h_i = p_i \times w_i, 1 \leq i \leq n \ , \tag{3}$$

with

$$\sum_{i=1}^{n} p_i = 1 \ . \tag{4}$$

MC techniques estimate $C^j h$, $j \geq 0$ by constructing a Markov chain of length j, with initial probabilities given by the vector p, and transition probabilities by P. The random walk visits a set of states in $\{1, \ldots, n\}$, and we denote the state visited in the i th step by k_i, $i \in [0, j]$. The probability of the initial state being α is given by

$$Prob(k_0 = \alpha) = p_\alpha \ , \tag{5}$$

and the transition probability is given by

$$Prob(k_i = \alpha | k_{i-1} = \beta) = P_{\alpha\beta} \ . \tag{6}$$

Consider random variables X_i defined as follows

$$X_0 = w_{k_0}, \ X_i = X_{i-1} \times W_{k_i k_{i-1}} \ . \tag{7}$$

If we let δ denote the Kronecker delta function ($\delta_{ij} = 1$ if $i = j$, and 0 otherwise), then it can be shown [3] that

$$E(X_j \delta_{ik_j}) = (C^j h)_i, \ 1 \leq i \leq n \ . \tag{8}$$

Therefore, for each random walk, $X_j \delta_{ik_j}$ can be used to estimate the i th component of $C^j h$. That is, in any random walk, the k_j th component is estimated as X_j, and all the other components are estimated as 0. We perform many random walks, maintaining a running sum for estimates of each component, and finally average over the number of walks. Note that in any single walk, the running sum for only one component needs to be updated.

We will later find it useful to estimate $\sum_{j=0}^{m} C^j h$ through

$$E(\sum_{j=0}^{m} W_j \delta_{ik_j}) = (\sum_{j=0}^{m} C^j h)_i \ . \tag{9}$$

Transition Probability Choice: Popular choices to satisfy (1) and (3) are to either (i) make all probabilities in a given column proportional to the magnitude of the corresponding element, or (ii) make all probabilities in each column equal.

Multiple Matrices We can easily extend the above to multiplying by more than one matrix. For example, we will find it useful to estimate $\sum_{j=0}^{m} (BC)^j Bh$, where $B, C \in \Re^{n \times n}$ and $h \in \Re^n$. Transition probabilities and weights for C and h are chosen with the constraints given earlier by (1) and (3). Similarly, transition probabilities and weights for B are chosen such that

$$B_{ij} = \hat{P}_{ij} \times \hat{W}_{ij}, 1 \leq i, j \leq n \ , \tag{10}$$

where $\hat{P}$ is the transition probability matrix and $\hat{W}$ is the weight matrix, with

$$\sum_{i=1}^{n} \hat{P}_{ij} = 1, 1 \leq j \leq n \ . \tag{11}$$

We construct a Markov chain of length $2m+1$ to estimate $\sum_{j=0}^{m} (BC)^j Bh$, with initial probabilities given by the vector p, and subsequent transition probabilities by $\hat{P}$ and P. The random walk visits a set of states in $\{1, \ldots, n\}$, and we denote the state visited in the i th step by k_i, $i \in [0, 2m+1]$. The probability of the initial state being some α is given by

$$Prob(k_0 = \alpha) = p_\alpha \ , \tag{12}$$

and the transition probabilities by

$$Prob(k_{2i+1} = \alpha | k_{2i} = \beta) = \hat{P}_{\alpha\beta}, 0 \le i \le m \ , \tag{13}$$

and

$$Prob(k_{2i} = \alpha | k_{2i-1} = \beta) = P_{\alpha\beta},\ 1 \le i \le m \ . \tag{14}$$

Consider random variables X_i defined as follows

$$X_0 = w_{k_0},\ X_{2i+1} = X_{2i}\hat{W}_{k_i k_{i-1}},\ 0 \le i \le m,\ X_{2i} = X_{2i-1}W_{k_i k_{i-1}},\ 1 \le i \le m \ . \tag{15}$$

It can be shown, along the same lines as with $\sum_{j=0}^{m} C^j b$, that

$$E(\sum_{j=0}^{m} X_{2j+1}\delta_{ik_{2j+1}}) = (\sum_{j=0}^{m} (BC)^j Bh)_i \ . \tag{16}$$

This enables us to estimate $\sum_{j=0}^{m}(BC)^j Bh$.

2.2 Linear Solvers

In order to solve $Ax = b$, $A \in \Re^{n\times n}$ and $x, b \in \Re^n$, the starting point of MC techniques is to split A as

$$A = N - M \ , \tag{17}$$

and write the fixed-point iteration [4]

$$x^{(m+1)} = N^{-1}Mx^{(m)} + N^{-1}b = Cx^{(m)} + h \ , \tag{18}$$

where $C = N^{-1}M$ and $h = N^{-1}b$. Then we get

$$x^{(m)} = C^m x^{(0)} + \sum_{i=0}^{m-1} C^i h \ . \tag{19}$$

The initial vector $x^{(0)}$ is often taken to be h for convenience, yielding

$$x^{(m)} = \sum_{i=0}^{m} C^i h \ . \tag{20}$$

$x^{(m)}$ converges to the solution as $m \to \infty$ if $\|C\| < 1$.

MC techniques construct a Markov chain to estimate the sum in (20), with the initial probabilities determined by h, and transition probabilities by C, as explained in Sect. 2.1. We wish to note that there are many different estimators available [5, 6, 1, 7, 8] (with the one we mentioned being a popular one, and similar to that introduced by Wasow [8]). However, the fundamental idea behind the estimators are essentially the same.

For effectiveness of the MC technique, efficient determination of C is considered important. Therefore current MC techniques choose N to be a diagonal matrix, thereby yielding $C = N^{-1}M$ efficiently. This yields, for example, the Jacobi method when N is taken to be the diagonal of A[1]. This perceived need for choosing N to be diagonal has resulted in the iterative schemes underlying current MC techniques having poor convergence properties. Though variance reduction, residual correction, and other techniques have been applied on top of this to get better accuracy, the fundamental limitation is that the MC techniques estimate a quantity that itself does not converge fast.

On the other hand, we note that MC techniques do not have to be based on the best possible iterative technique. Estimates can be obtained fast, and this fact may compensate for the underlying iterative scheme having poor convergence properties. Despite this fact, MC techniques have generally not been competitive with deterministic techniques, except for a limited number of applications. Furthermore, the convergence properties of the underlying iterative scheme restrict the systems to which the current MC techniques can be applied. Therefore there is a need for MC techniques based on better underlying iterations.

3 Non-diagonal Splitting

We wish to demonstrate that one can have efficient MC implementations of stationary methods even if we do not restrict ourselves to diagonal splittings. This can yield better convergence properties, at the potential cost of introducing inefficiencies at other points in the algorithm. However, we surmount these problems for our splitting through various means, as explained later. These ideas may also be generalized to other related splittings, which can yield efficient MC techniques for a wider variety of applications.

3.1 The Splitting

We choose N to be the diagonal and first subdiagonal of A. This yields N of the form

$$N = \begin{pmatrix} d_1 & & & & \\ s_2 & d_2 & & & \\ & s_3 & d_3 & & \\ & & & \ddots & \\ & & & s_n & d_n \end{pmatrix} . \tag{21}$$

(We assume $d_i \neq 0,\ 1 \leq i \leq n$, as with the Jacobi method. Otherwise N^{-1} does not exist.)

N^{-1} is a lower triangular matrix, and just computing $C = N^{-1}M$ would make this splitting noncompetitive. So we do not explicitly compute C, but rather, estimate the result of the recurrence

$$x^{(m+1)} = N^{-1}Mx^{(m)} + N^{-1}b \ , \tag{22}$$

[1] It is also popular to take N to be the identity matrix.

which yields the following when we take $x^{(0)} = b$.

$$x^{(m)} = \sum_{j=0}^{m} (N^{-1}M)^j N^{-1} b \ . \tag{23}$$

$x^{(m)}$ can be estimated as shown in Sect. 2.1. The number of steps in each random walk will be twice the number in the current techniques. However, if the method converges faster, then we will easily compensate for this.

Before discussing our technique further, we will determine N^{-1}.

3.2 Inverse of N

Each column of N^{-1} can be obtained by solving $Nx = e_i,\ 1 \le i \le n$, where e_i is the i th column of the identity matrix. We can then easily verify that N^{-1} is given by

$$N_{ij}^{-1} = \begin{cases} 0 & \text{if } i < j \\ \frac{1}{d_i} & \text{if i = j} \\ \frac{(-1)^{i-j}}{d_j} \Pi_{k=j+1}^{i} \frac{s_k}{d_k} & \text{otherwise} \end{cases} \tag{24}$$

We note the following two points:

- Any element of N^{-1} can be computed in constant time, using $O(n)$ storage and $O(n)$ pre-computation time as follows. Compute and store $T(i)$[2], $1 \le i \le n$, defined as follows

$$T(i) = \begin{cases} 1 & i = 1 \\ T(i-1)\frac{s_i}{d_i} & \text{otherwise} \end{cases} \tag{25}$$

  This takes $O(n)$ space and time.
  Any element of N^{-1} can be computed in constant time as follows

$$N_{ij}^{-1} = \begin{cases} 0 & \text{if } i < j \\ \frac{1}{d_i} & \text{if i = j} \\ \frac{(-1)^{i-j}}{d_j} \frac{T(i)}{T(j)} & \text{otherwise} \end{cases} \tag{26}$$

- The entire matrix N^{-1} can be computed in $O(n^2)$ time, for example, using (25) and (26). This contrasts with the $O(n^3)$ time required for general dense matrices.

We have not characterized the convergence properties of this splitting through a formal analysis, but since it appears to be more general than the Jacobi method, we surmise that it will converge at least for diagonally dominant systems, and empirical results seem to suggest this too. However, further work needs to be done to characterize the convergence properties, at least for matrices arising in specific applications.

[2] If some $s_i = 0$, then we need to make some modifications, as shown in [9].

One may attempt to generalize the current technique, by not choosing the sub-diagonal of A, along with the diagonal, as M, but rather, by choosing large elements of the lower triangle (or upper triangle) from each row of A. This may be advantageous in some applications where a few large entries occur at locations other than the diagonal or sub-diagonal.

Note that the MC technique has two factors that contribute to the error: (i) error from the underlying iterative technique, and (ii) stochastic error in estimating the iterative solution. Much effort has been placed by the MC community in reducing the second factor. These techniques, and also the Sequential MC idea, can be applied to our splitting too. We wish to motivate the search for MC implementations of better iterative processes by demonstrating that one need not be restricted to N being diagonal. Clearly, the splitting we have employed is not in any way as sophisticated as in modern deterministic stationary methods, not to mention non-stationary methods. The important contribution of this work is that we will show how this non-diagonal splitting can be efficiently implemented using MC techniques. These ideas will enable further improvement to MC techniques by permitting more general stationary methods, such as the simple extension mentioned in the previous paragraph.

4 Dense Matrix Implementation

The dense MC implementation of the above method is simple. Since A contains $\theta(n^2)$ elements, we will use $\theta(n^2)$ additional storage and $\theta(n^2)$ pre-computation time to enable efficient simulation. (For sparse systems, we cannot afford $\theta(n^2)$ storage or pre-computation time, and therefore the implementation is more complicated.)

Since M and b are explicitly available, and N^{-1} can be computed and stored in $O(n^2)$ space and time, as shown in Sect. 3.2, we can then estimate the solution using the procedure outlined in Sect. 2.1, taking $C = M$, $B = N^{-1}$ and $h = b$. We chose the probabilities to be proportional to the magnitude of the corresponding element. The alias method [10] is used for sampling, in view of its efficiency – the cost of determining the next state is essentially the cost of producing one random number, independent of the matrix size. This method requires some "information" on the probabilities to be pre-computed. We pre-compute this in an efficient manner, and accomplish it in time proportional to the number of elements. (Once this information is pre-computed, the explicit probabilities are no longer needed.) This requires $O(n^2)$ space and time for dense matrices. The weights are easily computed in $O(n^2)$ space and time [9].

5 Experimental Results

We next report some experimental results. We choose a matrix where the conventional technique will converge fast, in order to demonstrate different issues that affect the relative accuracy of the two techniques. The matrix A is chosen to be a 100×100 matrix with $A_{ij} = i - j$ except for the diagonal and subdiagonal elements. The subdiagonal is taken to be $(2/3)\times$ sum of the absolute values of all other elements of the column, excluding the diagonal. The diagonal is taken to be 1+ twice the sum of the absolute values of all the elements in the column. The vector b was taken to be $(1, 2, \cdots, n)^T$. The matrix A is diagonally dominant, ensuring that the conventional Jacobi-based technique converges. The parameter m of (20) and (23) was taken to be 2, since the iterations converge fast for this matrix. The sub-diagonal elements are moderately large, so that we can see the difference in performance between the two techniques. (At the other extreme, if the sub-diagonal were 0, then the two methods would estimate identical quantities.)

Figure 1 plots relative error ($||\text{Exact}-\text{Computed solution}||/||\text{Exact solution}||$) versus the "scaled" number of simulations. The scaling refers to the fact that, since our technique takes twice the number of steps that the conventional technique does, we multiply the number of simulations with our technique by a scaling factor of two, in order to make a fair comparison with the conventional one, in terms of time taken. For example, if the simulation number is reported as 1000, it implies 1000 simulations with the conventional technique, but only 500 with ours. The plot has two regions. When the number of simulations is small, the error is dominated by the contribution of the stochastic factor. In this region, the conventional method does better, since the actual number of simulations is greater for it. When the convergence of the underlying iteration is the dominant factor, then the new technique does better, as expected. Note that techniques that reduce the variance should be added in a real computation, and this will tend to reduce the stochastic error, and so our method would be become better at an earlier point (that is, with fewer simulations). For example, we demonstrate the use of a low discrepancy sequence (Scrambled Halton, with the scrambling due to Faure [11]), which makes our method better at a much earlier point.

6 Sparse Matrix Implementation

We now consider the case where A is sparse. The solution is still estimated using the procedure outlined in Sect. 2.1. Since A is sparse, so is M, and the space required for the weight matrix and the transition probability (implicitly stored as data for the alias method) is proportional to the number of non-zero elements in M. Weight and probability computation for the b vector is as for the dense case, since b is, in general, dense. However, N^{-1} is dense, and we can neither afford $O(n^2)$ computation time to determine it, nor $O(n^2)$ space to store it. We therefore use a sparse representation that enables us to store certain information in $O(n)$ space, with $O(n)$ pre-computation time. Using this, we will be able to generate the next state in constant time, and also compute the weight associated with that transition in constant time.

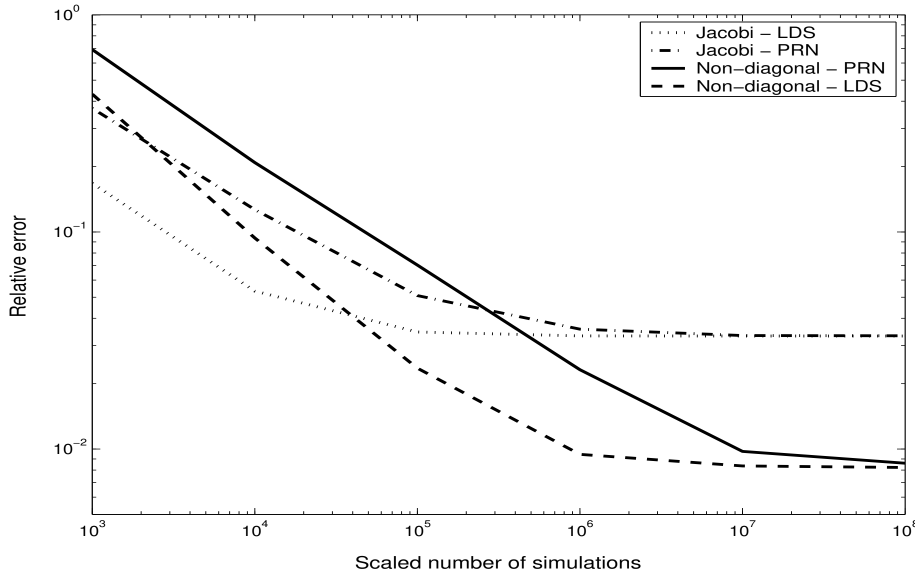

Fig. 1. Plot relative error versus scaled number of simulations. PRN refers to a pseudo-random number sequence, while LDS refers to a low discrepancy sequence.

We now show how we can accomplish the above goals. We assign transition probabilities[3] as follows

$$\hat{P}_{ij} = \begin{cases} 0 & \text{if } i < j \\ \frac{1}{n-j+1} & \text{otherwise} \end{cases} \tag{27}$$

Thus the probabilities for each element above the diagonal is 0, and all elements on and below the diagonal of each column have equal probabilities. Sampling from the uniform distribution can easily be accomplished in constant time, and does not require any additional memory.

The weights need to be chosen to satisfy $N^{-1}_{ij} = P_{ij}\hat{W}_{ij}$ (from (10)). Therefore, from (10) (with $B = N^{-1}$) and (27),

$$\hat{W}_{ij} = \begin{cases} 0 & \text{if } i < j \\ N^{-1}_{ij}(n-j+1) & \text{otherwise} \end{cases} \tag{28}$$

Any element N^{-1}_{ij} can be computed in constant time, as shown in Sect. 3.2, using $O(n)$ space and pre-computation time. Subsequently, $\hat{W}_{ij}$ is computed in constant time each time it is required, as defined by (28).

[3] If some $s_i = 0$, then we need to make some modifications, as shown in [9].

7 Conclusions

We have proposed efficient MC implementations for non-diagonal splitting, and also experimentally demonstrated its effectiveness on dense matrices. We have also proposed an effective sparse implementation, even though the explicit iteration matrix is dense. In fact, we believe that this is the major contribution of this work – demonstrating that efficient sparse implementations are possible even for splittings that lead to a dense C matrix. We expect further studies along these lines to permit stochastic solutions to a wider variety of applications, and have outlined ideas on generalization of the current work.

The sparse and dense implementations are based on the same underlying iterative scheme, and will therefore have identical convergence properties for the portion of the error arising from the iterative process. While the error from the stochastic simulations will converge to 0 with increase in the number of simulations, for both implementations, the actual magnitude of the error could differ significantly between the two implementations, since probabilities and weights are chosen in different manners. This needs to be studied empirically, once the software for the sparse technique is developed.

References

1. Forsythe, G.E., Leibler, R.A.: Matrix inversion by a Monte Carlo method. Mathematical Tables and Other Aids to Computation **4** (1950) 127–127
2. Srinivasan, A., Mascagni, M.: Monte Carlo techniques for estimating the Fiedler vector in graph applications. In: Lecture Notes in Computer Science – 2330, Springer-Verlag (2002) 635–645
3. Dimov, I.T., Karaivanova, A.N.: A power method with Monte Carlo iterations. In Iliev, Kaschiev, Margenov, Sendov, Vassilevski, eds.: Recent Advances in Numerical Methods and Appl. II. World Scientific (1999) 239–247
4. Heath, M.T.: Scientific Computing: An introductory survey. McGraw-Hill, New York (1997)
5. Curtiss, J.H.: Monte Carlo methods for the iteration of linear operators. Journal of Mathematical Physics **32** (1954) 209–232
6. 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. In: Symposium on Monte Carlo methods, University of Florida, 1954, 191–233, New York, John Wiley and Sons (1956)
7. Halton, J.H.: Sequential Monte Carlo. Proceedings of the Cambridge Philosophical Society **58 part 1** (1962) 57–78
8. Wasow, W.: A note on the inversion of matrices by random walks. Mathematical Tables and Other Aids to Computation **6** (1952) 78–78
9. Srinivasan, A.: Improved Monte Carlo linear solvers through non-diagonal splitting. Technical Report TR-030203, Department of Computer Science, Florida State University (2003) http://www.cs.fsu.edu/research/reports/TR-030203.ps.
10. Knuth, D.E.: The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, Third edition. Addison-Wesley, Reading, Massachusetts (1998)
11. Faure, H.: Good permutations for extreme discrepancy. Journal of Number Theory **42** (1992) 47–56