Stochastic linear solvers*

A. Srinivasan† *and V. Aggarwal*‡

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 [5] in 1950). However, with the development of modern deterministic numerical techniques, MC started losing its appeal in numerical linear algebra. There has been a recent revival of interest in MC linear algebra, partly because of advances in MC techniques, but more importantly, due to the increasing importance of applications where the use of MC techniques is attractive [11]. For example, the use of MC is promising in applications where approximate solutions are sufficient, such as in preconditioning, graph partitioning, information retrieval, and feature extraction. Furthermore, parallel MC is very latency tolerant, and so should be effective in a Grid-like environment. MC can also yield specific components of the solution. In addition, the convergence rate is independent of the size of the matrix.

The MC linear algebra techniques, in general, are based on the ability to perform stochastic matrix-vector multiplication. This permits the estimation of certain stationary techniques to solve linear systems and the power method to estimate the largest eigenvalue and corresponding eigenvector. The stochastic linear solver can in turn be used to estimate the smallest eigenvalue and corresponding eigenvector too. Other eigenvectors can also be obtained through deflation.

A major problem with current MC linear solver techniques, which we summarize in § 2, is that they are fundamentally based on stationary iterative methods with poor convergence properties. We proposed a different iterative process and evaluated it empirically with dense matrices in [9, 10], and we briefly outlined the idea behind the sparse implementation too. We summarize the splitting and the dense implementation in § 3, and discuss the sparse implementation in further detail in § 4. We then present experimental results illustrating its convergence properties in § 5, and summarize our conclusions in § 6.

*We wish to acknowledge computational resources provided by NCSA.

†Department of Computer Science, Florida State University, email: asriniva@cs.fsu.edu.

‡Department of Computer Science, Florida State University, email: viaggarw@cs.fsu.edu.

2 Current MC techniques

2.1 Matrix vector multiplication

The basis for MC linear algebra techniques is the ability to perform stochastic matrix-vector multiplication. So we first outline the idea behind stochastic matrix-vector multiplication. Further details can be found in [9].

Consider the matrix $C \in \Re^{n \times n}$ and vector $h \in \Re^n$. 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

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

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

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

Each random walk gives an estimate for the entire sum, with each step of the random walk updating a particular component of the running sum.

Examples of 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.

For example, if we make probabilities proportional to the magnitude of the elements, then the initial probabilities are given by

$$p_i = \frac{|h_i|}{\sum_{j=1}^{n} |h_j|}, \tag{10}$$

and the transition probabilities by

$$P_{ij} = \frac{|C_{ij}|}{\sum_{k=1}^{n} |C_{kj}|}. \tag{11}$$

The corresponding weights are given by

$$w_i = sign(h_i) \times \sum_{i=1}^{n} |h_i| \text{ and } W_{ij} = sign(C_{ij}) \times \sum_{k=1}^{n} |C_{kj}|. \tag{12}$$

Multiple matrices

We can easily extend the above technique 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). Transition probabilities $\hat{P}$ and weights $\hat{W}$ are similarly chosen for B too, and Markov chain of length $2m + 1$ to estimate $\sum_{j=0}^{m} (BC)^j Bh$, in a straight-forward generalization of the above technique. This is described in further detail in [9, 10].

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

$$A = N - M, \tag{13}$$

and write the fixed-point iteration [7]

$$x^{(m+1)} = N^{-1}Mx^{(m)} + N^{-1}b = Cx^{(m)} + h, \tag{14}$$

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

$$x^{(m)} = C^m x^{(0)} + \sum_{i=0}^{m-1} C^i h. \tag{15}$$

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

$$x^{(m)} = \sum_{i=0}^{m} C^i h. \tag{16}$$

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

MC techniques construct a Markov chain to estimate the sum in (16), with the initial probabilities determined by h, and transition probabilities by C, as explained in § 2.1. We wish to note that there are many different estimators available [1, 2, 5, 6, 12] (with the one we mentioned being a popular one, and similar to that introduced by Wasow [12]). 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. 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

In [9, 10], we demonstrated that one can have efficient MC implementations of stationary methods even if we do not restrict ourselves to diagonal splittings, and we discussed a dense matrix implementation in detail. We summarize the scheme below. 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.

3.1 The splitting

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

$$N = \begin{pmatrix} d_1 & & & & & \\ s_2 & d_2 & & & & \\ & s_3 & d_3 & & & \\ & & & \ddots & & \\ & & & & s_n & d_n \end{pmatrix}. \tag{17}$$

(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

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

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

$$x^{(m)} = \sum_{j=0}^{m} (N^{-1}M)^j N^{-1}b. \tag{19}$$

$x^{(m)}$ can be estimated as shown in § 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.

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. However, we note that MC techniques need not be based on the best deterministic iterative process, since they can compensate for slower convergence by producing their estimate fast. Furthermore, the focus of this work is to show that stochastic versions of a non-diagonal splitting can be efficiently implemented for sparse matrices. These ideas will enable further improvement to MC techniques by permitting more general stationary methods.

3.2 Dense matrix implementation

The matrix N^{-1} is lower triangular, and in general, the lower triangle is dense. While general matrix inversion takes $O(n^3)$ time, which is prohibitive, we showed in [9, 10] that N^{-1} requires just $O(n^2)$ time and space. The matrix A contains $\theta(n^2)$ elements, and we can use $\theta(n^2)$ additional storage and $\theta(n^2)$ pre-computation time to enable efficient simulation – each random walk of length m will take only $O(m)$ time.

4 Sparse matrix implementation

We now consider the case where A is sparse. The solution is still estimated using the procedure outlined in § 3, in particular, using (19). 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 [8], which we use for sampling) 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 the $O(n^2)$ 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.

We first observe that

$$N_{ij}^{-1} = \begin{cases} 0 & \text{if } i < j \\ \frac{1}{d_i} & \text{if i = j} \\ \frac{(-1)^{i-j}}{d_j} \Pi_{k=j+1}^{i} \frac{s_k}{d_k} & \text{otherwise} \end{cases} \tag{20}$$

If $s_k = 0,\ 2 \leq i \leq n$, then N_{ij}^{-1} will be zero if $i \geq k$ and $j < k$, as can be seen from (20). For each $j,\ 1 \leq j \leq n$, if there exists a $k,\ j+1 \leq k \leq n$, such $s_k = 0$, then define $L(j)$ to be the smallest such k. Otherwise, define $L(j)$ to be $n+1$. We note that the non-zero elements of column j are those in rows j to $L(j)-1$. $L(j),\ 1 \leq j \leq n$, can be computed in $O(n)$ time and space using the following recurrence

$$L(j) = \begin{cases} n+1 & j = n \\ L(j+1) & s_{j+1} \neq 0 \text{ and } j \neq n \\ j+1 & s_{j+1} = 0 \text{ and } j < n \end{cases} \tag{21}$$

Here we start by computing $L(n)$, and proceed in descending order, until we compute $L(1)$.

We now wish to define N_{ij}^{-1} to permit its efficient computation. We first define a function T by

$$T(i) = \begin{cases} 1 & i = 1 \text{ or } s_i = 0 \\ T(i-1)\frac{s_i}{d_i} & \text{otherwise} \end{cases} \tag{22}$$

We can precompute T using $O(n)$ time and space, using the definition above. Next we give a "computational" definition of N through

$$N_{ij}^{-1} = \begin{cases} 0 & \text{if } i < j \text{ or } i \geq L(j) \\ \frac{1}{d_i} & \text{if } i = j \\ \frac{(-1)^{i-j}}{d_j} \frac{T(i)}{T(j)} & \text{otherwise} \end{cases} \tag{23}$$

Since T has been precomputed, any component of N^{-1} can be obtained in a small constant time, using the above definition.

We next assign transition probabilities $\hat{P}$ such that the non-zero elements in each column of N^{-1} have equal probability of occurring. Thus $\hat{P}$ is defined by

$$\hat{P}_{ij} = \begin{cases} 0 & \text{if } i < j \text{ or } i \geq L(j) \\ \frac{1}{L(j)-j} & \text{otherwise} \end{cases} \tag{24}$$

We can obtain samples easily in a small constant time by suitably sampling from the uniform distribution.

Finally, we need to be choose weights $\hat{W}_{ij}$ that satisfy $N_{ij}^{-1} = \hat{P}_{ij}\hat{W}_{ij}$. Therefore

$$\hat{W}_{ij} = \begin{cases} 0 & \text{if } i < j \text{ or } i \geq L(j) \\ N_{ij}^{-1}(L(j) - j) & \text{otherwise} \end{cases} \tag{25}$$

Since any element N_{ij}^{-1} can be computed in constant time, any element of $\hat{W}_{ij}$ too can be computed in constant time, each time that it is required.

Simulations are performed as with the dense matrix, except that the probabilities for N^{-1} are chosen according to (24) and the corresponding weights by (25).

5 Experimental results

We wish to study the effectiveness of the splitting and of the sparse implementation. We 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. The sparse and dense implementations have the same error due to the iterative process. However, the stochastic errors can be qualitatively different, since they use different probabilities.

We will first discuss the properties of the dense implementation, and compare it with the conventional technique. We will then compare the stochastic errors in the sparse implementation with those from the dense implementation.

In the first experiment, we choose a dense 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 sub-diagonal 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 (16) and (19) 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.)

Fig. 1 plots relative error ($\|\text{Exact} - \text{Computed solution}\|_2 / \|\text{Exact solution}\|_2$) 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 [4]), which makes our method better at a much earlier point.

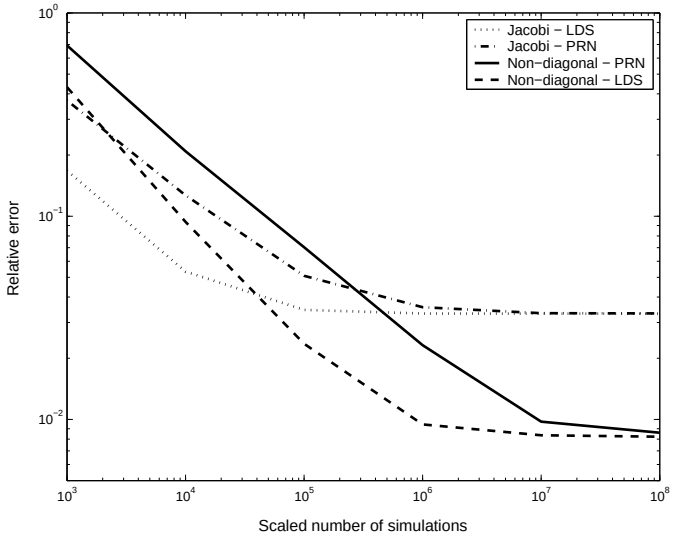

Figure 1. *Plot relative error versus scaled number of simulations. PRN refers to a pseudo-random number sequence, while LDS refers to a low discrepancy sequence.*

We next compare the stochastic component of the errors of the sparse and dense implementations in Fig. 2, with the same matrix as above. The relative error is computed with the value of the deterministic iterative process replacing that of the exact solution of the system. The number of simulations is not scaled, since both use the same walk length. Since the matrix is dense, the sparse implementation has no computational advantage. However, it is still be informative to evaluate it here.

We see that the error for the sparse implementation decreases at the same rate as for the dense one; however, the actual error is higher by a significant constant fact. We also note that the low discrepancy sequence does not help the sparse implementation much.

We next take the same matrix as above, but make all the elements zero, except the diagonal and four elements above and four elements below the diagonal of each column. The sparse implementations therefore uses much less memory than the dense one. We wish to determine if the accuracy too is acceptable. In this case, the sparse implementation improves, and with the low discrepancy sequence, it is

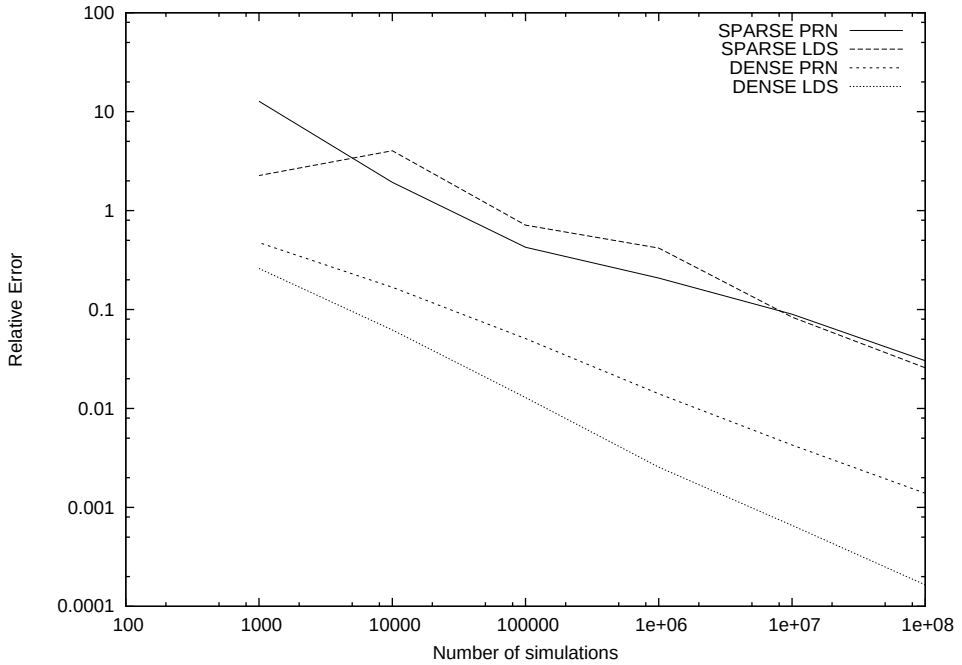

Figure 2. *Plot relative error versus number of simulations for a dense matrix. PRN refers to a pseudo-random number sequence, while LDS refers to a low discrepancy sequence.*

competitive with the dense implementation, as seen from Fig. 3.

The sparse implementation has a higher error since it chooses elements with equal probability in matrix N^{-1}, while the dense implementation chooses them with probability proportional to their magnitude. Elements with large values tend to lead to components with large magnitudes, and since the dense implementation chooses these more often, the large components are computed more accurately, leading to the norm of the error being smaller. A different definition of the error, for example, choosing it as the norm of the average relative error, might lead to a different result. An appropriate definition would depend on the needs of the application.

6 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. We expect further studies along these lines to permit stochastic solutions to a wider variety of applications.

The sparse and dense implementations of the non-diagonal splitting are based on the same underlying iterative scheme, and will therefore have identical con-

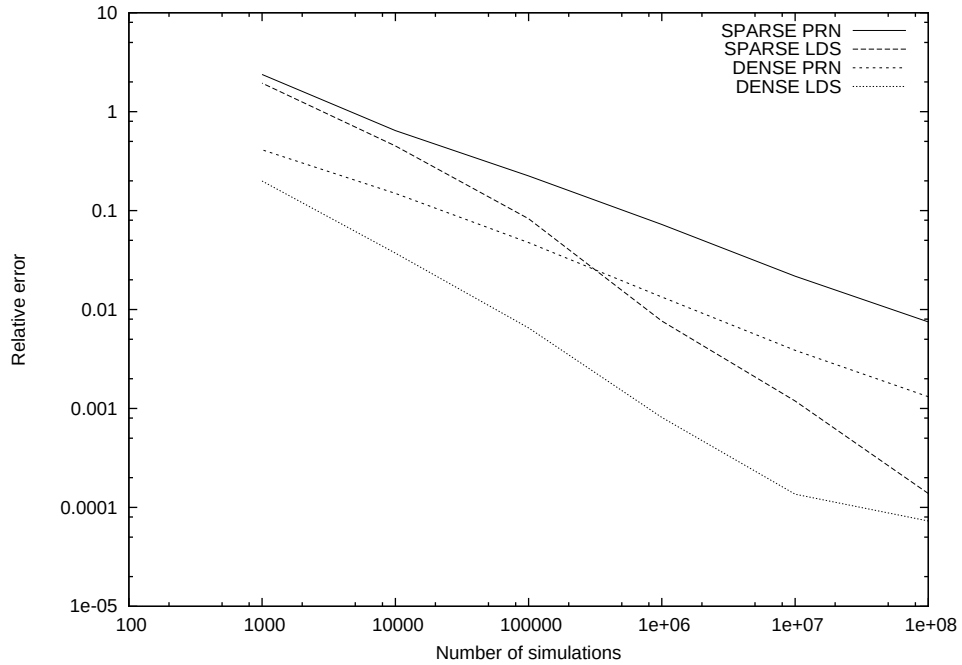

Figure 3. *Plot relative error versus number of simulations for a sparse matrix. PRN refers to a pseudo-random number sequence, while LDS refers to a low discrepancy sequence.*

vergence 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 was demonstrated empirically too.

The sparse implementation clearly uses much less space than the dense one, with sparse matrices. While the sparse implementation converges at the same rate as the dense one, its actual magnitude of the error is higher than that in the dense implementation, and we mentioned reasons for that above. Though the sparse technique with a low-discrepancy sequence performed well on a sparse matrix, the experiments, on the whole, suggest that choosing elements with equal probabilities is not that effective. We can modify this by choosing elements with varying probabilities, such that they can still be sampled efficiently, and adjust the weights accordingly. For example, we might sample from an exponentially decaying distribution, since the elements of N^{-1} decay for a diagonally dominant matrix. Clearly, this is an issue to which more thought needs to be given.

Bibliography

[1] J. H. Curtiss. Monte Carlo methods for the iteration of linear operators. *Journal of Mathematical Physics*, 32:209–232, 1954.

[2] J. H. Curtiss. 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, 1956. John Wiley and Sons.

[3] I. T. Dimov and A. N. Karaivanova. A power method with Monte Carlo iterations. In Iliev, Kaschiev, Margenov, Sendov, and Vassilevski, editors, *Recent Advances in Numerical Methods and Appl. II*, pages 239–247. World Scientific, 1999.

[4] H. Faure. Good permutations for extreme discrepancy. *Journal of Number Theory*, 42:47–56, 1992.

[5] G. E. Forsythe and R. A. Leibler. Matrix inversion by a Monte Carlo method. *Mathematical Tables and Other Aids to Computation*, 4:127–127, 1950.

[6] J. H. Halton. Sequential Monte Carlo. *Proceedings of the Cambridge Philosophical Society*, 58 part 1:57–78, 1962.

[7] M. T. Heath. *Scientific Computing: An introductory survey.* McGraw-Hill, New York, 1997.

[8] D. E. Knuth. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms, Third edition.* Addison-Wesley, Reading, Massachusetts, 1998.

[9] A. Srinivasan. 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] A. Srinivasan and V. Aggarwal. Improved Monte Carlo linear solvers through non-diagonal splitting. In *Lecture Notes in Computer Science: Proceedings of the 2003 International Conference on Computational Science and its Applications*. Springer Verlag, 2003 (to appear).

[11] A. Srinivasan and M. Mascagni. Monte Carlo techniques for estimating the Fiedler vector in graph applications. In *Lecture Notes in Computer Science – 2330*, pages 635–645. Springer-Verlag, 2002.

[12] W. Wasow. A note on the inversion of matrices by random walks. *Mathematical Tables and Other Aids to Computation*, 6:78–78, 1952.