JOURNAL OF COMPUTATIONAL PHYSICS 80, 362–373 (1989)

Targetted Stochastic Matrix Inversion

JOSEPH F. DREITLEIN AND GEORGE F. SOWERS

Physics Department, University of Colorado, Boulder, Colorado

Received October 9, 1987; revised February 29, 1988

The solution of linear systems by stochastic techniques is investigated as an alternative to the widely used elimination and relaxation procedures. The Pan–Reif algorithm is used to allow *any* non-singular matrix to be stochastically inverted. New algorithms are devised to overcome some of the efficiency problems of the original Von Neumann–Ulam method and its refinements. The method is tested on some simple physical problems to gauge its effectiveness.

1. INTRODUCTION

There are innumerable physical problems whose solutions await a fast and accurate method of large matrix inversion. For example, the computation of Green's functions on a latticized space or spacetime can be formulated as a matrix inversion problem. The requisite algorithms are still in a state of active development for both serial and parallel computer architecture [1].

In this paper, we develop further an alternative [2] to the widely used elimination and relaxation methods [3] for the solution of linear equations, such solutions being equivalent to matrix inversion processes. After outlining the general idea behind stochastic matrix inversion in Section 2, we show in Section 3 how the Pan–Reif algorithm can be used to condition any non-singular matrix for stochastic matrix inversion.

Two algorithms, to which Sections 4 and 5 are respectively dedicated, improve substantially the efficiency of computing a targetted inverse matrix element. (Targeted matrix inversion processes [4] compute a prescribed inverse matrix element, all that is needed in many computations of physical interest.) It should be noted that both algorithms are well suited to parallel processing. The algorithms are particularly suitable for sparse matrix inversion because in such cases the possible paths used in the inversion are severely restricted in number and are particularly easy to count and generate. An analysis of the random error involved in the computation shows that the error is typical of a stochastic process and goes as $1/\sqrt{N}$. The analysis leading to this result is essentially the same as that reported in Ref. [8].

The algorithms are tested on simple lattice problems which require targetted inverse matrix elements. Techniques, such as lookup table preparation, which

0021-9991/89 $3.00

increase the computational speed are mentionned and the results of calculations are reported in Section 6. Section 7 gives our conclusions about the efficiency and promise of stochastic matrix inversion.

2. Stochastic Matrix Inversion

Stochastic matrix inversion, suggested many years ago by Von Neumann and Ulam [5], involves finding a procedure for estimating the value of K^L in order to compute an approximant for the Neumann series for $M^{-1} = (1-K)^{-1}$. If the norm of K is less than one, the series converges. The next section shows how *any* inversion problem can be reduced to evaluating a convergent Neumann series.

We recapitulate here the salient points of the stochastic procedure. The matrix element $(K^L)_{n(0),\, n(L)}$ involves S^{L-1} terms of L products each, where $S \times S$ is the dimension of the matrix. Each term can be uniquely specified by picking a sequence of nodes $\{n\} \equiv \{n(0), n(1), ..., n(L)\}$ which is encountered in the product $K_{n(0),\, n(1)} K_{n(1),\, n(2)} \cdots K_{n(L-1),\, n(L)}$. A typical term in the sum will contribute a value, say V, to the matrix element $(K^L)_{n(0),\, n(L)}$.

In many of the inversion problems encountered in physics, the matrix is sparse and so many of the paths of nodes will lead to a vanishing term in the sum for $(K^L)_{n(0),\, n(L)}$. Only needed are paths with nonvanishing V. For example, if the matrix elements of $K_{i,j}$ are zero except when $j = i \pm 1$ with i, j taken modulo S then there are at most only 2^{L-1} non-zero valued paths encountered in evaluating K^L. Assume this case for the sake of discussion. Furthermore, let a particular one of the 2^{L-1} paths be selected with probability P by a procedure which takes into account the importance of the path (such a procedure will be detailed below). Then if a given path is stochastically selected, its contribution to the desired matrix element should be recorded by letting a random variable W assume the value (called the weight from now on)

$$W = V/P \tag{2.1}$$

if this particular path is selected. It follows that the expectation value of W is V, since in N tries it will be selected $N \times P$ times on the average. If the typical important paths are selected, one might hope that a relatively few paths will give a good estimate $\langle W \rangle$ for V. This is the essence of stochastic matrix inversion to which a few more details must no be added to make the method practical.

One procedure for implementing the stochastic matrix inversion process goes as follows. Take a random walk starting from the node $n(0)$. The first step leads to node $n(1)$ with some selected probability $p_{n(0),\, n(1)}$. Since the probability of winding up somewhere is one, the sum of probabilities of stepping to some node is one. If the value of the K matrix element between the two nodes is $V_{n(0),\, n(1)}$ then the weight associated with this step is taken as $V_{n(0),\, n(1)}/P_{n(0),\, n(1)} \equiv W_{n(0),\, n(1)}$. Continuing in this fashion, the contribution from the second step has the weight $W_{n(1),\, n(2)}$

formed from $V_{n(1),n(2)}$ and $P_{n(1),n(2)}$. After L steps, the value used in the estimates for the matrix element K^L to be recorded when this particular path is encountered is the product $W(L)$ of L terms of the weights computed in the walk:

$$W(L) \equiv \prod W_{n(i),n(j)}. \tag{2.2}$$

Since the path is chosen with probability

$$P(L) \equiv \prod P_{n(i),n(j)}, \tag{2.3}$$

the expectation value of $W(L)$ is the desired matrix element. A random walk of length L can be used to calculate the estimator not only for K^L but also the estimators for K^J with J between 0 and L. One merely uses Eq. (2.2) with L replaced by J and takes the product of J terms. If, as the random walk proceeds, the products are computed cumulatively and then summed, an estimator for the Neumann series which is truncated at term K^L results.

The procedure described calculates an estimator for $(M^{-1})_{n(0),n(L)}$. While the value of $n(0)$ can be selected as the starting point of the nodal walk, the value of $n(L)$ is only determined stochastically. If a particular inverse matrix element is desired, the procedure becomes computational inefficient especially when $L \gg 1$ and the matrices are sparse since the probability of hitting a particular final node $n(L)$ is small. What is needed is a procedure for procedure for targeting the random nodal walks at a prescribed final value $n(L)$. Sections 4 and 5 give two algorithms to accomplish the targetting efficiently. But first we show stochastic procedures can be applied to the inversion of any non-singular matrix.

3. Preparation of Matrices for Stochastic Matrix Inversion

All procedures for stochastic matrix inversion use a Neumann series expansion to compute the inverse of a matrix: $M^{-1} = (I-K)^{-1} = I + K + K^2 + \cdots$. The series will converge only if the matrix norm of K is less than unity.

If the norm of K is greater than one, the problem can be recast into a form which does involve a convergent Neumann series. The recasting of the problem requires an approximate inverse of M which shall be called W. Define the remainder matrix by

$$R(W) = I - (W \times M) \tag{3.1}$$

arranged to have a norm $|R(W)| < 1$. Then

$$M^{-1} = (W \times M)^{-1} W = (1 - R(W))^{-1} W \tag{3.2}$$

and $(I - R(W))^{-1}$ can be expanded in a convergent series expansion.

It is always possible to find such a suitable W, as pointed out by Pan and Reif

[6] and others [7]. An approximate inverse with the desired properties of boundedness is given in terms of the Hermitian conjugate $M^\dagger$ by

$$W = \lambda M^\dagger,$$
$$\lambda = \frac{1}{(\max_i \sum_j |M_{ij}|) \times (\max_j \sum_i |M_{ij}|)}. \tag{3.3}$$

The proof of this somewhat surprising construction of an approximate inverse is given in Pan and Reif. With this choice of W, the remainder matrix becomes

$$R(\lambda M^\dagger) = I - \lambda M^\dagger M \tag{3.4}$$

and requires one matrix multiplication to compute.

If M is sparse, then W is also sparse. This fact enables the procedures described in this paper to be extended to the stochastic calculation of

$$\sum_j (I - R(W))^{-1}_{ij} W_{jk} \tag{3.5}$$

in one of two alternative ways. The targetted values $(I - R(W))^{-1}_{ij}$ may be computed for the small number of indices for which W_{ij} is non-vanishing for a given k and then multiplied by these matrix elements which are then summed for the small number of terms. Alternatively, the matrix elements of $R^p \times W$ for a sequence of p values may be computed directly by an obvious modification of the procedure to be prescribed for calculating the matrix elements R^p.

4. Targetted Stochastic Matrix Inversion—Algorithm A

To avoid elaborate and possibly confusing notation and yet still present the basic idea of targetted stochastic matrix inversion, consider an $S \times S$ matrix with all elements vanishing except when $i = j \pm 1$ with all indices reduced modulo S. The random walk on nodes used to determine the estimator for the Neumann series consists of steps which move from node $n(k)$ to either node $n(k+1)$ or node $n(k-1)$. Again, to illustrate the ideas in a simple way, all sequences of nodes are selected with equal probability.

Suppose now that the random walk starts at a prescribed node $n(0)$ and ends at a predetermined node $n(L)$. Such walks cannot be determined by randomly stepping sequentially since such a procedure would not, in general, lead to the target node $n(L)$. Instead one can move a distance $D = (n(L) - n(0))$ from the starting node in L steps by moving E positive steps and O negative steps with the constraints

$$L = E + O,$$
$$D = E - O. \tag{4.1}$$

In general, there are many nodal walks which are of length L and proceed a distance D from the origin node. If there is at least one such walk, then there will be a total of F such walks where

$$F = \binom{E+O}{E} = \binom{L}{\frac{1}{2}(L+D)}. \tag{4.2}$$

It will always be true that both $L+D$ and $L-D$ are even if a walk can land on the prescribed site after L steps. It is not difficult to find an algorithm (Appendix A) to take the E steps and the O steps in a random sequence and so to generate an unbiased targetted walk. The issue to resolve is how to obtain the weight W which such a path will contribute to the Neumann series estimator.

First consider the construction of an estimator for K^L. The particular sequence of nodes generated by the random walk leads to a product of matrix elements which will be called V just as in the last section. The probability of taking this particular path is $1/F$. It is easy to see then that the random variable whose mean value is the estimator for the matrix K^L should receive a contribution of value

$$V \times F, \tag{4.3}$$

where V is determined by the particular path selected. The expected value $\langle W \rangle$ obtained by averaging the values of W over an ensemble of selected nodal paths will just be M^L. This procedure was described and tested in a previous work Ref. [4]. It represents an improvement over the method described in the last section if specific inverse matrix elements are needed rather than the entire inverse matrix. The increase in efficiency is very like the efficiency gained by using the relaxation rather than the elimination method for inversion of matrices which solve linear equations. However, it would be much more efficient if each path of nodes selected would generate an estimator for the whole Neumann series rather than just for one term. Such a method will now be presented.

Suppose in the process of walking randomly from $n(0)$ to $n(L)$, the goal displacement D is also realized on the Jth step. Then the estimator for K^J is to receive a contribution which depends upon the path value V and a factor $C = P^{-1}$. The path value V is obtained by multiplying the product of matrix elements determined by the sequence of nodes visited by the random walk just as described in Section 2. The computation of the weight $W = VC$ to be recorded for this "hit" contributing to $(K^J)_{n(0),\, n(L)}$ requires the value of the factor C obtained by combinatorial considerations. Recall that the number of paths which take a displacement D in L steps is

$$N(L, D) = \binom{L}{\frac{1}{2}(L+D)}. \tag{4.4}$$

If a path hits the targeted displacement D after J steps, then in the remaining $L-J$

steps the path must again wander to the target node $n(L)$. The number of paths of length $L-J$ which move from node $n(L)$ to node $n(L)$ is

$$N(L-J,0)=\binom{L-J}{\frac{1}{2}(L-J)}. \tag{4.5}$$

Therefore the probability of taking a path of length L which moves to target D in J steps and completes the journey in any way consistent with winding up with displacement D is

$$P=\frac{N(L-J,0)}{N(L,D)} \tag{4.6}$$

and so the weight of the path is

$$W=\frac{V}{P}=\frac{V\times L!(((1/2)(L-J))!)^2}{(L-J)!((1/2)(L+D))!((1/2)(L-D))!} \tag{4.7}$$

and should be recorded as an estimator for $(K^J)_{n(0),n(L)}$.

By keeping a record of the weights accumulated everytime a targetted walk lands on the target node, an estimator for the Neumann series rather than just a single term of the series is produced. The targetting of the path for L steps enhances the chance of hitting the target at $J<L$ steps since the path is restrained from diffusing too far from the target. The efficiency of targetting for a long path is thus partially shared by smaller length paths which hit the target.

5. Targetted Stochastic Matrix Inversion—Method *B*

An alternative method for targetted matrix inversion will now be presented. The main differences from the previously discussed procedure are the way of generating the paths and the concomitant changes in the method of computing estimators.

The method will be presented for a walk in two dimensions with unit steps chosen randomly in one of the four directions to nearest neighbors. To compute a path of length L which hits a target at a displacement $D=(DX, DY)$ from the origin, take the probability of making one of the four alternative steps to be

$$p(k)=\frac{N(L-1,D'(k))}{N(L,D)}. \tag{5.1}$$

Here $k=1, 2, 3, 4$ according to whether the step is in the $+x$, $-x$, $+y$, or $-y$ direction, respectively. The quantity $N(L, D)$ is the number of paths of length L suffering a displacement $D=(DX, DY)$ and $D'(k)$ is given by

$$
\begin{aligned}
D'(1) &= (DX-1, DY),\\
D'(2) &= (DX+1, DY),\\
D'(3) &= (DX, DY-1),\\
D'(4) &= (DX, DY+1);
\end{aligned}
\tag{5.2}
$$

$D'(k)$ is the displacement to the target after the step of type k has been taken. Subsequent steps are also chosen in accord with Eq. (5.1) but with L replaced by the number of steps remaining to be taken, D replaced by the current displacement to the goal, and D' replaced by the new displacement to the goal after the pending step is taken. The quantities $N(L, D)$ can readily be computed, in some cases analytically (for an example, consult Appendix B).

The algorithm for path generation given by Eq. (5.1) selects all possible paths in an unbiased way. Indeed the probability of traversing the sequence of nodes $N_0 = (X_0, Y_0)$, $N_1 = (X_1, Y_1)$, ..., $N_L = (X_L, Y_L)$ is clearly

$$
p(N_0 \to N_L) = \frac{N(L-1, D_1)}{N(L, D)} \times \frac{N(L-2, D_2)}{N(L-1, D_1)} \times \cdots \times \frac{N(1, D_{L-1})}{N(2, D_{L-2})}, \tag{5.3}
$$

where D_i is the residual displacement to be made after i steps. The probability of the path being taken is $1/N(L, D)$, since $N(1, D_{L-1}) = 1$.

At each step, if the target is hit, the "hit" table is updated by incrementing the counter tallying the number of times that the node is hit on the Jth step and adding the product of the matrix elements encountered in the walk to this node to a value table. After many walks are generated, the estimate for the value of K^J is obtain by looking up the number of times (H) the target was hit on the Jth step, the sum of the values of the matrix elements recorded (V) and the quantity $N(J, D)$. The stochastic estimator for K^J is then $(V/H)\,N(J, D)$.

If the walk should hit the point of origin, then the subsequent walk from the origin to the target also has information about the values of the matrix element desired. To use this information, one must open a register to accumulate the products of matrix elements encountered in the subsequent walk beginning at the revisited origin.

It remains to be shown that the stochastic matrix inversion method can be easily implemented in software and is competitive in efficiency and perhaps superior to other methods of large sparse matrix inversion. The next section gives an example of the use of this algorithm to solve a simple lattice problem.

6. Examples

The two algorithms for stochastic matrix inversion have been tested on simple problems—the computation of some internodal resistances of one- and two-dimensional networks of equal resistances. A formulation and solution of such problems

TABLE I

Computed results for $D = 1$ Network Using Algorithm A

Sample size	−2	−1	0	1	2
2000	3.16	1.98	0	1.85	3.18
2000	3.16	1.94	0	1.82	3.19
2000	3.12	1.86	0	1.75	3.12
Exact	4.00	2.00	0	2.00	4.00

TABLE II

Computed Results for $D = 2$ Network Using Algorithm A

Sample size	(0, 0)	(1, 1)	(2, 2)	(3, 3)	(4, 4)
300	0.00	0.59	0.83	0.90	0.97
300	0.00	0.63	0.82	0.91	0.95
300	0.00	0.62	0.83	0.91	0.98
Exact	0.00	0.64	0.85	0.98	1.07

TABLE III

Computed Results for $D = 2$ Network Using Algorithm B (50 Walks)

Steps per walk	1	2	3
6	0.598	0.718	0.742
10	0.611	0.756	0.800
16	0.619	0.795	0.848
20	0.622	0.795	0.867
24	0.625	0.803	0.882
Exact	0.637	0.849	0.976

TABLE IV

Number of Estimates for $(K^J)_{0,0}$

Length	2	4	6	8	10	12	14	16	18	20	22	24
6	30	27	50									
10	46	36	27	30	50							
16	53	33	18	23	18	22	31	50				
20	51	30	33	19	16	19	26	20	28	50		
24	59	33	30	27	19	23	15	18	16	16	29	50

by (untargetted) stochastic matrix inversion has already been given [8]. The basic computational problem is the inversion of the admittance matrix. The examination of this type of problem as a test for the algorithms was motivated by the similarity of the features of the matrix with those encountered in quantum field fermion simulations.

The algorithms were encoded in the C language and run on a microcomputer since the matrices to be inverted were effectively small. The code using algorithm A creates a lookup table for the weights given by Eq. (4.7), first for a one-dimensional network of resistances ($D=1$). For the $D=2$ case, considered next, the lookup table for the weights is computed from the formula

$$W(J)=\frac{(1/4)^J \times N(L, D_X, D_Y)}{N(L-J, 0, 0)}, \tag{6.1}$$

where $N(L, D_X, D_Y)$ is given by Eq. (B.4), L is the number of steps in the walk and J is the step on which the target is hit. A small number of paths were selected in the unbiased manner described in Appendix A. The results are reported in Tables I and II. Table I gives the values of resistances found for a $D=1$ linear network of 1 Ω resistances. The comparison of pairs of values which ideally are equal (at nodes which are symmetrically placed with respect to node 0) gives some idea of the random error incurred. The deviation computed from the exact results also includes systematic errors such as truncation of the Neumann series. Table II summarizes the results for a two-dimensional homogeneous resistive network of 1 Ω resistances. Only 500 paths were selected out of $\approx 5.49 \times 10^{226}$ different paths for the 50 step walk for the target $D_X=0$, $D_Y=0$. The results are accurate to within a few percent.

The same problem of a $D=2$ resistive network was attacked using Algorithm B. Fifty walks were taken in each case and L, the number of steps, was varied from 6 to 24. The results are presented in Table III. In addition, Tables IV and V have been compiled to show the number of estimates obtained for K^J for 50 walks. It should be noted just how efficient Algorithm B is in obtaining hits for estimates of K^J.

Since the numbers computed in all these examples depend on the difference of two large numbers, the errors tend to amplify especially in the $D=1$ case. The

TABLE V

Number of Estimates for $(K^J)_{0,3}$

Length	2	4	6	8	10	12	14	16	18	20	22	24
6			50									
10			7	20	50							
16			3	7	11	19	24	50				
20			1	6	4	9	9	9	18	50		
24			3	6	5	10	9	13	10	18	26	50

examples in this sense bring out sharply the errors in the methods. Even in such unfavorable circumstances, the estimates obtained are quite acceptable for many purposes.

7. Conclusions

The method of large matrix inversion by stochastic estimates has been developed to the stage which we believe is maximally efficient. For problems requiring specific inverse matrix elements of very large matrices to an accuracy of a few percent, the method presented should be considered as an alternative, probably superior, to the non-stochastic methods of inversion. We are currently applying the stochastic matrix inversion algorithms to a problem in quantum field theory involving interacting bosons and fermions.

APPENDIX A—Unbiased Selection of Paths

Suppose that a random walk on a two-dimensional square lattice is taken with L steps result in a displacement $D=(DX, DY)$. Haltering steps, those which loop back to the same node, are allowed in this example. The constraints on the number of positive steps in the X direction (XR), the number of negative steps in the X direction (XL), the number of steps in the positive Y direction (YU), the number of steps in the negative Y direction (YD), and the number of haltering steps (H) are

$$\begin{aligned} L &= XR + XL + YU + YD + H \\ DX &= XR - XL \\ DY &= YU - YD. \end{aligned} \tag{A.1}$$

The number of different paths of length L which undergo a displacement D is given by the multinomial coefficient

$$N(L, D) = \sum \frac{L!}{XR!\, XL!\, YU!\, YD!\, H!}, \tag{A.2}$$

where the sum is over all nonnegative values of XR, XL, YU, YD, and H which obey the constraints of Eq. (A.1).

Suppose that XL and YD are also specified. The number of paths with the specifications L, DX, DY, XL, and YD is given by

$$\frac{L!}{(DX+XL)!\, XL!(DY+YD)!\, YD!\, (L-DX-DY-2XL-2YD)!} \tag{A.3}$$

TABLE VI

Probability of a Path with a Specified (XL, YD)

(XL, YD)	Probability
(0, 0)	1/61
(0, 1)	12/61
(1, 0)	12/61
(1, 1)	24/61
(0, 2)	6/61
(2, 0)	6/61

These numbers may be looked upon as the relative probability for the targetted path having the characterizing values (XL, YD). The possible values of (XL, YD) with non-vanishing probability are those for which the arguments of the factorials in the denominator of Eq. (A.3) are non-negative. One must choose the value of (XL, YD) according to their relative probabilities.

For example, if $L=4$ and $D=(0, 0)$, there are 61 different possible paths. Table VI gives the probability of a path with a specified (XL, YD). In this example, it is of interest to note that there are 13, 25, 13, and 61 paths returning to the origin after one, two, three, and four steps, respectively. The number of untargetted walks with $L=4$ is 625.

Once (XL, YD) is chosen, the five values XR, XL, YU, YD, and H are fully determined by (XL, YD) along with L and (DX, DY). The walk can now be generated sequentially in an unbiased fashion using a generalization of Algorithm S found in Knuth [9]. Let z_i $(i=1, 2, 3, 4, 5)$ take on the values xr, xl, yu, yd, and h and be the number of steps of each type remaining after $L-l$ steps, where $l=xr+xl+yu+yd+h$ is the remaining number of steps. Then the probability of taking a step of type i is

$$p_i=\frac{z_i}{l}. \tag{A.4}$$

If the kind of step is determined sequentially by these probabilities then an unbiased targetted walk will be generated.

APPENDIX B—Counting the Number of Paths

The number of paths of length L which exhibit a displacement $D=(DX, DY)$ for a two-dimensional walk can be given by a closed analytic expression. A generating function for counting paths is

$$(e^{i\phi_1}+e^{-i\phi_1}+e^{i\phi_2}+e^{-i\phi_2})^L=\sum N(L, D)\, e^{iD\cdot\phi}, \tag{B.1}$$

where $\phi = (\phi_1, \phi_2)$. Hence

$$N(L, D) = \frac{1}{4\pi^2} \iint (e^{i\phi_1} + e^{-i\phi_1} + e^{i\phi_2} + e^{-i\phi_2})^L \, e^{-iD\cdot\phi} \, d\phi_1 \, d\phi_2. \tag{B.2}$$

The evaluation of the integral is easy if new variables are introduced:

$$\begin{aligned} \phi &= \tfrac{1}{2}(\phi_1 - \phi_2) \\ \Phi &= \tfrac{1}{2}(\phi_1 + \phi_2). \end{aligned} \tag{B.3}$$

The result of the integration is

$$\begin{aligned} &L!^2/(\tfrac{1}{2}(L - DX - DY))!\,(\tfrac{1}{2}(L + DX + DY))! \\ &/(\tfrac{1}{2}(L - DX + DY))!\,(\tfrac{1}{2}(L + DX - DY))! \end{aligned} \tag{B.4}$$

if the target can be reached by taking L steps. Otherwise, $N(L, D)$ is zero.

REFERENCES

1. *Science* **228**, 1297 (1985).
2. *Phys. Rev. Lett.* **49**, 183 (1982).
3. W. H. PRESS, B. FLANNERY, S. A. TEUKOLSKY, AND W. T. VETTERLING, *Numerical Recipes* (Cambridge Univ. Press, New York, 1986), Chap. 2.
4. T. A. DEGRAND, J. DREITLEIN, AND D. TOMS, *Phys. Lett. B* **120**, 391 (1983).
5. See J. M. HAMMERSLEY AND D. C. HANDSCOMB, *Monte Carlo Methods* (Wiley, New York, 1964).
6. V. PAN AND J. REIF, in *Proceedings, 17th Annual ACM Symposium in the Theory of Computing, 1985*, p. 143.
7. A. BEN-ISRAEL AND D. COHEN, *SIAM J. of Num. Anal.* **3**, 410 (1966).
8. J. DREITLEIN, *Phys. Rev. A* **31**, 1658 (1985).
9. D. KNUTH, *The Art of Computer Programming, Semi-Numerical Algorithms* (Addison-Wesley, Reading, MA, 1971), p. 122.