A Randomizing Butterfly Transformation Useful in Block Matrix Computations

D. Stott Parker
stott@cs.ucla.edu
Computer Science Department
University of California
Los Angeles, CA 90024-1596

July 12, 1995

Abstract

We present a new randomization scheme that preconditions an input linear system by multiplying it with nonsingular random matrices. At the expense of these multiplications, and making the linear system dense if it was not already, this approach makes blocks of the resulting system nonsingular with probability 1.

Specifically, in this paper we consider random matrices that are 'random butterflies', and call their application a *Random Butterfly Transformation* (RBT). RBTs can be performed efficiently, and in particular should be useful on parallel and vector machines with architectures that support FFT-like computations.

Block Gaussian elimination is an important example of an algorithm with which this RBT can be used. Gaussian elimination is complicated by *pivoting*, which handles degenerate matrices having zero elements on the diagonal. Pivoting can significantly complicate the algorithm, increase data movement, and reduce speed, particularly on high-performance computers. We show that this preconditioning has the effect of eliminating the need for pivoting (with probability 1).

The RBT approach also suggests many interesting developments beyond the elimination of pivoting. Randomization appears to offer new possibilities for linear systems solution that are well-suited to high-performance computers.

1 Motivation

The theory and practice in matrix computations differ in significant ways. For example, the emphases in theoretical texts (such as [9]) have diverged from those in pragmatic texts (such as [8]). The extent of this divergence stands in strong contrast with classical treatments such as [11], which tried to bridge both theory and practice. The realities of using modern computer architectures, finite precision arithmetic hardware, and existing programming languages and software libraries make much of this divergence understandable. Nevertheless, one wonders whether the two could be brought closer together.

One striking difference between theory and practice centers around block matrix computations. Theoretical discussions focus heavily on block decompositions, while practical discussions emphasize iterative, column-at-a-time computation.

For example, consider implementing a block-Gaussian Elimination step on square submatrices of a square matrix, i.e., on matrices divided into **quadrants**:

$$\begin{pmatrix} I & 0 \\ -CA^{-1} & I \end{pmatrix} \begin{pmatrix} A & B \\ C & D \end{pmatrix} = \begin{pmatrix} A & B \\ 0 & D - CA^{-1}B \end{pmatrix}.$$

A practitioner's brow will probably furrow upon seeing this formula. "You wouldn't actually do it that way. For example, what about the roundoff error in computing A^{-1}?" After assurances that we will not compute A^{-1} explicitly, but instead will compute $X = CA^{-1}$ by solving the linear system $A^\top Y = C^\top$, $X = Y^\top$, the brow will still be furrowed. "Still, the roundoff might be terrible. And what if A is singular?" These concerns encourage a retreat to conservative methods.

While recursive specification of matrix operations using this quadrant style is elegant, it is usually viewed as impractical. The block-Gaussian Elimination step above, for example, works only when the submatrix A is nonsingular, the upper left quadrants of both A and $D - CA^{-1}B$ are nonsingular, the upper left quadrants of each of these are nonsingular, etc. Although this recursive condition is known to hold for some important classes of input matrices, such as symmetric positive definite matrices, it certainly will not hold for all.

Resorting to a conservative column-at-a-time style is a pity, of course, since clearly a block-at-a-time style might yield greater parallelism. It is worthwhile considering alternatives. In [19], we reconsider the approach of Wise [28], who studies recursive algorithms on quadrant decompositions of the matrix. Although Wise's numerical analysis is naive, he develops recursive algorithms that can be run very effectively on some parallel computers.

The basic issue here is that matrix computations that are easy to state are complicated by *degeneracy*. Gaussian Elimination is a good case in point, as degeneracy requires it to include *pivoting*, which dynamically permutes rows and columns of the input matrix so that large (nonzero) matrix elements are moved to the diagonal for use in elimination. Pivoting is a necessity in practice, but it interrupts the algorithm's data flow, introduces data movement that can slow high-performance computers, and generally prevents recursion.

The communications overhead of pivoting can be significant on a parallel machine. A single pivot can take $O(n)$ steps on machines with limited interconnection schemes; in such a machine, the $O(n^2)$ data movements potentially required by pivoting, together with the synchronization they require, can actually take longer than the $O(n^3)$ arithmetic operations required by Gaussian Elimination. Significant overhead is also generated on both MIMD and SIMD machines. A number of authors have considered scheduling algorithms for Gaussian Elimination on MIMD machines [23]. For example, Bampis et al. [1] point out that MIMD overhead (communication and idle time) can run to $O(n^3)$, even when pivoting is ignored; Veldhorst [27] has considered the effect of partial pivoting. In [22], implementations of Gaussian Elimination with pivoting on SIMD hypercubes are detailed, and experiments show how the communications overhead grows as the number of processors increases.

There is an alternative to pivoting. The primary role of pivoting is to avoid degeneracies of the input matrix. Pivoting can therefore be replaced by any transformation of the matrix that avoids degeneracy. In this paper we develop recursive randomization algorithms that accomplish this. The algorithms are relatively efficient and are well-suited to many high-performance computers. While they do incur greater roundoff error than would result with pivoting, they work effectively on well-conditioned matrices, and provide a new perspective on connecting theory with practice.

2 Randomization as a Method to Avoid Degeneracy

The purpose of this paper is to show how artifacts of degeneracy like pivoting can be escaped with randomization, and in particular how block algorithms can be developed with very simple randomizing transforms. First, let us sketch the idea of randomizing linear transformations, and how they can be used. The rest of the paper shows that certain transformations, Random Butterfly Transformations, actually have the desired properties.

2.1 The Basic Idea

Let M be a nonsingular real $n \times n$ matrix, let U and V be nonsingular random $n \times n$ matrices. (By a random matrix, we mean a matrix whose elements are chosen independently from a suitable distribution. We can *guarantee* such matrices to be nonsingular by choosing them from a nonsingular class of matrices, like orthogonal matrices. More on this shortly.)

Our basic idea is that, given the randomness of U and V, *we can perform Gaussian Elimination on the product matrix UMV without any pivoting.* Intuitively UMV is sufficiently 'random' that, with probability 1, pivoting is never needed. After Gaussian Elimination we can undo the effects of U and V.

We will justify this idea formally later. Taking it on faith for the moment, notice that it permits many standard computations. For example, to solve $M\mathbf{x} = \mathbf{b}$, we can instead solve $(UMV)\mathbf{y} = U\mathbf{b}$. using Gaussian Elimination without pivoting. Afterwards, it follows that $\mathbf{x} = V\mathbf{y}$.

Putting it another way, Gaussian Elimination needs pivoting only for degenerate matrices: nonsingular matrices that happen to yield zeroes on the diagonal. In the space of matrices, degenerate matrices have measure zero: almost every matrix is nondegenerate. So, we transform any problem involving a possibly degenerate matrix M into a problem involving a (randomized, almost certainly) nondegenerate matrix UMV, and then transform back (derandomize) the resulting solution. In this paper, we consider block Gaussian elimination, in which the blocks are (recursively) quadrants of the matrix. So the randomization need only (recursively) guarantee nonsingularity of these quadrants.

2.2 Example: Using Randomization instead of Pivoting

To illustrate the basic idea, consider solving the linear system $M\mathbf{x} = \mathbf{b}$ where

$$M = \begin{pmatrix} 0 & 2 \\ 1 & 0 \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} 2 \\ 3 \end{pmatrix}$$

with randomized Gaussian elimination. With the random matrices

$$U = \begin{pmatrix} +.6226 & +.5170 \\ +.6226 & -.5170 \end{pmatrix}, \qquad V = \begin{pmatrix} +.5377 & +.6759 \\ +.5377 & -.6759 \end{pmatrix}$$

we obtain the randomized problem $\quad U\,M\,V\,\mathbf{y} = U\,\mathbf{b}, \quad$ where

$$\begin{aligned} UMV &= \begin{pmatrix} +.6226 & +.5170 \\ +.6226 & -.5170 \end{pmatrix} \begin{pmatrix} 0 & 2 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} +.5377 & +.6759 \\ +.5377 & -.6759 \end{pmatrix} &= \begin{pmatrix} 1.004 & -.4207 \\ .2780 & -1.048 \end{pmatrix} \\ U\,\mathbf{b} &= \begin{pmatrix} +.6226 & +.5170 \\ +.6226 & -.5170 \end{pmatrix} \begin{pmatrix} 2 \\ 3 \end{pmatrix} &= \begin{pmatrix} 3.113 \\ -.517 \end{pmatrix}. \end{aligned}$$

Gaussian elimination can be applied directly to this problem without any use of pivoting, and gives

$$\mathbf{y} \quad = \quad \begin{pmatrix} 3.721 \\ 1.480 \end{pmatrix},$$

so the solution to $M\mathbf{x} = \mathbf{b}$ is

$$\mathbf{x} \quad = \quad V\,\mathbf{y} \quad = \quad \begin{pmatrix} 3.001 \\ 1.001 \end{pmatrix}$$

— correct to three digits.

This technique always works, with probability 1. Why? As we will show, if the input matrix is nonsingular, the determinant of any quadrant of the randomized matrix is a polynomial of random values, and, moreover, with probability 1, this polynomial is not identically zero (and the quadrant is nonsingular). Block algorithms on quadrant submatrices therefore work correctly.

In fact, this approach will succeed with almost any nonsingular random transformations U and V. The trick, however, is to use 'sparse' random transformations that always work, are numerically stable, and are computationally efficient. The butterfly matrices discussed below achieve all three criteria.

3 Preliminaries

3.1 Submatrices

Definition 1 *The* **submatrix of A with rows $i_1, \ldots, i_p$ and columns** $j_1, \ldots, j_q$ *(in that order) is denoted $A[\, i_1, \ldots, i_p \,|\, j_1, \ldots, j_q \,]$, and consists of the indicated elements of A:*

$$A[\, i_1, \ldots, i_p \,|\, j_1, \ldots, j_q \,] \quad = \quad \begin{pmatrix} a_{i_1,j_1} & a_{i_1,j_2} & \cdots & a_{i_1,j_q} \\ a_{i_2,j_1} & a_{i_2,j_2} & \cdots & a_{i_2,j_q} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i_p,j_1} & a_{i_p,j_2} & \cdots & a_{i_p,j_q} \end{pmatrix}.$$

Henceforth, we assume matrix row and column indexing begins with 1.

For example, the submatrix of A with rows 3, 2 and columns 1, 3, 1, 4 is

$$A[\,3,2\,|\,1,3,1,4\,] \quad = \quad \begin{pmatrix} a_{31} & a_{33} & a_{31} & a_{34} \\ a_{21} & a_{23} & a_{21} & a_{24} \end{pmatrix}.$$

We also follow the convention that, when $p = 0$ or $q = 0$, the submatrix is 1. So,

$$A[\;|\;] \quad = \quad 1.$$

Another notation for submatrices will be useful later:

Definition 2 *Given a $(2n) \times (2n)$ matrix $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$, define the $n \times n$* **submatrix of M with signature** $\delta_1, \ldots, \delta_n; \epsilon_1, \ldots, \epsilon_n$ *by*

$$M\langle\, \delta_1, \ldots, \delta_n \,;\, \epsilon_1, \ldots, \epsilon_n \,\rangle \quad = \quad \left(m_{ij}^{\delta_i,\epsilon_j} \right)$$

where

$$m_{ij}^{\delta_i,\epsilon_j} \quad = \quad \begin{cases} a_{ij} & \text{if } \delta_i = 0,\ \epsilon_j = 0 \\ b_{ij} & \text{if } \delta_i = 0,\ \epsilon_j = 1 \\ c_{ij} & \text{if } \delta_i = 1,\ \epsilon_j = 0 \\ d_{ij} & \text{if } \delta_i = 1,\ \epsilon_j = 1. \end{cases}$$

Example 1 *If $n = 2$ and*

$$M \quad = \quad \begin{pmatrix} a_{11} & a_{12} & b_{11} & b_{12} \\ a_{21} & a_{22} & b_{21} & b_{22} \\ c_{11} & c_{12} & d_{11} & d_{12} \\ c_{21} & c_{22} & d_{21} & d_{22} \end{pmatrix}$$

then

$$M\langle\, 0,0\,;\,1,0 \,\rangle \quad = \quad \begin{pmatrix} b_{11} & a_{12} \\ b_{21} & a_{22} \end{pmatrix}, \qquad M\langle\, 0,1\,;\,1,0 \,\rangle \quad = \quad \begin{pmatrix} b_{11} & a_{12} \\ d_{21} & c_{22} \end{pmatrix}.$$

$M\langle\, \delta_1, \ldots, \delta_n \,;\, \epsilon_1, \ldots, \epsilon_n \,\rangle$ is the submatrix of M selecting only the rows $(1 + n\,\delta_1)$, …, $(n + n\,\delta_n)$ and columns $(1 + n\,\epsilon_1)$, …, $(n + n\,\epsilon_n)$, assuming that rows and columns of M are numbered from 1 to $2n$. In other words,

$$M\langle\, \delta_1, \ldots, \delta_n \,;\, \epsilon_1, \ldots, \epsilon_n \,\rangle \quad = \quad M[\,(1+n\,\delta_1), \ldots, (n+n\,\delta_n) \,|\, (1+n\,\epsilon_1), \ldots, (n+n\,\epsilon_n)\,].$$

So $M\langle\, 0,0 \,;\, 1,0 \,\rangle = M[\,1,2\,|\,3,2\,]$, while $M\langle\, 0,1 \,;\, 1,0 \,\rangle = M[\,1,4\,|\,3,2\,]$.

3.2 Unitary Transformations

Definition 3 *Recall that a* **unitary matrix** *is a complex square matrix U for which its* **Hermitian adjoint** *(conjugate transpose) U^* is its inverse: $UU^* = U^*U = I$. For example, the matrix*

$$O_{2n} \quad = \quad \frac{1}{\sqrt{2}} \begin{pmatrix} I_n & I_n \\ I_n & -I_n \end{pmatrix}$$

is unitary. Also, if Θ is any real diagonal matrix, then $exp(\mathrm{i}\Theta)$ is unitary.

A **unitary transformation** *of a square matrix M is any product*

$$U^* \; M \; V$$

where U and V are unitary matrices.

Unitary transformations have important computational properties that we will review later.

3.3 Random Matrices

Definition 4 *A* **random real index** *θ is a real-valued integrable function $\theta : \mathcal{D} \rightarrow \mathcal{R}$ with a strictly increasing (absolutely) continuous probability distribution on a real interval sample space $\mathcal{D}$ of nonzero measure.*

A **random value** *r is a 1-to-1 function of a random real index θ. Here we use complex-valued functions like $exp(\mathrm{i}\theta)$, and insist they be integrable with respect to the probability distribution of θ.*
A **nonzero random value** *is a random value whose function never takes the value zero.*
A **random matrix** *is a matrix whose complex-valued entries are functions of independent random values sharing a common distribution.*

If r is a random value, a property $P(r)$ **holds with probability 1** *when $P(r)$ is true on all of $\mathcal{D}$ except possibly a subset of measure 0. (Equivalently, $P(r)$* **holds almost surely**, *or $P(r)$* **holds almost everywhere**.*)* [1]

For example, a diagonal matrix Θ whose entries are random real indices θ with a continuous distribution on the interval $[0, 2\pi)$ is a random matrix. The random diagonal matrix with diagonal entries $exp(\mathrm{i}\,\theta)$, a function of a random index θ with continuous distribution on the interval $[0, 2\pi)$, is always nonsingular and unitary.

[1] To support implementation with floating point numbers, we obviously need the requirement that the dimension n of any matrix be 'small' relative to the number of floating point numbers available. If there are 2^w possible floating point numbers, clearly $n \ll 2^w$ is necessary for the matrices to be considered random in the sense intended here. Furthermore, the phrase "with probability 1" actually must be understood as relative to floating point numbers. If floating point numbers have t-bit precision, then the probability that two randomly-chosen floating point numbers are equal is 2^{-t}, and the phrase must be replaced by "with probability $1 - O(2^{-t})$". See the remarks after Theorem 1.

3.4 Randomness and Zeroes of Polynomials

Lemma 1 *Suppose $p(x) = ax + b$, where a, b are fixed complex values. Let r be a nonzero random value. With probability 1, if $p(r) = 0$ then $a = b = 0$.*

Proof
Assume fixed values a, b, and that $p(r) = 0$ for nonzero r. Thus it must be the case that either $a = b = 0$, or $a \neq 0$, $b \neq 0$, and $r = -b/a$. Because r is a random value, however, $r \neq -b/a$ with probability 1. □

Theorem 1 *Suppose $p(x_1, \ldots, x_n)$ is a fixed multivariate polynomial in n variables, in which each variable appears with degree at most 1. Let $r_1, \ldots, r_n$ be independently chosen nonzero random values. With probability 1, if $p(r_1, \ldots, r_n) = 0$ then $p = 0$.*

Proof
By induction on n. Lemma 1 establishes the base case, where $n = 1$. For $n > 1$ we can collect terms in x_n and write

$$p(x_1, \ldots, x_n) \quad = \quad a(x_1, \ldots, x_{n-1})\; x_n \; + \; b(x_1, \ldots, x_{n-1})$$

where a and b are polynomials in which variables appear with degree at most one. Suppose $p(r_1, \ldots, r_n) = 0$, for independent nonzero random r_i. Then by the previous lemma, with probability 1 $a(r_1, \ldots, r_{n-1}) = b(r_1, \ldots, r_{n-1}) = 0$, so by induction $a = b = 0$, and $p = 0$. □

When floating point numbers are used in computing these polynomials, the probability in Lemma 1 can decline to a value less than 1. For example, the probability that two randomly chosen t-bit floating-point numbers differ is $1 - 2^{-t}$. In this case Theorem 1 would find $p = 0$ with probability $(1 - 2^{-t})^n = 1 - n\,2^{-t} + O(2^{-2t})$. This is still "with probability $1 - O(2^{-t})$". Since they are distracting and are not likely to be significant (most computers today support $t \geq 56$), we omit these $O(2^{-t})$ terms from now on, but the phrase "with probability 1" should be treated with care.

4 Quadrant Randomizing Transformations

In this section we identify matrices U and V such that, given any nonsingular matrix M, the transform

$$U^* \; M \; V$$

has the property that its quadrants are all nonsingular, with probability 1. Specifically, we prove theorems like the following:

> **Theorem** If $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ is a $(2n) \times (2n)$ nonsingular matrix, and R, S are nonsingular independent random diagonal $n \times n$ matrices, then with probability 1 each quadrant in the matrix
>
> $$\begin{aligned} \widetilde{M} \quad &= \quad \begin{pmatrix} \widetilde{A} & \widetilde{B} \\ \widetilde{C} & \widetilde{D} \end{pmatrix} \quad = \quad \begin{pmatrix} I & R \\ I & -R \end{pmatrix} \begin{pmatrix} A & B \\ C & D \end{pmatrix} \begin{pmatrix} I & I \\ S & -S \end{pmatrix} \\ &= \quad \begin{pmatrix} A + BS + RC + RDS & A - BS + RC - RDS \\ A + BS - RC - RDS & A - BS - RC + RDS \end{pmatrix} \end{aligned}$$
>
> is nonsingular.

The basic idea of the proof is to show that the quadrants must be nonsingular because their determinants are nonzero, with probability 1. Each such determinant is a polynomial in the diagonal elements of the matrices R, S. Because the original matrix M is nonsingular, the polynomial is not identically zero. Therefore, by Theorem 1, with probability 1 the value of the determinant polynomial is nonzero, and with probability 1 each quadrant is nonsingular.

The point is that matrix computations can be broken into blocks by recursive application of this transformation. Since the transformation is linear, it can be used in solving linear systems by mapping them to (nonsingular) randomized linear systems, solving their block subsystems, and then mapping back. Specifically, linear systems involving each of the quadrants ($\widetilde{A}$, $\widetilde{B}$, $\widetilde{C}$, $\widetilde{D}$) of $\widetilde{M}$ can be recursively solved since they are nonsingular.

4.1 Random Butterfly Transformations

We define a class of matrices that make computationally efficient transformations.

Definition 5 *A* **butterfly matrix** *will be defined here as any nonsingular* $(2n) \times (2n)$ *matrix*

$$\begin{pmatrix} P & Q \\ R & S \end{pmatrix}$$

where P, Q, R, *and* S *are diagonal* $n \times n$ *matrices.*

A **random butterfly transformation (RBT)** *of a matrix* M *is any product*

$$U^* \; M \; V$$

where U *and* V *are random butterfly matrices.*

In this paper, we focus first on RBTs in which U and V are butterflies of the form

$$\begin{pmatrix} R^0 & R^0 \\ R^1 & -R^1 \end{pmatrix} \;=\; \begin{pmatrix} R^0 & 0 \\ 0 & R^1 \end{pmatrix} \begin{pmatrix} I_n & I_n \\ I_n & -I_n \end{pmatrix} \;=\; (R^0 \,\oplus\, R^1) \;\sqrt{2}\, O_{2n}$$

where R^0 and R^1 are diagonal nonsingular $n \times n$ matrices. These matrices are unitary under a variety of conditions, such as that $\sqrt{2}R^0$ and $\sqrt{2}R^1$ are both unitary (i.e., both have the form $exp(\mathrm{i}\Theta)$ for real diagonal Θ). Later we relax this focus and consider arbitrary random butterflies.

4.2 Butterfly Transformations and Polynomials

Theorem 2 *Let* R^0, R^1, S^0, S^1 *be independent nonsingular diagonal* $n \times n$ *matrices*

$$R^0 = diag(r_1^0, \ldots, r_n^0), \;\; R^1 = diag(r_1^1, \ldots, r_n^1), \;\; S^0 = diag(s_1^0, \ldots, s_n^0), \;\; S^1 = diag(s_1^1, \ldots, s_n^1).$$

and $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$, $U^* = \begin{pmatrix} R^0 & R^1 \\ R^0 & -R^1 \end{pmatrix}$, $V = \begin{pmatrix} S^0 & S^0 \\ S^1 & -S^1 \end{pmatrix}$. *Then the first quadrant in the transformation* $\widetilde{M} = U^*MV$ *is* $\widetilde{A} = R^0AS^0 + R^0BS^1 + R^1CS^0 + R^1DS^1$, *and*

$$det\,\widetilde{A} \;=\; \sum_{\delta_1=0}^{1} \cdots \sum_{\delta_n=0}^{1} \sum_{\epsilon_1=0}^{1} \cdots \sum_{\epsilon_n=0}^{1} \; det\, M\langle\, \delta_1, \ldots, \delta_n \,;\, \epsilon_1, \ldots, \epsilon_n \,\rangle \; r_1^{\delta_1} \cdots r_n^{\delta_n}\, s_1^{\epsilon_1} \cdots s_n^{\epsilon_n}.$$

Proof Letting σ range over permutations of $\{1,\ldots,n\}$, the determinant is

$$\begin{aligned}
det\ \widetilde{A} &= \sum_{\sigma} (-1)^{\sigma} \prod_{i=1}^{n} (r_i^0\, a_{i,\sigma(i)}\, s_{\sigma(i)}^0 + r_i^0\, b_{i,\sigma(i)}\, s_{\sigma(i)}^1 + r_i^1\, c_{i,\sigma(i)}\, s_{\sigma(i)}^0 + r_i^1\, d_{i,\sigma(i)}\, s_{\sigma(i)}^1) \\
&= \sum_{\sigma} (-1)^{\sigma} \prod_{i=1}^{n} \left(\sum_{\delta=0}^{1} \sum_{\epsilon=0}^{1} r_i^{\delta}\ m_{i,\sigma(i)}^{\delta,\epsilon}\ s_{\sigma(i)}^{\epsilon} \right) \\
&= \sum_{\delta_1=0}^{1} \cdots \sum_{\delta_n=0}^{1} \sum_{\epsilon_1=0}^{1} \cdots \sum_{\epsilon_n=0}^{1} \ \sum_{\sigma} (-1)^{\sigma} \prod_{i=1}^{n} m_{i,\sigma(i)}^{\delta_i,\epsilon_{\sigma(i)}}\ \ r_i^{\delta_i}\ s_{\sigma(i)}^{\epsilon_{\sigma(i)}} \\
&= \sum_{\delta_1=0}^{1} \cdots \sum_{\delta_n=0}^{1} \sum_{\epsilon_1=0}^{1} \cdots \sum_{\epsilon_n=0}^{1} det\ M\langle\, \delta_1,\ldots,\delta_n\ ;\ \epsilon_1,\ldots,\epsilon_n \,\rangle\ r_1^{\delta_1}\cdots r_n^{\delta_n}\ s_1^{\epsilon_1}\cdots s_n^{\epsilon_n}. \qquad \square
\end{aligned}$$

Theorem 2 shows $det\ \widetilde{A}$ is a multivariate polynomial in the variables r_1^0, r_1^1, …, r_n^0, r_n^1, s_1^0, s_1^1, …, s_n^0, s_n^1. The degree of each variable is at most 1, and the coefficient of $r_1^{\delta_1}\cdots r_n^{\delta_n}\, s_1^{\epsilon_1}\cdots s_n^{\epsilon_n}$ is $det\ M\langle\, \delta_1,\ldots,\delta_n\ ;\ \epsilon_1,\ldots,\epsilon_n \,\rangle$. An important corollary results when $R^0 = S^0 = I$:

Theorem 3 *Let $R = diag(r_1,\ldots,r_n)$, $S = diag(s_1,\ldots,s_n)$, $\widetilde{A} = A + BS + RC + RDS$. Then*

$$det\ \widetilde{A} \;=\; \sum_{\delta_1=0}^{1} \cdots \sum_{\delta_n=0}^{1} \sum_{\epsilon_1=0}^{1} \cdots \sum_{\epsilon_n=0}^{1} det\ M\langle\, \delta_1,\ldots,\delta_n\ ;\ \epsilon_1,\ldots,\epsilon_n \,\rangle\ (r_1)^{\delta_1}\cdots(r_n)^{\delta_n}(s_1)^{\epsilon_1}\cdots(s_n)^{\epsilon_n}.$$

Example 2 *With $n = 2$, we obtain*

$$M \;=\; \left(\begin{array}{cc|cc} a_{11} & a_{12} & b_{11} & b_{12} \\ a_{21} & a_{22} & b_{21} & b_{22} \\ \hline c_{11} & c_{12} & d_{11} & d_{12} \\ c_{21} & c_{22} & d_{21} & d_{22} \end{array}\right), \qquad \widetilde{A} \;=\; \begin{pmatrix} a_{11} + b_{11}\, s_1 + c_{11}\, r_1 + d_{11}\, r_1\, s_1 & a_{12} + b_{12}\, s_2 + c_{12}\, r_1 + d_{12}\, r_1\, s_2 \\ a_{21} + b_{21}\, s_1 + c_{21}\, r_2 + d_{21}\, r_2\, s_1 & a_{22} + b_{22}\, s_2 + c_{22}\, r_2 + d_{22}\, r_2\, s_2 \end{pmatrix}$$

$$\begin{aligned}
det\ \widetilde{A} = {} & (a_{11}\, a_{22} - a_{12}\, a_{21}) \\
& + (a_{22}\, c_{11} - a_{21}\, c_{12})\, r_1 + (a_{11}\, c_{22} - a_{12}\, c_{21})\, r_2 + (a_{22}\, b_{11} - a_{12}\, b_{21})\, s_1 + (a_{11}\, b_{22} - a_{21}\, b_{12})\, s_2 \\
& + (c_{11}\, c_{22} - c_{12}\, c_{21})\, r_1\, r_2 + (a_{22}\, d_{11} - b_{21}\, c_{12})\, r_1\, s_1 + (c_{11}\, b_{22} - a_{21}\, d_{12})\, r_1\, s_2 \\
& + (b_{11}\, c_{22} - a_{12}\, d_{21})\, r_2\, s_1 + (a_{11}\, d_{22} - b_{12}\, c_{21})\, r_2\, s_2 + (b_{11}\, b_{22} - b_{12}\, b_{21})\, s_1\, s_2 \\
& + (c_{22}\, d_{11} - c_{12}\, d_{21})\, r_1\, r_2\, s_1 + (c_{11}\, d_{22} - c_{21}\, d_{12})\, r_1\, r_2\, s_2 \\
& + (b_{22}\, d_{11} - b_{21}\, d_{12})\, r_1\, s_1\, s_2 + (b_{11}\, d_{22} - b_{12}\, d_{21})\, r_2\, s_1\, s_2 \\
& + (d_{11}\, d_{22} - d_{12}\, d_{21})\, r_1\, r_2\, s_1\, s_2.
\end{aligned}$$

This determinant is a polynomial in r_1, r_2, s_1, s_2, in which the maximum degree of each variable is 1. Each coefficient in the determinant polynomial is the determinant of the 2×2 submatrix $M\langle\, \delta_1, \delta_2\ ;\ \epsilon_1, \epsilon_2 \,\rangle$. For example, the coefficient of $r_2\, s_1 = (r_1)^0(r_2)^1(s_1)^1(s_2)^0$ is det $M\langle\, 0, 1\ ;\ 1, 0 \,\rangle = (b_{11}c_{22} - a_{12}d_{21})$.

4.3 Semi-Nonsingularity

Definition 6 *A $(2n) \times (2n)$ matrix M is said to be* **semi-nonsingular** *if at least one of the 4^n $n \times n$ submatrices $M\langle\, \delta_1,\ldots,\delta_n\ ;\ \epsilon_1,\ldots,\epsilon_n \,\rangle$ is nonsingular.*

Virtually all $(2n) \times (2n)$ matrices are semi-nonsingular. For example, when $n = 1$ only the zero matrix fails to be semi-nonsingular. With $n > 1$ it is challenging to construct a nontrivial example of a matrix that fails to be semi-nonsingular. For $n = 2$ only peculiar matrices like

$$M \;=\; \begin{pmatrix} t & u & v & w \\ tx/w & ux/w & vx/w & x \\ ty/w & uy/w & vy/w & y \\ tz/w & uz/w & vz/w & z \end{pmatrix}$$

are not semi-nonsingular — and actually every 4×4 matrix that is not semi-nonsingular is necessarily singular. Failing to be semi-nonsingular ultimately forces a matrix to be singular:

Theorem 4 *Every $(2n) \times (2n)$ nonsingular matrix is also semi-nonsingular.*

Proof
The idea of the proof is that every nonsingular matrix M has a decomposition $M = LUP$, where P is a $(2n) \times (2n)$ permutation matrix, and using P we can identify a nonsingular submatrix $M\langle\, \delta_1, \ldots, \delta_n \,;\, \epsilon_1, \ldots, \epsilon_n \,\rangle$. Thus M is also semi-nonsingular. The proof uses graph-theoretic tools common in combinatorics [2].

When M is nonsingular, an LUP decomposition can be obtained for it in a standard way as follows. First, delete the $(2n)$-th row from M; the result is a $(2n-1) \times (2n)$ matrix of rank $(2n-1)$. From this matrix, we can then delete a column j_{2n} to obtain a $(2n-1) \times (2n-1)$ nonsingular matrix. Next, repeat this process of deleting row i and some associated column j_i for each value of i from $(2n-1)$ down to 1, each time obtaining an $(i-1) \times (i-1)$ nonsingular matrix. The result of this deletion process is a sequence of values $j_{2n}, \ldots, j_2, j_1$ such that if we permute the columns of M so that j_1 is first, j_2 second, ..., and j_{2n} last, then each leading principal submatrix (submatrix with rows and columns $1, \ldots, k$) in the permuted matrix is nonsingular.

This property is sufficient to permit us to decompose the permuted matrix into LU form. Let P be the permutation such that $P(i) = j_i$, which is represented by the $2n \times 2n$ permutation matrix $P = (\delta_{j_i,i})$. Then $MP^{-1} = MP^{\top}$ is M with the columns permuted as just described. So if $MP^{-1} = LU$, then $M = LUP$, and $(MP^{-1})[\,1, \ldots, k \,|\, 1, \ldots, k\,]$ is nonsingular for $1 \le k \le 2n$.

Define $(i \operatorname{Mod} n) \stackrel{def}{=} ((i-1) \bmod n) + 1$. Corresponding to P we can now construct a bipartite graph R with two partitions of n vertices each, and $2n$ undirected edges (possibly with duplicate edges), such that vertex $(i \operatorname{Mod} n) \in \{1, \ldots, n\}$ in the left partition is connected to vertex $(j_i \operatorname{Mod} n) \in \{1, \ldots, n\}$ in the right partition whenever $P(i) = j_i$, In other words, each edge $(i \operatorname{Mod} n)$ — $(j_i \operatorname{Mod} n)$ corresponds to a pair $(i, P(i)) = (i, j_i)$. Also, each vertex x in the left partition corresponds to the two matrix rows $\{x, x+n\}$, and each vertex y in the right partition corresponds to the two matrix columns $\{y, y+n\}$. This graph is thus **Eulerian** since every vertex in the graph has 2 incident edges (we permit duplicate edges). Therefore, for any m, each subset X of m vertices in the left partition is connected to at least m vertices $R(X)$ in the right partition. The **Hall condition** $|\,R(X)\,| \ge |\,X\,|$ is necessary and sufficient for there to be a **matching** in the graph: a set of n edges such that every vertex in the graph is incident upon exactly one edge. (This is Hall's famous theorem on Systems of Distinct Representatives; see e.g. [2].)

Not only does the graph have a matching, but because the graph is Eulerian, we can construct a matching simply by forming an arbitrary tour through (each connected component of) the graph, and including every other edge on the tour in the matching.

Construct any matching for this graph. The matching then corresponds to a set of n pairs $(i_1, P(i_1))$, $(i_2, P(i_2))$, ..., $(i_n, P(i_n))$. In each pair i_k is either k or $k+n$, and by construction there are no two distinct values k, k' such that $P(i_k) = P(i_{k'})$ or $P(i_k) = P(i_{k'}) + n$.

We claim the submatrix $M[\,i_1, \ldots, i_n \,|\, P(i_1), \ldots, P(i_n)\,]$ is nonsingular. It is the submatrix of MP^{-1} resulting from deleting row i and column i, for each i not in the set $\{i_1, \ldots, i_n\}$. In other words,

$$M[\,i_1, \ldots, i_n \,|\, P(i_1), \ldots, P(i_n)\,] \quad = \quad (MP^{-1})[\,i_1, \ldots, i_n \,|\, i_1, \ldots, i_n\,].$$

MP^{-1} is nonsingular since it has an LU decomposition. But if $X = LU$ is an LU decomposition of a matrix X, the submatrix obtained by deleting the i-th row and column of X is exactly the product of the submatrices of L and of U obtained by deleting their i-th rows and columns. So this submatrix has an LU decomposition, and is also nonsingular.

Finally, for $1 \le k \le n$, put $\delta_k = \lfloor (i_k - 1)/n \rfloor$ and $\epsilon_k = \lfloor (P(i_k) - 1)/n \rfloor = \lfloor (j_{i_k} - 1)/n \rfloor$. Then the nonsingular submatrix just obtained is $M\langle\, \delta_1, \ldots, \delta_n \,;\, \epsilon_1, \ldots, \epsilon_n \,\rangle$. $\Box$

Example 3 *Consider the permutation matrix*

$$M \quad = \quad \left(\begin{array}{cccc|cccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right).$$

Permutation matrices force a particular selection of P: we must choose $P = M$ in order for MP^{-1} to be diagonal. The bipartite graph for this matrix P is then as in Figure 1. Suppose that we select

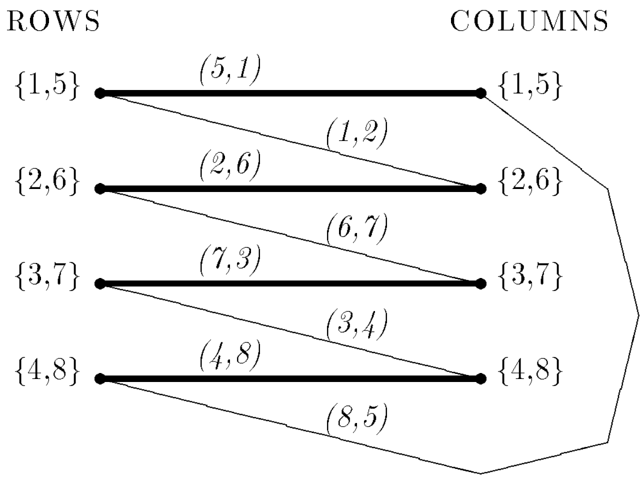

Figure 1: Bipartite graph corresponding to the permutation P

the horizontal edges as a matching, as indicated with darker edges in the figure. These edges are labeled with the pairs (5,1), (2,6), (7,3), (4,8). Then the submatrix

$$M[5,2,7,4 \mid 1,6,3,8] \quad = \quad M\langle\, 1,0,1,0\,;\, 0,1,0,1\,\rangle \quad = \quad \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

is nonsingular as claimed.

4.4 Quadrant Randomizing Transformations

Theorem 5 *Let $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ be a nonsingular $(2n) \times (2n)$ matrix, and R, S be nonsingular independent random diagonal $n \times n$ matrices. With probability 1, each quadrant in the matrix*

$$\widetilde{M} \;=\; \begin{pmatrix} \widetilde{A} & \widetilde{B} \\ \widetilde{C} & \widetilde{D} \end{pmatrix} \;=\; \begin{pmatrix} I & R \\ I & -R \end{pmatrix}\begin{pmatrix} A & B \\ C & D \end{pmatrix}\begin{pmatrix} I & I \\ S & -S \end{pmatrix}$$

is nonsingular.

Proof
The upper left quadrant $\widetilde{A}$ of the transform

$$\widetilde{M} \;=\; \begin{pmatrix} \widetilde{A} & \widetilde{B} \\ \widetilde{C} & \widetilde{D} \end{pmatrix} \;=\; \begin{pmatrix} A+BS+RC+RDS & A-BS+RC-RDS \\ A+BS-RC-RDS & A-BS-RC+RDS \end{pmatrix}$$

has determinant $det\,\widetilde{A} = det\,(A+BS+RC+RDS)$, which from Theorem 3 we know is a polynomial in the diagonal elements r_i, s_i. Since M is semi-nonsingular by Theorem 4, at least one of its submatrices $M\langle\, \delta_1,\ldots,\delta_n\,;\,\epsilon_1,\ldots,\epsilon_n\,\rangle$ has a nonzero determinant, and the polynomial is not identically zero. By Theorem 1, since the polynomial $det\,\widetilde{A}$ is not identically zero, when evaluated with random values r_i and s_i it is nonzero with probability 1. Thus $\widetilde{A} = A+BS+RC+RDS$ is nonsingular with probability 1. A similar argument will show that each of the other quadrants is nonsingular with probability 1, since altering the signs of the entries in R and S will not otherwise affect the argument. □

The theorem says that $\widetilde{A}$ is nonsingular with probability 1, that is, regardless of the nonsingular input matrix M, randomly chosen R and S will virtually guarantee that the inverse of $\widetilde{A}$ exists. (The theorem does not say anything about the fact that singular matrices have measure 0 in the space of all matrices.)

Example 4 *When $n = 1$,*

$$\widetilde{M} \;=\; \begin{pmatrix} 1 & r \\ 1 & -r \end{pmatrix}\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} 1 & 1 \\ s & -s \end{pmatrix} \;=\; \begin{pmatrix} a+bs+cr+drs & a-bs+cr-drs \\ a+bs-cr-drs & a-bs-cr+drs \end{pmatrix}$$

for scalars a, b, c, d and random values r, s. The determinant of the first quadrant of this matrix is $a+bs+cr+drs$, and is identically zero iff $a=b=c=d=0$.

With $n = 2$, let us consider a specific semi-nonsingular matrix. For example, with

$$M \;=\; \begin{pmatrix} A & B \\ C & D \end{pmatrix} \;=\; \left(\begin{array}{cc|cc} 2 & 0 & 0 & 0 \\ 0 & 0 & 5 & 0 \\ \hline 0 & 3 & 0 & 0 \\ 0 & 0 & 0 & 7 \end{array}\right)$$

and $R = \begin{pmatrix} r_1 & 0 \\ 0 & r_2 \end{pmatrix}$, $S = \begin{pmatrix} s_1 & 0 \\ 0 & s_2 \end{pmatrix}$, *each of $A+RC$, $A+BS$, $B+RD$, $C+DS$ is singular, but* $\widetilde{A} = A+BS+RC+RDS = \begin{pmatrix} 2 & 3r_1 \\ 5s_1 & 7r_2s_2 \end{pmatrix}$, *is nonsingular with probability 1, as its determinant* $14r_2s_2 - 15r_1s_1$ *is nonzero with probability 1.*

One might suspect that for very sparse matrices M like permutation matrices, one or more of the quadrants of $\widetilde{M}$ could be singular. This is not so. For example, given the permutation matrix

$$M \;=\; \begin{pmatrix} A & B \\ C & D \end{pmatrix} \;=\; \left(\begin{array}{cccc|cccc} 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{array}\right),$$

the upper left quadrant $\widetilde{A} = A + BS + RC + RDS$ *satisfies*

$$det\ \widetilde{A} \quad = \quad det \begin{pmatrix} r_1 & 1 & 0 & 0 \\ 0 & s_2 & r_2\, s_3 & 0 \\ 0 & 0 & r_3 & 1 \\ r_4\, s_1 & 0 & 0 & s_4 \end{pmatrix} \quad = \quad r_1\, r_3\, s_2\, s_4 - r_2\, r_4\, s_1\, s_3.$$

Since M *is semi-nonsingular (for example, the submatrix* $M\langle\, 1,0,1,0\,;\,0,1,0,1\,\rangle$ *is nonsingular),* $det\,\widetilde{A}$ *cannot be zero.*

4.5 Generalization to Arbitrary Butterflies

The theorems above prove random butterfly matrices give an effective quadrant randomizing transform with random butterfly matrices of the form

$$\begin{pmatrix} I & I \\ R & -R \end{pmatrix}.$$

These theorems can be readily generalized to hold for **random arbitrary butterfly matrices**

$$\begin{pmatrix} P & Q \\ R & S \end{pmatrix}$$

where P, Q, R, S are (nonsingular) diagonal random matrices. Specifically, Theorems 2 and 5 can be generalized for arbitrary butterflies:

Theorem 6 *Let* M *be a nonsingular* $(2n) \times (2n)$ *matrix. With probability 1, each quadrant in the Random Butterfly Transform* $\widetilde{M} = U^*\, M\, V$ *is nonsingular when* U *and* V *are independent random arbitrary butterfly matrices.*

Proof
Let $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$, $U^* = \begin{pmatrix} R^0 & R^1 \\ \cdots & \cdots \end{pmatrix}$, $V = \begin{pmatrix} S^0 & \cdots \\ S^1 & \cdots \end{pmatrix}$, with nonsingular random diagonal matrices R^0, R^1, S^0, S^1. Then $\widetilde{A} = R^0 A S^0 + R^0 B S^1 + R^1 C S^0 + R^1 D S^1$ is the first quadrant of $\widetilde{M} = U^* M V$, just as in Theorem 2. If we define $R = (R^0)^{-1} R^1$, $S = S^1 (S^0)^{-1}$, then both R and S are nonsingular random diagonal matrices, and $(R^0)^{-1} \widetilde{A} (S^0)^{-1} \;=\; A + BS + RC + RDS$. This is identical to the quadrant expression in Theorems 3 and 5, which can be used to show the first quadrant of $\widetilde{M}$ is nonsingular, because $det\,\widetilde{A} \;=\; det\,(R^0)\, det\,(S^0)\, det\,(A + BS + RC + RDS) \;\neq\; 0$. A similar argument can be used for the remaining quadrants. □

It is important in this theorem that U and V be independent. We cannot just let $U = V$, and then restrict our RBTs to similarity transformations $\widetilde{M} = U^* M U = U^{-1} M U$. Unfortunately, this tempting simpler transformation loses the properties found earlier: Theorem 3 fails to hold when $U = V$. For example, if $U = V = \begin{pmatrix} I & I \\ R & -R \end{pmatrix}$ where

$$R \quad = \quad \exp(\mathrm{i}\,\Theta) \quad = \quad \begin{pmatrix} \exp(\mathrm{i}\,\theta_1) & 0 \\ 0 & \exp(\mathrm{i}\,\theta_2) \end{pmatrix}$$

and

$$M \quad = \quad \begin{pmatrix} A & B \\ C & D \end{pmatrix} \quad = \quad \left(\begin{array}{cc|cc} 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ \hline 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{array}\right),$$

then — *regardless of the values of* θ_1, θ_2 — the first quadrant of U^*MU is

$$\widetilde{A} \quad = \quad (A + BR + R^*C + R^*DR) \quad = \quad \begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix},$$

which is *singular*. (The result can be singular whenever $B = C = 0$ and both A and D are permutation matrices, for any value of n.) On the other hand, if we pick the butterfly matrices $U = \begin{pmatrix} I & I \\ R & -R \end{pmatrix}$, $V = \begin{pmatrix} I & I \\ S & -S \end{pmatrix}$, where $S = \exp(\mathrm{i}\,\Theta')$, the first quadrant of U^*MV is

$$\widetilde{A} \quad = \quad (A + BS + R^*C + R^*DS) \quad = \quad \begin{pmatrix} \exp(\mathrm{i}\,(\theta_1' - \theta_1)) & 1 \\ 1 & \exp(\mathrm{i}\,(\theta_2' - \theta_2)) \end{pmatrix},$$

which is nonsingular with probability 1. It is essential to keep U and V independent.

Theorem 6 is significant, since it shows that the RBT can be used with *unitary* random arbitrary butterfly matrices. If Θ is a random real diagonal matrix, then

$$\frac{1}{\sqrt{2}} \begin{pmatrix} I & \exp(\mathrm{i}\Theta) \\ I & -\exp(\mathrm{i}\Theta) \end{pmatrix}, \qquad \frac{1}{\sqrt{2}} \begin{pmatrix} I & \exp(\mathrm{i}\Theta) \\ \exp(-\mathrm{i}\Theta) & -I \end{pmatrix}$$

and the random reflection and rotation matrices

$$\begin{pmatrix} \cos(\Theta) & \sin(\Theta) \\ \sin(\Theta) & -\cos(\Theta) \end{pmatrix}, \qquad \begin{pmatrix} \cos(\Theta) & -\sin(\Theta) \\ \sin(\Theta) & \cos(\Theta) \end{pmatrix}$$

are all unitary random arbitrary butterfly matrices. (The small example in Section 2.2 actually uses random reflection matrices for U and V.) As mentioned earlier, unitary matrices give very well-behaved linear transforms, and are important in practical implementations of the RBT.

5 Implementation of a Block RBT Linear System Solver

5.1 A Randomized Recursive Linear System Solver

We can obtain a 'block LU' factorization of any nonsingular $(2n) \times (2n)$ matrix A as

$$\begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \quad = \quad \begin{pmatrix} A_{00} & 0 \\ A_{10} & I \end{pmatrix} \begin{pmatrix} I & \mathcal{A}_{01} \\ 0 & \mathcal{A}_{11} \end{pmatrix}$$

provided the first quadrant A_{00} is also nonsingular, by solving

$$\begin{aligned} A_{00}\ \mathcal{A}_{01} &= A_{01} \\ \mathcal{A}_{11} &= A_{11} \ - \ A_{10}\ \mathcal{A}_{01}. \end{aligned}$$

Solving the first of these equations requires solution of a block linear system of half the size. With this factorization, the $(2n) \times (2n)$ matrix-matrix block linear system

$$\begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \begin{pmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{pmatrix} \quad = \quad \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix}$$

can be solved recursively, by reducing the problem to the pair of block triangular linear systems

$$\begin{pmatrix} A_{00} & 0 \\ A_{10} & I \end{pmatrix} \begin{pmatrix} \mathcal{B}_{00} & \mathcal{B}_{01} \\ \mathcal{B}_{10} & \mathcal{B}_{11} \end{pmatrix} = \begin{pmatrix} B_{00} & B_{01} \\ B_{10} & B_{11} \end{pmatrix} \qquad \begin{pmatrix} I & \mathcal{A}_{01} \\ 0 & \mathcal{A}_{11} \end{pmatrix} \begin{pmatrix} X_{00} & X_{01} \\ X_{10} & X_{11} \end{pmatrix} = \begin{pmatrix} \mathcal{B}_{00} & \mathcal{B}_{01} \\ \mathcal{B}_{10} & \mathcal{B}_{11} \end{pmatrix}$$

which simplify to solution of the following 'back-solution' equations:

$$\begin{array}{rcl} A_{00}\ \mathcal{B}_{00} &=& B_{00} \\ A_{00}\ \mathcal{B}_{01} &=& B_{01} \\ \mathcal{B}_{10} &=& B_{10} - A_{10}\ \mathcal{B}_{00} \\ \mathcal{B}_{11} &=& B_{11} - A_{10}\ \mathcal{B}_{01} \end{array} \qquad \begin{array}{rcl} \mathcal{A}_{11}\ X_{10} &=& \mathcal{B}_{10} \\ \mathcal{A}_{11}\ X_{11} &=& \mathcal{B}_{11} \\ X_{00} &=& \mathcal{B}_{00} - \mathcal{A}_{01}\ X_{10} \\ X_{01} &=& \mathcal{B}_{01} - \mathcal{A}_{01}\ X_{11}. \end{array}$$

Using the same approach, we can solve the $(2n) \times (2n)$ matrix-vector block linear system

$$\begin{pmatrix} A_{00} & A_{01} \\ A_{10} & A_{11} \end{pmatrix} \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \end{pmatrix}$$

by reducing it to the pair of block triangular linear systems

$$\begin{pmatrix} A_{00} & 0 \\ A_{10} & I \end{pmatrix} \begin{pmatrix} \boldsymbol{\beta}_0 \\ \boldsymbol{\beta}_1 \end{pmatrix} = \begin{pmatrix} \mathbf{b}_0 \\ \mathbf{b}_1 \end{pmatrix} \qquad \begin{pmatrix} I & \mathcal{A}_{01} \\ 0 & \mathcal{A}_{11} \end{pmatrix} \begin{pmatrix} \mathbf{x}_0 \\ \mathbf{x}_1 \end{pmatrix} = \begin{pmatrix} \boldsymbol{\beta}_0 \\ \boldsymbol{\beta}_1 \end{pmatrix}$$

which can be solved by recursively solving the following equations:

$$\begin{array}{rcl} A_{00}\ \boldsymbol{\beta}_0 &=& \mathbf{b}_0 \\ \boldsymbol{\beta}_1 &=& \mathbf{b}_1 - A_{10}\ \boldsymbol{\beta}_0 \end{array} \qquad \begin{array}{rcl} \mathcal{A}_{11}\ \mathbf{x}_1 &=& \boldsymbol{\beta}_1 \\ \mathbf{x}_0 &=& \boldsymbol{\beta}_0 - \mathcal{A}_{01}\ \mathbf{x}_1. \end{array}$$

Accumulating these equations leads to the following recursive algorithms, when n is a power of 2:

to solve $A\mathbf{x} = \mathbf{b}$:

$$\begin{array}{l} \text{solve } A_{00}\ \mathcal{A}_{01} = A_{01} \\ \text{solve } A_{00}\ \boldsymbol{\beta}_0 = \mathbf{b}_0 \\ \mathcal{A}_{11} := A_{11} - A_{10}\,\mathcal{A}_{01} \\ \boldsymbol{\beta}_1 := \mathbf{b}_1 - A_{10}\ \boldsymbol{\beta}_0 \\ \text{solve } \mathcal{A}_{11}\ \mathbf{x}_1 = \boldsymbol{\beta}_1 \\ \mathbf{x}_0 := \boldsymbol{\beta}_0 - \mathcal{A}_{01}\ \mathbf{x}_1. \end{array}$$

to solve $AX = B$:

$$\begin{array}{l} \text{solve } A_{00}\ \mathcal{A}_{01} = A_{01} \\ \text{solve } A_{00}\ \mathcal{B}_{00} = B_{00} \\ \text{solve } A_{00}\ \mathcal{B}_{01} = B_{01} \\ \mathcal{B}_{10} := B_{10} - A_{10}\ \mathcal{B}_{00} \\ \mathcal{B}_{11} := B_{11} - A_{10}\ \mathcal{B}_{01} \\ \mathcal{A}_{11} := A_{11} - A_{10}\ \mathcal{A}_{01} \\ \text{solve } \mathcal{A}_{11}\ X_{10} = \mathcal{B}_{10} \\ \text{solve } \mathcal{A}_{11}\ X_{11} = \mathcal{B}_{11} \\ X_{00} := \mathcal{B}_{00} - \mathcal{A}_{01}\ X_{10} \\ X_{01} := \mathcal{B}_{01} - \mathcal{A}_{01}\ X_{11}. \end{array}$$

We can terminate both recursions for 1×1 matrices with $x = b/a$.

Assuming we use a matrix multiplication algorithm with complexity $M(n) = n^3$ and a matrix-vector multiplication algorithm with complexity $V(n) = n^2$, the time bound recurrences for solving square $n \times n$ matrix-matrix equations ($AX = B$) and $n \times n$ matrix-vector equations ($A\mathbf{x} = \mathbf{b}$) when n is a power of 2 are

$$\begin{array}{rcl} T_{\text{matrix-matrix}}(n) &=& 5\,T_{\text{matrix-matrix}}(n/2) + 5\,M(n/2) \\ T_{\text{matrix-vector}}(n) &=& T_{\text{matrix-matrix}}(n/2) + M(n/2) + 2\,T_{\text{matrix-vector}}(n/2) + 2\,V(n/2) \end{array}$$

where both recursions terminate with $T(1) = 1$. These have solutions

$$\begin{array}{rcl} T_{\text{matrix-matrix}}(n) &=& \frac{5}{3}\,n^3 + O(n^{\log_2(5)}) \\ T_{\text{matrix-vector}}(n) &=& \frac{4}{9}\,n^3 + O(n^{\log_2(5)}). \end{array}$$

The small constant factors in these complexity figures show that from the standpoint of performance the recursive algorithms can be competitive with established methods. Of course an improved matrix multiplication algorithm will only make them look better.

The algorithm above does not explicitly handle singular quadrants A_{00} of the matrix A. This is where randomization comes in. If the recursive algorithms above randomize the A matrix with the RBT upon entry, and derandomize the solution (X or $\mathbf{x}$) upon exit, the problem with singularity will be avoided (with probability 1). Notice $\mathcal{A}_{11} = A_{11} - A_{10}A_{00}^{-1}A_{01}$ (the **Schur complement** of A with respect to A_{00} [17]) is nonsingular because $det\,A = det\,A_{00}\,det\,\mathcal{A}_{11}$. The additional overhead of recursively performing this randomization is only $O(n^2 \log n)$, which does not significantly affect the complexity of the algorithm, although it is greater than the $O(n^2)$ overhead of partial pivoting.

5.2 Implementation

To test the concepts presented above, we implemented the algorithm above in FORTRAN 77 and ran it on a Sun Sparcstation 20 at UCLA. Results are tabulated in Figure 2 for 11 different classes of matrices, including five random classes in [25] and six classes that have known exact solutions (the first 3 of which appear in [12]):

Matrix class	Problem: $A\mathbf{x} = \mathbf{b}$	Exact solution $\mathbf{x}$
`normal`	normal random a_{ij} and b_i ($\mu = 0,\ \sigma = 1$)	
`[-1,1]`	uniform random a_{ij} and b_i, drawn from $[-1, 1]$	
`[0,1]`	uniform random a_{ij} and b_i, drawn from $[0, 1]$	
`{-1,1}`	discrete random a_{ij} and b_i ($pr(-1) = pr(1) = 0.5$)	
`{0,1}`	discrete random a_{ij} and b_i ($pr(0) = pr(1) = 0.5$)	
`|i-j|`	$a_{ij} = \|i-j\|, \quad b_i = 1$	$\mathbf{x} = (\frac{1}{n-1}, 0, \ldots, 0, \frac{1}{n-1})$
`max(i,j)`	$a_{ij} = max(i,j), \quad b_i = i$	$\mathbf{x} = (1, 0, \ldots, 0)$
`C(i+j,j)`	$a_{ij} = \begin{pmatrix} i+j-2 \\ j-1 \end{pmatrix}, \quad b_i = 1$	$\mathbf{x} = (1, 0, \ldots, 0)$
`Hadamard`	$A = \bigotimes^{\log_2(n)} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \quad b_i = 1$	$\mathbf{x} = (1, 0, \ldots, 0)$
`Permute`	A is a random permutation, $\quad (A^{-1}b)_i = i$	$\mathbf{x} = (1, 2, 3, \ldots, n)$
`Turing`	$a_{ij} = \left\{ \begin{array}{cc} 0 & i < j \\ 1 & i = j \\ -1 & i > j \end{array} \right\}, \quad b_i = 1$ (Wilkinson's test)	$\mathbf{x} = (1, 2, 4, \ldots, 2^{n-1})$

The 'differences' and 'errors' are the indicated average or maximal differences of the x_i values, where errors are normalized by the max norm of the computed RBT $\mathbf{x}$ vector. They give a kind of relative error.

These results are for sequential computation, and are not optimized at all. However, they show that the approach is already usable, and gives numerical results comparable in quality to that of LINPACK's `dgefa/dgesl`. Rounding errors do grow rapidly with the problem size n, and are a definite concern. The final columns in the table give averages and maxima of 'relative errors'

$$\frac{|\,b_i^{RBT} \;-\; b_i\,|}{\|\,\mathbf{b}^{RBT}\,\|_\infty}$$

where the b_i values are either the computed solution from LINPACK, or the exact solution. The ill-conditioned problems `C(i+j,j)` and `Turing` have respective condition numbers 2^{4n} (spectral norm) and $n\,2^{n-1}$ (row sum norm). Edelman [4] has shown that normally distributed random matrices are very well-behaved, with expected spectral norm approximately $\sqrt{2n}$, so it is not surprising that the RBT gives respectable results for these matrices.

Matrix Class	Matrix Rank	Good Runs	RBT avg sec	LINPACK avg sec	avg Error RBT-LINPACK	max Error RBT-LINPACK	avg Error RBT-exact	max Error RBT-exact
normal	32	256	0.0348	0.0339	0.00D+00	0.00D+00		
[-1,1]	32	256	0.0344	0.0345	0.00D+00	0.00D+00		
[0,1]	32	256	0.0343	0.0338	0.00D+00	0.00D+00		
{-1,1}	32	256	0.0338	0.0336	0.00D+00	0.00D+00		
{0,1}	32	256	0.0332	0.0331	0.00D+00	0.00D+00		
|i-j|	32	256	0.0398	0.0394	0.00D+00	0.00D+00	4.01D-15	1.03D-14
max(i,j)	32	256	0.0521	0.0519	0.00D+00	0.00D+00	0.00D+00	0.00D+00
C(i+j,j)	32	256	0.0324	0.0318	0.00D+00	0.00D+00	2.73D-02	8.75D-01
Hadamard	32	256	0.0248	0.0245	0.00D+00	0.00D+00	0.00D+00	0.00D+00
Permute	32	256	0.0160	0.0154	0.00D+00	0.00D+00	0.00D+00	0.00D+00
Turing	32	256	0.0142	0.0137	0.00D+00	0.00D+00	0.00D+00	0.00D+00
normal	64	128	0.3486	0.2489	5.02D-14	5.14D-12		
[-1,1]	64	128	0.3478	0.2490	1.13D-13	1.11D-11		
[0,1]	64	128	0.3497	0.2506	5.34D-14	9.13D-12		
{-1,1}	64	128	0.3505	0.2405	9.43D-14	2.01D-11		
{0,1}	64	128	0.3603	0.2378	2.63D-14	2.40D-12		
|i-j|	64	128	0.3527	0.2809	1.02D-14	3.36D-14	7.04D-15	3.36D-14
max(i,j)	64	128	0.3539	0.3668	1.22D-14	3.39D-14	1.22D-14	3.39D-14
C(i+j,j)	64	128	0.3542	0.2435	2.79D-13	2.15D-12	1.37D-02	8.75D-01
Hadamard	64	128	0.3665	0.1342	3.17D-17	3.33D-16	3.17D-17	3.33D-16
Permute	64	128	0.3739	0.0594	2.15D-13	4.28D-10	2.15D-13	4.28D-10
Turing	64	128	0.3691	0.0511	2.87D+01	9.20D+02	2.87D+01	9.20D+02
normal	128	64	2.7582	1.8309	1.53D-12	9.91D-11		
[-1,1]	128	64	2.7580	1.8321	1.76D-12	2.22D-10		
[0,1]	128	64	2.7607	1.8341	1.97D-12	3.51D-10		
{-1,1}	128	64	2.7632	1.7840	9.66D-13	3.40D-11		
{0,1}	128	64	2.8124	1.7781	7.88D-13	4.16D-11		
|i-j|	128	64	2.7583	2.0176	3.88D-14	1.36D-13	1.98D-14	9.25D-14
max(i,j)	128	64	2.7623	2.7174	5.30D-14	1.56D-13	5.30D-14	1.56D-13
C(i+j,j)	128	64	2.7637	1.8017	1.16D-08	1.52D-07	6.84D-03	8.75D-01
Hadamard	128	64	2.8064	0.7399	4.98D-17	4.72D-16	4.98D-17	4.72D-16
Permute	128	64	3.3132	0.2297	3.12D-13	1.25D-11	3.12D-13	1.25D-11
Turing	128	64	2.8141	0.1961	3.58D+20	2.29D+22	3.58D+20	2.29D+22
normal	256	32	22.7859	14.4842	8.84D-10	9.93D-08		
[-1,1]	256	32	22.8084	14.4721	1.95D-10	2.03D-08		
[0,1]	256	32	22.5521	14.3256	1.60D-10	1.49D-08		
{-1,1}	256	32	22.5440	14.0975	2.77D-11	3.06D-09		
{0,1}	256	32	22.8136	14.0919	5.99D-11	3.19D-09		
|i-j|	256	32	22.4093	15.3754	8.64D-14	3.33D-13	5.43D-14	2.18D-13
max(i,j)	256	32	22.2184	20.6891	1.11D-13	5.75D-13	1.11D-13	5.75D-13
C(i+j,j)	256	32	22.4988	14.1470	5.26D-02	3.61D+00	6.38D-02	6.48D+00
Hadamard	256	32	22.7675	4.2462	5.99D-17	6.66D-16	5.99D-17	6.66D-16
Permute	256	32	27.5203	0.9067	3.03D-12	1.44D-10	3.03D-12	1.44D-10
Turing	256	32	22.7016	0.7635	1.84D+58	2.36D+60	1.84D+58	2.36D+60
normal	512	20	257.0444	123.1675	4.34D-10	2.50D-08		
[-1,1]	512	20	258.0085	123.4554	1.36D-10	5.43D-09		
[0,1]	512	20	256.6910	122.6309	1.74D-10	1.10D-08		
{-1,1}	512	20	257.8989	121.8171	7.54D-09	4.99D-07		
{0,1}	512	20	258.9514	121.4883	3.21D-10	1.69D-08		
|i-j|	512	20	257.9167	129.1167	3.29D-13	2.31D-12	1.37D-13	5.50D-13
max(i,j)	512	20	256.7432	165.0134	3.10D-13	1.27D-12	3.10D-13	1.27D-12
C(i+j,j)	512	20	257.3357	123.7720	1.73D+13	8.85D+15	3.24D+13	1.66D+16
Hadamard	512	20	257.6069	25.6450	7.10D-17	8.14D-16	7.10D-17	8.14D-16
Permute	512	20	279.3145	3.8960	2.45D-11	1.07D-09	2.45D-11	1.07D-09
Turing	512	20	267.4214	3.4297	9.94+134	2.54+137	9.94+134	2.54+137

Figure 2: Results of simple implementation: Block RBT vs. LINPACK's `dgefa/dgesl`

Since FORTRAN 77 does not support recursion, the program does its own recursion stack maintenance. The algorithm does have structure that helps in implementation: all equations above take the two forms

$$\begin{aligned} A\mathcal{X} &= B \\ \mathcal{X} &= X - AB, \end{aligned}$$

where X is not referenced further after $\mathcal{X}$ is defined. These can be implemented as destructive assignments overwriting the value of X with $\mathcal{X}$, although the recursion requires saving prior values of A_{01} and A_{11}.

The current implementation cuts off recursion for systems of size 32 or less, and relies on the LINPACK `dgefa/dgesl` routines at that point. This technique reduces the overhead of recursion — stack maintenance, saving variables and arrays from overwriting, etc. It also tends to reduce roundoff error, since partial pivoting stabilizes Gaussian elimination. The optimal cutoff point will depend on the environment. The table entries for $n = 32$ show the overhead in the RBT implementation. The major overhead of the implementation is random number generation, which dominates the execution time for small n. Currently we use the `zufall` random number generation package [21]. The RBT transformations used are Hadamard-like random reflection matrices with angles $\theta = (1 + r/100)\,\pi/4$, where r is a uniform random index ranging over $[-1/2, 1/2]$. Different transformations would have different properties, but these matrices are unitary and, as we show in the next section, unitary butterflies have good numerical properties.

The current implementation is sequential, but we are developing implementations that will exploit the algorithm's ample parallelism.

5.3 Error Analysis

We can extract the essence of the algorithm's stability properties with a simple argument. For a more detailed error analysis of Gaussian Elimination using the RBT, see [18].

It is a classical result [6, 8] that backwards error bounds are derivable for Gaussian elimination (with or without pivoting) that are proportional to the *growth factor*

$$\rho(A) \quad = \quad \frac{\max_{ijk} \; |\widehat{a}_{ij}^{(k)}|}{\| \; A \; \|_\infty}$$

where the quantities $\widehat{a}_{ij}^{(k)}$ are the iterates of the A matrix computed by Gaussian elimination. Only weak bounds are known for the growth factor for both partial pivoting and complete pivoting, and for partial pivoting these grow exponentially in the rank of A.

In [17] we present explicit formulas for the results of Gaussian Elimination, noting that the $a_{ij}^{(k)}$ are always ratios of determinants of submatrices of A. Specifically,

$$a_{ij}^{(k+1)} \quad = \quad \frac{det\ A[\,1,\ldots,k,\ i\,|\,1,\ldots,k,\ j\,]}{det\ A[\,1,\ldots,k\,|\,1,\ldots,k\,].}$$

Thus the growth factor is formally determined not by the condition number $\kappa(A)$, but by the near-degeneracy of submatrices of A. If no submatrices of A are degenerate, then the growth factor of A is small.

This result and the discussion preceding it allow us to resolve the basic issues in error analysis here, when the RBT uses unitary random butterflies, as in the implementation above:

- *Randomization with unitary butterflies cannot increase the problem condition number.*
  It is well-known [9] that **unitarily equivalent norms** of a matrix, such as the spectral norm

$\| \cdot \|_2$, are invariant under unitary transformations. Thus $\| U^* M V \|_2 = \| M \|_2$ when U and V are unitary. Consequently the spectral condition number

$$\kappa_2(M) \quad = \quad \| M \|_2 \, \| M^{-1} \|_2$$

is also invariant under unitary transformations: when the RBT uses unitary butterflies U and V, then since $\kappa_2(U^*AV) = \kappa_2(A)$, the original matrix A has exactly the same condition number as its RBT.

- *Randomization with unitary butterflies itself cannot produce a large E matrix.*
  When the RBT uses unitary butterflies, it is numerically stable. To see this, if a transformed matrix U^*AV has error E, then

  $$(U^* A V) + E \quad = \quad (U^* A V) + (U^* F V) \quad = \quad U^*(A+F)V$$

  and the resulting error $F = UEV^*$ has (unitarily equivalent) norm $\| F \| = \| E \|$, of the same magnitude as the error obtained by Gaussian elimination on the original problem.

- *Randomization can change a problem's growth factor.*
  Randomization does modify the submatrices of the input matrix, and thus can modify its growth factor. With the test matrices above, the randomized growth factors seem very much like those of the original. In fact the growth factor after randomization with the RBT becomes a ratio of averages of determinants of submatrices of the original matrix — essentially an average of the growth factors obtained over all possible pivoting sequences (good and bad) on the original matrix [18]. There remains an interesting question: *can we choose the RBT matrices U^* and V so that the growth factor of U^*AV is small?* This is currently open, and is discussed further in [18]. The techniques of [17] also might succeed in obtaining explicit formulas for error bounds.

The results of the tests with the block algorithm above are consistent with this perspective.

The block algorithm above is not exactly Gaussian elimination, and so does not compute exactly $\widehat{a}_{ij}^{(k+1)}$ (though it does compute something like them). A thorough error analysis for the algorithm will require giving bounds on the computed partial results. However, the algorithm above was not so much intended for its specifics as for its representativeness. It is consistent with the expectation from the sketchy analysis above that the RBT approach is less stable than partial pivoting. It also is consistent with the expectation that for problems whose growth factors (submatrix condition numbers) are small, the RBT gives acceptable error.

6 Concluding Remarks

The Random Butterfly Transformation (RBT) pre- and post-multiplies a given matrix with nonsingular random butterfly matrices. In this paper we have described the use of RBTs in transforming nonsingular matrices into nonsingular matrices with nonsingular quadrants. This transformation has various applications in computational linear algebra. A noteworthy possibility is the development of new Gaussian elimination algorithms that operate without pivoting.

We have shown that RBTs have three important properties. First, they can be applied very efficiently, particularly on parallel and vector machines with architectures that support FFT-like computations. When unitary butterflies are used, RBTs can also be inverted easily. Second, RBTs are benign numerically, although their use in conjuction with other algorithms (like block Gaussian elimination) does not guarantee numerical stability. Third, and most important, they guarantee that the quadrants of the resulting randomized matrix are nonsingular (with probability 1).

The RBT approach also suggests other interesting avenues for work. Randomization appears to offer new strategies for linear systems solution that are well-suited to high-performance computers. High-performance machines tend to favor algorithmic simplicity. The RBT trades naive time complexity (the number of 'steps' performed by a sequential algorithm) for algorithmic simplicity, and this may give better performance.

Acknowledgement

I am indebted to Brad Pierce for a very careful reading of the paper, and many insightful suggestions for improving it.

References

[1] E. Bampis, J.C. Konig, D. Trystram, "Impact of communications on the complexity of the Parallel Gaussian Elimination", *Parallel Computing* **17**:1, 55–61, 1991.

[2] B. Bollobás, *Combinatorics: Set Systems, Hypergraphs, Families of Vectors, and Combinatorial Probability*, NY: Cambridge University Press, 1986.

[3] J.R. Bunch, J. Hopcroft, "Triangular Factorization and Inversion by Fast Matrix Multiplication", *Math. of Computation* **28**:125, 231–236, 1974.

[4] A. Edelman, "Eigenvalues and Condition Numbers of Random Matrices", *SIAM J. Matrix Anal. Appl.* **9**:4, 543–560, 1988.

[5] D.K. Fadeev, V.N. Fadeeva, *Computational Methods of Linear Algebra*, San Francisco, W.H. Freeman, 1963.

[6] G.E. Forsythe, C.B. Moler, *Computer Solution of Linear Algebraic Systems*, Prentice-Hall, 1967.

[7] V.L. Girko, *Theory of Random Determinants*, Boston: Kluwer Academic Publishers, 1990.

[8] G.H. Golub, C.F. Van Loan, *Matrix Computations: Second Edition*, Baltimore: Johns Hopkins University Press, 1989.

[9] R.A. Horn, C.R. Johnson, *Matrix Analysis*, NY: Cambridge University Press, 1985; reprinted 1992.

[10] R.A. Horn, C.R. Johnson, *Topics in Matrix Analysis*, NY: Cambridge University Press, 1991.

[11] A.S. Householder, *The Theory of Matrices in Numerical Analysis*, NY: Dover Books, 1964.

[12] R.T. Gregory, D.L. Karney, *A Collection of Matrices for Testing Computational Algorithms*, Wiley-Interscience, 1969.

[13] A.W. Marshall, I. Olkin, *Inequalities: Theory of Majorization and Its Applications*, NY: Academic Press, 1979.

[14] M. Marcus, H. Minc, *A Survey of Matrix Theory and Matrix Inequalities*, NY: Dover, 1964.

[15] J. von Neumann, H.H. Goldstine, "Numerical inverting of matrices of high order", *Bull. Amer. Math. Soc.* **53**, pp. 1021–1099, 1947.

[16] D.S. Parker, "Notes on Shuffle/Exchange-Type Switching Networks", *IEEE Trans. Comput.* **C-29:3**, 213-222 (March 1980).

[17] D.S. Parker, "Explicit Formulas for the Results of Gaussian Elimination", Technical Report CSD-950025, UCLA Computer Science Department, 1995.

[18] D.S. Parker, "Random Butterfly Transformations with applications in Computational Linear Algebra", Technical Report CSD-950023, UCLA Computer Science Department, 1995.

[19] D.S. Parker, D. Lê, "How to Eliminate Pivoting from Gaussian Elimination — by Randomizing Instead", Technical Report CSD-950022, UCLA Computer Science Department, 1995.

[20] D.S. Parker, D. Lê, "Quadtree Matrix Algorithms Revisited: Basic Issues and their Resolution", Technical Report CSD-950028, UCLA Computer Science Department, 1995.

[21] W.P. Petersen, "Lagged Fibonacci series random number generators for the NEC SX-3", *International Journal of High Speed Computing*, **6**:3, 387–398, 1994.
`http://www.netlib.org/random/zufall.f`

[22] F.F. Rivera, R. Doallo, J.D. Bruguera, E.L. Zapata, R. Peskin, "Gaussian Elimination with Pivoting on Hypercubes", *Parallel Computing* **14**:1, 51–60, 1990.

[23] Y. Robert, *The Impact of Vector and Parallel Architectures on the Gaussian Elimination Algorithm*, NY: Halsted Press, 1990.

[24] G.W. Stewart, *Introduction to Matrix Computations*, Academic Press, 1973.

[25] L.N. Trefethen, R.S. Schreiber, "Average-Case Stability of Gaussian Elimination", *SIAM J. Matrix Anal. Appl.* **11**:3, pp. 335–360, July 1990.

[26] C. Van Loan, *Computational Frameworks for the Fast Fourier Transform*, Philadelphia, PA: SIAM Press, 1992.

[27] M. Veldhorst, "Gaussian Elimination with Partial Pivoting on an MIMD computer", *J. Parallel Distr. Computing* **6**, 62–68, 1989.

[28] David S. Wise, Matrix Algebra and Applicative Programming, in Gilles Kahn (ed.), *Functional Programming Languages and Computer Architecture* (Proceedings), Portland, Oregon, USA, Springer LNCS # 274, pp. 134-153, September 1987.