$\mathbb{F}_2$ Lanczos revisited

Michael Peterson, Chris Monico

Department of Mathematics and Statistics
Texas Tech University
e-mail: cmonico@math.ttu.edu

Draft of December 8, 2006

Abstract

We present a new variant of the block Lanczos algorithm for finding vectors in the kernel of a symmetric matrix over $\mathbb{F}_2$. Our algorithm is at least as efficient as that of Peter Montgomery [6], while the sequence of matrices W_i constructed here have different algebraic properties that may be useful in eventually providing a provable upper bound on the time required to solve this problem. Namely, our W_i satisfy $W_i^T W_j = 0$ for $i \neq j$ as opposed to $W_i^T A W_j = 0$ in [6].

1 Introduction

The problem of solving large sparse systems of linear equations over $\mathbb{F}_2$ arises in several situations. Our primary motivation here is specifically to solve linear homogeneous systems $B\boldsymbol{x} = 0$ such as those arising from integer factorization algorithms like the Number Field Sieve [1]. In most of what follows, we will assume A to be symmetric and $A\boldsymbol{x} = 0$ the problem to be solved. But note that if B is not symmetric, one may take $A = B^T B$ and under suitable hypothesis, recover vectors in the kernel of B from vectors in the kernel of A.

For a matrix B over $\mathbb{F}_2$ which is, say $500,000 \times 501,000$, using Gaussian elimination to solve $B\boldsymbol{x} = 0$ is unacceptable to say the least. Even if such a matrix B were sparse, after several pivoting operations it would quickly become dense requiring on the order of 30 GB of storage. Since Gaussian elimination over $\mathbb{F}_2$ is an $O(n^3)$ algorithm for $n \times n$ matrices, the runtime situation is no better than the storage. There are variants of Gaussian Elimination, like the 'structured Gaussian Elimination' of [2] which perform better in these instances, but runtime and storage problems still persist at even slightly larger sizes.

Don Coppersmith [3] and Peter Montgomery [6] have already given Block Lanczos variants which work well over $\mathbb{F}_2$. The big advantage of Lanczos methods in the current situation is that the given matrix is used only in a 'black-box' form. That is, no operations are ever performed on the matrix itself. Instead, the matrix is simply applied to various vectors and calculations are carried out on the results. So if we begin with a large but sparse matrix, the total storage requirements are not much more than those required to store the sparse matrix in a form which is convenient for computing its product on vectors (small block matrices, in fact). Furthermore, the Lanczos variants offer a big runtime advantage over Gaussian Elimination in the case where the given matrix is sparse.

Between [3] and [6], Montgomery's algorithm seems to be more efficient in practice, while Coppersmith's retains hints of geometric motivation. Our goal here is to combine the ideas of both of these papers into an algorithm which is both geometrically motivated and efficient. In fact, the runtime of our algorithm is essentially identical to Montgomery's. But the sequence of subspaces produced in our algorithm is decidedly different, and satisfies different structural identities (the sequence of subspaces produced in [6] are *pairwise $A-$orthogonal*, while the subspaces produced in this paper are *pairwise orthogonal*). Perhaps these structural differences will eventually lead to a provable upper bound on the time required to find a vector in $\ker(A)$ over $\mathbb{F}_2$.

For completeness we explicitly include and expand on the ideas from [4] for quickly computing certain matrix products which arise during the course of this algorithm. Finally, we will give some experimental data resulting from the application of this algorithm to some matrices arising from the NFS factorization of several integers.

This paper constitutes an expansion on work of the first author in his Master's Thesis [7].

2 Gram-Schmidt Lanczos over $\mathbb{R}$

We consider here a variation on the Lanczos algorithm [5] over $\mathbb{R}^N$ to find a vector $\boldsymbol{x} \in \ker(A)$, where A is symmetric $N \times N$ and singular. To be clear, the method we present here is almost certainly numerically unstable, but our eventual goal is to work over $\mathbb{F}_2$ anyway.

It begins by choosing a random $\boldsymbol{y} \in \mathbb{R}^N$ and applying the Gram-Schmidt process to the sequence of vectors $A^1\boldsymbol{y}, A^2\boldsymbol{y}, A^3\boldsymbol{y}, \ldots$. For vectors $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^N$, let $\mathrm{Proj}\,(\boldsymbol{u}; \boldsymbol{v})$ denote the projection of $\boldsymbol{u}$ onto $\boldsymbol{v}$, so that $\mathrm{Proj}\,(\boldsymbol{u}; \boldsymbol{v}) = \frac{\boldsymbol{u}\cdot\boldsymbol{v}}{\boldsymbol{v}\cdot\boldsymbol{v}}\boldsymbol{v}$. Set

$$\begin{aligned} \boldsymbol{w}_0 &= Ay \\ \boldsymbol{w}_{n+1} &= A\boldsymbol{w}_n - \sum_{j=0}^{n} \mathrm{Proj}\,(A\boldsymbol{w}_n; \boldsymbol{w}_j), \qquad \text{for } n \geq 0. \end{aligned}$$

Then the collection $\{\boldsymbol{w}_0, \ldots, \boldsymbol{w}_k\}$ is an orthogonal collection of vectors having the same span as $\{A\boldsymbol{y}, \ldots, A^{k+1}\boldsymbol{y}\}$.

The key ingredient in the efficiency of this class of algorithms is the clever observation by Lanczos [5] that the computation of $\boldsymbol{w}_{n+1}$ can be simplified. By construction, we have that $\boldsymbol{w}_i \cdot \boldsymbol{w}_j = \boldsymbol{0}$ for $i \neq j$. Then for $i \leq n-2$ we have

$$\begin{aligned} A\boldsymbol{w}_n \cdot \boldsymbol{w}_i &= \boldsymbol{w}_n^T A^T \boldsymbol{w}_i = \boldsymbol{w}_n^T A \boldsymbol{w}_i \\ &= \boldsymbol{w}_n \cdot \left(\boldsymbol{w}_{i+1} + \sum_{j=0}^{i} \mathrm{Proj}\,(A\boldsymbol{w}_i; \boldsymbol{w}_j) \right) \\ &= \boldsymbol{w}_n \cdot \boldsymbol{w}_{i+1} + \sum_{j=0}^{i} \boldsymbol{w}_n \cdot (\alpha_j \boldsymbol{w}_j) \quad \text{for some } \alpha_j \in \mathbb{R} \\ &= 0. \end{aligned}$$

It follows that for $j \leq n-2$, $\mathrm{Proj}\,(A\boldsymbol{w}_n; \boldsymbol{w}_j) = \boldsymbol{0}$, so that for each $n > 1$,

$$\boldsymbol{w}_{n+1} = A\boldsymbol{w}_n - \mathrm{Proj}\,(A\boldsymbol{w}_n; \boldsymbol{w}_{n-1}) - \mathrm{Proj}\,(A\boldsymbol{w}_n; \boldsymbol{w}_n).$$

If $\boldsymbol{w}_0, \ldots, \boldsymbol{w}_k$ are all nonzero, it follows that they are linearly independent since they are pairwise orthogonal. But since these are vectors in $\mathbb{R}^N$, it follows that the sequence $\{\boldsymbol{w}_0, \boldsymbol{w}_1, \ldots\}$ defined as above must eventually become zero, so there is a least positive integer m for which $\boldsymbol{w}_{m+1} = \boldsymbol{0}$. In parallel with the computation of the $\boldsymbol{w}_j$, we compute

$$\boldsymbol{x} = \sum_{i=0}^{m} \mathrm{Proj}\,(\boldsymbol{y}; \boldsymbol{w}_i) = \sum_{i=0}^{m} \frac{\boldsymbol{w}_i \cdot \boldsymbol{y}}{\boldsymbol{w}_i \cdot \boldsymbol{w}_i} \boldsymbol{w}_i.$$

If $\boldsymbol{y}$ is in the span of the $\boldsymbol{w}_j$, then $\boldsymbol{x} = \boldsymbol{y}$. However, if A is singular this is very unlikely to happen. We claim that $\boldsymbol{x} - \boldsymbol{y} \in \ker(A)$. For this, set $\boldsymbol{u} = \boldsymbol{y} - \boldsymbol{x} = \boldsymbol{y} - \sum \mathrm{Proj}\,(\boldsymbol{y}; \boldsymbol{w}_i)$. Then

$$A\boldsymbol{u} = \boldsymbol{w}_0 - \sum \frac{\boldsymbol{w}_i \cdot \boldsymbol{y}}{\boldsymbol{w}_i \cdot \boldsymbol{w}_i} A\boldsymbol{w}_i,$$

so that $A\boldsymbol{u}$ is in the span of $\boldsymbol{w}_0, \ldots, \boldsymbol{w}_m$. But $\boldsymbol{w}_j^T \boldsymbol{u} = 0$ for all j and so using the symmetry of A we have that

$$\begin{aligned}
\mathrm{Proj}\,(A\boldsymbol{u}; \boldsymbol{w}_j) &= \boldsymbol{w}_j (\boldsymbol{w}_j^T \boldsymbol{w}_j)^{-1} (A\boldsymbol{w}_j)^T \boldsymbol{u} \\
&= \boldsymbol{w}_j (\boldsymbol{w}_j^T \boldsymbol{w}_j)^{-1} \left(\boldsymbol{w}_{j+1} + \sum_{i=0}^{j} \mathrm{Proj}\,(A\boldsymbol{w}_j; \boldsymbol{w}_i) \right)^T \boldsymbol{u} \\
&= \boldsymbol{w}_j (\boldsymbol{w}_j^T \boldsymbol{w}_j)^{-1} \left(\boldsymbol{w}_{j+1}^T \boldsymbol{u} + \sum_{i=0}^{j} c_{i,j} \boldsymbol{w}_i^T \boldsymbol{u} \right) = \boldsymbol{0} \quad (\text{ for some } c_{i,j} \in \mathbb{R} \,).
\end{aligned}$$

The goal of the rest of this paper is to describe a similar technique which will work over $\mathbb{F}_2$.

3 Notations

Throughout, K will be fixed (as either 32 or 64), depending on the architecture on which this algorithm is to be implemented. For matrices X, Y, we let $(X|Y)$ denote the column-wise concatenation of X and Y. Similarly, $\left(\frac{X}{Y}\right)$ denotes the row-wise concatenation of X and Y.

If V_i is an $r \times c_i$ matrix for $i = 0, 1, \ldots, k$, we abuse notation slightly and define the subspace generated by $V_0, \ldots, V_k$ to be

$$\langle V_0, \ldots, V_k \rangle = \{Z \,:\, Z = V_0 U_0 + \cdots + V_k U_k \quad \text{for some } c \geq 1 \text{ and } c_i \times c \text{ matrices } U_i\}.$$

That is, $Q \in \langle V_0, \ldots, V_k \rangle$ iff Q has r rows and every column of Q is in the column span of $(V_0|V_1|\cdots|V_k)$.

4 The algorithm over $\mathbb{F}_2$

The most obvious obstruction to extending the technique of Section 2 to $\mathbb{F}_2$ is that there exist nonzero vectors over $\mathbb{F}_2$ which are self-orthogonal. This presents a formal problem with the calculations from the previous section, as division by zero would occur often. However, the real problem is more fundamental; projection onto such a vector cannot be performed in any reasonable sense.

Over $\mathbb{F}_2$ where addition is simply XOR, we can add vectors of dimension K for the same cost as adding vectors of dimension 1. For this reason alone, it is already natural to consider a block Lanczos variant in this case. But more importantly, if $\mathcal{V}$ is a subspace of dimension K over $\mathbb{F}_2$, while we may not always be able to project onto $\mathcal{V}$, we can almost always find a 'large' subspace $\mathcal{W}$ of $\mathcal{V}$ for which projection onto $\mathcal{W}$ is well-defined. The $\mathbb{F}_2$-vector spaces which do admit well-defined projection are described by the following proposition.

Proposition 4.1 *Let $\mathcal{W}$ be a subspace of $\mathbb{F}_2^N$ having a basis of column vectors $\mathcal{W} = \mathrm{Colsp}(W)$ with W^TW invertible. Then each $\boldsymbol{u} \in \mathbb{F}_2^N$ can be uniquely written as $\boldsymbol{u} = \boldsymbol{w} + \boldsymbol{v}$ with $\boldsymbol{w} \in \mathcal{W}$ and $\mathcal{W}^T\boldsymbol{v} = \boldsymbol{0}$. Furthermore, this property is independent of the choice of basis for $\mathcal{W}$.*

Proof: Let $\boldsymbol{u} \in \mathbb{F}_2^N$ and set $\boldsymbol{w} = W(W^TW)^{-1}W^T\boldsymbol{u}$ and $\boldsymbol{v} = \boldsymbol{u} + \boldsymbol{w}$. Then $\boldsymbol{w} \in \mathrm{Colsp}(W) = \mathcal{W}$, $\boldsymbol{u} = \boldsymbol{v} + \boldsymbol{w}$ and

$$\begin{aligned} W^T\boldsymbol{v} &= W^T\boldsymbol{u} + W^T\boldsymbol{w} \\ &= W^T\boldsymbol{u} + W^TW(W^TW)^{-1}W^T\boldsymbol{u} = \boldsymbol{0}, \end{aligned}$$

so that $\mathcal{W}^T\boldsymbol{v} = 0$ as desired.

Suppose now that $\boldsymbol{w'}$ is another vector in $\mathrm{Colsp}(W)$ so that $W^T(\boldsymbol{u}+\boldsymbol{w'}) = \boldsymbol{0}$. Then $\boldsymbol{w'} \in \mathrm{Colsp}(W) \Rightarrow \boldsymbol{w'} = W\boldsymbol{\alpha}$ for some $\boldsymbol{\alpha} \in \mathbb{F}_2^N$. So we have $\boldsymbol{0} = W^T(\boldsymbol{u} + \boldsymbol{w'}) = W^T\boldsymbol{u} + W^TW\boldsymbol{\alpha}$, whence $W^TW\boldsymbol{\alpha} = W^T\boldsymbol{u}$. Since W^TW is invertible, it follows that $\boldsymbol{\alpha} = (W^TW)^{-1}W^T\boldsymbol{u}$. Left multiplying both sides by W we find

$$\boldsymbol{w'} = W\boldsymbol{\alpha} = W(W^TW)^{-1}W^T\boldsymbol{u} = \boldsymbol{w},$$

and so uniqueness follows as desired.

Finally, note that if U is another basis of column vectors for $\mathcal{W} = \mathrm{Colsp}(W) = \mathrm{Colsp}(U)$, then $U = WA$ for some invertible matrix A. It follows that $U^TU = A^TW^TWA = A^T(W^TW)A$ is invertible, so that the result is independent of the choice of basis. □

If W^TW is invertible and U has the same number of rows as W, we therefore define

$$\mathrm{Proj}\,(U; W) := W(W^TW)^{-1}W^TU.$$

This notation is both convenient and compatible with intuition since $\mathrm{Colsp}(\mathrm{Proj}\,(U; V))$ is then the projection of $\mathrm{Colsp}(U)$ onto $\mathrm{Colsp}(V)$.

The underlying idea of this algorithm is to produce a sequence of orthogonal subspaces spanning an $A-$cyclic subspace of $\mathbb{F}^N$, and use projection onto the orthogonal subspaces to solve the given problem locally. The following proposition summarizes the properties we desire of these subspaces, and how they will be used to solve the kernel problem over $\mathbb{F}_2$. For simplicity, the proposition is stated strictly in matrix terms, but the subspace interpretation is obvious.

Proposition 4.2 *Let A be a symmetric $N \times N$ matrix over $\mathbb{F}_2$. Suppose $W_0, W_1, \ldots, W_{m+1}$ is a sequence of matrices satisfying*

1. *W_i is $N \times k_i$.*
2. *$W_i^T W_i$ is invertible for $i \in \{0, 1, \ldots, m\}$.*
3. *$W_i^T W_j = 0$ for all distinct $i, j \in \{0, 1, \ldots, m+1\}$.*
4. *$AW_i \in \langle W_0, W_1, \ldots, W_{m+1} \rangle$ for all $i \in \{0, 1, \ldots, m\}$.*

Then $(W_0|W_1|\cdots|W_m)$ has full rank. If $Y \in \langle W_0, \ldots, W_m \rangle$, then $Y = \sum_{j=0}^m \mathrm{Proj}\,(Y; W_j)$. Furthermore, if Y is any $N \times k$ matrix with $AY \in \langle W_0, \ldots, W_{m+1} \rangle$ and

$$X = \sum_{i=0}^m \mathrm{Proj}\,(Y; W_i),$$

then $\mathrm{rank}\,(A(X+Y)) \leq 2 \cdot \mathrm{rank}\,(W_{m+1})$.

Proof: To see first that the $N \times (\sum k_i)$ matrix $(W_0|W_1|\cdots|W_m)$ has full rank, notice that any linear dependence on the columns of this matrix can be expressed as

$$0 = W_0 C_0 + W_1 C_1 + \cdots + W_m C_m,$$

for some $k_j \times 1$ matrices C_j. It follows from the hypotheses that for each i, $0 = W_i^T(W_0 C_0 + W_1 C_1 + \cdots + W_m C_m) = W_i^T W_i C_i$. But since $W_i^T W_i$ is invertible, we have $C_i = 0$, and so the columns are linearly independent as desired.

Observe now that if $Y = \sum_{j=0}^m W_j U_j$ then for each j

$$\begin{aligned} \mathrm{Proj}\,(Y; W_j) = W_j(W_j^T W_j)^{-1} W_j^T Y &= \sum_{i=0}^m W_j(W_j^T W_j)^{-1} W_j^T W_i U_i \\ &= W_j(W_j^T W_j)^{-1} W_j^T W_j U_j = W_j U_j, \end{aligned}$$

so that $\sum \mathrm{Proj}\,(Y; W_j) = Y$, proving the second statement.

For the last statement, we begin with the observation that for each $0 \leq j \leq m$,

$$W_j^T(X+Y) = W_j^T \left(\sum_{i=0}^m W_i(W_i^T W_i)^{-1} W_i^T Y \right) + W_j^T Y = W_j^T Y + W_j^T Y = 0.$$

By a similar calculation, $W_{m+1}^T(X+Y) = W_{m+1}^T Y$.

Now since $A(X+Y) \in \langle W_0, \ldots, W_{m+1} \rangle$, we have that $A(X+Y) = W_{m+1}V + W$ for some V and some $W \in \langle W_0, \ldots, W_m \rangle$. Using the previous part of this proposition, it follows that $W = \sum_{i=0}^m \mathrm{Proj}\,(A(X+Y) + W_{m+1}V; W_i) = \sum_{i=0}^m \mathrm{Proj}\,(A(X+Y); W_i)$, which we will now compute.

By hypothesis, we have that for each $0 \leq i \leq m$, $AW_i = \sum_{j=0}^{m+1} W_j U_{j,i}$ for some $N \times k_i$ matrices $U_{j,i}$. Since A is symmetric, it follows that for each $0 \leq i \leq m$,

$$\begin{aligned}
\text{Proj}\,(A(X+Y); W_i) &= W_i(W_i^T W_i)^{-1} W_i^T A(X+Y) \\
&= W_i(W_i^T W_i)^{-1}(AW_i)^T(X+Y) \\
&= W_i(W_i^T W_i)^{-1}\left(\sum_{j=0}^{m+1} W_j U_{j,i}\right)^T (X+Y) \\
&= W_i(W_i^T W_i)^{-1}\left(\sum_{j=0}^{m+1} U_{j,i}^T (W_j^T(X+Y))\right) \\
&= W_i(W_i^T W_i)^{-1} U_{m+1,i}^T W_{m+1}^T Y.
\end{aligned}$$

This gives that

$$A(X+Y) = W_{m+1}V + W = W_{m+1}V + \left(\sum_{i=0}^{m} W_i(W_i^T W_i)^{-1} U_{m+1,i}^T\right) W_{m+1}^T Y.$$

Both terms in this expression have rank at most $\text{rank}\,(W_{m+1})$, and since the rank of a sum is at most the sum of the ranks, it follows that $\text{rank}\,(A(X+Y)) \leq 2 \cdot \text{rank}\,(W_{m+1})$ as desired. □

The remainder of this section is devoted to showing how to produce a collection $\{W_i\}$ of matrices (subspaces) satisfying the conditions of Proposition 4.2, with $2 \cdot \text{rank}\,(W_{m+1})$ small compared to $\text{rank}\,(X+Y)$ so that we may recover vectors in the kernel of A via simultaneous elementary column operations.

The goal is to follow the ideas of Section 2 as closely as possible. To that end, we would attempt to simply choose a random $N \times k$ matrix Y, and set $W_0 = AY$ and $n = 1$. Assuming $(W_i^T W_i)$ is invertible for $0 \leq i < n$, we set

$$E = AW_{n-1} - \sum_{i=0}^{n-1} \text{Proj}\,(AW_n; W_i). \tag{4.1}$$

If $E^T E$ is invertible, we set $W_n = E$. As in Section 2, we would have the crucial observation for efficiency that all but the last two of these projections are identically zero. However, handling the case where $E^T E$ is not invertible introduces some complication. Loosely, the idea is to start with Equation 4.1, but restrict to a maximal dimensional subspace of $\text{Colsp}(E)$ which we can project onto.

Note first that the vectors in $\text{Colsp}(E)$ which are orthogonal to all of $\text{Colsp}(E)$ form a subspace $\mathcal{D}$ of $\text{Colsp}(E)$. Furthermore, $E^T E$ is invertible iff $\mathcal{D} = \{0\}$. If $\widetilde{U}$ is an invertible matrix so that $E\widetilde{U} = [C|D]$ where $\text{Colsp}(D) = \mathcal{D}$, we have

$$(E\widetilde{U})^T(E\widetilde{U}) = \left[\begin{array}{c|c} C^T C & C^T D \\ \hline D^T C & D^T D \end{array}\right] = \left[\begin{array}{c|c} C^T C & 0 \\ \hline 0 & 0 \end{array}\right],$$

and C^TC must be invertible since $\mathrm{Colsp}(D) = \mathcal{D}$. Since $(E\widetilde{U})^T(E\widetilde{U}) = \widetilde{U}^TE^TE\widetilde{U}$ and $\widetilde{U}$ is invertible, it also follows that that $\mathrm{rank}\,(C^TC) = \mathrm{rank}\,(E^TE)$. The only question remaining is how to find such a $\widetilde{U}$. For this, compute E^TE and find an invertible U so that $U(E^TE)$ is in reduced-row echelon form. Then if $\mathrm{rank}\,(E^TE) = r$ we have

$$U(E^TE) = \left[\frac{V}{0}\right], \quad \text{where } V \text{ has } r \text{ rows and full rank.}$$

Since U is invertible we have $\mathrm{rank}\,(UE^TEU^T) = r$. Furthermore, since the bottom $K - r$ rows of UE^TEU^T are zero and this matrix is symmetric, it follows that we have

$$(EU^T)^T(EU^T) = \left[\begin{array}{c|c} C^TC & 0 \\ \hline 0 & 0 \end{array}\right],$$

where C is the first r columns of EU^T and $\mathrm{rank}\,(C^TC) = r$. It also follows that $\mathcal{D}$ is precisely the span of the last $K - r$ columns of EU^T.

Suppose now that $\{W_0, W_1, \ldots, W_{n-1}\}$ have been computed satisfying the first three hypotheses of Proposition 4.2 and W_n is to be computed. We first compute

$$E_n = AW_{n-1} - \sum_{j=0}^{n-1} \mathrm{Proj}\,(AW_{n-1}; W_j). \tag{4.2}$$

If $E_n^TE_n$ is invertible, we can simply take $W_n = E_n$. Very roughly speaking, this happens with probability only 0.42 or so (see Section 8 for an explanation). In the event $E_n^TE_n$ is not invertible, we follow the preceding discussion and find an invertible U so that $U(E_n^TE_n)$ is RREF and set W_n equal to the first $r = \mathrm{rank}\,(E_n^TE_n)$ columns of E_nU^T. We also set D_{n+1} equal to the last $K - r$ columns of E_nU^T. We clearly cannot discard D_{n+1} or the dimension of the W_i's obtained in this way would quickly drop to zero. But by construction, D_{n+1} is orthogonal to all of $W_0, W_1, \ldots, W_n$ making it an excellent candidate for inclusion in W_{n+1}. We will argue in Section 8 that if the sequence of E_n obtained in this way behaved like random, we would have that the expected value of r is about $K - 0.76$ with a standard deviation small compared to $K = 32$ or 64 (for sufficiently large N). Experimental data will be given in that section to support this assumption.

So at the next iteration we will compute

$$\widetilde{E}_{n+1} = AW_n - \sum_{j=0}^{n} \mathrm{Proj}\,(AW_n; W_j),$$

and taking $E_{n+1} = [D_{n+1}|\widetilde{E}_{n+1}]$, proceed essentially as above. However, in order to be able to simplify the calculation of Equation 4.2 in subsequent iterations, we will require that $\mathrm{Colsp}(D_{n+1}) \subseteq \mathrm{Colsp}(W_{n+1})$ and so some care is required in the RREF computation. We have

$$T_{n+1} = E_{n+1}^TE_{n+1} = \left[\begin{array}{c|c} 0 & D_{n+1}^T\widetilde{E}_{n+1} \\ \hline \widetilde{E}_{n+1}^TD_{n+1} & \widetilde{E}_{n+1}^T\widetilde{E}_{n+1} \end{array}\right].$$

With the assumptions above on the size of D_{n+1}, we find that the number of columns in D_{n+1} should be relatively small, say at most 5 or 6. Since $D_{n+1}^T \widetilde{E}_{n+1}$ then has far more columns than rows and $\widetilde{E}_{n+1}$ is more-or-less independent of D_{n+1}, it is extremely likely that $D_{n+1}^T \widetilde{E}_{n+1}$ does have full rank. Then if $D_{n+1}^T \widetilde{E}_{n+1}$ has full rank, its rows are linearly independent and we can guarantee the inclusion $\mathrm{Colsp}(D_{n+1}) \subseteq \mathrm{Colsp}(W_{n+1})$ by first putting this submatrix of T_{n+1} in RREF and using the leading entries of these rows as pivots in the RREF computation of T_{n+1}. In practice, since the number of rows of $D_{n+1}^T \widetilde{E}_{n+1}$ is small, it suffices just to put T_{n+1} into RREF using an algorithm which always searches for pivot elements from top to bottom, choosing the first possible one. In the very unlikely event that $D_{n+1}^T \widetilde{E}_{n+1}$ does not have full rank, the algorithm should terminate with a failure message. Note that Montgomery's algorithm admits the same remote possibility of failure, and that both his and the current algorithm could be modified to allow for this possibility by adding an additional recurrence term later. However, the possibility is so unlikely in practice that it is not worth the runtime penalty or the complication of doing so.

The algorithm terminates when we eventually encounter some W_{n+1} which would be zero by the above method. Note that this is guaranteed to happen by the full-rank implication of Proposition 4.2. When this happens, we take $m = n$ and $W_{m+1} = D_{n+1}$ for the purpose of applying the last part of Proposition 4.2.

This is already the essence of the algorithm. The remainder of this paper is devoted to more efficiently computing the W_j that would be obtained in this way.

5 Simplifying the computation

In this section, we simplify the recurrence and replace some of the more expensive matrix multiplications with less expensive calculations by observations very similar to those of Montgomery [6], carried over to the present case.

The iterative portion of the algorithm proceeds as follows: Given a sequence $W_0, W_1, \ldots, W_n$ satisfying the first three hypothesis of Proposition 4.2 and a matrix D_{n+1} of small rank as described above, we compute

$$E_{n+1} = AW_n - \sum_{j=0}^{n} \mathrm{Proj}\,(AW_n; W_j), \tag{5.1}$$

and set $T_{n+1} = [D_{n+1}|E_{n+1}]^T [D_{n+1}|E_{n+1}]$. Find U_{n+1} as described above so that $U_{n+1}T_{n+1}$ is in RREF and take W_{n+1} as the first $k_{n+1} = \mathrm{rank}\,(T_{n+1})$ columns of $[D_{n+1}|E_{n+1}]U_{n+1}^T$ and D_{n+2} as the remaining $K - k_{n+1}$ columns. In the sequel, we also assume that $\mathrm{Colsp}(D_{n+1}) \subseteq \mathrm{Colsp}(W_{n+1})$ for the reasons given above. This assumption will be further justified with experimental data in the Section 8.

First we show that, as in Section 2, *most* of the $\mathrm{Proj}\,(AW_n; W_j)$ are zero so that the calculation of E_{n+1} in Equation 5.1 may be greatly simplified. For this, notice that we have by the construction above for each j, $[W_j|D_{j+1}] = [D_j|E_j]U_j^T$ where U_j is invertible and D_j or D_{j+1} (or both) may be empty (i.e., have no columns). It follows that for $i < n-2$

$$W_n^T AW_i = W_n^T \left(E_{i+1} + \sum_{j=0}^{i} \mathrm{Proj}\,(AW_i; W_j) \right) = W_n^T E_{i+1} = W_n^T [W_{i+1}|D_{i+2}](U_{i+1}^T)^{-1}.$$

Then $i+1 < i+2 < n \Rightarrow W_n^T W_{i+1} = 0$. Furthermore, since $\mathrm{Colsp}(D_{i+2}) \subseteq \mathrm{Colsp}(W_{i+2})$ and $W_n^T W_{i+2} = 0$, it follows that $W_n^T D_{i+2} = 0$ so that $W_n^T A W_i = 0$. Thus for $i < n-2$ we have by the symmetry of A that $\mathrm{Proj}\,(AW_n; W_i) = W_i (W_i^T W_i)^{-1} W_i^T A W_n = W_i (W_i^T W_i)^{-1} \left(W_n^T A W_i\right)^T = 0$. So under the assumptions above, E_{n+1} may be computed by

$$E_{n+1} = AW_n - \sum_{j=n-2}^{n} \mathrm{Proj}\,(AW_n; W_j). \tag{5.2}$$

Now we show how to inductively remove some of the more expensive matrix products occurring in the above expression.

We begin by observing that

$$U_{n+1} T_{n+1} U_{n+1}^T = [W_{n+1}|D_{n+2}]^T [W_{n+1}|D_{n+2}] = \left[\begin{array}{c|c} W_{n+1}^T W_{n+1} & W_{n+1}^T D_{n+2} \\ \hline D_{n+2}^T W_{n+1} & D_{n+2}^T D_{n+2} \end{array}\right]. \tag{5.3}$$

Since $U_{n+1} T_{n+1}$ is already computed and U_{n+1} is a small $(K \times K)$ square matrix, the left-hand side of Equation 5.3 may be computed cheaply; using this we set $J_{n+1} = (W_{n+1}^T W_{n+1})^{-1}$ for use in the next few iterations.

We also have $W_n^T A W_{n+1} = (W_{n+1}^T A W_n)^T = \left(W_{n+1}^T \left(E_{n+1} + \sum_{j=0}^{n} \mathrm{Proj}\,(AW_n; W_j)\right)\right)^T = E_{n+1}^T W_{n+1}$, and this quantity may be found as the lower left $k_n \times k_{n+1}$ submatrix of

$$T_{n+1} U_{n+1}^T = [D_{n+1}|E_{n+1}]^T [W_{n+1}|D_{n+2}] = \left[\begin{array}{c|c} D_{n+1}^T W_{n+1} & D_{n+1}^T D_{n+2} \\ \hline E_{n+1}^T W_{n+1} & E_{n+1}^T D_{n+2} \end{array}\right], \tag{5.4}$$

which is again an inexpensive calculation. So we set $F_n = W_n^T A W_{n+1}$ in this way, and save it for use in the next iteration.

Similarly, $W_{n-1}^T A W_{n+1} = \left(W_{n+1}^T A W_{n-1}\right)^T = (W_{n+1}^T E_n)^T = E_n^T W_{n+1}$. Then since $E_n = [W_n|D_{n+1}](U_n^T)^{-1} P_{n-1}$, where P_{n-1} is a matrix selecting the last k_{n-1} columns, it follows that

$$W_{n-1}^T A W_{n+1} = P_{n-1}^T U_n^{-1} \left[\frac{W_n^T}{D_{n+1}^T}\right] W_{n+1} = P_{n-1}^T U_n^{-1} \left[\frac{0}{D_{n+1}^T W_{n+1}}\right].$$

The quantity $D_{n+1}^T W_{n+1}$ is already known from Equation 5.4 in the calculation of F_n. In this way, we set $G_n = W_{n-1}^T A W_{n+1}$ as the last k_{n-1} rows of $U_n^{-1} \left[\frac{0}{D_{n+1}^T W_{n+1}}\right]$. Observe immediately that if $k_n = K$ then $D_{n+1} = 0$ so that $G_n = 0$ in this case.

Inductively, we have that the calculation of E_{n+1} may be simplified to

$$E_{n+1} = AW_n + W_n J_n W_n^T A W_n + W_{n-1} J_{n-1} F_{n-1} + W_{n-2} J_{n-2} G_{n-1}, \tag{5.5}$$

and the final term may even be omitted if $k_{n-2} = K$ (which happens for approximately $1/2$ of all iterations).

Finally, we simplify the computation of

$$X_{n+1} = X_n + \mathrm{Proj}\,(Y; W_{n+1}),$$

from which the solution is produced. For this, we have $\mathrm{Proj}\,(Y; W_{n+1}) = W_{n+1}J_{n+1}W_{n+1}^T Y$ and we wish to remove the expensive computation of $W_{n+1}^T Y$. We will assume inductively that $S_j = W_j^T Y$ is known for $j \leq n$ and show how to find $S_{n+1} = W_{n+1}^T Y$. Since

$$W_{n+1} = (\text{ the first } k_{n+1} \text{ columns of })[D_{n+1}|E_{n+1}]U_{n+1}^T,$$

we have

$$S_{n+1} = W_{n+1}^T Y = (\text{ the first } k_{n+1} \text{ rows of })U_{n+1}\begin{bmatrix} D_{n+1}^T Y \\ \hline E_{n+1}^T Y \end{bmatrix}.$$

We need now to find $D_{n+1}^T Y$ and $E_{n+1}^T Y$. Using Equation 5.5 to find $E_{n+1}^T Y$, we have

$$\begin{aligned} E_{n+1}^T Y &= W_n^T AY + W_n^T AW_n J_n^T W_n^T Y + F_{n-1}^T J_{n-1}^T W_{n-1}^T Y + G_{n-1}^T J_{n-2}^T W_{n-2}^T Y \\ &= W_n^T AY + H_n^T J_n^T S_n + F_{n-1}^T J_{n-1}^T S_{n-1} + G_{n-1}^T J_{n-2}^T S_{n-2}, \end{aligned}$$

where we have set $H_n = W_n^T AW_n$ since this quantity has already been computed while finding E_{n+1} from Equation 5.5. If we assume that $\mathrm{Colsp}(AY) \subseteq \mathrm{Colsp}(W_0)$, then the first term vanishes for $n \geq 1$. Then since

$$D_{n+2} = (\text{ the last } K - k_{n+1} \text{ columns of })[D_{n+1}|E_{n+1}]U_{n+1}^T,$$

it follows that

$$D_{n+2}^T Y = (\text{ the last } K - k_{n+1} \text{ rows of })U_{n+1}\begin{bmatrix} D_{n+1}^T Y \\ \hline E_{n+1}^T Y \end{bmatrix}.$$

Thus, if we assume inductively that $S_j = W_j^T Y$ and $V_{j+1} = D_{j+1}^T Y$ are known for $j \leq n$, we have that for $n \geq 1$

$$E_{n+1}^T Y = H_n^T J_n^T S_n + F_{n-1}^T J_{n-1}^T S_{n-1} + G_{n-1}^T J_{n-2}^T S_{n-2} \tag{5.6}$$

$$C = U_{n+1}\begin{bmatrix} V_{n+1} \\ \hline E_{n+1}^T Y \end{bmatrix} \tag{5.7}$$

$$S_{n+1} = \text{ the first } k_{n+1} \text{ rows of } C \tag{5.8}$$

$$V_{n+2} = \text{ the last } K - k_{n+1} \text{ rows of } C. \tag{5.9}$$

Note: if we proceed as above, it could easily happen that we end with $W_{m+1} = 0$ and $D_{m+2} \neq 0$. In this case, we will not have $A(X+Y) = 0$ identically. However, in practice, the rank of D_j is small (say, less or equal 5, justified by the heuristic argument and experimental data in Section 8). Then by Proposition 4.2, $\mathrm{rank}\,(A(X+Y)) \leq 2 \cdot \mathrm{rank}\,(D_{m+2})$ [1] will be small as well. Suppose that $\dim\ker(A)$ is sufficiently large, say $\dim\ker(A) \geq 2K$, which is not an unreasonable assumption if we are trying to find nearly K linearly independent vectors in the kernel. Then since Y is chosen randomly and $X \in \langle W_0, W_1, \ldots, W_n\rangle$, it is extremely likely that $X+Y$ will have full rank. Then if $X+Y$ has full (or near full) rank and $A(X+Y)$ has small rank, we produce *nearly* K linearly independent vectors in $\ker(A)$ by performing simultaneous column operations on $A(X+Y)$ and $X+Y$.

[1] In fact, it seems to be the case here that $\mathrm{rank}\,(A(X+Y)) \leq \mathrm{rank}\,(D_{m+2})$, but we have been unable to prove this.

6 A compact description

Algorithm 6.1 $\mathbb{F}_2$-*Lanczos Kernel*
Input: *A singular symmetric $N \times N$ matrix A over $\mathbb{F}_2$ ('black-box' form is sufficient, as the algorithm performs no operations on A itself; we require only the ability to compute AZ for $N \times K$ matrices Z over $\mathbb{F}_2$, where $K = 32$ or 64 according to the machine architecture).*
Output: *Several (up to K) linearly independent vectors in the kernel of A.*

1. *Choose a random $N \times K$ matrix Y so that Y and $(AY)^T(AY)$ both have full rank. Set $n \leftarrow 0$, $W_n \leftarrow AY$, $J_n \leftarrow (W_n^T W_n)^{-1}$, $S_n \leftarrow W_n^T Y$, $k_{n-1}, k_n \leftarrow 64$ and all other variables zero.*

2. *Compute AW_n and $H_n \leftarrow W_n^T AW_n$. Set $P \leftarrow J_n H_n$, $Q \leftarrow J_{n-1}F_{n-1}$, $R \leftarrow J_{n-2}G_{n-1}$, and $E_{n+1} \leftarrow AW_n + W_n P + W_{n-1}Q$. If $k_{n-1} < 64$ then do $E_{n+1} \leftarrow E_{n+1} + W_{n-2}R$.*

3. *Compute $T \leftarrow [D_{n+1}|E_{n+1}]^T[D_{n+1}|E_{n+1}]$. Find an invertible U_{n+1} so that $U_{n+1}T$ is in reduced row-echelon form (see note below) and set $k_{n+1} = \operatorname{rank}(T)$. Set W_{n+1} to be the first k_{n+1} columns of $[D_{n+1}|E_{n+1}]U_{n+1}^T$ and D_{n+2} the remaining columns (or zero if there are no columns). If $W_{n+1} = 0$, goto Step 5. Otherwise, use the fact that*

$$(U_{n+1}T)^T = \left[\begin{array}{c|c} D_{n+1}^T W_{n+1} & D_{n+1}^T D_{n+2} \\ \hline E_{n+1}^T W_{n+1} & E_{n+1}^T D_{n+2} \end{array}\right],$$

to set $F_n \leftarrow E_{n+1}^T W_{n+1}$ as the lower left $k_n \times k_{n+1}$ sub-matrix. If $k_n < 64$, set $G_n \leftarrow$(the last k_{n-1} rows of)$U_n^{-1}\left[\frac{0}{D_{n+1}^T W_{n+1}}\right]$. Finally, compute $U_{n+1}TU_{n+1}^T$ and use the fact that

$$U_{n+1}TU_{n+1}^T = \left[\begin{array}{c|c} W_{n+1}^T W_{n+1} & W_{n+1}^T D_{n+2} \\ \hline D_{n+2}^T W_{n+1} & D_{n+2}^T D_{n+2} \end{array}\right],$$

to set $J_{n+1} \leftarrow (W_{n+1}^T W_{n+1})^{-1}$.

4. *If $n = 0$, compute $E_{n+1}^T Y$ directly. Otherwise, compute it via*

$$E_{n+1}^T Y = P^T S_n + Q^T S_{n-1} + R^T S_{n-2}.$$

Compute $U_{n+1}\left[\frac{V_{n+1}}{E_{n+1}^T Y}\right]$, and set S_{n+1} to be the first k_{n+1} rows of this quantity and V_{n+2} to be the last $K - k_{n+1}$ rows. Set $X \leftarrow X + W_{n+1}J_{n+1}S_{n+1}$. Set $n \leftarrow n + 1$ and goto Step 2.

5. *Set Z to be the nonzero columns in the reduced column echelon form of $X+Y$. Compute AZ and perform simultaneous column operations on AZ and Z to produce $K-d$ vectors in $\ker(A)$, where $d = \operatorname{rank}(AZ)$.*

Note: The RREF computation in Step 3 search for pivot elements from top to bottom, choosing the first possible row each time.

7 Efficiency considerations

In this section, we describe and elaborate on several tricks pointed out in [4] for efficient implementation. In practice, for large matrices A, the most time consuming operations involved in Algorithm 6.1 are (in this order):

(i) Computing AW for a given $N \times K$ matrix W.

(ii) Computing $W^T V$ for $N \times K$ matrices W and V.

(iii) Computing WU where W is $N \times K$ and U is $K \times K$.

We will describe Coppersmith's trick for speeding up the computations in (ii) above in the case when W and V are dense (the corresponding products which appear explicitly in the algorithm usually do involve W, V, U which are dense). A very similar trick is used to speed up the calculation (iii) above.

It is well-known that any method for performing (ii) can also be used to help speed up the calculations of AW for many A which arise in practice. In particular, for matrices B arising from the NFS or QS, we have that B is overall quite sparse, but that some rows are dense (i.e., those corresponding to 'small' primes or a quadratic character base); furthermore, the dense rows in these cases occur together in 'clumps'. The suggestion in this case is that one partition B into blocks of dense rows and sparse rows, where each dense block consists of exactly K dense rows. Store the sparse portions of B by simply recording the locations of nonzero entries, and store the dense blocks with K entries per machine word. To use this Lanczos method, one takes the symmetric matrix $A = B^T B$, and the products $B^T Z$, BZ may then be computed blockwise using the straightforward method on the sparse rows. For the dense rows, use the same Coppersmith trick that we are about to describe for computing (ii) from above.

Suppose now that W, V are both $N \times K$ dense with $N \gg K$ and stored as $W = (w_0, w_1, \ldots, w_{N-1})^T$, $V = (v_0, v_1, \ldots, v_{N-1})^T$ where w_i and v_i are $K-$bit machine words. That is, the $(i,j)-$th entry of W is the $j-$th bit of w_i, and similarly for V. The obvious method for computing the product $W^T V$ would be to go through W entry by entry, adding (via XOR) the appropriate rows of V to various memory locations as necessary. It is easy to see that if W has half of its entries nonzero, that this requires about $NK/2$ XOR operations. Coppersmith's trick will reduce this to $NK/8 + c_1$ or $NK/16 + c_2$, where c_1, c_2 are some constants that do not depend on N; the exact reduction is machine dependent, and some variations may even admit more reduction, depending on machine word size, available memory, cache considerations and so on. Loosely, the idea is to group together common additions which would be performed several times using the naive approach.

To simplify the discussion now, let us assume that $K = 64$ (i.e., a 64 bit architecture). We will partition a 64 bit machine word x into 8 subwords: $x = (\rho_0(x)|\rho_1(x)|\cdots|\rho_7(x))$, where each $\rho_j(x)$ is an eight bit word. Create eight temporary arrays, $C_0, C_1, \ldots, C_7$ each of which holds 2^8 eight bit words and initialize all entries of these to zero. To compute $W^T V$, first perform the following operations.

for $i = 0..N-1$ do
  for $j = 0..7$ do
    $C_j[\rho_j(w_i)] \leftarrow C_j[\rho_j(w_i)] \oplus v_i$

where $\oplus$ denotes bitwise modulo 2 addition (i.e., XOR). Now consider, for example, the first row of the product $U = W^T V$. If $\pi_j(x)$ denotes the $j-$th bit of x, then the first row of $W^T V$ can be computed from the C_0 array via

$$u_0 = \sum_{\substack{0 \le i < N \\ \pi_0(w_i)=1}} v_i = \sum_{\substack{0 \le k < 2^8 \\ \pi_0(k)=1}} C_0[k],$$

where both sums are again bitwise XOR. We can compute every row of U in this way

$$u_t = \sum_{\substack{0 \le k < 2^8 \\ \pi_r(k)=1}} C_q[k], \qquad \text{where } q = \left\lfloor \frac{t}{8} \right\rfloor, \qquad 0 \le t = 8q + r < 64. \tag{7.1}$$

Computing U in the obvious way from this equation requires about 8128 XOR operations. As an alternative to directly using Equation 7.1, we present another method for recovering U from the $C_0, \ldots, C_7$ arrays.

for $j = 0$ to 7
  $k \leftarrow 0$
  while $(k < 8)$
    $u_{8j+k} \leftarrow C_j[1] \oplus C_j[3] \oplus C_j[5] \oplus \cdots \oplus C_j[2^{8-k} - 1]$
    $k \leftarrow k + 1$
    if $(k < 8)$ then
      for $i = 0$ to 2^{8-k}
        $C_j[i] \leftarrow C_j[2i] \oplus C_j[2i + 1]$
      end for
    end if
  end while
end for

The number of XOR operations used by this method is roughly 4096, or about half as many as direct calculation via 7.1, at the added expense of some additional coding complexity.

Notice that we could have instead used a partition of 64 bit machine words into four subwords of 16 bits each. In this case, we have only four arrays C_0, C_1, C_2, C_3 each holding 2^{16} words of 16 bits each and the cost of the initial calculation of the C_j arrays drops to only $4N$ versus $8N$ as above. However, the cost of reassembling the final matrix product increases to about 2^{21} XOR operations. Since the reassembly cost is fixed (e.g. does not depend on N), this coarser subdivision is asymptotically better by a factor of $1/2$. Of course, in practice, the cutoff where the coarser subdivision is faster is highly implementation dependent, but the obvious estimate puts it around $N \approx 2^{19}$.

8 Expected size of D_n and experimental results

In this section, we first give a heuristic argument that the dimensions of the spaces $\mathrm{Colsp}(D_n)$ should be small. We then give some supporting data obtained by applying the algorithm in this paper to several matrices which arose from the NFS factorization of some integers. The algorithm was implemented with $K = 64$ and all timings reported here are for a 64 bit AMD Athlon processor.

The sequence of matrices $[D_{n+1}|E_{n+1}]$ computed in Step 3 of Algorithm 6.1 is surely not a random sequence. Nevertheless, the computational evidence suggests that $\mathrm{rank}\,(T) = \mathrm{rank}\,([D_{n+1}|E_{n+1}]^T[D_{n+1}|E_{n+1}])$ does behave very much like $\mathrm{rank}\,(W^TW)$ where W is a random $N \times K$ full rank matrix over $\mathbb{F}_2$. In [8], Sendrier computes for $N \geq 2K$ the number of (N, K) linear block codes over $\mathbb{F}_q$ whose hull has dimension ℓ. Consider $G = W^T$ as the generator matrix for a linear block code $\mathcal{C}$ over $\mathbb{F}_2$. Then the hull of this code, $\mathcal{C} \cap \mathcal{C}^\perp$, is precisely (the transpose of) the set of vectors in $\mathrm{Colsp}(W)$ which are orthogonal to all of $\mathrm{Colsp}(W)$. It follows from Theorem 3 of [8] that if K is fixed and W is a uniform random variable on the space of $N \times K$ binary matrices with full rank, then for $0 \leq \ell \leq K$

$$\mathrm{Prob}(\mathrm{corank}\,(W^TW) = \ell) = 2^{-\ell(\ell+1)/2} \prod_{\ell+1 \leq i \leq K} \left(1 - 2^{-i}\right) \prod_{1 \leq i \leq \frac{K-\ell}{2}} \left(1 - 4^{-i}\right)^{-1} \left(1 + O\left(\frac{K}{2^{N/2-\ell}}\right)\right).$$

The limiting values of this expression as $N \to \infty$ are given for comparison in the last row of Table 1. We also remark that the expected rank for such W tends to $K - 0.7644997803$, and we obtain the same numerical values as [6]. However, the analysis given there does not seem to directly apply to our situation here unless we are missing an obvious reduction from the column-reduced echelon form of W to the $K \times K$ case.

The following table gives the results from applying our $K = 64$ implementation of Algorithm 6.1 to matrices arising from the NFS factorizations of various integers. The first column is the dimensions of the original matrix M, from which we build a symmetric matrix $A = M^TM$. The second column is the actual 'wall-clock' time, measured in minutes, and the third is the number of iterations required (e.g., the value of n at step 5). The fourth column is the average dimension of $\mathcal{D}_n = \mathrm{Colsp}(D_n)$, and the remaining columns are the frequencies with which $\dim(\mathcal{D}_n)$ took on the specified values (neglecting D_1, which is zero by construction). The last row gives the values predicted by the argument above as $N \to \infty$. In none of these experiments did we encounter any occurrences of $\dim(\mathcal{D}_n) \geq 5$. The limiting values corresponding to $\delta = 6, 7, 8$ are roughly 6.8×10^{-7}, 5.4×10^{-9} and 2.1×10^{-11} respectively.

We should also remark that the matrices used in these experiments had varying densities which affects the overall runtime of this algorithm inasmuch as denser matrices B require more time to compute B^TBX.

9 Conclusions and Future Work

In this paper, we gave a new variant of the Lanczos algorithm over the binary field $\mathbb{F}_2$, which finds several vectors in the kernel of a symmetric $N \times N$ matrix A whose kernel is sufficiently large (say, of dimension at least twice the size of the desired number of kernel vectors). We

				Frequency of $\dim(\mathcal{D}_n) = \delta$					
size	time	iterations	avg.	$\delta = 0$	$\delta = 1$	$\delta = 2$	$\delta = 3$	$\delta = 4$	$\delta = 5$
51362×51706	0.5	812	.815	.4236	.4002	.1466	.0246	0	0
418172×420407	161	6612	.7603	.4267	.4094	.1422	.0206	.0012	0
547795×550684	247	8663	.7738	.4138	.4217	.1431	.0196	.0017	0
579364×582582	263	9163	.777	.4133	.4204	.1438	.0210	.0015	0
709413×713281	443	11217	.7576	.4253	.4140	.1393	.0203	.0010	0
limiting			.7645	.4194	.4194	.1398	.0200	.0013	.00004

Table 1: Experimental results $K = 64$, $A = M^T M$

also reproduced, with some additional detail, observations by Coppersmith [4] which lead to more efficient implementation.

Each iteration of the algorithm requires a single 'black-box' computation of the form AX and a number of several smaller matrix calculations; each iteration of our algorithm requires about the same work as in the variation given by Montgomery [6]. Our conjecture is that this algorithm needs *about* $N/(K - 0.7645)$ iterations where K is the machine word size in bits (typically 32 or 64), which is also the conjectured number of iterations needed by [6]. However, the sequence of subspaces produced here have different algebraic properties than [6], so perhaps this variation will eventually prove useful in proving some rigorous results.

This research was partially supported by a grant from the Texas Tech University Research Enhancement Fund (REF).

References

[1] In *The Development of the Number Field Sieve*, volume 1554 of *Lecture Notes in Mathematics*, New York, 1993. Springer Verlag.

[2] E. A. Bender and E. R. Canfield. An approximate probabilistic model for structured gaussian elimination. *Journal of Algorithms*, 31:271–290, 1999.

[3] Don Coppersmith. Solving linear equations over GF(2): block Lanczos algorithm. *Linear Algebra Appl.*, 192:33–60, 1993. Computational linear algebra in algebraic and related problems (Essen, 1992).

[4] Don Coppersmith. Solving homogeneous linear equations over GF(2) via block Wiedemann algorithm. *Math. Comp.*, 62(205):333–350, 1994.

[5] C. Lanczos. *Applied Analysis*. Prentice Hall,, Englewood Cliffs, NJ, 1956.

[6] Peter L. Montgomery. A block Lanczos algorithm for finding dependencies over GF(2). In *Advances in cryptology—EUROCRYPT '95 (Saint-Malo, 1995)*, volume 921 of *Lecture Notes in Comput. Sci.*, pages 106–120. Springer, Berlin, 1995.

[7] M. Peterson. Parallel block lanczos for solving large binary systems. Master's thesis, Texas Tech University, 2006.

[8] Nicolas Sendrier. On the dimension of the hull. *SIAM Journal on Discrete Mathematics*, 10(2):282–293, 1997.