A modified block Lanczos algorithm with fewer vectors

Emmanuel Thomé

▶ **To cite this version:**

Emmanuel Thomé. A modified block Lanczos algorithm with fewer vectors. Joppe W. Bos; Arjen K. Lenstra. Topics in Computational Number Theory inspired by Peter L. Montgomery, Cambridge University Press, pp.175-188, 2017, 978-1-107-10935-3. 10.1017/9781316271575.008 . hal-01293351

HAL Id: hal-01293351
https://inria.hal.science/hal-01293351

Submitted on 24 Mar 2016

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A modified block Lanczos algorithm with fewer vectors

Emmanuel Thomé

INRIA Nancy / LORIA
615 rue du jardin botanique
54600 Villers-lès-Nancy

Abstract. The block Lanczos algorithm proposed by Peter Montgomery is an efficient means to tackle the sparse linear algebra problem which arises in the context of the number field sieve factoring algorithm and its predecessors. We present here a modified version of the algorithm, which incorporates several improvements: we discuss how to efficiently handle homogeneous systems and how to reduce the number of vectors stored in the course of the computation. We also provide heuristic justification for the success probability of our modified algorithm.
While the overall complexity and expected number of steps of the block Lanczos is not changed by the modifications presented in this article, we expect these to be useful for implementations of the block Lanczos algorithm where the storage of auxiliary vectors sometimes has a non-negligible cost.

1 Linear systems for integer factoring

For factoring a composite integer N, algorithms based on the technique of combination of congruences look for several pairs of integers (x, y) such that

$$x^2 \equiv y^2 \mod N.$$

This equality is hoped to be non trivial for at least one of the obtained pairs, letting $\gcd(x - y, N)$ unveil a factor of the integer N.

Several algorithms use this strategy: the CFRAC algorithm, the quadratic sieve and its variants, and the number field sieve. Pairs (x, y) as above are obtained by combining relations which have been collected as a step of these algorithms. Relations are written multiplicatively as a set of valuations. All the algorithms considered seek a multiplicative combination of these relations which can be rewritten as an equality of squares. This is achieved by solving a system of linear equations defined over $\mathbb{F}_2$, where equations are parity constraints on

⋆ This article is based on work by the author contributed as chapter 7 in *Topics in Computational Number Theory inspired by Peter L. Montgomery*, by Joppe W. Bos and Arjen K. Lenstra, to be published by Cambdridge University Press.
⋆⋆ April 8, 2016; Version for this file: `d76dfc6`

each valuation considered, and unknowns indicate whether or not relations are to be selected as part of the combination.

We are therefore facing a linear algebra problem. Writing the relations collected as the rows of a matrix M with coefficients in $\mathbb{F}_2$, we are to find several solutions to the homogeneous linear system

$$x^T M = 0.$$

To fix notations, we let the matrix M be square of size $N \times N$. It is noteworthy that the matrix M is extremely sparse, as can be illustrated by data from some factoring experiments: for the factoring of RSA-512 in 1999, the matrix M had $N \approx 7 \times 10^6$ and 62 non-zero coefficients per row, and for the RSA-768 factorization in 2009, the matrix M had $N \approx 2 \times 10^8$ and 144 non-zero coefficients per row.

This sparsity property can be exploited to yield efficient algorithms which solve the linear system in a "black-box" fashion, that is, without ever modifying the matrix M. The only access to the matrix M which is allowed to such algorithms is the operation of multiplying M (or its transpose) by a vector, and obtain the result. The interesting black-box algorithms are those which solve the linear system using at most $O(N)$ times this operation. For sparse matrices, this approach is considerably cheaper than "dense" algorithms which do not exploit the sparsity property, with regard to both the time and space complexity (which would, for dense algorithms, be $O(N^\omega)$ and $O(N^2)$).

2 The standard Lanczos algorithm

Dealing with sparse linear systems is an important topic which goes beyond computational number theory. Among the sparse algorithms which can be employed (reviewed in early works such as [6]), we find the conjugate gradient and the Lanczos algorithms, which were both originally stated in the context of solving numerical systems occurring, for example, in the context of the solution of partial differential equations. With some adaptation work, it is possible to use these algorithms over finite fields, with limitations which we will mention in §3. The Wiedemann algorithm [10] was proposed as a method particularly well adapted to finite fields. We will discuss in §9 how it compares with the Lanczos and block Lanczos algorithms.

As a first step towards presenting the block Lanczos algorithm, we give here an overview of how the standard Lanczos algorithm can be used to solve homogeneous or inhomogeneous linear systems over finite fields. Arguments which appear in the justification of the standard Lanczos are also important to the block Lanczos context, which explains this preliminary overview. Within this section, we assume that the base field is $\mathbb{F}_p$ for some prime p.

Briefly put, the Lanczos algorithm is the Gram-Schmidt orthogonalization process applied to a Krylov subspace. We need to work with a symmetric matrix A defined over $\mathbb{F}_p$. Different problems can be stated, for example depending on whether we intend to solve a homogeneous or inhomogeneous linear system.

Another distinction comes from the linear system which we want to solve in the first place. While in some cases it does indeed define a symmetric matrix A, it may also be that we form A as $A = MM^T$, and solve a linear system involving A as a derived means of solving one involving M. Such a strategy would be natural in the prospect of solving the linear systems as defined in §1. In that case, the matrix A is never actually computed, and the black box "multiplication by A" is instead realized as the composition of the two black boxes multiplying by M^T and M.

For expository purposes, we assume in this section that we have a right-hand side vector $b \in \mathbb{F}_p^N$, and intend to solve for $x \in \mathbb{F}_p^N$ the equation

$$Ax = b.$$

The matrix A being symmetric, we may consider the inner product defined from A as $v^T Aw$ for vectors $v, w \in \mathbb{F}_p^N$. We say that v and w are A-orthogonal whenever $v^T Aw = 0$. A vector is A-isotropic if it is A-orthogonal to itself.

The Lanczos algorithm focuses on the sequence of Krylov subspaces of $\mathbb{F}_p^N$ defined as $V_i = \left\langle v_0, Av_0, A^2v_0, \ldots, A^i v_0 \right\rangle$, where $v_0 = b$. It is clear that the sequence of subspaces $(V_i)_{i \geq 0}$ is strictly increasing up to some index, and then stationary.

We define a sequence of vectors $(v_i)_{i \geq 0}$, computed so as to satisfy the two following conditions:

$$v_i \text{ is } A\text{-orthogonal to } v_j \text{ whenever } i \neq j, \tag{1}$$

$$V_i = \langle v_0, \ldots, v_i \rangle . \tag{2}$$

We proceed by induction, and assume that a sequence of vectors v_0 to v_i has been computed so that the two conditions above hold. We now see how to compute v_{i+1}. We begin by noting (using condition (2) inductively) that

$$V_{i+1} = \langle v_0 \rangle + AV_i = \langle v_0 \rangle + AV_{i-1} + \langle Av_i \rangle = V_i + \langle Av_i \rangle$$

so that setting v_{i+1} to be any vector within the affine subspace $V_i + Av_i$ fulfils condition (2) for index $i + 1$. In order to satisfy condition (1), we let:

$$v_{i+1} = Av_i - \sum_{j \leq i} \frac{v_j^T A^2 v_i}{v_j^T A v_j} v_j.$$

We leave for further discussion the important question of the non-degeneracy of the denominators in the expression of v_{i+1}.

It turns out that the equation above defining v_{i+1} can be simplified. Indeed, because $Av_j \in V_{j+1}$, we have that $v_j^T A^2 v_i = 0$ whenever $j < i - 1$. This implies that only two terms in the sum above are non-zero, yielding the following shorter equation for defining v_{i+1}:

$$v_{i+1} = Av_i - c_{i+1,i} v_i - c_{i+1,i-1} v_{i-1},$$

$$c_{i+1,i} = \frac{v_i^T A^2 v_i}{v_i^T A v_i} v_i, \quad c_{i+1,i-1} = \frac{v_{i-1}^T A^2 v_i}{v_{i-1}^T A v_{i-1}} v_{i-1}.$$

Note also that we have $Av_{i-1} \in v_i + V_{i-1}$, so that $v_{i-1}^T A^2 v_i = v_i^T A v_i$. We can then simplify the expression of $c_{i+1,i-1}$ as

$$c_{i+1,i-1} = \frac{v_i^T A v_i}{v_{i-1}^T A v_{i-1}} v_{i-1}.$$

The sequence of vectors $(v_i)_{i \geq 0}$ can thus be computed with a simple recurrence procedure, requiring only a short amount of history to be updated from each iteration to the next (namely, the vectors v_{i+1} and v_i as well as the scalar $v_i^T A v_i$).

We now discuss the termination of the computation of the sequence of vectors $(v_i)_{i \geq 0}$. It is clear that v_{i+1} can be computed only as long as the following condition holds:

$$\forall j \leq i,\ v_j^T A v_j \neq 0. \tag{3}$$

We assume that condition (3) holds until some index m (not included), and that $v_m = 0$. This implies $V_m = V_{m-1}$. Define now x as

$$x = \sum_{i<m} \frac{v_i^T b}{v_i^T A v_i} v_i.$$

By construction*, we have $Ax - b \in V_{m-1}$. Vectors v_i for indices $i < m$ form an A-orthogonal basis of V_{m-1}, therefore $(Ax-b)^T A v_i = 0$ for all $i < m$ because of the expression of x. It follows from (1) and (3) that we have $Ax = b$. Computing the summands of x can be done at the same time as the sequence $(v_i)_{i \geq 0}$ is computed, adding the need for one extra vector of $\mathbb{F}_p^N$.

Condition (3) may fail to hold without reaching $v_m = 0$, however. This is because the positive characteristic setting does not forbid A-isotropic vectors: it may happen that $v_m^T A v_m = 0$ without $v_m = 0$. In this case, the algorithm fails. In [4], Eberly and Kaltofen show that condition (3) is equivalent to the matrix $H(A,b) = (b^T A^{i+j+1} b)_{0 \leq i < m}$ being of generic rank profile (all leading principal minors are non-zero). They further show how it is possible to control the failure probability with appropriate randomization, under the assumption that the coefficient field is large enough.

3 The case of characteristic two

When $p = 2$, the standard Lanczos algorithm cannot work, as A-isotropic vectors are bound to occur. This problem is an incurable failure condition in the finite field case, but an analogous mishap can also be encountered in the numerical case: if some $v_i^T A v_i$ happens to be very close to zero, then numerical instability occurs.

Techniques to address this issue have been proposed in the numerical context quite early on, namely the look-ahead Lanczos algorithm [9] which suggests to

* This argument uses the fact that we have chosen $v_0 = b$. Had we chosen v_0 arbitrarily, then we would need to assume $b \in V_m$.

compute v_{i+1} from several of the previous iterates. In the context of integer factorization and linear systems defined over $\mathbb{F}_2$, early techniques suggested, e.g. in [6], to overcome the issue of A-isotropic vectors were quite inefficient, requiring for example to do all computations in a field $\mathbb{F}_{2^k}$ for some k. Coppersmith [2] and Montgomery [7], in a somewhat simpler form, proposed to efficiently solve this problem by taking inspiration from the look-ahead technique, and more importantly by considering several vectors simultaneously.

The theoretical benefit is that if we consider a block of n vectors $\mathbf{v}$ (represented by a matrix of size $N \times n$), the matrix $\mathbf{v}^T A\mathbf{v}$ might fail to be invertible, but its rank defect may be expected to be reasonably small, thereby allowing the algorithm to proceed.

Considering blocks of vectors is also a great practical benefit when dealing with sparse matrices defined over $\mathbb{F}_2$. Multiplying a sparse matrix by a vector requires, for each matrix coefficient, to access a single coefficient (hence a single bit) of the input vector. Despite the fact that memory access probably reaches the nearby bits of the input vector as well, these do not matter and one expects that their value is most often discarded. When a block of n vectors is considered, and n is equal to the machine word size (say, $n = 64$), then there is a natural alternative way to proceed. Storing blocks of vectors as N-element arrays of n-bit machine words, it is possible to compute simultaneously the product of a sparse matrix by a block of vectors in essentially the same number of distinct memory accesses than required for doing a single matrix-times-vector operation. The question is then whether such an approach leads to a modification of the Lanczos algorithm which requires fewer iterations.

4 Orthogonalizing a sequence of subspaces

The key to the block Lanczos algorithm is the idea of considering a sequence of subspaces of dimension larger than 1. We use boldface letters to denote blocks of n vectors, and the notation $\langle \mathbf{v} \rangle$ denotes the subspace of $\mathbb{F}_p^N$ spanned by the n columns of $\mathbf{v}$. We extend this trivially to $\langle \mathbf{v}_0, \mathbf{v}_1 \rangle$ As in the case of the standard Lanczos, we define notions which are related to the inner product defined by the matrix A. We say that spaces $\langle \mathbf{v} \rangle$ and $\langle \mathbf{w} \rangle$ are A-orthogonal whenever $\mathbf{v}^T A\mathbf{w} = 0$. It is clear that $\mathbf{v}^T A\mathbf{w}$ is an $n \times n$ matrix with coefficients in $\mathbb{F}_p$.

We first describe a naive extension of the Lanczos algorithm to the setting of blocks of vectors, and explain why it does not work (at least not if p may be small). We need to define an analogue to the sequence of mutually orthogonal vectors $(v_i)_{i \geq 0}$ considered in the standard Lanczos algorithm. Let us fix an arbitrary vector block $\mathbf{v}_0$ as a starting point (to be discussed in §7). We may attempt to define a sequence of vector spaces with $\mathbf{v}_i^T A\mathbf{v}_i$ non-singular as follows.

- Set $\mathbf{t} = A\mathbf{v}_i - \sum_{j \leq i} \mathbf{v}_j (\mathbf{v}_j^T A\mathbf{v}_j)^{-1} (\mathbf{v}_j^T A^2 \mathbf{v}_i)$.
- Define $\mathbf{v}_{i+1}$ as a maximal set of columns within $\mathbf{t}$ so that $\mathbf{v}_{i+1}^T A\mathbf{v}_{i+1}$ is invertible.

The key problem with the approach above is that $\mathbf{v}_{i+1}$ is a block of possibly fewer vectors than $\mathbf{v}_i$: when $\mathbf{t}^T A\mathbf{t}$ above is not of full rank, some vectors are discarded and not selected in $\mathbf{v}_{i+1}$. This implies that after some steps, the expected dimension of the block $\langle \mathbf{v}_i \rangle$ collapses to zero, with no further progress possible. (A rule of thumb expecting a rank defect of 1 with probability $\frac{1}{p}$ predicts that no more than np steps can be done before this collapse.)

To address this issue, Montgomery suggested an idea related to the look-ahead Lanczos [9] (but apparently discovered independently): allow to build orthogonal subspaces from a larger number of the previous iterates. For notational ease, we depart slightly here from the notations used in [7]. We define sequences $(\mathbf{v}_i)_{i\geq 0}$, $(\mathbf{d}_i)_{i\geq 0}$, and $(\mathbf{w}_i)_{i\geq 0}$, where $\mathbf{v}_i \in \mathbb{F}_p^{N\times n}$, $\mathbf{w}_i \in \mathbb{F}_p^{N\times n}$, and $\mathbf{d}_i \in \mathbb{F}_p^{n\times n}$ diagonal with entries in $\{0,1\}$. We require, for all $i \geq 0$:

$$\mathbf{w}_i = \mathbf{v}_i \mathbf{d}_i,$$

$$[\mathbf{w}_i^T A \mathbf{w}_i]_{\mathbf{d}_i} \neq 0 \text{ (principal minor marked by } \mathbf{d}_i) \tag{4}$$

$$\mathbf{w}_j^T A \mathbf{v}_i = 0 \text{ whenever } j < i. \tag{5}$$

The diagonal matrix $\mathbf{d}_i$ essentially encodes the choice of a subset of $\{1,\ldots,n\}$, which justifies the notation $[\mathbf{w}_i^T A\mathbf{w}_i]_{\mathbf{d}_i}$ for the principal minor attached to this set (note that we have $[\mathbf{v}_i^T A\mathbf{v}_i]_{\mathbf{d}_i} = [\mathbf{w}_i^T A\mathbf{w}_i]_{\mathbf{d}_i}$). It is clear that condition (5) also implies $\mathbf{w}_i^T A\mathbf{w}_j = 0$ whenever $i \neq j$.

In Montgomery's algorithm, the sequence of orthogonal subspaces is the sequence $\mathbf{w}_i$, which are formed from as many columns from $\mathbf{v}_i$ as possible (condition 4 imposes that the inner product defined by A is non-degenerate on $\langle \mathbf{w}_i \rangle$). Vectors from $\mathbf{v}_i$ which are *not* selected in $\mathbf{w}_i$, instead of being dropped, are considered again for selection in the next iterations.

As for the standard Lanczos algorithm, we explain how the conditions above can be satisfied with an explicit inductive construction. The starting point of each iteration is the vector block $\mathbf{v}_i$. The first step is to compute $\mathbf{d}_i$ (and hence $\mathbf{w}_i$) so as to satisfy condition (4). In a second step, we compute $\mathbf{v}_{i+1}$ so as to satisfy condition (5).

5 Construction of the next iterate

We first discuss how to compute $\mathbf{d}_i$ (and hence $\mathbf{w}_i$) from $\mathbf{v}_i$. We need the following lemma:

Lemma 1. *Let $\mathbf{X} \in \mathbb{F}_p^{n\times n}$ be a symmetric matrix of rank r, and $S \subset \{1,\ldots,n\}$ be indices of r independent columns of $\mathbf{X}$. Then the principal minor $[\mathbf{X}]_S$ is non-zero.*

To see this, assume without loss of generality that $S = \{1,\ldots,r\}$. Columns of indices $r+1$ and above can be expressed as combinations of the first r columns. We may write a matrix $\mathbf{\Sigma} = \left(\begin{array}{c|c} 1_r & * \\ \hline 0 & 1_{n-r} \end{array}\right)$ so that $\mathbf{X\Sigma}$ has only its r first columns non zero. The matrix $\mathbf{X}' = \mathbf{\Sigma}^T\mathbf{X\Sigma}$ has its last $n-r$ rows and columns equal

to zero, so that only its leading $r \times r$ submatrix is non-zero. Since $\mathbf{X}'$ has rank r and this submatrix coincides with the leading $r \times r$ submatrix of $\mathbf{X}$, this is saying that $[\mathbf{X}]_S \neq 0$, as claimed.

Lemma 1 implies that computing $\mathbf{d}_i$ so as to satisfy condition (4) only amounts to Gaussian elimination on the $n \times n$ matrix $\mathbf{v}_i^T A \mathbf{v}_i$.

An inverse of the submatrix whose row and column indices are encoded by $\mathbf{d}_i$ can be computed from the same Gaussian elimination procedure. Therefore, we assume that a by-product of the computation of $\mathbf{d}_i$ is an $n \times n$ matrix $\mathbf{w}_i^{\text{inv}}$ such that:

$$\begin{aligned} \mathbf{w}_i^{\text{inv}} &= \mathbf{w}_i^{\text{inv}} \mathbf{d}_i = \mathbf{d}_i \mathbf{w}_i^{\text{inv}}, \\ \mathbf{d}_i &= \mathbf{w}_i^{\text{inv}} (\mathbf{w}_i^T A \mathbf{w}_i) = \mathbf{w}_i^{\text{inv}} (\mathbf{v}_i^T A \mathbf{v}_i) \mathbf{d}_i \end{aligned}$$

The former condition above expresses the fact that $\mathbf{w}_i^{\text{inv}}$ is zero outside the row and column indices encoded by $\mathbf{d}_i$, while the latter expresses the fact that it is an inverse to the corresponding submatrix. Note that this construction implies that $\mathbf{w}_i^{\text{inv}}$ is symmetric.

Assuming $\mathbf{d}_i$ and $\mathbf{w}_i$ have been derived from $\mathbf{v}_i$, we now build $\mathbf{v}_{i+1}$ from $A\mathbf{w}_i$ and $\mathbf{v}_i$. Given that $A\mathbf{w}_i$ has, by construction, $n - \text{rank}(\mathbf{d}_i)$ zero columns, we complete it with the columns of $\mathbf{v}_i$ which were not selected in $\mathbf{w}_i$. We then write $\mathbf{v}_{i+1}$ as follows.

$$\begin{aligned} \mathbf{t} &= A\mathbf{v}_i \mathbf{d}_i + \mathbf{v}_i (1 - \mathbf{d}_i), \\ \mathbf{v}_{i+1} &= \mathbf{t} - \sum_{j \leq i} \mathbf{w}_j \mathbf{w}_j^{\text{inv}} \mathbf{w}_j^T A \mathbf{t}. \end{aligned}$$

It is clear from the quantities computed so far that $\mathbf{v}_{i+1}$ is A-orthogonal to $\mathbf{w}_j$ for all $j \leq i$, which is condition (5). We remark that we could have used, as Montgomery does, the vector block $A\mathbf{w}_i + \mathbf{v}_i = \mathbf{t} + \mathbf{w}_i$ instead of the value chosen above for $\mathbf{t}$, and this would have led to the same value for $\mathbf{v}_{i+1}$.

6 Simplifying the recurrence equation

The previous section defines a complete set of equations for determining $\mathbf{v}_i$. However the expression above for $\mathbf{v}_{i+1}$ is a very deep recurrence, which would lead to poor time and space complexity. We therefore need, as is done in the standard Lanczos algorithm, to show that the recurrence equation can be simplified.

We first restate the recurrence relations from the previous section, and introduce some auxiliary notation $\mathbf{c}_{i+1,j}$.

$$\begin{aligned} \mathbf{v}_{i+1} &= A\mathbf{v}_i \mathbf{d}_i + \mathbf{v}_i (1 - \mathbf{d}_i) - \sum_{j \leq i} \mathbf{w}_j \mathbf{c}_{i+1,j}, \\ \mathbf{c}_{i+1,j} &= \mathbf{w}_j^{\text{inv}} \mathbf{w}_j^T A (A\mathbf{v}_i \mathbf{d}_i + \mathbf{v}_i (1 - \mathbf{d}_i)). \end{aligned} \tag{6}$$

Condition (5) implies that the second summand in the expression of $\mathbf{c}_{i+1,j}$ is zero for $j < i$. We now examine the first summand for $j < i$. Consider equation (6) for index j, and multiply by $\mathbf{d}_j$. We obtain:

$$\begin{aligned} A\mathbf{w}_j &= \mathbf{v}_{j+1}\mathbf{d}_j + \mathcal{O}(\langle \mathbf{w}_0, \ldots, \mathbf{w}_j \rangle), \\ \mathbf{w}_j^T A^2 \mathbf{w}_i &= \mathbf{d}_j \mathbf{v}_{j+1}^T A \mathbf{v}_i \mathbf{d}_i, \end{aligned}$$

where the notation $\mathcal{O}(V)$, for V a subspace of $\mathbb{F}_p^N$, denotes any vector block whose columns belong V.

The equations above yield the following simpler form for $\mathbf{c}_{i+1,i-1}$:

$$\mathbf{c}_{i+1,i-1} = \mathbf{w}_{i-1}^{\text{inv}} \mathbf{v}_i^T A \mathbf{v}_i \mathbf{d}_i.$$

Better, for $j+1 < i$, we consider equation (6) for index $j+1$ and multiply it by $1 - \mathbf{d}_{j+1}$. We obtain the following equation, from which we rewrite $\mathbf{v}_{j+1}$ in an interesting way:

$$\begin{aligned} \mathbf{v}_{j+2}(1 - \mathbf{d}_{j+1}) &= \mathbf{v}_{j+1}(1 - \mathbf{d}_{j+1}) + \mathcal{O}(\langle \mathbf{w}_0, \ldots, \mathbf{w}_{j+1} \rangle), \\ \mathbf{v}_{j+1} &= \mathbf{w}_{j+1} + \mathbf{v}_{j+1}(1 - \mathbf{d}_{j+1}) \\ &= \mathbf{v}_{j+2}(1 - \mathbf{d}_{j+1}) + \mathcal{O}(\langle \mathbf{w}_0, \ldots, \mathbf{w}_{j+1} \rangle). \end{aligned}$$

This implies, for $j = i-1$, $j = i-2$, and more generally for any $j < i$ (repeatedly using the last fact):

$$\begin{aligned} \mathbf{c}_{i+1,i-1} &= \mathbf{w}_{i-1}^{\text{inv}} \mathbf{v}_i^T A \mathbf{v}_i \mathbf{d}_i, \\ \mathbf{c}_{i+1,i-2} &= \mathbf{w}_{i-2}^{\text{inv}} (1 - \mathbf{d}_{i-1}) \mathbf{v}_i^T A \mathbf{v}_i \mathbf{d}_i, \\ \mathbf{c}_{i+1,j} &= \mathbf{w}_j^{\text{inv}} \left(\prod_{k=j+1}^{i-1} (1 - \mathbf{d}_k) \right) \mathbf{v}_i^T A \mathbf{v}_i \mathbf{d}_i \end{aligned} \tag{7}$$

We remark that this expression for $\mathbf{c}_{i+1,i-2}$ is simpler than in [7].

In the normal course of the computation, it is easy to ensure that $(1-\mathbf{d}_i)(1-\mathbf{d}_{i+1}) = 0$: this expresses the fact that column indices which are not selected among the independent columns in $\mathbf{v}_i^T A \mathbf{v}_i$ used to define $\mathbf{d}_i$ have to be given priority when defining $\mathbf{d}_{i+1}$ at the next step. As long as this can be achieved, we obtain that whenever $j \leq i-3$, we have $\mathbf{c}_{i+1,j} = 0$. In [7], Montgomery does exactly like this, and computes each iterate $\mathbf{v}_{i+1}$ with access to only the three previous iterates $\mathbf{v}_i$, $\mathbf{v}_{i-1}$, and $\mathbf{v}_{i-2}$.

The particular form of equation (7), however, allows to write a simpler recurrence, which has the advantage of limiting the storage needs of the algorithm. Define

$$\begin{aligned} \mathbf{p}_i &= \mathbf{v}_{i-1}\mathbf{w}_{i-1}^{\text{inv}} + \mathbf{v}_{i-2}\mathbf{w}_{i-2}^{\text{inv}}(1 - \mathbf{d}_{i-1}) + \cdots, \\ &= \sum_{j<i} \mathbf{v}_j \mathbf{w}_j^{\text{inv}} \prod_{k=j+1}^{i-1} (1 - \mathbf{d}_k). \end{aligned}$$

By equations (6) and (7), we see that the contribution of all iterates before $\mathbf{w}_i$ in the expression of $\mathbf{v}_{i+1}$ can be simplified as:

$$\sum_{j<i} \mathbf{w}_j \mathbf{c}_{i+1,j} = \mathbf{p}_i \mathbf{v}_i^T A \mathbf{v}_i \mathbf{d}_i.$$

The computation of the sequence of vector blocks $(\mathbf{v}_i)_{i\geq 0}$ from a starting vector block $\mathbf{v}_0$ can now be summarized. At the start, we have $\mathbf{p}_0 = 0$. For all $i \geq 0$, we proceed through the following steps.

- Compute $\mathbf{v}_i^T A \mathbf{v}_i$ and $\mathbf{v}_i^T A^2 \mathbf{v}_i$. Deduce $\mathbf{d}_i$ (giving, or not, priority to indices not selected in $\mathbf{d}_{i-1}$ — it makes no difference) and $\mathbf{w}_i^{\text{inv}}$. If $\mathbf{d}_i = 0$, terminate (see §7).
- Compute

$$\begin{aligned}
\mathbf{c}_{i+1,i} &= \mathbf{w}_i^{\text{inv}}(\mathbf{v}_i^T A^2 \mathbf{v}_i \mathbf{d}_i + \mathbf{v}_i^T A \mathbf{v}_i (1 - \mathbf{d}_i)), \\
\mathbf{v}_{i+1} &= A\mathbf{v}_i \mathbf{d}_i + \mathbf{v}_i(1 - \mathbf{d}_i) - \mathbf{v}_i \mathbf{c}_{i+1,i} - \mathbf{p}_i \mathbf{v}_i^T A \mathbf{v}_i \mathbf{d}_i, \\
\mathbf{p}_{i+1} &= \mathbf{v}_i \mathbf{w}_i^{\text{inv}} + \mathbf{p}_i(1 - \mathbf{d}_i)
\end{aligned}$$

- Memorize $\mathbf{v}_{i+1}$ and $\mathbf{p}_{i+1}$ for the next iteration.

7 Termination

As the computation of the sequence of vector block proceeds, we clearly have

$$\begin{aligned}
\langle \mathbf{w}_0, \ldots, \mathbf{w}_i \rangle &\subset \langle \mathbf{v}_0, A\mathbf{v}_0, \ldots, A^k \mathbf{v}_0, \ldots \rangle, \\
\dim \langle \mathbf{w}_0, \ldots, \mathbf{w}_i \rangle &= \sum_{j \leq i} \operatorname{rank} \mathbf{d}_j = \sum_{j \leq i} \operatorname{rank} \mathbf{v}_j^T A \mathbf{v}_j, \\
\dim \langle \mathbf{v}_0, A\mathbf{v}_0, \ldots, A^k \mathbf{v}_0, \ldots \rangle &\leq N.
\end{aligned}$$

Therefore, the number of iterations can be studied by first examining the expected rank of $\mathbf{v}_j^T A \mathbf{v}_j$. Montgomery writes in [7] the generating function for the rank defect of an arbitrary $n \times n$ symmetric matrix over $\mathbb{F}_p$. For $p = 2$, the result obtained is that the expected rank defect is 0.764.... We thus have $\mathrm{E}[\operatorname{rank} \mathbf{d}_i] \approx N - 0.764$, from which we expect that at most an expected value of $\frac{N}{n-0.764}$ iterations are computed.

The actual termination condition which causes the iterative process to stop at index m (more exactly, become stationary, if we consider $\mathbf{w}_m^{\text{inv}} = 0$ to be a legitimate value) is when we reach $\mathbf{d}_m = 0$, which means that $\mathbf{v}_m^T A \mathbf{v}_m = 0$. When the block dimension n is exceptionally small, this might happen sooner than the expected value computed above, out of bad luck. We consider here that the block dimension is large enough, so that this situation does not happen.

Heuristically, we expect a large intersection of $\langle \mathbf{v}_m \rangle$ with the null space of A, which allows to find close to n solutions to the homogeneous linear system $Ax = 0$.

We provide here some justification for this fact. Let δ_0 be a vector block with $A\delta_0 = 0$, and let $\mathbf{v}_0$ be an arbitrary vector block. We consider the two sequences corresponding to $\mathbf{v}_0$ and $\mathbf{v}'_0 = \mathbf{v}_0 + \delta_0$. It is easy to see that both sequences evolve synchronously, as the matrices $\mathbf{v}_i^T A \mathbf{v}_i$ are equal for both sequences at each step. Let $\mathbf{\Delta}_i = \left(\mathbf{v}'_i - \mathbf{v}_i \middle| \mathbf{p}'_i - \mathbf{p}_i \right)$. We have

$$\begin{aligned} \mathbf{\Delta}_{i+1} &= \mathbf{\Delta}_i \times \mathfrak{S}_i, \\ \mathfrak{S}_i &= \left(\begin{array}{c|c} (1 - \mathbf{d}_i) - \mathbf{c}_{i+1,i} & \mathbf{w}_i^{\text{inv}} \\ \hline -\mathbf{v}_i^T A \mathbf{v}_i \mathbf{d}_i & (1 - \mathbf{d}_i) \end{array} \right). \end{aligned}$$

We claim that $\mathfrak{S}_i$ is invertible. Indeed, we have:

$$\left(\begin{array}{c|c} 1 - \mathbf{d}_i & \mathbf{d}_i \\ \hline \mathbf{d}_i & 1 - \mathbf{d}_i \end{array} \right) \times \mathfrak{S}_i \times \left(\begin{array}{c|c} 1 & 0 \\ \hline \mathbf{v}_i^T A^2 \mathbf{v}_i \mathbf{d}_i & 1 - \mathbf{d}_i + \mathbf{d}_i \mathbf{v}_i^T A \mathbf{v}_i \mathbf{d}_i \end{array} \right) = \left(\begin{array}{c|c} (1 - \mathbf{d}_i) - \mathbf{d}_i \mathbf{v}_i^T A \mathbf{v}_i \mathbf{d}_i & 0 \\ \hline -\mathbf{w}_i^{\text{inv}} (\mathbf{v}_i^T A \mathbf{v}_i (1 - \mathbf{d}_i)) & 1 \end{array} \right).$$

The latter matrix is clearly of full rank. A consequence is that $\operatorname{rank} \mathbf{\Delta}_m = \operatorname{rank} \mathbf{\Delta}_0$, and that $\operatorname{rank}(\mathbf{v}'_m - \mathbf{v}_m)$ is expected to be close to $\operatorname{rank}(\delta_0)$. We thus have no reason to expect that $\mathbf{v}_m$ is an abnormally poor supply of elements of the null space of A.

In the case which is relevant for integer factorization problems, we want to solve the equation $x^T M = 0$, and use the block Lanczos algorithm with $A = MM^T$. In this case, we expect (as above, heuristically) that the intersection of $\langle \mathbf{v}_i \rangle$ with the (left) null space of M is large enough to obtain close to n solutions to the linear system (provided the null space itself is large enough).

The block Lanczos algorithm can also be used to solve inhomogeneous linear systems, to some extent. In this case, we assume that the null space dimension is small compared to the block dimension n. We want to solve $Ax = b$ for several vectors b. We set the starting vector block $\mathbf{v}_0$ with our vectors b, and complete with random vectors so as to form an n-dimensional vector block. We compute

$$\mathbf{x} = \sum_{i<m} \mathbf{v}_i \mathbf{w}_i^{\text{inv}} \mathbf{v}_i^T \mathbf{v}_0.$$

Note that $\mathbf{x}$ can be computed online at little extra cost, since for $i \geq 1$ we have

$$\left(\mathbf{v}_0^T \mathbf{v}_{i+1} \middle| \mathbf{v}_0^T \mathbf{p}_{i+1} \right) = \left(\mathbf{v}_0^T \mathbf{v}_i \middle| \mathbf{v}_0^T \mathbf{p}_i \right) \times \mathfrak{S}_i$$

with $\mathfrak{S}_i$ as above. Maintaining the evolution of $\mathbf{x}$ throughout the computation of the sequence costs some extra memory.

By construction, we have $A\mathbf{x} - \mathbf{v}_0 \in \langle \mathbf{v}_m \rangle$. In [7], Montgomery argues that heuristically, we have $\langle A\mathbf{v}_m \rangle \subset \langle \mathbf{v}_m \rangle$. Based on the assumption that the null space of A is small enough, we hope to find linear combinations of the columns of $A\mathbf{x} - \mathbf{v}_0$ and $A\mathbf{v}_m$ which provide some solutions to $Ax = b$.

8 Implementation in parallel

Several implementations of the block Lanczos algorithm exist, and adapt reasonably well to parallel computing environments. Different processors (which can be different nodes communicating via message passing, or simply processor threads) can collectively compute the sequence $(\mathbf{v}_i)_{i\geq 0}$. It is useful to organize processors in a two-dimensional (possibly toroidal) mesh, following the explanation in [8]. For simplicity, we assume the mesh has size $d \times d$. Each processor "owns" part of the data: all vectors considered in the algorithm are divided in d^2 fragments, and the matrix M itself is also spread across processes, in d^2 fragments. An example organization, assuming that M has dimension $N_1 \times N_2$ (both assumed to be divisible by d^2), distributes data as follows for the processor on row i and column j (both indexed from 0) within the mesh:

- For vector blocks of size $N_1 \times n$, row indices $[x, x+\frac{N_1}{d^2}-1]$ with $x = (di+j)\frac{N_1}{d^2}$.
- For vector blocks of size $N_2 \times n$, row indices $[x, x+\frac{N_2}{d^2}-1]$ with $x = (dj+i)\frac{N_2}{d^2}$.
- Sub-block of M at position (i, j) when split in blocks of size $\frac{N_1}{d} \times \frac{N_2}{d}$.

In fact, load balancing has to be taken into account, so that the distribution may actually be slightly different, or equivalently we may need to permute rows and columns of B adequately.

In this setting, many operations on vectors can be performed locally. The only collective operation at each step is the multiplication by $A = MM^T$, decomposed into $u^T \leftarrow v^T M$ first, then $v \leftarrow Mu$, where u and v are vector blocks of size $N_2 \times n$ and $N_1 \times n$, respectively. Communication goes as follows. After $u^T \leftarrow v^T M$, processors on the same mesh column need to share their results so as to form N_2/d valid coefficients of the resulting vector u. For the operation $v \leftarrow Mu$, the processors in this same mesh column all need these same N_2/d input coefficients. Therefore, the communication operation required after each of these two products is in fact a pretty common pattern. In the Message Passing Interface, this operation is called "All-reduce", and is usually well optimized and tuned on most serious MPI implementations.

The other operations within each iteration are either of moderate cost (dot products, or multiplication of vectors by $n \times n$ matrices) or totally negligible (arithmetic directly involving $n \times n$ matrices). It should be noted however that the parallelization of the block Lanczos algorithm can only go as far as the communication speed allows, since synchronization has to occur after each multiplication by $A = MM^T$.

9 Recent developments

The block Lanczos algorithm has been successful in factoring projects since its inception, including record computations until 2005. Compared to the block Wiedemann algorithm [3], the block Lanczos algorithm seems to need a smaller number of multiplications of matrices by blocks of vectors. With blocking dimension n, block Lanczos requires $2N/(n - 0.764)$ products in total (counting two

for each iteration). The block Wiedemann algorithm requires instead $\frac{N}{m} + 2\frac{N}{n}$ products, depending on the two blocking dimensions m and n. When these are chosen straightforwardly as $m = n$, the algorithm needs $3N/n$ products. This comparison can shift towards being in favour of the block Wiedemann algorithm in two ways. First, if for example when $m = 4n$ is a valid choice, only $2.25N/n$ products are needed. Also, if large blocking dimensions can be considered (say we use blocking dimensions m' and n' that are two appropriate multiples of n), then by [5, Theorem 7], only $(1 + o(1))N/n$ products are needed, which is better than block Lanczos. However, the reason why the block Wiedemann algorithm has been preferred in most factoring records since 2005 is simply because of the better distribution opportunities it offers, a criterion which has been most important given the composite nature of the hardware platforms used.

One may wonder whether the block Lanczos algorithm can be profitably used in the context of the computation of discrete logarithms, in particular with the number field sieve variants. An artifact of the number field sieve for discrete logarithms, called Schirokauer maps, divides the presentation of the linear algebra problem in two different settings. Given a sparse matrix M defined over a large finite field, the Schirokauer maps form a dense matrix block $\mathbf{S}$ (with very few columns, but with large coefficients) such that the linear system to be solved can be written as $(M \mid \mathbf{S})x = 0$. It is not, however, the only way to proceed: any vector x such that $Mx \in \langle \mathbf{S} \rangle$ is a satisfactory solution. As it turns out, this approach is viable both in the block Lanczos algorithm, as discussed in §7, as well as in the block Wiedemann algorithm, as discussed in [3, §8]. In both cases, this is possible as long as the number of columns of the block $\mathbf{S}$ is less than the block dimension n.

References

1. Bos, J.W., Lenstra, A.K.: Topics in Computational Number Theory inspired by Peter L. Montgomery. Cambdridge University Press (2016), to appear
2. Coppersmith, D.: Solving linear equations over GF(2): Block Lanczos algorithm. Linear Algebra Appl. 192, 33–60 (Jan 1993)
3. Coppersmith, D.: Solving linear equations over GF(2) via block Wiedemann algorithm. Math. Comp. 62(205), 333–350 (Jan 1994)
4. Eberly, W., Kaltofen, E.: On randomized Lanczos algorithm. In: Küchlin, W.W. (ed.) ISSAC '97. p. 176–183. ACM Press (1997), extended abstract
5. Kaltofen, E.: Analysis of Coppersmith's block Wiedemann algorithm for the parallel solution of sparse linear systems. Math. Comp. 64(210), 777–806 (Apr 1995)
6. LaMacchia, B.A., Odlyzko, A.M.: Solving large sparse linear systems over finite fields. In: Menezes, A.J., Vanstone, S.A. (eds.) CRYPTO'90. LNCS, vol. 537, pp. 109–133. Springer, Berlin, Germany, Santa Barbara, CA, USA (Aug 11–15, 1990)
7. Montgomery, P.L.: A block Lanczos algorithm for finding dependencies over gf(2). In: Guillou, L.C., Quisquater, J.J. (eds.) EUROCRYPT'95. LNCS, vol. 921, pp. 106–120. Springer, Berlin, Germany, Saint-Malo, France (May 21–25, 1995)
8. Montgomery, P.L.: Parallel block Lanczos (2000), slides of presentation at RSA-2000, dated January 17, 2000

9. Parlett, B.N., Taylor, D.R., Liu, Z.A.: A look-ahead Lanczos algorithm for unsymmetric matrices. Math. Comp. 44(169), 105–124 (Jan 1985)
10. Wiedemann, D.H.: Solving sparse linear equations over finite fields. IEEE Trans. Inform. Theory IT–32(1), 54–62 (Jan 1986)